Как настроить балансировку нагрузки с помощью Nginx.

Современные приложения могут обрабатывать множество запросов одновременно, и при этом, даже при высокой нагрузке, они должны возвращать пользователям корректную информацию. Масштабировать приложения можно разными способами:

- Вертикальное масштабирование -— просто «накинуть» оперативной памяти, мощностей процессора — арендовать или купить более мощный сервер. На ранних этапах развития приложения это просто, но у такого подхода есть недостатки — цена и ограничения современного железа.

  
  80d59a9d-c558-4e7b-acd6-34af128d41cb?width=1292&height=482

- Горизонтальное масштабирование — добавить больше инстансов приложения. Поднять второй сервер, развернуть на нём точно такое же приложение и каким-то образом распределять трафик между этими экземплярами приложений.

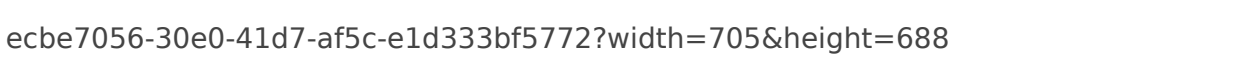
  
  588116fc-513c-4e07-8e74-a2ecbf72a2d9?width=1600&height=696

Горизонтальное масштабирование, с одной стороны, может быть дешевле и менее ограничивать нас в железе — можно просто добавить ещё инстансов приложения. Однако теперь нам необходимо распределять пользовательские запросы между разными экземплярами приложения.

Балансировка нагрузки (load balancing) — это способ распределения запросов к приложению (сетевого трафика) между некоторым количеством устройств.

Балансировщик нагрузки — это программа-посредник, которая располагается между пользователем и группой приложений. Общая логика следующая:

1. Пользователь заходит на сайт по определенному домену, за которым скрывается IP-адрес балансировщика нагрузки.
2. Балансировщик на основе настроек определяет, на какой из экземпляров приложения перенаправлять трафик от пользователя.
3. Пользователь получает ответ от нужного экземпляра приложения.

   
   ecbe7056-30e0-41d7-af5c-e1d333bf5772?width=705&height=688

Какие проблемы решает балансировка нагрузки?

- **Повышение доступности приложения:** Балансировщики нагрузки обладают функционалом для обнаружения аварийных ситуаций на серверах. В случае отказа одного из серверов балансировщик может автоматически перенаправить трафик на другой адрес, обеспечивая бесперебойную работу приложения для пользователей.
- **Масштабируемость:** Одной из основных задач балансировщика является распределение нагрузки между экземплярами приложения. Это позволяет применять горизонтальное масштабирование, добавляя новые экземпляры приложения и увеличивая общую производительность системы.
- **Улучшение безопасности:** Балансировщики нагрузки могут включать в себя функции, связанные с безопасностью. Они могут отслеживать трафик, фильтровать запросы, обеспечивать маршрутизацию через фаерволы и другие механизмы, что способствует повышению безопасности приложения.

Какие есть инструменты балансировки сетевого трафика?

Существует довольно много приложений, которые могут выступать в качестве балансировщика нагрузки, однако одним из самых популярных является Nginx. Его и рассмотрим в данном гайде.

Nginx

Nginx — это универсальный веб-сервер. Он отличается хорошей производительностью, низким потреблением ресурсов и своими широкими возможностями. Nginx можно использовать как:

- Веб-сервер
- Реверс-прокси и балансировка нагрузки
- Почтовый прокси-сервер
- И многое другое.

На сайте можно узнать подробнее про возможности Nginx. Ну а мы перейдём к практике.

Установка Nginx на Ubuntu

Nginx можно установить на все популярные дистрибутивы Linux: Ubuntu, CentOS и другие. В статье мы будем использовать Ubuntu. Чтобы установить Nginx, используем следующие команды:

```
sudo apt update
sudo apt install nginx
```

Чтобы убедиться, что всё прошло успешно, можно использовать команду:

```
systemctl status nginx
```

Файлы конфигураций для Nginx находятся в каталоге `/etc/nginx/sites-available/`. По умолчанию в этом каталоге создаётся файл `default`. В нём мы и будем писать нашу конфигурацию.

Пример настройки

начала откроем файл конфигурации по умолчанию:

```
cd /etc/nginx/sites-available/

sudo nano default
```

Поместим сюда следующую конфигурацию:

```
   upstream application{
      server 10.2.2.11; # ip-адреса серверов для распределения запросов между ними
      server 10.2.2.12;
      server 10.2.2.13;
   }

   server {
      listen 80; # по этому порту будет открываться nginx


      location / {
         # описываем, куда перенаправлять трафик от nginx
          proxy_pass http://application;
      }
   }
```

Для настройки балансировки нагрузки в Nginx в конфигурации нужно определить два блока:

- `upstream` — определяет адреса серверов, между которыми будет распределяться сетевой трафик. Тут мы указываем IP-адреса, порты и, при необходимости, методы балансировки нагрузки. Их мы обсудим далее.
- `server` — определяет способ, с помощью которого Nginx будет получать запросы. Обычно тут указывается порт, доменное имя и другие параметры.
- Путь `proxy_pass` — описывает, куда эти запросы надо перенаправлять. Это название указанного выше `upstream`.

Таким образом, Nginx используется не только как балансировщик нагрузки, но ещё и как обратный прокси (reverse proxy). Реверс-прокси — это сервер, который располагается между клиентом и экземплярами приложений. Он перенаправляет запросы от клиентов в бэкенд, и при этом может обеспечивать нас дополнительными функциями, например, такими как SSL-сертификаты, логирование и др.

Методы балансировки нагрузки

Round Robin

Существует довольно много методов балансировки. Nginx по умолчанию использует алгоритм Round Robin. Он довольно прост. Допустим, у нас есть приложения 1, 2 и 3. Балансировщик нагрузки отправит первый запрос на первое приложение:

> 6fe3b70b-dc2b-414b-ad69-ae8c6dc21640?width=444&height=304

Затем на 2:

> 8ddde93c-8f0f-49de-a01d-4f65b6e79cec?width=439&height=271

Затем на 3:

> 9534b017-c8ae-45b7-849a-7598b4ab81c8?width=418&height=276

И далее снова на 1:

> cc60e815-29e5-4c7e-a710-35907e27b610?width=438&height=293

Рассмотрим на примере. Я развернул два приложения и настроил балансировку нагрузки с помощью Nginx для них.

```
upstream application {
   server 172.25.208.1:5002#first
   server 172.25.208.1:5001; #second
}
```

Давайте посмотрим, как это работает на практике:

Первый запрос отправляет нас на первый сервер, второй запрос — на второй сервер, а затем снова на первый. Однако этот алгоритм имеет ограничение — экземпляры бэкенда будут простаивать просто потому, что ждут своей очереди.

Round Robin с указанием весов

Чтобы избежать простоя серверов, можно использовать некоторый числовой приоритет. У каждого сервера появляется свой *вес*, который определяет, как много трафика распределяется на конкретный экземпляр приложения. Таким образом мы гарантируем, что более мощные серверы получат больше трафика.

В Nginx приоритет указывается с помощью *server weights* следующим образом:

```
upstream application{
      server 10.2.2.11 weight=5;
      server 10.2.2.12 weight=3;
      server 10.2.2.13 weight=1;
   }
```

При такой настройке сервер с адресом 10.2.2.11 получит наибольшее количество трафика, так как у него указан наибольший вес.

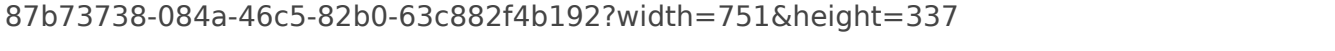
Такой подход более надёжен, чем обычный Round Robin, однако он всё ещё имеет недостаток — вручную мы можем указать вес, основываясь на мощности сервера, но при этом сами запросы также могут различаться по скорости выполнения: есть более долгие и тяжёлые, а есть быстрые и незатратные.

Рассмотрим этот метод на примере.

Настройка:

```
upstream application {
   server 172.25.208.1:5002 weight=3; #first
   server 172.25.208.1:5001 weight=1; #second
}
```

Результат:



Как мы видим, теперь каждый четвёртый запрос отправляется на второй сервер.

Least Connection

Что, если отойти от Round Robin? Распределять запросы между серверами можно не просто по порядку, а, например, основываясь на каких-либо параметрах. Отлично подойдёт количество активных соединений с сервером.

Алгоритм Least Connection обеспечивает равномерное распределение нагрузки между экземплярами приложения, как раз основываясь на количестве соединений с сервером. Чтобы его настроить, в блоке upstream надо указать `least_conn;`:

```
upstream application{
      least_conn;
      server 10.2.2.11;
      …
  }
```

Вернёмся к нашему примеру.

Чтобы проверить работу этого алгоритма, я написал скрипт, который отсылает 500 запросов параллельно и смотрит, на какое из приложений попал каждый запрос.

Вот вывод этого скрипта:

ae66ccbf-d410-42cc-8882-1194728d4e72?width=316&height=45

Кроме того, этот алгоритм может использоваться вместе с весами для адресов, по аналогии с Round Robin. В этом случае веса будут обозначать количество соединений с этим адресом по отношению к другим адресам — например, в случае с весами 1 и 5, на адрес с весом 5 попадёт в пять раз больше соединений, чем на адрес с весом 1.

Пример такой настройки:

```
upstream application{
      least_conn;
      server 10.2.2.11 weight=5;
      …
   }
```

А вот настройка для примера:

```
upstream loadbalancer {
   least_conn;
   server 172.25.208.1:5002 weight=3; #first
   server 172.25.208.1:5001 weight=1; #second
}
```

И вывод скрипта:

4a2496fb-8cc3-4ca3-b3bd-65bdb90cd7d7?width=322&height=93

Как мы видим, запросов на первый сервер ровно в три раза больше, чем на второй.

IP Hash

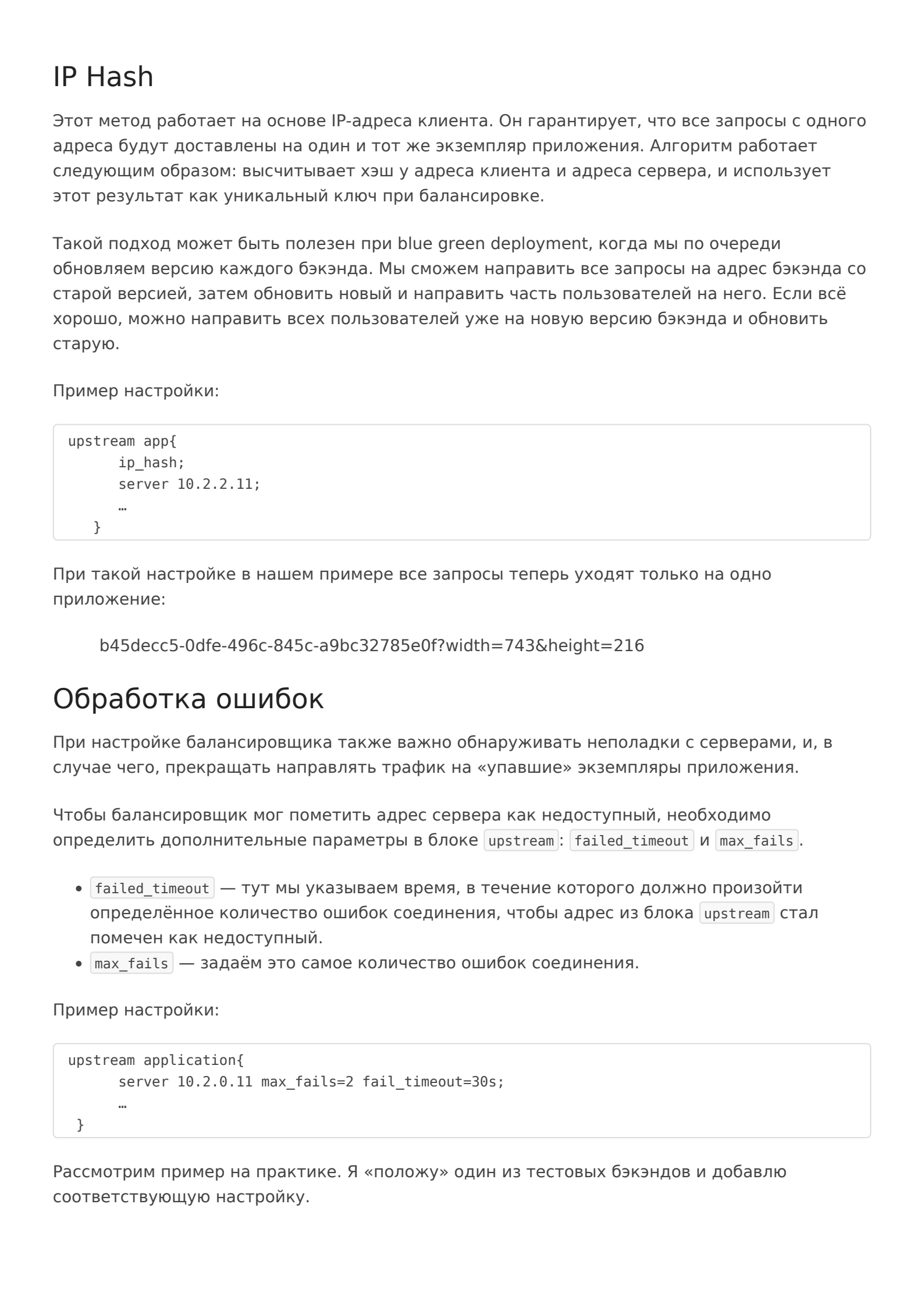
Этот метод работает на основе IP-адреса клиента. Он гарантирует, что все запросы с одного адреса будут доставлены на один и тот же экземпляр приложения. Алгоритм работает следующим образом: высчитывает хэш у адреса клиента и адреса сервера, и использует этот результат как уникальный ключ при балансировке.

Такой подход может быть полезен при blue green deployment, когда мы по очереди обновляем версию каждого бэкэнда. Мы сможем направить все запросы на адрес бэкэнда со старой версией, затем обновить новый и направить часть пользователей на него. Если всё хорошо, можно направить всех пользователей уже на новую версию бэкэнда и обновить старую.

Пример настройки:

```
upstream app{
      ip_hash;
      server 10.2.2.11;
      …
   }
```

При такой настройке в нашем примере все запросы теперь уходят только на одно приложение:



Обработка ошибок

При настройке балансировщика также важно обнаруживать неполадки с серверами, и, в случае чего, прекращать направлять трафик на «упавшие» экземпляры приложения.

Чтобы балансировщик мог пометить адрес сервера как недоступный, необходимо определить дополнительные параметры в блоке `upstream`: `failed_timeout` и `max_fails`.

- `failed_timeout` — тут мы указываем время, в течение которого должно произойти определённое количество ошибок соединения, чтобы адрес из блока `upstream` стал помечен как недоступный.
- `max_fails` — задаём это самое количество ошибок соединения.

Пример настройки:

```
upstream application{
      server 10.2.0.11 max_fails=2 fail_timeout=30s;
      …
 }
```

Рассмотрим пример на практике. Я «положу» один из тестовых бэкэндов и добавлю соответствующую настройку.

e7d009e0-7307-4891-9607-e68431e1a5ba?width=1020&height=217

Первый экземпляр бэкэнда из примера теперь отключен.


88aa0bfd-46f4-4c20-9d31-7de36925e08e?width=769&height=290

Nginx перенаправляет трафик только на второй сервер.

Сравнительная таблица алгоритмов распределения трафика


ac141a50c2fd35cba03e1.png


5f876dfe29b588865610f.png

Заключение

В этой статье мы погрузились в тему load balancing. Узнали, какие существуют методы балансировки нагрузки в Nginx и разобрали их на примере.

Revision #1
Created 2023-10-27 20:04:32 UTC by odiljonov
Updated 2023-10-27 20:05:00 UTC by odiljonov