Балансировка нагрузки на основе HAproxy **и** VRRP (keepalived)

VRRP

VRRP (Virtual Router Redundancy Protocol) - сетевой протокол, предназначенный для увеличения доступности маршрутизаторов, выполняющих роль шлюза по умолчанию. Это достигается путём объединения группы маршрутизаторов в один виртуальный маршрутизатор и назначения им общего IP-адреса, который и будет использоваться как шлюз по умолчанию для компьютеров в сети

https://ru.wikipedia.org/wiki/VRRP

Особенности:

- failover маршрутизаторов
- быстрое переключение между маршрутизаторами
- работа в режиме Active/Active (балансировка трафика)
- поддержка многими вендорами
- возможность использования нескольких маршрутизаторов в роли резервных

VRRP

Терминология:

VRRP-маршрутизатор (VRRP Router) - маршрутизатор, на котором работает протокол VRRP. Он может участвовать в одном или более виртуальных маршрутизаторах

Виртуальный маршрутизатор (Virtual Router, VR) - группа маршрутизаторов в одной сети с одним VIP и VRID

Владелец IP-адреса (IP Address Owner) - VRRP-маршрутизатор, который использует IP-адрес, назначенный виртуальному маршрутизатору, как реальный IP-адрес, присвоенный интерфейсу

VRRP-объявление (ADVERTISEMENT) - сообщения, которые отправляет Master-маршрутизатор

VRRP

Терминология:

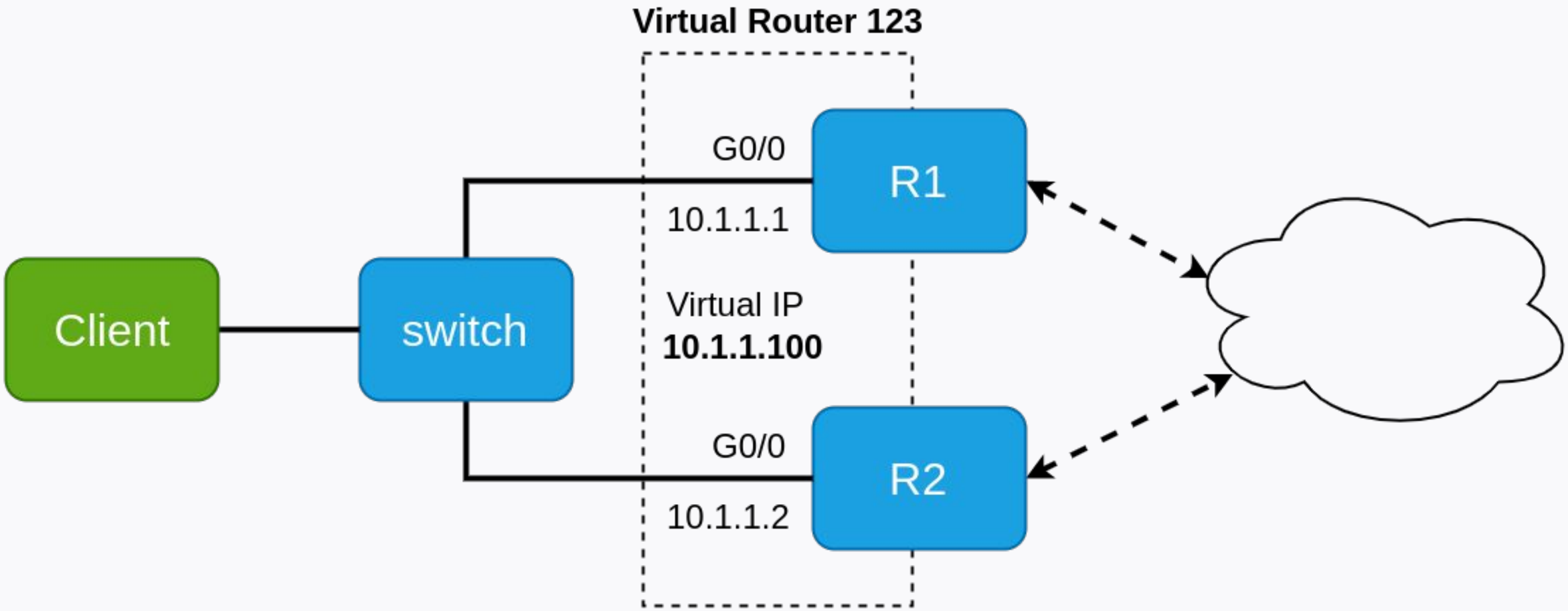
Виртуальный IP-адрес (VIP) - это IP-адрес, присвоенный интерфейсу одного из маршрутизаторов, которые составляют Virtual Router

Virtual Router Master или VRRP Master router - VRRP-маршрутизатор, который отвечает за отправку пакетов, отправленных на IP-адрес, который ассоциирован с виртуальным маршрутизатором, и за ответы на ARP-запросы, отправленные на этот адрес. Если владелец IP-адреса доступен, то он всегда становится Master

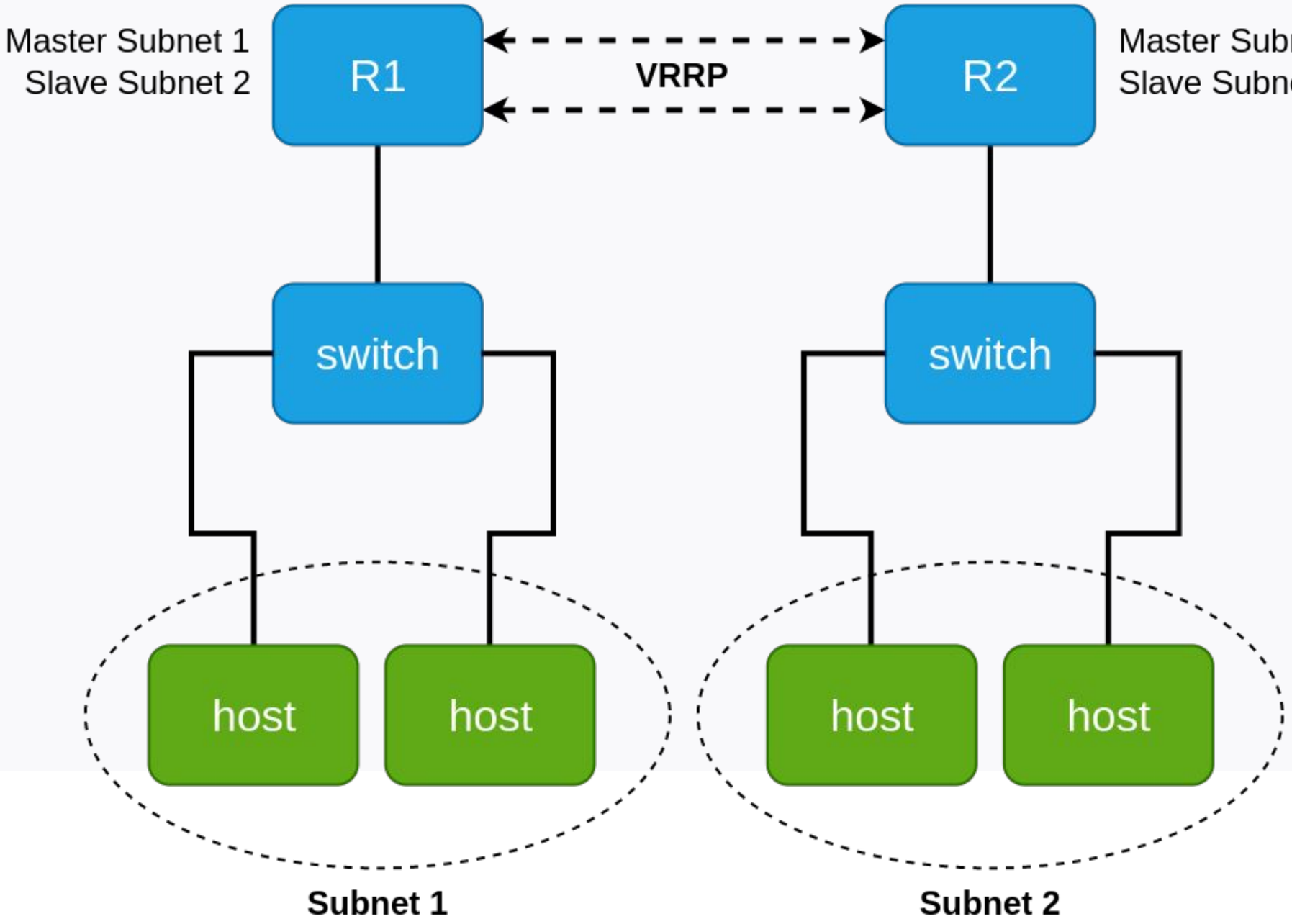
Virtual Router Backup или VRRP Backup router - один или несколько маршрутизаторов, готовых взять на себя роль VRRP Master router, если текущий VRRP Master router станет недоступным

Виртуальный MAC-адрес (Virtual MAC) - 0000:5E00:01xx, где xx — номер группы VRRP

VRRP: **схемы применения**



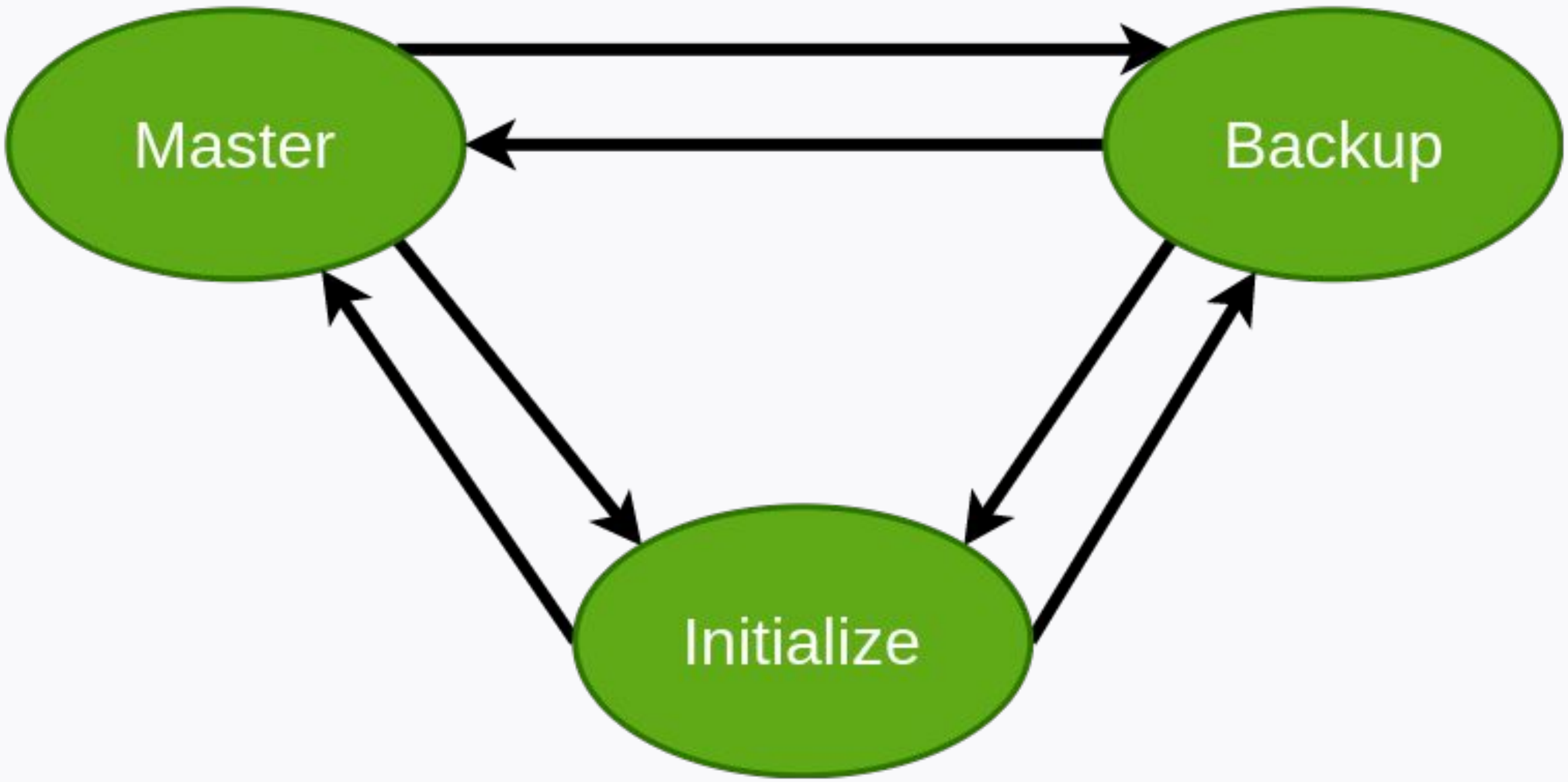
VRRP: **схемы применения**



VRRP: **принцип действия**

1. Маршрутизатор с максимальным приоритетом получает статус **Master Router**
2. Остальные маршрутизаторы получают статус **Backup Router**
3. С интерфейса **Master`a** на MAC адрес 01:00:5e:00:00:xx и групповой адрес 224.0.0.18 multicast`ом отправляются Hello-пакеты
4. Если **Backup Router** не получают сообщения в течении трех **Adver Timer (Master Down Timer)**, то новым **Master Router** становится маршрутизатор с наибольшим приоритетом, либо маршрутизатор с наибольшим IP

VRRP: **принцип действия**



VRRP: **параметры**

Идентификатор виртуального маршрутизатора (VRID) - настраиваемое значение в диапазоне от 1 до 255. Нет значения по умолчанию

Приоритет - значение приоритета (1 - 254)

Резервируемые значения приоритета:

- 255 - владелец адреса
- 0 - мастер

IP-адрес - один или более IP-адресов, присвоенные виртуальному маршрутизатору. Нет значения по умолчанию

Advertisement Interval - временной интервал между отправкой объявлений. По умолчанию 1 секунда

VRRP: **параметры**

Skew Time - время (в секундах), которое используется для отклонения Master Down Interval. Рассчитывается по формуле: ((256 - Priority) / 256)

Master Down Interval - временной интервал, после которого Backup Router станет Master Router. Рассчитывается по формуле: (3 * Advertisement Interval) + Skew time

Режим preempt - контролирует то, будет ли Backup-маршрутизатор с более высоким приоритетом пытаться перехватить на себя роль Master у текущего Master-маршрутизатора с более низким приоритетом

Тип аутентификации - должен быть уникальным. Пакет, тип аутентификации которого не совпадает с настроенным типом аутентификации, или у которого указан неизвестный тип аутентификации, должен быть отброшен. Значения 0, 1 и 2

Данные аутентификации - устанавливаются равным 0

Keepalived

Keepalived - это демон, с помощью которого реализуется отказоустойчивость и балансировка в Linux системах

https://www.keepalived.org

Особенности:

- демон написан на C
- для работы использует модуль ядра **netfilter ip_vs**
- помимо протокола VRRP для более быстрой сходимости имеет реализацию протокола **BFD** https://tools.ietf.org/html/rfc5881

Keepalived

Установка keepalived:

```
yum install -y keepalived
```

Включение параметра ядра net.ipv4.ip_nonlocal_bind:

```
sysctl net.ipv4.ip_nonlocal_bind=1
```

Необходимо для того, чтобы дать возможность демону keepalived добавить виртуальный IP адрес на интерфейс

Keepalived: **пример конфигурации**

Пример конфигурации keepalived на Master Router:

```
vrrp_script chk_haproxy {
  script "killall -0 haproxy"
  interval 2
  weight 2
}

vrrp_instance VI_1 {
  interface eth1
  state MASTER
  virtual_router_id 1
  priority 101
  virtual_ipaddress {
    10.0.26.81
  }
  track_script {
    chk_haproxy
  }
  authentication {
    auth_type PASS
    auth_pass secret_password
        }
}
```

Keepalived: **пример конфигурации**

Пример конфигурации keepalived на Backup Router:

```
vrrp_script chk_haproxy {
  script "killall -0 haproxy"
  interval 2
  weight 2
}

vrrp_instance VI_1 {
  interface eth1
  state MASTER
  virtual_router_id 1
  priority 100
  virtual_ipaddress {
    10.0.26.81
  }
  track_script {
    chk_haproxy
  }
  authentication {
    auth_type PASS
    auth_pass secret_password
        }
}
```

HAproxy

HAProxy - серверное программное обеспечение для обеспечения высокой доступности и балансировки нагрузки для TCP и HTTP-приложений, посредством распределения входящих запросов на несколько обслуживающих серверов

https://ru.wikipedia.org/wiki/HAProxy

Особенности:

- написан на C
- TCP прокси-сервер / балансировщик с проверкой состояния бэкендов
- разные алгоритмы определения доступности сервера
- HTTP / HTTPS реверс-прокси
- SSL терминирование, с SNI/NPN/ALPN и OCSP stapling в комплекте
- L4 / L7 балансировщик
- контентный коммутатор (разбор HTTP трафика по заголовкам)
- поддерживает: HTTP keepalive, HTTP/2, IPv6, UNIX-сокеты, сжатие

HAproxy

Версия HAProxy 2.0 stable, что нового:

https://www.opennet.ru/opennews/art.shtml?num=50904

- представлен новый API Data Plan, позволяющий на лету управлять настройками HAProxy через REST Web API. В том числе можно динамически добавлять и удалять бэкенды и серверы, создавать ACL, изменять маршрутизацию запросов, изменять привязки обработчиков к IP
- упрощена настройка логов при запуске в изолированных контейнерах - лог теперь можно направить в stdout и stderr, а также в любой существующий файловый дескриптор (например, "log fd@1 local0")
- добавлена официальная поддержка режима End-to-End HTTP/2 (обработка всех стадий в HTTP/2, в том числе обращений к бэкенду, а не только взаимодействие прокси с клиентом)

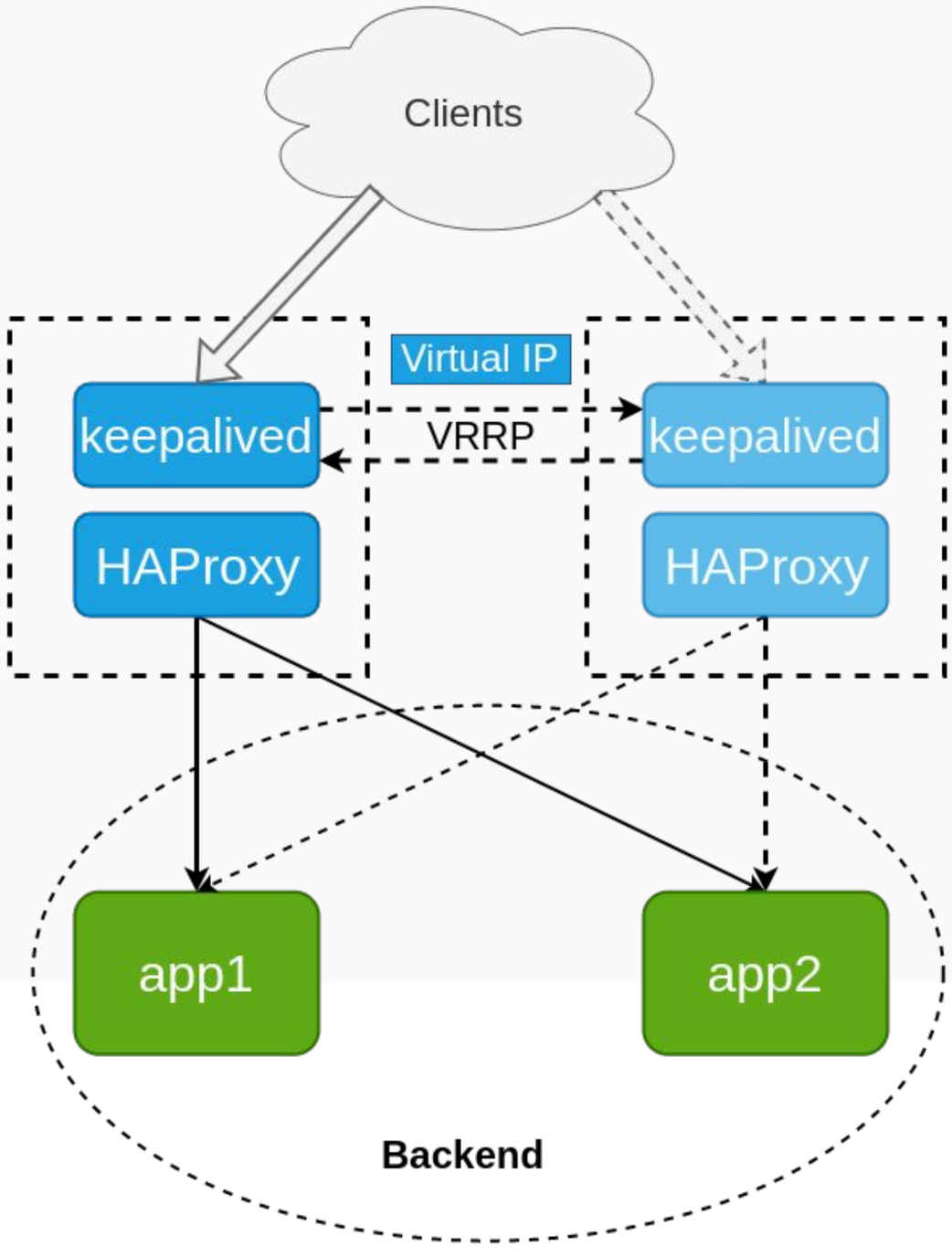
HAproxy

Версия HAProxy 2.0 stable, что нового:

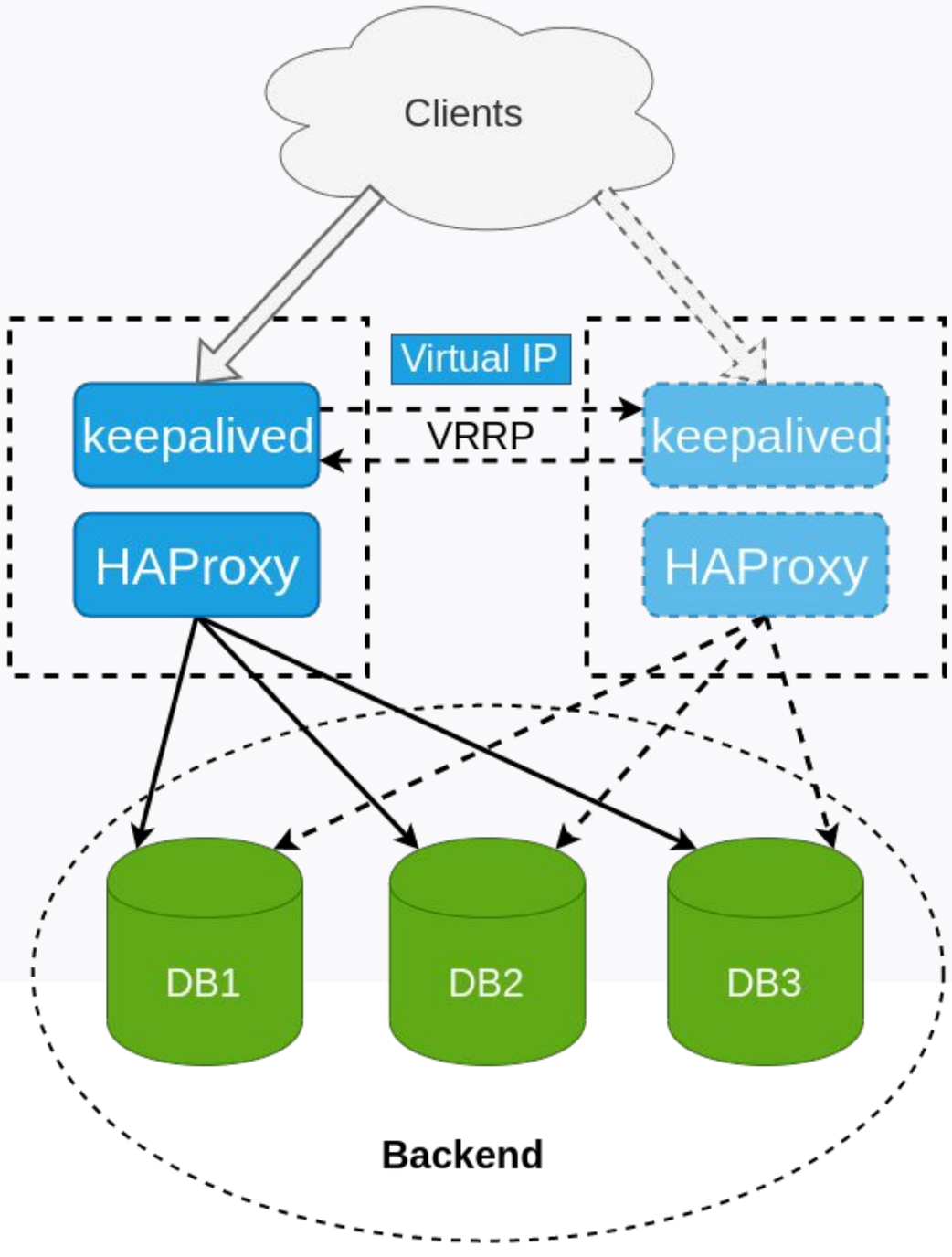
https://www.opennet.ru/opennews/art.shtml?num=50904

- добавлен внешний обработчик spoa-mirror (/usr/sbin/spoa-mirror) для зеркалирования запросов на отдельный сервер (например, для копирования части рабочего трафика для тестирования экспериментального окружения на реальной нагрузке)
- представлен HAProxy Kubernetes Ingress Controller для обеспечения интеграции с платформой Kubernetes
- добавлена встроенная поддержка экспорта статистики в систему мониторинга Prometheus

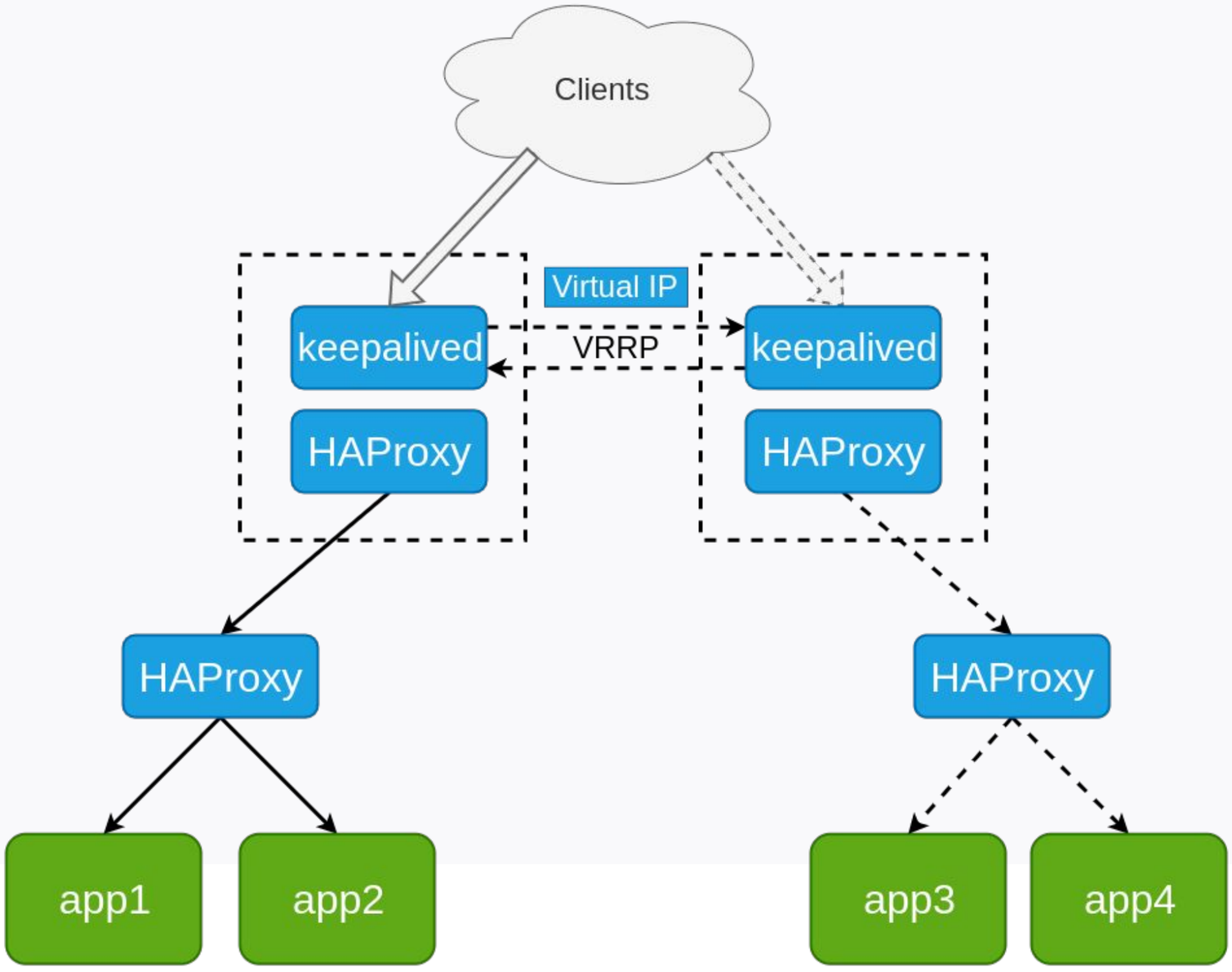
HAproxy: **схемы применения**



HAproxy: **схемы применения**



HAproxy: **схемы применения**



HAProxy: **конфигурация**

Основные секции конфигурации:

Global определяет общую конфигурацию (logging, user, group...)

Defaults определяет настройки по-умолчанию (mode, maxconn...)

Listen объединяет в себе описание для фронтенда и бэкенда и содержит полный список прокси

Frontend определяет, каким образом перенаправлять запросы к бэкенду в зависимости от того, что за запрос поступил от клиента

Backend содержит список серверов и отвечает за балансировку нагрузки между ними в зависимости от выбранного алгоритма

HAProxy: **конфигурация**

Примеры Frontend.

Обработка HTTP и HTTPS запросов:

```
frontend http_frontend *:80
  mode http
  redirect scheme https code 301 if !{ ssl_fc }

frontend https_frontend_ssl_pass
  mode tcp
  bind *:443
  default_backend https-backend
```

HAProxy: **конфигурация**

ACL - Access Lists HAProxy для принятия решения о направлении трафика в зависимости от контента в запросе

acl <имя правила> <критерий> [flags] [operator] [<value>] ...

Примеры ACL:

```
frontend http-in
    bind *:80
    acl url_appX path_beg -i /appX/
    use_backend appX-backend if url_appX
    default_backend appZ-backend
```

HAProxy: **конфигурация**

ACL - Access Lists HAProxy для принятия решения о направлении трафика в зависимости от контента в запросе

acl <имя правила> <критерий> [flags] [operator] [<value>] ...

Примеры ACL:

```
acl url_static  path_beg       /static /images /img /css
acl url_static  path_end       .gif .png .jpg .css .js
acl host_www    hdr_beg(host) -i www
acl host_static hdr_beg(host) -i img. video. download. ftp.
```

HAProxy: **конфигурация**

Алгоритмы балансировки (выбор бэкенда):

weight - регулировка пропорциональной нагрузки на бэкенд на основе параметра “weight” (вес), задается в пределах 1-256

roundrobin - каждый сервер получает запросы пропорционально своему весу, при этом веса серверов могут меняться на лету

static-rr - то же, что и roundrobin, только изменение весов на лету не дает никакого эффекта

leastconn - выбирает сервер с наименьшим количеством активных соединений

first - выбирает первый сервер с доступными слотами для соединения

source - сервер назначается на основе хэша IP-адреса отправителя запроса и весов серверов

HAProxy: **конфигурация**

Алгоритмы балансировки (выбор бэкенда):

uri - сервер выбирается на основе адреса (без параметров) страницы

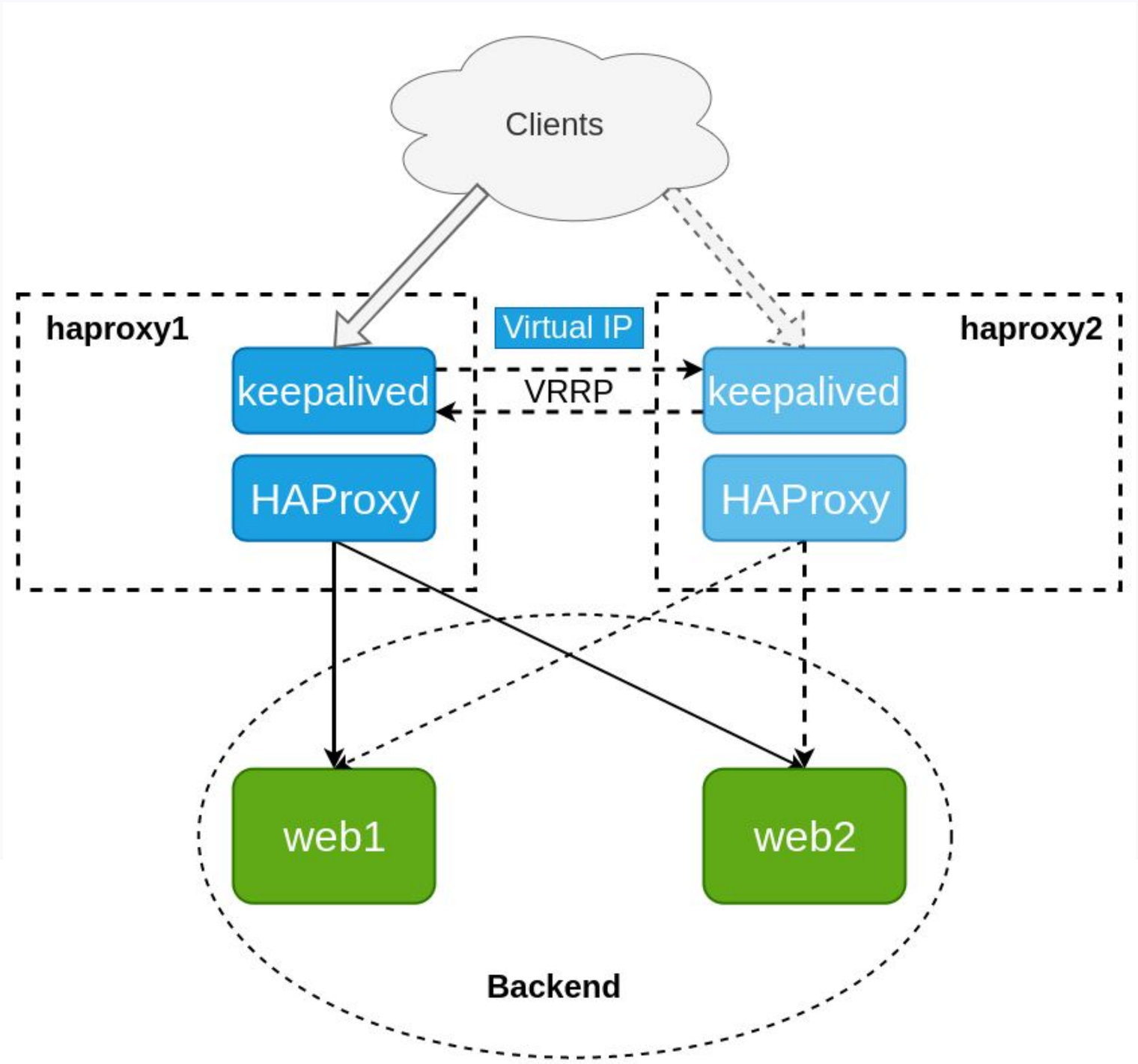
url_param - сервер выбирается на основе GET-параметров запроса

hdr - сервер выбирается на основе заголовков запроса

rdp-cookie - сервер выбирается на основе cookie (если они не установлены, то применяется обычный **round robin**)

Схема тестового стенда

Схема тестового стенда



Ваши вопросы?