GeekBrains

Основные сервисы на Linux для предприятия

Балансировка нагрузки

Рассмотрим различные методы балансировки нагрузки. Настроим IPVS для балансировки трафика между двумя серверами

Оглавление

Балансировка нагрузки

- Scale UP
- Scale Out
- DNS Load Balancing
  - Преимущества балансировки:
  - Недостатки балансировки
- Балансировка на L3/L4 модели OSI
  - Алгоритмы планировщика
    - Round-Robin (Round-Robin Scheduling)
    - Round-Robin с использованием весовых коэффициентов (Weighted Round-Robin Scheduling)
    - Минимум подключений (Least-Connection)
    - Минимум подключений с использованием весовых коэффициентов (Weighted Least-Connections) (используется по умолчанию)
    - Минимум подключений с учетом местоположения (Locality-Based Least-Connection Scheduling)
    - Минимум подключений с учетом местоположения и репликацией (Locality-Based Least-Connection Scheduling with Replication Scheduling)
    - Хеш получателя (Destination Hash Scheduling)
    - Хеш отправителя (Source Hash Scheduling)

Балансировка на 7м уровне модели OSI

Балансировка трафика на сетевом оборудовании - ECMP

Собираем все вместе

Настройка IPVS

Домашнее задание

Дополнительные материалы

Используемая литература

Балансировка нагрузки

Рассмотрим балансировку нагрузки на примере, который хорошо известен всем нам — веб-сайт.

Люди загружают ваш сервер с веб-сайтом, делая к нему множество запросов, чтобы посмотреть его содержимое в браузере. Для этого и нужны сайты: никто не хочет ресурс, который не смотрят и не посещают пользователи.

В какой-то момент количество запросов может быть гораздо больше, чем сервер может обработать. Из-за этого скорость его работы будет ниже, чем обычно, и с ростом числа пользователей он будет становиться всё медленнее и медленнее, пока полностью не выйдет из строя. Очевидно, что это не то, что вы хотите.

Для решения проблемы вам нужно больше ресурсов, и тут есть два варианта:

- Вы можете купить сервер с большим количеством ресурсов для замены текущего (Scaleup).
- Вы можете купить ещё один небольшой сервер для работы вместе с существующим (Scaleout).

Scaleup

Scaleup-масштабирование довольно сильно распространено для приложений, которым нужно больше ресурсов. Например, база данных стала настолько большой, что не помещается в память, как раньше. Возможно, диски переполнены или база стала обрабатывать больше запросов, чем раньше, и требует большей вычислительной мощности.

Базы данных, как правило, хороший пример для scaleup, потому что традиционно у них возникают серьёзные проблемы при работе на более чем одном сервере. Это потому, что многие вещи, которые работают на одном сервере, невозможно заставить работать параллельно на нескольких серверах. Например, как вы можете эффективно распределять таблицы между серверами? Это действительно

сложная проблема, и ее решение привело к появлению нескольких новых типов баз данных, таких как MongoDB и CouchDB.

Scaleup-масштабирование может быть довольно дорогим, так как высокопроизводительные компоненты растут в цене не линейно, а на порядок.

Scaleout

Scaleout-масштабирование подходит, когда у вас есть два, три или больше серверов, которые предоставляют сервис. В таком масштабировании, если вам начинает не хватать ресурсов, вы можете просто продолжать добавлять серверы. Конечно, в какой-то момент вы начнете сталкиваться с проблемами пространства в стойках и энергопотребления/охлаждения, но у вас наверняка будет больше вычислительной мощности. Также при scaleout-масштабировании система более отказоустойчива, чем при scaleup. Если один сервер выйдет из строя, у вас все равно останутся другие, которые могут взять на себя нагрузку.

У scaleout-масштабирования есть одна серьезная проблема — как заставить разные серверы работать вместе, чтобы они создавали впечатление единого? Решением этой проблемы занимается балансировщик нагрузки.

DNS Load Balancing

Балансировка нагрузки на DNS — самая простая в реализации. DNS можно использовать для быстрого решения многих проблем с производительностью, поскольку он позволяет направлять входящий трафик на любой из серверов. Он обладает также целым рядом проблем, поэтому такой вид балансировки применяется совместно с каким-либо еще методом.

Как вы помните, DNS сопоставляет доменные имена с IP-адресами и наоборот. При этом, никто не запрещает нам для какого-либо конкретного доменного имени возвращать несколько IP-адресов. IP-адреса используются по очереди. Например, у вас есть несколько веб-серверов, обслуживающих домен example.com.

```
www.example.com 192.168.1.1
www.example.com 192.168.1.2
```

DNS-сервер будет переключаться между этими IP-адресами всякий раз, когда кто-либо запрашивает у него имя www.example.com. По сути, это означает, что одна часть соединений будет обслужена 192.168.1.1, а другая — 192.168.1.2 (в идеальном мире — в пропорции 50 на 50). Это не считается идеальной балан, так как: если один из серверов выйдет из строя, DNS-сервер не узнает об этом и всё равно отправит клиенту его IP-адрес. Также у DNS-клиентов есть кеш, которым они постоянно пользуются, чтобы не спрашивать адреса у авторитетных DNS-серверов. Таким же кешем обладают DNS-сервера провайдера, который предоставляет услугу доступа в интернет для клиентов.

Что касается фактической балансировки нагрузки, например, при помощи BIND, она довольно проста в том, как она работает. BIND использует простой метод под названием round-robin для распределения соединений по группе серверов, которые он знает для домена. Он делает это последовательно (переходя первым, вторым, третьим и т. д.). Чтобы добавить балансировку нагрузки DNS на ваш сервер, вам нужно добавить несколько записей A для домена. В случае с example.com:

```
example.com.        IN      A       123.123.123.123
example.com.        IN      A       123.123.123.124
example.com.        IN      A       123.123.123.125
example.com.        IN      A       123.123.123.126
```

или:

```
example.com.        IN      A       123.123.123.123
                    IN      A       123.123.123.124
                    IN      A       123.123.123.125
                    IN      A       123.123.123.126
```

Теперь можно проверить работоспособность:

```
root@test:~# nslookup example.com 127.0.0.1
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   example.com
Address: 123.123.123.123
Name:   example.com
Address: 123.123.123.126
Name:   example.com
Address: 123.123.123.124
Name:   example.com
Address: 123.123.123.125
```

Преимущества балансировки:

- Простота и легкость настройки. Вы можете легко добавить очередной хост к доменному имени вашего сервиса и это сразу же включит его в балансировку. При этом вам не требуется менять что-либо в самом сервисе. Все настройки происходят на DNS.
- Легко чинить. Набор инструментов для траблшутинга DNS прост и понятен.
- Все сервисы уже в наличии. Невозможно представить себе инфраструктуру, в которой нет DNS-сервера. А это значит, что вы легко сможете добавить сервис DNS-балансировки.

Недостатки балансировки

Кроме преимуществ, всегда есть набор недостатков, о которых следует помнить.

Веб представляет собой протокол общения между клиентом и сервером, который не помнит предыдущие запросы. То есть каждый последующий запрос независим от предыдущего. Чтобы напомнить участникам обмена данными о чем именно идёт речь, у каждого клиента есть уникальный идентификатор — cookie, который присутствует в каждом запросе. Но, если следующий запрос будет отбалансирован на другой сервер, данные о cookie на нём могут отсутствовать в этот момент времени.

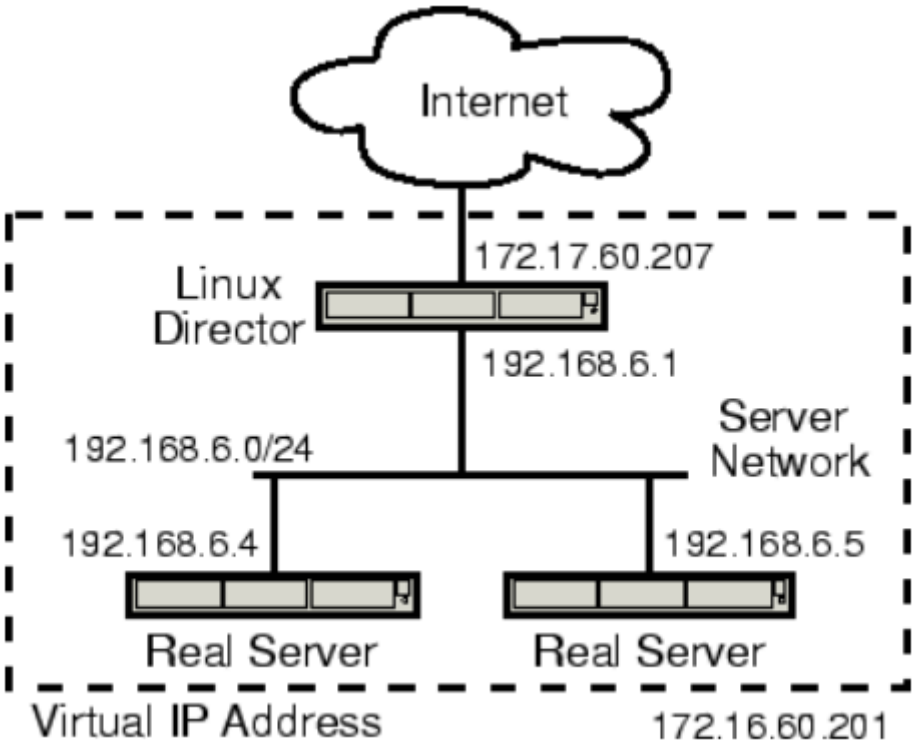
Нет возможности контроля над нагрузкой, а также над DNS-инфраструктурой интернет-провайдеров. Сам по себе механизм балансировки нагрузки при помощи DNS неверно называть балансировкой. Это скорее распределение нагрузки: в случае применения этого метода у нас нет возможности динамически что-либо перестраивать, указывать, какой клиент каким сервером должен быть обслужен. Мы зависим от настроек резолверов провайдера и от удачи.

Инертность в изменениях настроек — в DNS существует значение TTL, которое указывает, сколько хранить в кеше ответ. Если по какой-то причине один из серверов в группе балансировки вышел из строя, запросы на него всё ещё будут приходить как минимум на время значения TTL, а так как ничто не мешает провайдерам кешировать ответы на любое количество времени, такая ситуация может длиться несколько дней.

Балансировка на L3/L4 модели OSI

Кроме распределения нагрузки при помощи разных A-записей в DNS, можно использовать настоящую балансировку, которую уже можно контролировать и которая может достаточно точно распределять нагрузку между серверами.

Есть достаточно большое количество решений от вендоров и opensource-решений, но один из самых простых, понятных и производительных вариантов — Linux Virtual Server (LVS). Фактически LVS — модуль ядра Linux и выглядит как коммутатор TCP/UDP-сессий (иногда LVS называют маршрутизатором TCP-сессий). Он обеспечивает балансировку нагрузки на сетевом уровне, когда не требуется смотреть глубже внутрь пакета. Приложение будет взаимодействовать с неким виртуальным IP-адресом (VIP), который находится на LVS-балансировщике (Director), а он уже скрывает за собой группу реальных серверов (real servers), которые отвечают на запросы пользователя.



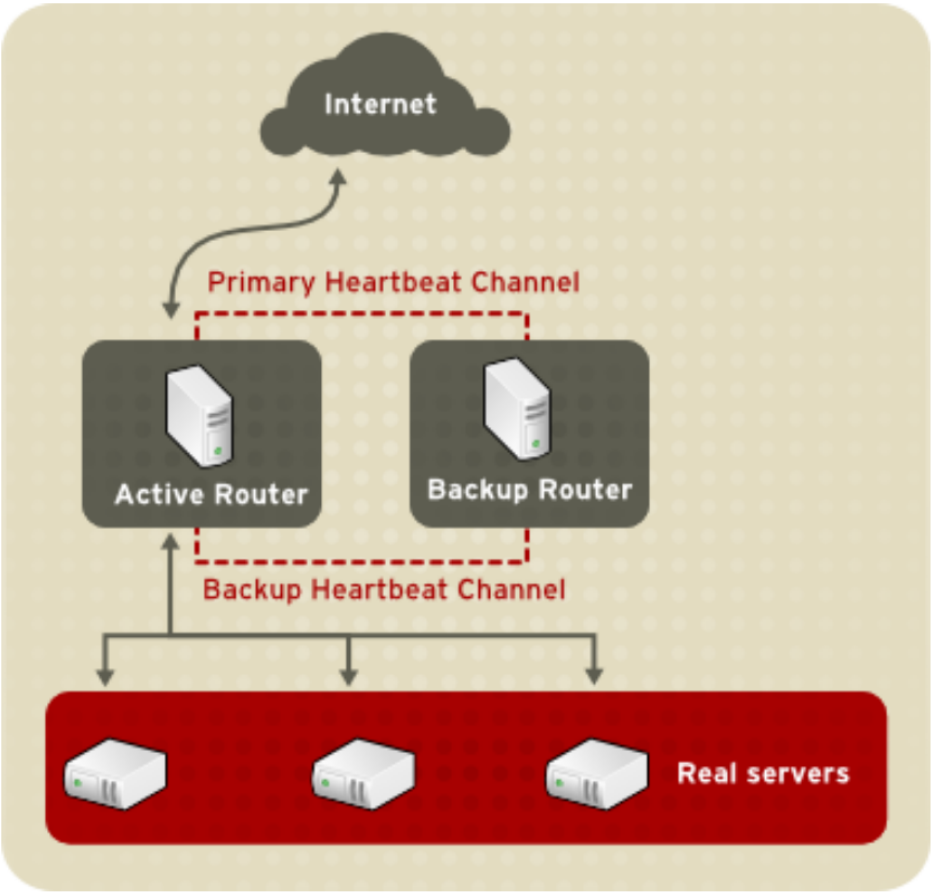
Получается, что набор серверов, на которых живёт и обрабатывает запросы клиентов приложение, скрывается за одним виртуальным IP-адресом. Терминология:

- Director — LVS-сервер, осуществляющий балансировку TCP/UDP-сессий. Не терминирует на себе пользовательские сессии.
- Real (realserver) — сервер, на котором находится приложение для обслуживания клиентов. На нем терминируются TCP/UDP-сессии пользователей.
- VIP или Virtual IP — IP нашего сервиса. Находится на директоре и представляет собой группу реалов. Пользователь видит сервис целиком как этот VIP и взаимодействует с ним, не подразумевая, что по факту это IP балансировщика нагрузки.
- DIP и RIP — IP директора и реальных серверов. Кроме виртуальных IP-адресов, у всех участников сервиса есть еще и реальные IP-адреса, находящиеся на интерфейсах.

Для повышения отказоустойчивости LVS-серверов с одинаковым VIP может быть несколько. В простейшем виде они могут представлять собой пару Active-Standby, а также сервис проверки состояния между Active и Standby-нодами.

Например, на изображении показана простая конфигурация LVS, состоящая из двух уровней. На первом уровне расположены два LVS-сервера (LVS-routers) — active и backup. Каждый LVS-сервер имеет два сетевых интерфейса: один для внешней сети и один для внутренней, что позволяет контролировать трафик между двумя сетями. В этом примере активный LVS-сервер использует Network Address Translation, или NAT, для распределения трафика из внешней сети между произвольным количеством реальных серверов, расположенных на втором уровне, и, в свою очередь,

предоставляющих доступ к необходимым сервисам. Реальные серверы в этом примере подключены к выделенному сегменту внутренней сети и направляют весь внешний трафик обратно через активный LVS-маршрутизатор. Для пользователей внешней сети все серверы выглядят как единое целое.



В таком дизайне в любой отдельно взятый момент времени активен только один LVS-сервер. Одна из основных задач активного LVS-сервера — перенаправление запросов, поступающих на виртуальные IP-адреса, к реальным серверам. Перенаправление основывается на одном из восьми поддерживаемых алгоритмов балансировки нагрузки.

Активный маршрутизатор также динамически отслеживает текущее состояние сервисов на реальных серверах при помощи простых запросов. Делается это, чтобы понять, что реальный сервер всё ещё жив и может обслуживать запросы пользователей. Если реальный сервер провалил healthcheck-запрос, он убирается из сервиса балансировки, так как запросы, отправленные на такой сервер, не будут обработаны и пользователь получит ошибку.

Периодически активный и резервный LVS-серверы обмениваются heartbeat-сообщениями через основной внешний интерфейс, а в случае сбоя — через внутренний интерфейс. Если резервный узел не может получить heartbeat-сообщение в течение заданного интервала времени, он инициирует процедуру восстановления после отказа и берёт на себя роль активного сервера.

Одно из преимуществ использования LVS — его способность выполнять гибкую балансировку нагрузки уровня IP в пуле реальных серверов. Такая гибкость объясняется наличием большого количества алгоритмов планировщика, которые администратор может выбрать на этапе настройки LVS. Балансировка нагрузки в LVS значительно превосходит такие методы, как Round-Robin DNS, когда иерархическая структура DNS и кеширование на клиентской машине может привести к разбалансировке нагрузки. В дополнение, низкоуровневая фильтрация, используемая в LVS-сервере, имеет ряд преимуществ перед перенаправлением запросов уровня приложения, так как балансировка нагрузки на уровне сетевых пакетов требует меньших вычислительных затрат и обеспечивает лучшую масштабируемость.

Используя планировщик, активный маршрутизатор может принимать во внимание активность реального сервера, а также, опционально, назначенный администратором весовой коэффициент при распределении запросов к сервису. Использование назначенных весовых коэффициентов позволяет выбрать для индивидуальных реалов произвольные приоритеты. При использовании соответствующего режима планировщика можно построить пул реальных серверов на базе различных аппаратных и программных конфигураций. При этом активный LVS-сервер может равномерно распределять нагрузку между реальными серверами.

Механизм планировщика LVS реализован в наборе патчей ядра, известных как модули IP Virtual Server или IPVS. Эти модули реализуют коммутацию на 4 уровне (L4), что позволяет работать с множеством серверов, используя всего один IP-адрес.

Для эффективного отслеживания состояния и коммутации пакетов IPVS строит в ядре таблицу IPVS. Эта таблица используется активным LVS-сервером для перенаправления запросов с адреса виртуального сервера на адреса реальных серверов в пуле, а также для перенаправления ответов реальных серверов клиентам. Таблица IPVS постоянно обновляется при помощи утилиты ipvsadm, которая добавляет и убирает реальные серверы в зависимости от их доступности.

Алгоритмы планировщика

Структура, которую приобретает таблица IPVS, зависит от алгоритма планировщика, который администратор выбирает для каждого виртуального сервера. Ниже представлены алгоритмы планировщика, предоставляемые CentOS 7 для обеспечения максимальной гибкости кластерных сервисов.

Round-Robin (Round-Robin Scheduling)

Последовательно, по очереди распределяет поступающие запросы в пуле реальных серверов. При использовании этого алгоритма все реальные сервера считаются равными, вне зависимости от текущей загрузки и мощности. Эта модель работы планировщика схожа с round-robin DNS, но, в отличие от него, не подвержена разбалансировке, связанной с кешированными DNS-запросами.

Round-Robin с использованием весовых коэффициентов (Weighted Round-Robin Scheduling)

Последовательно распределяет поступающие запросы в пуле реальных серверов, при этом больше загружает серверы с большей производительностью. Производительность обозначается назначенным администратором весовым коэффициентом, который впоследствии корректируется с использованием динамической информации о загрузке.

Алгоритм Round-Robin с использованием весовых коэффициентов предпочтителен для случаев, когда есть значительная разница в производительности реальных серверов.

Минимум подключений (Least-Connection)

Распределяет большее количество запросов серверам с наименьшим количеством активных подключений. Поскольку производится отслеживание активных подключений к активным серверам с помощью таблицы IPVS, этот алгоритм использует динамическое планирование, что делает его лучшим выбором для ситуаций, когда возможен большой разброс значений нагрузки. Он наилучшим образом подходит для пула реальных серверов, где все ноды имеют примерно одинаковую производительность. Если же мощности реальных серверов значительно отличаются, алгоритм «Минимум подключений с использованием весовых коэффициентов» будет лучшим выбором.

Минимум подключений с использованием весовых коэффициентов (Weighted Least-Connections) (используется по умолчанию)

Распределяет большее количество запросов серверам с наименьшим количеством активных подключений в соответствии с их производительностью. Производительность обозначается назначенным администратором весовым коэффициентом, который впоследствии корректируется с использованием динамической информации о загрузке. Введение весовых коэффициентов делает этот алгоритм идеальным для случаев, когда пул реальных серверов содержит различные по производительности конфигурации.

Минимум подключений с учётом местоположения (Locality-Based Least-Connection Scheduling)

Распределяет большее количество запросов серверам с наименьшим количеством активных подключений в соответствии с IP адресата. Этот алгоритм разработан для использования в кластерах кеширующих прокси-серверов. Он направляет пакеты с определенного IP-адреса серверу, обслуживающему этот адрес только в случаях, когда сервер не загружен полностью. В противном случае, если имеются серверы с менее чем половинной нагрузкой, пакет направляется наименее загруженному серверу. IP адресата также передается на обслуживание этому серверу.

Минимум подключений с учётом местоположения и репликацией (Locality-Based Least-Connection Scheduling with Replication Scheduling)

Распределяет большее количество запросов серверам с наименьшим количеством активных подключений в соответствии с IP адресата. Этот алгоритм также разработан для использования в

кластерах кеширующих прокси-серверов. От предыдущего алгоритма его отличает то, что IP-адрес закрепляется за подмножеством реальных серверов. Запросы передаются серверу из этого подмножества, имеющему минимальное количество подключений. Если все реальные серверы из этого подмножества перегружены, алгоритм резервирует новый сервер, выбирая наименее загруженный из пула и добавляя его к подмножеству, обслуживающему данный IP. После этого наиболее загруженный реальный сервер удаляется из подмножества для предотвращения избыточной репликации.

Хеш получателя (Destination Hash Scheduling)

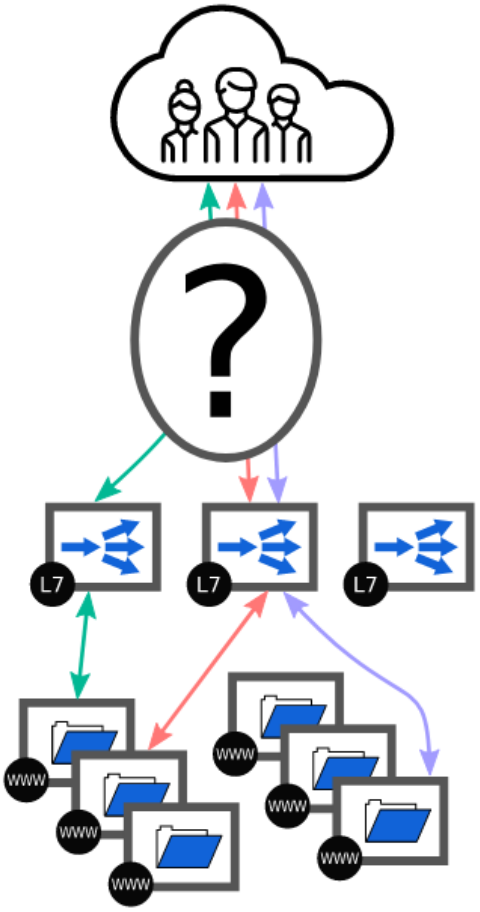
Распределяет запросы в пуле реальных серверов в соответствии с IP получателя из статической hash-таблицы. Этот алгоритм спроектирован для работы с кластером кеширующих прокси-серверов.

Хеш отправителя (Source Hash Scheduling)

Распределяет запросы в пуле реальных серверов в соответствии с IP отправителя из статической hash-таблицы. Этот алгоритм спроектирован для работы с кластером кеширующих прокси-серверов.

Балансировка на 7-м уровне модели OSI

Смысл этого уровня балансировки в терминации L4-сессии на себе, обработка запроса пользователя и передача этого запроса на набор бэкендов. Кроме этого, есть возможность проверять здоровье бекэнд-серверов, терминировать на себе дорогостоящее с точки зрения потребления ресурсов TLS-соединение, роутинг по HTTP-заголовкам, перезапись заголовков, рейт-лимитинг запросов от не аутентифицированных пользователей и так далее. Так как в этой ситуации все запросы требуют сохранения состояний (то есть являются stateful) алгоритм балансировки тут немного сложнее.



Так как L7-балансировщики терминируют на себе TCP-сессии клиентов, установка L7-балансировщиков (это ещё называется front-end load balancer) перед бэкенд-серверами дает вам несколько преимуществ:

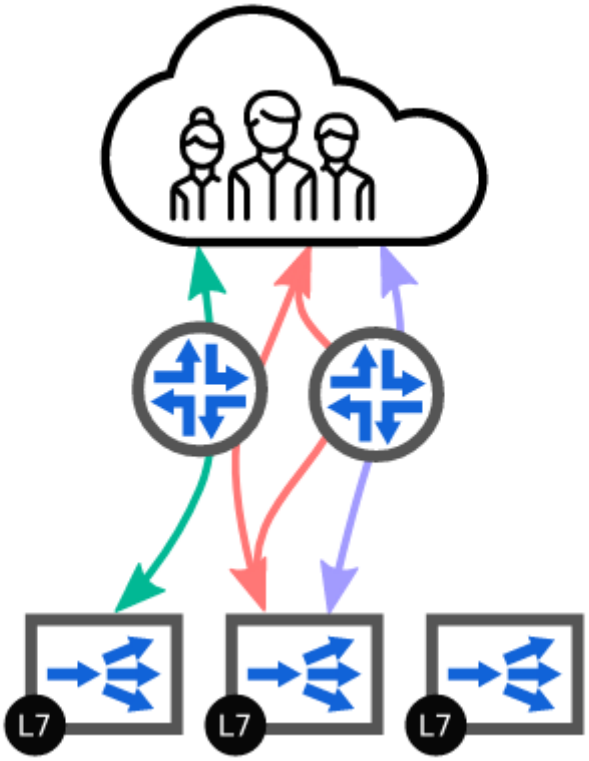
- Сессия между серверами может быть постоянно открытой. Это позволяет не тратить время на установку соединения, экономя ресурсы и уменьшая задержку. Запрос можно повторить ещё раз, и пользователь не получит сообщение об ошибке.
- Клиенты могут использовать другой IP-протокол или, например, MTU, нежели взаимодействие между балансировщиком и бэкендом. Например, внутри вашего дата-центра серверы могут общаться по IPv6, а запрос от клиента на L7-балансировщик придет по IPv4.
- Бэкенд-серверы не будут заниматься низкоуровневыми сервисами для обслуживания TCP-сессий, например, path MTU discovery до клиента, избегание большого количества TIME-WAIT-состояния сокетов и так далее.

Примеры L7-балансировщиков: NGINX, Envoy, Traefik, HAProxy

Балансировка трафика на сетевом оборудовании — ECMP

ECMP — equal cost multi path, или балансировка трафика по равноценным путям.

В некоторых случаях, особенно внутри дата-центров, у вас есть более чем 1 равноценный путь между отправителем и получателем.



Так как пути равноценны, сетевому устройству нет разницы, в какой их трёх возможных путей отправить пакет. И тут вступают в механизмы балансировки на самом маршрутизаторе. Их два:

- Per-packet (попакетно) — механизм работы описан в самом названии метода. Первый пакет отправляется в один путь, второй — во второй, третий — в третий и так далее. Применять этот механизм нельзя, так как даже в идеальных условиях нельзя гарантировать, что пакеты придут получателю в таком же порядке, как были отправлены. Поэтому этот механизм не применяется в современном мире и практически не реализуется в железе.
- Per-flow (по потокам) — в качестве единицы для балансировки используется вся сессия целиком. Пакеты внутри одной сессии всегда будут отправлены по одному пути. В случае для TCP/UDP-трафика, все пакеты в пределах одной сессии будут иметь одинаковый набор (Source IP, Destination IP, Source Port, Destination Port, Protocol), так называемый 5-Tuple. Это позволяет устройству понять, какие пакеты принадлежат какому потоку и позволяют взять хеш от этого набора данных, чтобы выбрать, в какой исходящий интерфейс отправлять пакеты.

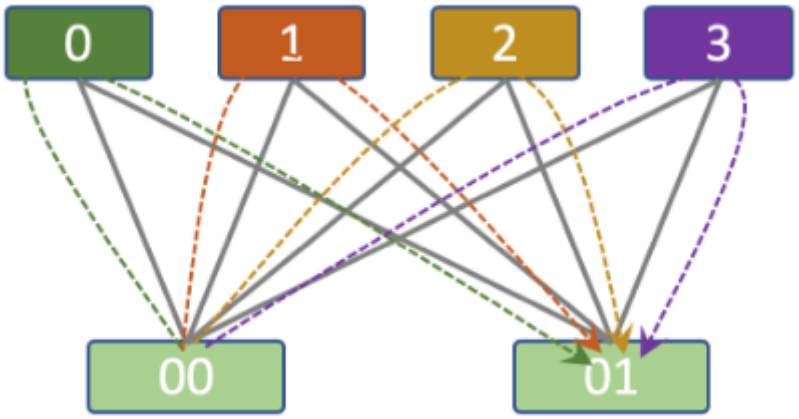
Балансировка на сетевом оборудовании происходит в железе и не затрачивает дополнительных ресурсов устройства.

В простейшем случае, формула для выбора пути следования — остаток от деления результате хеша от 5-tuple на количество возможных путей (hash % path_count).

Представьте ситуацию, что у нас есть 4 пути и 5 TCP-сессий. Результаты выполнения хеш-функции от 5-tuple для каждой из сессий: 4, 16, 21, 11, 6.

Берем остаток от деления хеша на количество путей (4):

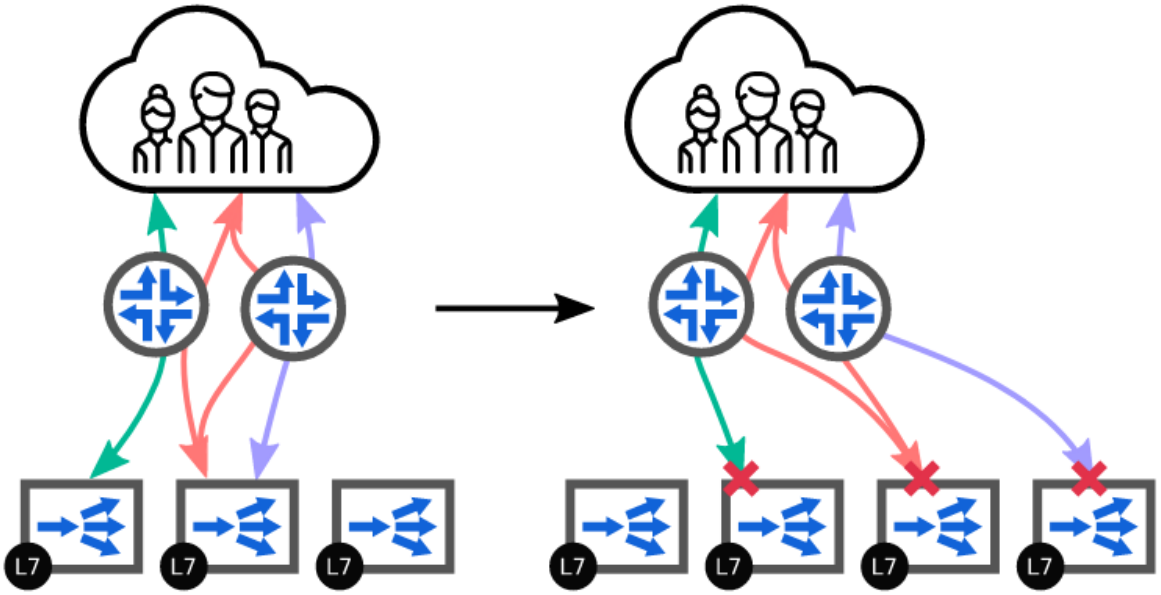
- 4 % 4 = 0
- 16 % 4 = 0
- 21%4 = 1
- 11 % 4 =3
- 6%4 = 2



Получается, что две сессии с хешами 4 и 16 будут отправлены через путь 0, и по одной сессии — во все остальные пути.

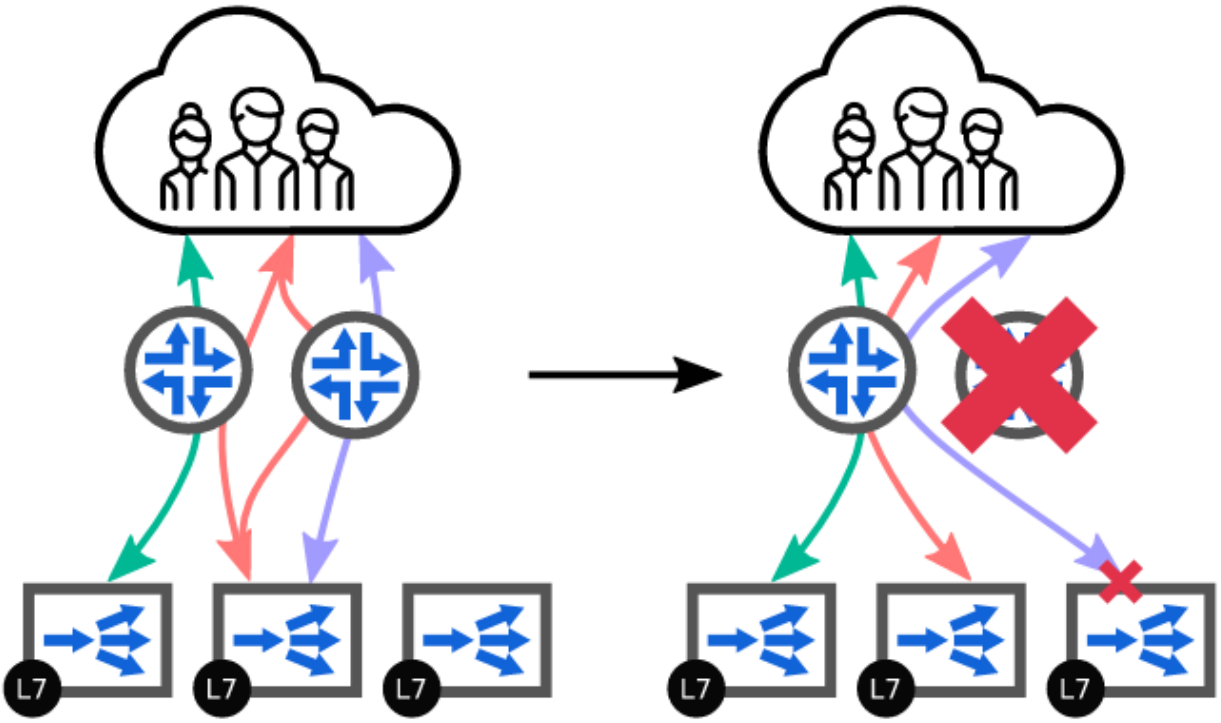
Исходя из такого расчета, можно понять, что в случае любого изменения количества путей, таблица отправки трафика будет пересчитана. Это может привести к тому, что сессия, которая ранее жила на сервере 1, будет разбалансирована на сервер 2. Сервер 2, в свою очередь, не имеет представления о такой сессии, а значит, она будет закрыта. Механизм сломается на некоторое время, если мы добавим количество серверов в бекэнд для балансировки или роутеров/путей в сеть.

В первом случае мы добавили еще один L7-балансировщик к группе из трёх балансировщиков.



Это немедленно увеличило количество возможных путей, что перебалансировало текущие сессии пользователей.

Аналогичная ситуация произойдёт в случае отказа одного из роутеров или добавления нового между клиентами и серверами.



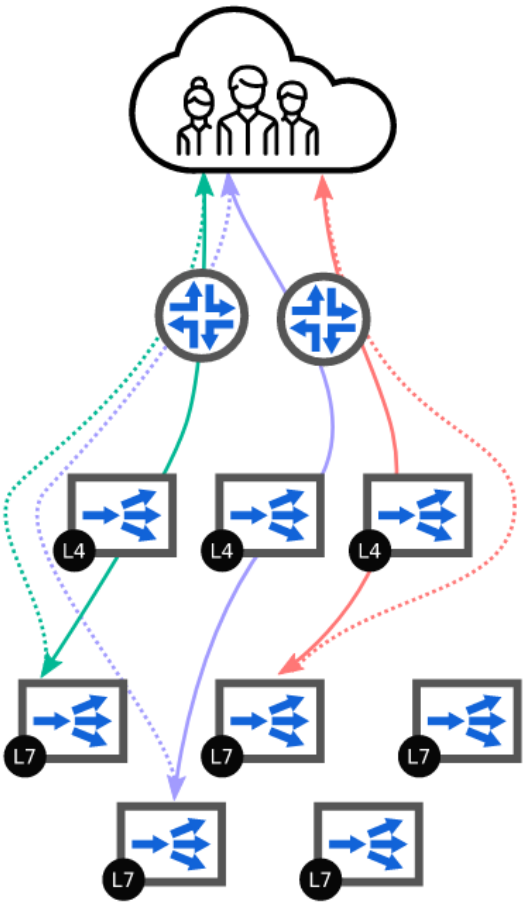
Современные сетевые устройства имеют функцию resilient caching (consistent hashing), которая позволяет бороться с этой проблемой.

Собираем все вместе

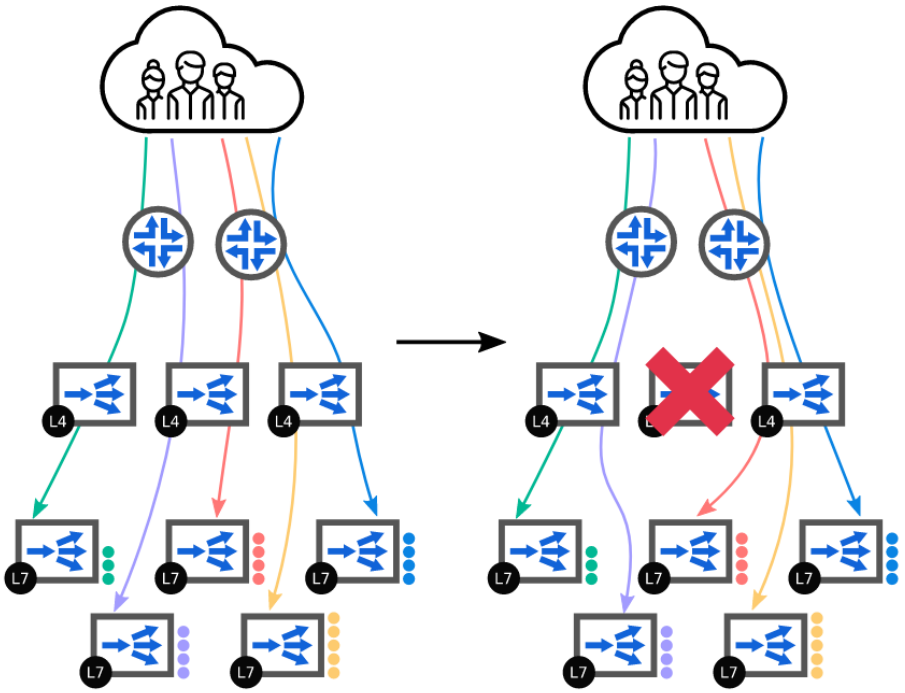
Для высокопроизводительного и отказоустойчивого сервиса мы собираем все возможности балансировки вместе. В начале мы говорили о том, что проще всего реализовать Active-Standby-схему с IPVS. Но в таком случае у нас простаивает 50% ресурсов, ожидая, что с Active-нодой что-либо случится.

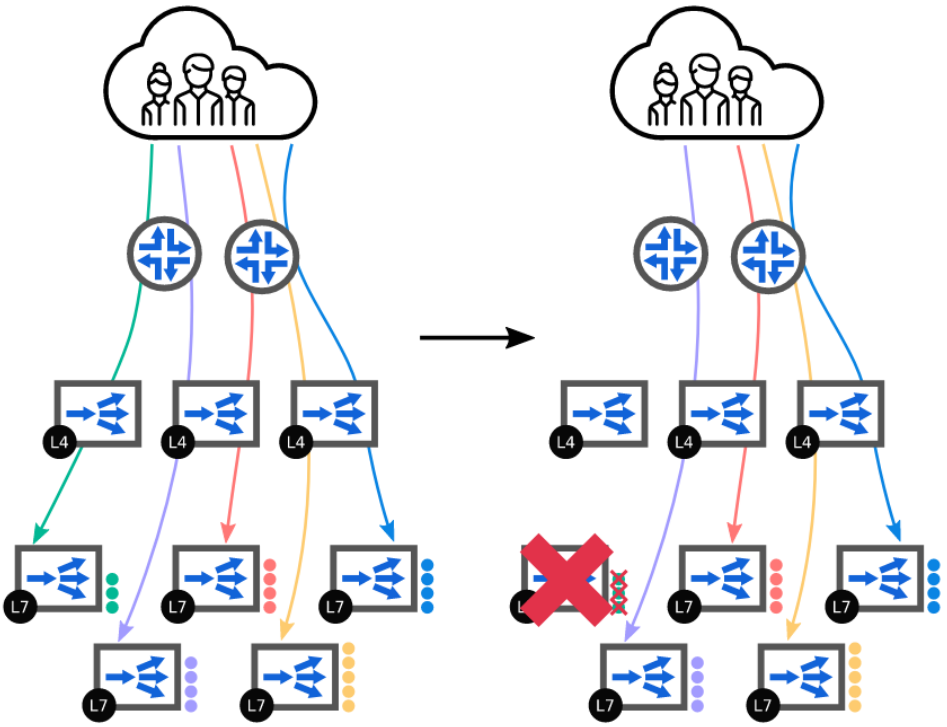
Вместо этого мы можем сделать первый уровень балансировки с Active-Active IPVS-балансировщиками и назначить на них одинаковый VIP. Этот VIP проанонсировать в сеть, что даст нам несколько ECMP-путей внутри сети.

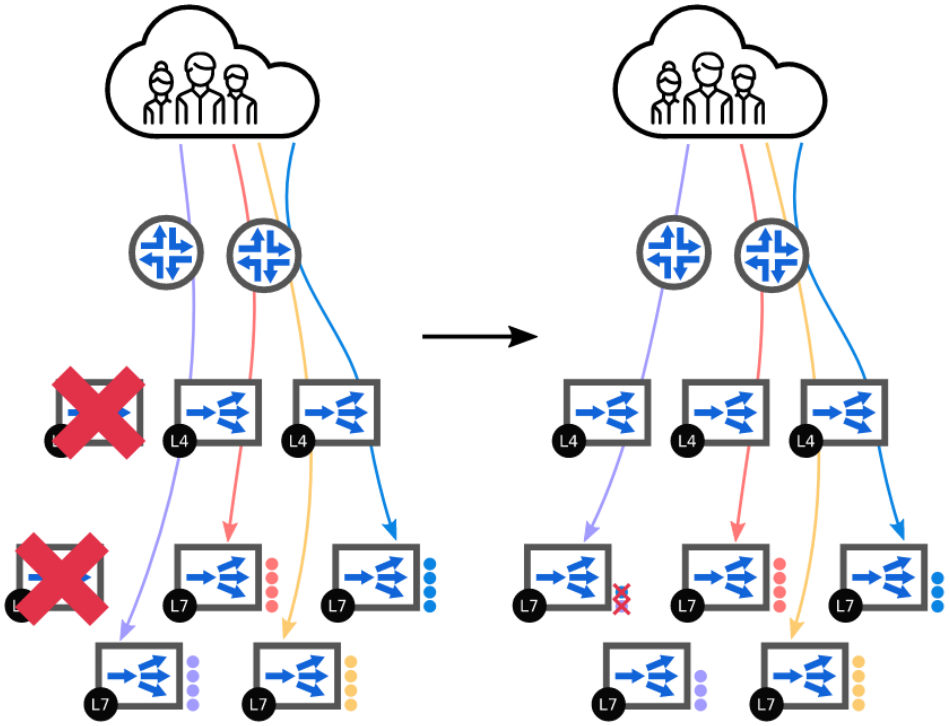
Реалами для IPVS-балансировщика будут не бэкенды, а ещё один уровень с L7-балансировщиками.



Для этого нам потребуется сервис healthcheck, который может быть предоставлен пакетом keepalived, consistent hashing на IPVS-балансировщиках, чтобы гарантированно отправлять одного и того же клиента с IPVS на один и тот же L7-балансировщик. Кроме этого, потребуется resilient hasing на сетевом оборудовании, чтобы, если у нас происходят какие-то изменения на сети, текущие сессии не перебалансировались, а новые уже выбирали свой путь исходя из изменившихся условий.


L4
L4
L4
L7
L7
L7
L7
L7
L4
L4
L7
L7
L7
L7
L7


L4
L4
L4
L7
L7
L7
L7
L7
L4
L4
L4
L7
L7
L7
L7
L7



Настройка IPVS

Правила проброса пакетов крайне просты: задаём виртуальный сервис, определяемый парой VIP:port. Сервис может быть TCP или UDP. Здесь же задаём метод ротации узлов (планировщик, scheduler). Далее задаём набор серверов из нашей фермы, также парой RIP:port, указываем метод проброса пакетов и вес, если того требует выбранный планировщик.

Устанавливаем пакет ipvsadm — утилиту для взаимодействия с модулем ядра ip_vs, — который, в свою очередь, обрабатывает соединения:

```
[root@centos7 ~]# yum install -y ipvsadm Dependencies Resolved

================================================================================
 Package         Arch           Version              Repository    Size
================================================================================
Installing:
 ipvsadm         x86_64         1.27-7.el7           base          45 k

Transaction Summary
================================================================================
```

Проверяем, что команда работает:

```
[root@centos7 ~]# ipvsadm -l
```

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
```

Добавляем бэкенды в виде док-контейнеров с Nginx. Узнаем, какие IP-адреса будут у контейнеров:

```
[root@centos7 ~]# yum install docker
Dependencies Resolved

================================================================================
 Package               Arch   Version                     Repository
                                                                        Size
================================================================================
Installing:
 docker                x86_64 2:1.13.1-103.git7f2769b.el7.centos
                                                          extras   18 M
Installing for dependencies:
 PyYAML                x86_64 3.10-11.el7                base    153 k
 atomic-registries     x86_64 1:1.22.1-29.gitb507039.el7 extras   35 k
 container-selinux     noarch 2:2.107-3.el7              extras   39 k
 container-storage-setup
                       noarch 0.11.0-2.git5eaf76c.el7    extras   35 k
 containers-common     x86_64 1:0.1.37-3.el7.centos      extras   21 k
 docker-client         x86_64 2:1.13.1-103.git7f2769b.el7.centos
                                                          extras  3.9 M
 docker-common         x86_64 2:1.13.1-103.git7f2769b.el7.centos
                                                          extras   97 k
 libseccomp            x86_64 2.3.1-3.el7                base     56 k
 libyaml               x86_64 0.1.4-11.el7_0             base     55 k
 oci-register-machine  x86_64 1:0-6.git2b44233.el7       extras  1.1 M
 oci-systemd-hook      x86_64 1:0.2.0-1.git05e6923.el7_6 extras   34 k
 oci-umount            x86_64 2:2.5-3.el7                extras   33 k
 python-backports      x86_64 1.0-8.el7                  base    5.8 k
 python-backports-ssl_match_hostname
                       noarch 3.5.0.1-1.el7              base     13 k
 python-ipaddress      noarch 1.0.16-2.el7               base     34 k
 python-pytoml         noarch 0.1.14-1.git7dea353.el7    extras   18 k
 python-setuptools     noarch 0.9.8-7.el7                base    397 k
 subscription-manager-rhsm-certificates
                       x86_64 1.24.13-3.el7.centos       updates 228 k
 yajl                  x86_64 2.0.4-4.el7                base     39 k

Transaction Summary
================================================================================

[root@centos7 ~]# systemctl start docker
[root@centos7 ~]# mkdir /srv/A /srv/B
[root@centos7 ~]# echo "This is A" > /srv/A/index.html
[root@centos7 ~]# docker run --rm -d -v "/srv/A:/usr/share/nginx/html" --name
nginx-A nginx
```

```
Unable to find image 'nginx:latest' locally
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
8d691f585fa8: Pull complete
5b07f4e08ad0: Pull complete
abc291867bca: Pull complete
Digest: sha256:922c815aa4df050d4df476e92daed4231f466acc8ee90e0e774951b0fd7195a4
Status: Downloaded newer image for docker.io/nginx:latest
469cc29613702428b733302f8a61af408b6c3fc217715c391308d652d9e8242c
[root@centos7 ~]# docker inspect -f '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx-A
172.17.0.2
[root@centos7 ~]# echo "This is B" > /srv/B/index.html
[root@centos7 ~]# docker run --rm -d -v "/srv/B:/usr/share/nginx/html" --name
nginx-B nginx
bd192d3852d29c3bee8e53c4d74f393387cba718b444be54b72b52bbbec2444c
[root@centos7 ~]# docker inspect -f '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' nginx-B
172.17.0.3
```

Проверяем, что мы можем запросить index.html у каждого из контейнеров:

```
[root@centos7 ~]# curl 172.17.0.2
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.17.5</center>
</body>
</html>
```

Как видно, мы получаем forbidden, скорее всего из-за неверного selinux-контекста:

```
[root@centos7 ~]# ls -laZ /srv/*/index.html
-rw-r--r--. root root unconfined_u:object_r:var_t:s0   /srv/A/index.html
-rw-r--r--. root root unconfined_u:object_r:var_t:s0   /srv/B/index.html
[root@centos7 ~]# semanage fcontext -a -t httpd_sys_content_t /srv/A/index.html
[root@centos7 ~]# semanage fcontext -a -t httpd_sys_content_t /srv/B/index.html
[root@centos7 ~]# restorecon -v /srv/A/index.html
restorecon reset /srv/A/index.html context
unconfined_u:object_r:var_t:s0->unconfined_u:object_r:httpd_sys_content_t:s0
[root@centos7 ~]# restorecon -v /srv/B/index.html
restorecon reset /srv/B/index.html context
unconfined_u:object_r:var_t:s0->unconfined_u:object_r:httpd_sys_content_t:s0
[root@centos7 ~]# ls -laZ /srv/*/index.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0
/srv/A/index.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0
/srv/B/index.html
```

Проверяем еще раз:

```
[root@centos7 ~]# curl 172.17.0.2
This is A
[root@centos7 ~]# curl 172.17.0.3
This is B
```

Создаем новый IPVS-сервис. На нашем сервере уже есть IP-адрес, мы будем использовать его для теста:

```
[root@centos7 ~]# ip a s ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 00:0c:29:2f:32:2f brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.41/24 brd 192.168.1.255 scope global noprefixroute dynamic
ens33
       valid_lft 19530sec preferred_lft 19530sec
    inet6 fe80::fdc8:d699:3e8d:5c3b/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[root@centos7 ~]# ipvsadm -A -t 192.168.1.41:80 -s rr
[root@centos7 ~]# ipvsadm -l -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.41:80 rr
```

Добавляем реалы:

```
[root@centos7 ~]# ipvsadm -a -t 192.168.1.41:80 -r 172.17.0.2 -m
[root@centos7 ~]# ipvsadm -a -t 192.168.1.41:80 -r 172.17.0.3 -m
[root@centos7 ~]# ipvsadm -l -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.41:80 rr
  -> 172.17.0.2:80                Masq    1      0          0
  -> 172.17.0.3:80                Masq    1      0          0
```

Обратите внимание, что IPVS не открывает у себя сокет для обслуживания трафика на 80 порту:

```
[root@centos7 ~]# netstat -tulpan | grep 80
[root@centos7 ~]#
```

Открываем порт на файерволе:

```
[root@centos7 ~]#  firewall-cmd --zone=public --add-service=http --permanent
success
[root@centos7 ~]#  firewall-cmd --zone=public --add-service=http
success
```

Проверяем работоспособность на 1000 запросов:

```
@DESKTOP-C89VMG0:~$ for i in `seq 1 1000`; do curl http://192.168.1.41 -s; done
| sort | uniq -c
    500 This is A
    500 This is B
```

Проверяем рейт:

```
[root@centos7 ~]# ipvsadm -l -n --rate
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port                 CPS    InPPS   OutPPS    InBPS   OutBPS
  -> RemoteAddress:Port
TCP  192.168.1.41:80                    23       93       93     5751     9670
  -> 172.17.0.2:80                      12       47       47     2886     4854
  -> 172.17.0.3:80                      12       46       46     2864     4816
```

Практическое задание

1. На сервере Server1 установить ipvsadm.
2. На сервере Server1 установить docker и запустить 2 контейнера с Nginx.
3. Создать интерфейс dummy2 и назначить ему IP-адрес 111.111.111.111/32.
4. Проанонсировать этот IP в сеть из трёх серверов.
5. Настроить ipvs для передачи http запросов с сервера server3 в оба контейнера в Nginx используя механизм балансировки round-robin.
6. Удостовериться, что запросы балансируются между контейнерами путем проверки access.log внутри каждого из них.

Дополнительные материалы

1. Nginx в качестве балансировщика нагрузки.
2. Nginx-балансировка нагрузки на сервера Apache.
3. Load balancing with HAProxy, Nginx and Keepalived in Linux.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Product Documentation for Red Hat Enterprise Linux 7.
2. Practical Load Balancing: Ride the Performance Tiger by Eelco Plugge, David Hows, Peter Membrey. ISBN: 9781430236801