Ранее я немного касался настройки nginx в качестве web сервера или proxy сервера для web приложения. Сегодня я расскажу, как настроить балансировку нагрузки бэкендов с помощью nginx. В качестве примера возьму простую ситуацию, когда трафик будет распределяться между тремя одинаковыми серверами.

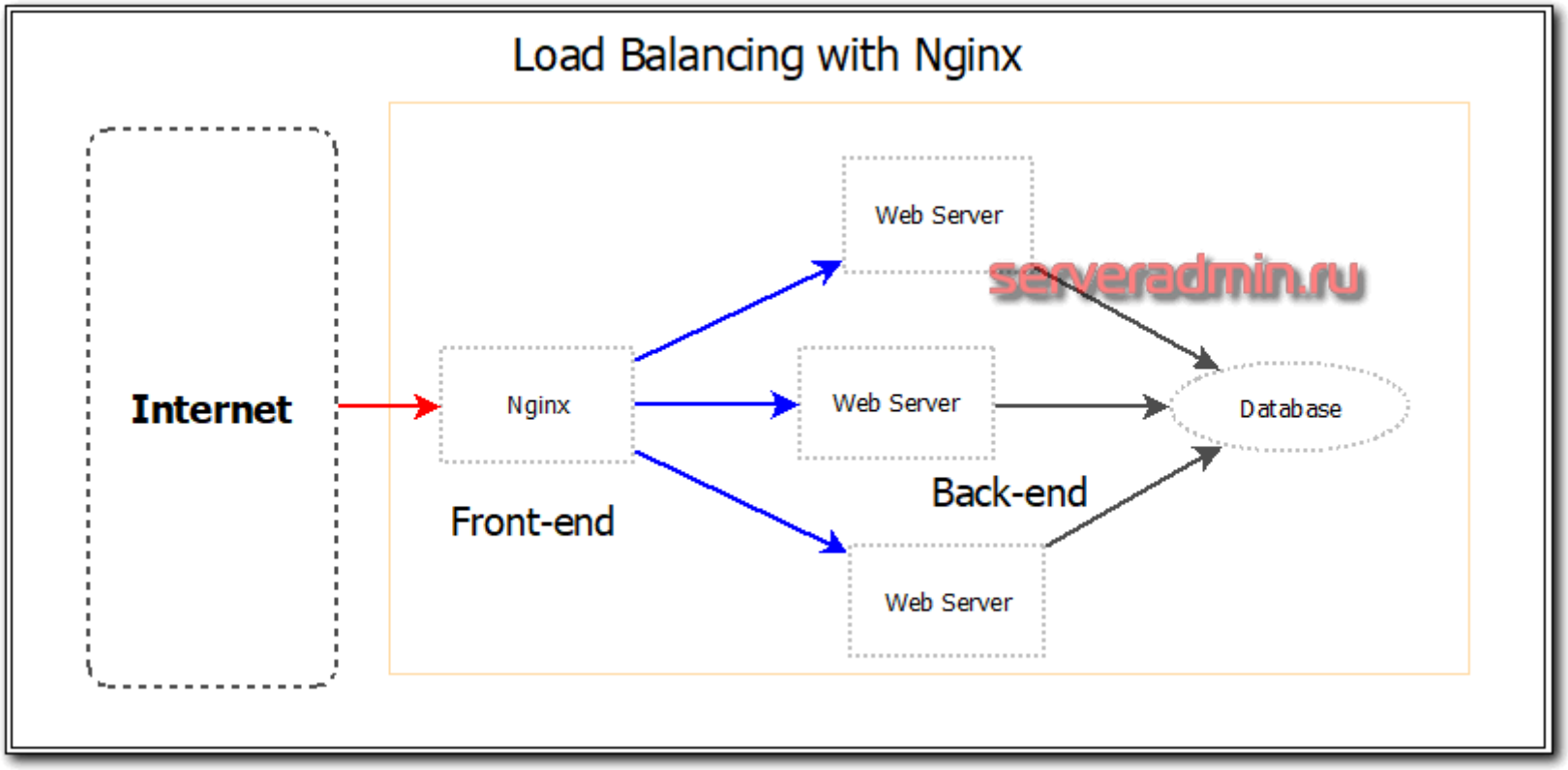
Содержание:

Введение

Ранее я подробно рассказывал о базовой настройке nginx и настройки его в качестве proxy сервера. Так же у меня есть подробная статья по настройке web сервера на базе Centos 7. Сегодня я расскажу о том, как настроить балансировку нагрузки с помощью nginx. Чаще всего это необходимо для балансировки нагрузки между бэкендами и обеспечения отказоустойчивости в работе web сервиса. Существуют различия в возможностях тонкой настройки балансировщика в бесплатной и платной версии nginx plus. Это один из первых моментов, где я столкнулся с тем, что мне стали необходимы функции платной версии, которых не было в бесплатной. Но цена nginx plus велика.

Я рассмотрю простой пример, где у нас в качестве web сервера выступает nginx, который распределяет запросы на три равноценных сервера. В случае

выхода из строя одного из серверов, он все запросы будет адресовать оставшимся в работе серверам. Теоретически это должно проходить незаметно для клиентов. Практически же есть масса нюансов в работе этого инструмента. Постараюсь обо всем этом рассказать.



Добавление бэкендов

Для того, чтобы начать балансировать нагрузку, необходимо добавить бэкенды в настройки nginx. Для примера, я возьму отдельный виртуальный хост. Идем в его конфиг и в самое начало добавляем три бэкенда через директиву **upstream**.

```
upstream cache-api {
    server 10.32.18.6:8080;
    server 10.32.18.7:8080;
    server 10.32.18.8:8080;
}
```

Сейчас я не касаюсь вопросов тонкой настройки балансировки. Будем идти от простого к сложному. На текущий момент мы добавили три сервера, на которые будет распределяться нагрузка. Далее в настройках виртуального хоста добавляем location, запросы к которому будем равномерно распределять.

```
location / {
    proxy_pass http://cache-api/;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Real-IP $remote_addr;
    }
```

Этого минимального набора настроек достаточно, чтобы nginx начал равномерно распределять запросы между двумя серверами. Но в реальной ситуации требуется более детальная настройка балансировщика. Для этого используются следующие параметры:

weight	Задает вес сервера, по умолчанию 1. Чем больше вес сервера, тем пропорционально больше запросов он будет принимать от балансировщика.
max_conns	Ограничивает максимальное число одновременных активных соединений к проксируемому серверу Значение по умолчанию равно 0 и означает, что ограничения нет.
max_fails	Задаёт число неудачных попыток работы с сервером, которые должны произойти в течение времени, заданного параметром fail_timeout, чтобы сервер считался недоступным на период времени, также заданный параметром fail_timeout. Дефолтное значение - 1.
fail_timeout	Задаёт время, в течение которого должно произойти заданное число неудачных попыток работы с сервером для того, чтобы сервер считался недоступным и время, в течение которого сервер будет считаться недоступным. По умолчанию параметр равен 10 секундам.
backup	Помечает сервер как запасной сервер. На него будут передаваться запросы в случае, если не работают основные серверы.
down	Помечает сервер как постоянно недоступный.

Подробнее об этом написано в официальной документации, в описании модуля ngx_http_upstream_module. К примеру, конфиг бэкендов для балансировки может быть таким.

```
server 10.32.18.6:8080 max_fails=2 fail_timeout=10s;
server 10.32.18.7:8080 max_fails=2 fail_timeout=10s;
server 10.32.18.8:8080 max_fails=2 fail_timeout=10s;
```

С такими настройками после двух неудачных попыток соединения в течении 10 секунд, бэкенд будет выведен из работы на те же 10 секунд.

Метод балансировки

Соединения к серверам для балансировки нагрузки могут распределяться по различным правилам. Существуют несколько методов распределения запросов. Я перечислю основные:

- **round-robin** - используется по умолчанию. Веб сервер равномерно распределяет нагрузку на сервера с учетом их весов. Специально указывать этот метод в конфигурации не надо.
- **least-connected** - запрос отправляется к серверу с наименьшим количеством активных подключений. В конфигурации данный параметр распределения запросов устанавливается параметром least_conn.
- **ip-hash** - используется хэш функция, основанная на клиентском ip адресе, для определения, куда направить следующий запрос. Используется для привязки клиента к одному и тому же серверу. В предыдущих методах один и тот же клиент может попадать на разные серверы.
- **hash** - задаёт метод балансировки, при котором соответствие клиента серверу определяется при помощи хэшированного значения ключа. В качестве ключа может использоваться текст, переменные и их комбинации.
- **random** - балансировка нагрузки, при которой запрос передаётся случайно выбранному серверу, с учётом весов.

В платной версии существуют дополнительный более продвинутый метод распределения нагрузки - **least_time**, при котором запрос передаётся серверу с наименьшими средним временем ответа и числом активных соединений с учётом весов серверов.

Пример настройки:

```
upstream cache-api {
```

```
    ip_hash;
    server 10.32.18.6:8080;
    server 10.32.18.7:8080;
    server 10.32.18.8:8080;
}
```

Проблемы балансировки нагрузки

Проблем при балансировки нагрузки с помощью nginx может быть масса. В обычном рабочем проекте нельзя просто взять и разделить нагрузку на несколько серверов. Само приложение, его БД должны быть готовы к этому. Первое, с чем вы столкнетесь - как правильно определить, что с сервером проблемы. В бесплатной версии nginx нет никаких инструментов для того, чтобы определить, что ваш бэкенд отвечает правильно.

Допустим, один из серверов перегружен и он отдает неправильные ответы. То есть он жив, отвечает, но ответы нам не подходят. Балансировщик будет считать, что все в порядке, запросы на проблемный сервер будут продолжать идти. Nginx исключит его из списка бэкендов только тогда, когда он полностью перестанет отвечать. Но это лишь малая часть проблем, которые могут приключиться. По моему опыту, чаще всего сервера не отваливаются полностью, а начинают тупить или отдавать, к примеру 502 ошибку. В бесплатной версии nginx будет считать, что все в порядке.

Для того, чтобы анализировать ответ бэкенда и в зависимости от этого ответа, решать, в каком состоянии находится сервер, вам необходим модуль ngx_http_upstream_hc_module. Он доступен только в коммерческой подписке. С помощью этого модуля можно тестировать код ответа, наличие или отсутствие определённых полей заголовка и их значений, а также содержимое тела ответа. Без этих данных качественно настроить работу балансировщика трудно.

Отмечу еще несколько полезных настроек, на которые надо обратить внимание, при настройке балансировки нагрузки с помощью nginx:

- **proxy_connect_timeout** - задаёт таймаут для установления соединения с проксированным сервером. При отсутствии ответа за указанное время сервер будет считаться неработающим. Дефолтное значение 60 секунд. Это очень много, если у вас большие нагрузки. За минуту скопится огромное количество висящих соединений, которые мог бы обработать другой бэкенд.
- **proxy_read_timeout** - задаёт таймаут при чтении ответа проксированного сервера. Дефолт тоже 60 секунд. Чаще всего имеет смысл уменьшить значение.

Полный конфиг балансировщика nginx

Привожу пример полного конфига виртуального хоста для балансировки нагрузки на примере nginx. Это не пример с рабочего сервера, так что возможны какие-то мелкие ошибки или опечатки. Не тестировал конфиг, так как это не готовое how to, а просто рекомендации и описание. Составил его чтобы было представление, как все это выглядит в единой конфигурации.

```
log_format upstream '$remote_addr - $host [$time_local] "$request" '
    'request_length=$request_length '
    'status=$status bytes_sent=$bytes_sent '
    'body_bytes_sent=$body_bytes_sent '
    'referer=$http_referer '
    'user_agent="$http_user_agent" '
    'upstream_status=$upstream_status '
    'request_time=$request_time '
    'upstream_response_time=$upstream_response_time '
    'upstream_connect_time=$upstream_connect_time '
    'upstream_header_time=$upstream_header_time';


upstream cache-api {
    ip_hash;
    server 10.32.18.7:8080 max_fails=2 fail_timeout=10s;
    server 10.32.18.6:8080 max_fails=2 fail_timeout=10s;
    server 10.32.18.8:8080 max_fails=2 fail_timeout=10s;
}

server {
    listen 443 ssl http2;
    server_name cache-api.sample.com;
    access_log /var/log/nginx/cache-api-access.log upstream;
    error_log /var/log/nginx/cache-api-error.log;
```

```
    ssl_certificate /etc/ssl/sample.com.crt;
    ssl_certificate_key /etc/ssl/sample.com.key;

    root /var/www/html;

    location / {
    proxy_pass http://cache-api/;
    proxy_read_timeout 15;
    proxy_connect_timeout 3;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Real-IP $remote_addr;
    }
}
```

За подробностями параметров секции с proxy_pass переходите в отдельную статью на эту тему.

Заключение

> Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении. Расскажи, как сделать правильно!

Мне не приходилось пробовать в деле никаких других балансировщиков, кроме Nginx. Знаю, что есть haproxy, но попробовать так и не дошли руки. Бесплатная версия nginx очень слабо подходит для полноценной балансировки крупного проекта, либо я просто не понимаю, как его правильно использовать. Реально не хватает тех фич, которые есть в Nginx Plus. До того момента, как не начал использовать балансировщик, не понимал толком, что там такого в платной версии. Теперь прекрасно понимаю :)

Готовые примеры балансировки с использованием фич Nginx Plus приведены в этой статье. Так же обращаю внимание на формат логов, который для удобства стоит подправить под использование бэкендов. Этот вопрос я рассмотрел отдельно в статье про мониторинг производительности бэкендов с

помощью elk stack.

Буду рад любым комментариям, ссылкам, советам по существу затронутой темы. Я в ней новичок. Ни на что не претендую, поделился своим опытом.