Как использовать Apache в качестве обратного прокси при помощи mod_proxy на Ubuntu 16.04

04.04.2017 11:23

#Apache #Linux/Ubuntu #Python

Введение

Обратный прокси-сервер – это тип прокси-сервера, который ретранслирует HTTP/HTPPS-запросы на один или несколько бэкенд-серверов. Обратные прокси-серверы полезны, так как многие современные веб-приложения для обработки HTTP-запросов используют бэкенд-серверы приложений (к которым пользователи не должны иметь доступ напрямую), а также часто поддерживают только базовые функции HTTP.

Поэтому вы можете использовать обратный прокси-сервер для того, чтобы предупредить возможность пользователям получить прямой доступ к основным серверам приложений. Также использование обратного прокси-сервера поможет вам перераспределить нагрузку от поступающих запросов на несколько других серверов приложений. Благодаря этому улучшится производительность и увеличится отказоустойчивость.

Из этой статьи вы узнаете о том, как настроить Apache для использования в качестве обратного прокси-сервера при помощи mod_proxy – модуля Apache для перенаправления соединений на один или несколько бэкенд-серверов в этой же сети. В этом руководстве используется простой бэкенд, написанный с использованием фреймворка Flask, но вы можете использовать любой бэкенд на ваше усмотрение.

Требования

Для того, чтобы выполнить необходимые действия, вам понадобится:

- сервер с установленной ОС Ubuntu 16.04 и пользователем, который может выполнять команды sudo;
- установленный на вашем сервере Apache.

Шаг 1: включение необходимых модулей Apache

Существует множество модулей Apache, которые идут в комплекте и доступны к использованию, но не активированы после установки. Поэтому необходимо включить те модули, которые будут использоваться в этом руководстве.

Нужный нам модуль – это mod_proxy, а также несколько дополнений, которые расширяют его функционал и позволяют поддерживать различные сетевые протоколы. Если перечислять более конкретно, то понадобятся:

- mod_proxy – главный модуль Apache для перенаправления соединений; благодаря ему Apache может выступать в качестве шлюза для основных серверов приложений;
- mod_proxy_http – позволит использовать прокси для HTTP;
- mod_proxy_balancer и mod_lbmethod_byrequests – добавляют функции балансировки нагрузки для бэкенд-серверов.

Для того, чтобы активировать модули, выполните команды ниже в соответствующем порядке:

```
$ sudo a2enmod proxy
$ sudo a2enmod proxy_http
$ sudo a2enmod proxy_balancer
$ sudo a2enmod lbmethod_byrequests
```

Затем перезапустите Apache для того, чтобы изменения вступили в силу.

```
$ sudo systemctl restart apache2
```

Теперь Apache будет выступать в качестве обратного прокси-сервера для HTTP-запросов. Следующий шаг не является обязательным, но он поможет протестировать работу Apache и убедиться, что все работает так, как нужно. Для этого необходимо создать два базовых бэкенд-сервера, однако если у вас они уже есть в наличии, то вы можете сразу перейти к шагу 3.

Шаг 2: создание тестовых бэкенд-серверов

Запуск нескольких базовых бэкенд-серверов – отличная возможность протестировать, корректно ли работает Apache. Поэтому необходимо создать два сервера, которые будут отвечать на HTTP-запросы, печатая строку текста. Один будет отвечать “Hello world!”, другой “ Hello Timeweb!”.

Примечание. Если говорить не о тестовом запуске, то обычно бэкенд-серверы отдают одинаковый тип контента. Однако для тестирования будет удобнее, если это будет два сервера с разными сообщениями, потому что так будет проще отследить, что балансировка нагрузки работает корректно.

Flask – это микрофреймворк Python, который используется для создания веб-приложений. В этом руководстве мы будем использовать Flask, так как для

написания базового приложения требуется всего несколько строк. Вам необязательно знать Python для того, чтобы выполнить дальнейшие действия.

Для начала обновите список доступных пакетов:

```
$ sudo apt-get update
```

Теперь установите Pip, рекомендованная система управления пакетами Python.

```
$ sudo apt-get -y install python3-pip
```

И уже при помощи Pip установите Flask:

```
sudo pip3 install flask
```

Теперь, когда все необходимые компоненты установлены, можно перейти к созданию файла, в котором будет находиться необходимый для первого бэкенд-сервера код.

Файл можно создать в домашней директории пользователя, под которым вы авторизованы.

```
$ nano ~/backend1.py
```

Скопируйте в файл текст, который приведен ниже, а затем сохраните и закройте файл.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
 return 'Hello world!'
```

Первые две строчки инициализируют фреймворк Flask. Единственная функция home() будет отдавать строку текста (“Hello world!”). А за то, чтобы этот текст появлялся в ответ на HTTP-запросы, отвечает @app.route('/').

Второй бэкенд-сервер будет точно таким же, как и первый, кроме единственного отличия в тексте.

Поэтому для начала необходимо скопировать первый файл:

```
$ cp ~/backend1.py ~/backend2.py
```

Теперь откройте новый скопированный файл:

```
$ nano ~/backend2.py
```

И в последней строчке вместо “Hello world!” напишите, к примеру, “Hello Timeweb!”:

```
return 'Hello Timeweb!'
```

Команда ниже запустит первый бэкенд-сервер, его порт 8080.

```
$ FLASK_APP=~/backend1.py flask run --port=8080 >/dev/null 2>&1 &
```

Второй сервер тоже необходимо запустить, его порт 8081:

```
$ FLASK_APP=~/backend2.py flask run --port=8081 >/dev/null 2>&1 &
```

Теперь протестируем работу обоих серверов.

Сначала первого:

```
$ curl http://127.0.0.1:8080/
```

В ответ вы должны получить “Hello world!”.

Затем второй:

```
$ curl http://127.0.0.1:8081/
```

В этом случае вы увидите в терминале надпись “Hello Timeweb!”.

Примечание. Для того, чтобы закрыть оба тестовых сервера (например, после выполнения всех действий в этом руководстве), вы можете просто выполнить следующую команду: killall flask.

Далее мы изменим конфигурационный файл Apache для того, чтобы появилась возможность использовать его в качестве обратного прокси-сервера.

Шаг 3: изменение изначальных установок для включения прокси

В этой части необходимо внести изменения для того, чтобы изначальный виртуальный хост Apache выполнял роль обратного прокси-сервера для одного или нескольких бэкенд-серверов.

Сначала нужно открыть конфигурационный файл Apache в редакторе nano (или в другом редакторе на ваш выбор):

```
$ sudo nano /etc/apache2/sites-available/000-default.conf
```

Внутри этого файла найдите блок с первой строкой <VirtualHost *:80> . Первый пример ниже продемонстрирует, как изменить этот файл так, чтобы использовать обратный прокси для одного бэкенд-сервера, а второй пример – для установки балансировки нагрузки для нескольких бэкенд-серверов.

1 пример: обратное прокси для одного бэкенд-сервера

Скопируйте текст ниже **вместо** всего текста, расположенного в блоке VirtualHost, то есть чтобы в итоге блок выглядел вот так:

```
<VirtualHost *:80>
 ProxyPreserveHost On

 ProxyPass / http://127.0.0.1:8080/
 ProxyPassReverse / http://127.0.0.1:8080/
</VirtualHost>
```

Если вы продолжаете следовать этому руководству и используете тестовые серверы, которые создали ранее, то скопируйте 127.0.0.1:8080, как и написано в примере. Если у вас есть свои собственные серверы приложений, то используйте их адреса.

В блоке используется три директивы:

- ProxyPreserveHost – заставляет Apache передать оригинальный заголовок Host бэкенд-серверу. Это полезно, так как в этом случае бэкенд-сервер получает адрес, который используется для доступа к приложению;
- ProxyPass – основная директива для настройки прокси. В данном случае она указывает, что все, что идет после корневого адреса URL (/), должно быть отправлено на бэкенд-сервер по указанному адресу. Например, если Apache получит запрос /primer, то он подключится к http://ваш_бэкенд-сервер/primer и отправит соответствующий ответ;
- ProxyPassReverse – должна иметь такие же настройки, как и ProxyPass. Она сообщает Apache, как изменить заголовки в ответе от бэкенд-сервера. Таким образом гарантируется, что браузер клиента будет перенаправлен на прокси-адрес, а не на адрес бэкенд-сервера.

После внесения изменений Apache необходимо перезапустить:

```
$ sudo systemctl restart apache2
```

Теперь, если вы наберете в браузере адрес своего сервера, вы увидите ответ от вашего бэкенд-сервера вместо приветственной страницы Apache.

2 пример: балансировка нагрузки между несколькими бэкенд-серверами

Если у вас есть несколько бэкенд-серверов, будет хорошей идеей при использовании прокси распределить трафик между ними; сделать это можно при помощи функции балансировки нагрузки утилиты mod_proxy.

Как и в первом примере, тут вам тоже необходимо заменить текст в блоке VirtualHost на следующий:

```
<VirtualHost *:80>
<Proxy balancer://mycluster>
 BalancerMember http://127.0.0.1:8080
 BalancerMember http://127.0.0.1:8081
</Proxy>

 ProxyPreserveHost On

 ProxyPass / balancer://mycluster/
```

```
 ProxyPassReverse / balancer://mycluster/
</VirtualHost>
```

В целом текст похож на предыдущий, однако вместо указания одного бэкенд-сервера появляется новый блок Proxy, в котором указано несколько серверов. Блок называется balancer://mycluster (название можно изменить) и состоит из одного или нескольких BalancerMembers, которые определяют адреса лежащих в основе бэкенд-серверов.

Директивы ProxyPass и ProxyPassReverse используют пул балансировки нагрузки под названием mycluster вместо конкретного сервера.

Вы можете использовать адреса тестовых серверов (как указано выше), либо заменить их на адреса своих серверов.

Для того, чтобы изменения вступили в силу, перезапустите Apache:

```
$ sudo systemctl restart apache2
```

Теперь проведите такой же тест, как и в первом примере: введите IP-адрес вашего сервера в браузер и вместо стандартного приветствия Apache вы увидите один из ответов бэкенд-серверов. Если вы используете тестовые серверы, то это будет либо “Hello world!”, либо “Hello Timeweb!”. Обновите страницу несколько раз и, если вы увидели оба текста, значит все работает корректно.

Заключение

Теперь вы знаете, как настроить Apache в качестве обратного прокси-сервера для одного или нескольких внутренних серверов. mod_proxy можно эффективно использоваться для того, чтобы настраивать обратный прокси для серверов с приложениями, написанных на различных языках программирования и технологиях, таких как Python, Django или Ruby и Ruby on Rails. Также mod_proxy можно использовать для балансировки нагрузки между несколькими бэкенд-серверами для сайтов с большой нагрузкой, чтобы обеспечить высокую доступность таких ресурсов.

mod_proxy и mod_proxy_http самая популярная комбинация модулей, однако есть несколько других, которые поддерживают другие сетевые протоколы. Хотя в этом руководстве они не использовались, их тоже можно выделить отдельным списком:

- mod_proxy_ftp для FTP;
- mod_proxy_connect для SSL-туннеля;
- mod_proxy_ajp для протокола AJP (Apache JServ Protocol);
- mod_proxy_wstunnel для веб-сокетов.

Узнать более подробно о них вы можете в официальной документации mod_proxy Apache: http://httpd.apache.org/docs/current/mod/mod_proxy.html