716,11

Рейтинг

ДомКлик

Место силы

Морскоu сегодня в 11:02

Липкие сессии для самых маленьких [Часть 1]

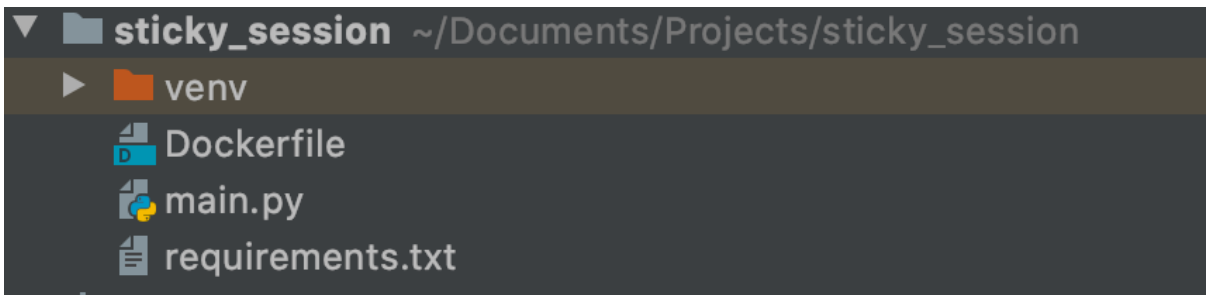
Блог компании ДомКлик, Python, Nginx

Tutorial

Липкие сессии (**Sticky-session**) — это особый вид балансировки нагрузки, при которой трафик поступает на один определенный сервер группы. Как правило, перед группой серверов находится балансировщик нагрузки (**Nginx**, **HAProxy**), который и устанавливает правила распределения трафика между доступными серверами.

В первой части цикла мы посмотрим как создавать липкие сессии с помощью *Nginx*. Во второй же части разберем создание подобной балансировки средствами *Kubernetes*.

Перед тем как настроить *nginx*, сделаем простенький сервис на фреймворке *FastAPI*. Создадим проект с виртуальным окружением Python 3.6+. В директории проекта должны находится следующие файлы:



Файл **requirements.txt** содержит несколько зависимостей:

```
fastapi==0.63.0
uvicorn==0.13.3
```

main.py содержит следующий код:

```
from fastapi import FastAPI
from uuid import uuid4

app = FastAPI()
uuid = uuid4()


@app.get("/")
async def root():
    return {'uuid': uuid}
```

Обратите внимание на переменную **uuid**, которая инициализируется вместе с *FastAPI* приложением. Переменная будет жить, пока работает сервер. Собственно, по значению этой переменной мы будем точно знать, что попали на тот же самый экземпляр приложения. Перед тем как запустить сервис нужно установить зависимости:

```
pip install -r requirements.txt
```

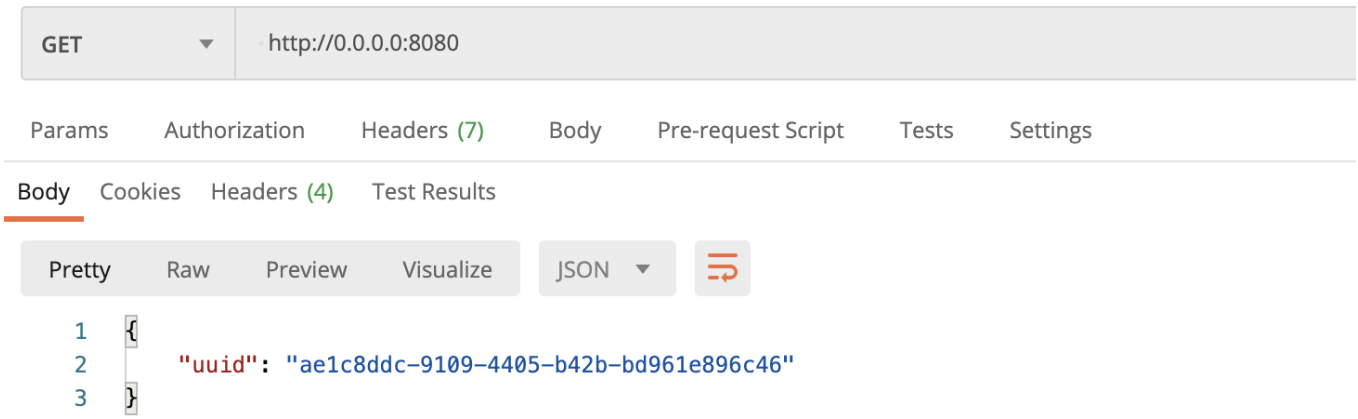
Запустить сервис можно командой:

```
uvicorn main:app --port 8080
```

Вывод будет такой:

```
INFO:     Started server process [72271]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
```

Проверим работу сервиса с помощью Postman, указываем `0.0.0.0:8080` и нажимаем Send. Сервер ответит сгенерированным *uuid*:



Рассмотрим содержание **Dockerfile**. Предполагается, что у вас есть хотя бы небольшой опыт работы с контейнерами.

```
FROM python:3.8

WORKDIR app

COPY . /app
```

```
RUN pip install -r requirements.txt

EXPOSE 8080

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Это очень похоже на то, что мы сделали вручную: создали окружение, установили зависимости, запустили проект. Ну, разве что порт не публиковали.

Соберём образ (не забываем про точку):

```
docker build -t sticky:0.0.1 .
```

```
 ---> Running in 1de3ee7b2c1c
Removing intermediate container 1de3ee7b2c1c
 ---> a63825b07f76
Successfully built a63825b07f76
Successfully tagged sticky:0.0.1
```

И запустим контейнер с пробросом портов:

```
docker run -p 8080:8080 sticky:0.0.1
```

```
(venv) ➜  sticky_session git:(master) ✗ docker run -p 8080:8080 sticky:0.0.1
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Пощелкайте Postman убедитесь, что все также работает. Теперь сделаем запуск приложения в несколько реплик.

В корне директории проекта создадим файлик ***docker-compose.yaml***. Вставим следующий код:

```
version: '3.4'

services:
    web:
        build:
            context: .
        ports:
            - "8080-8081:8080"
```

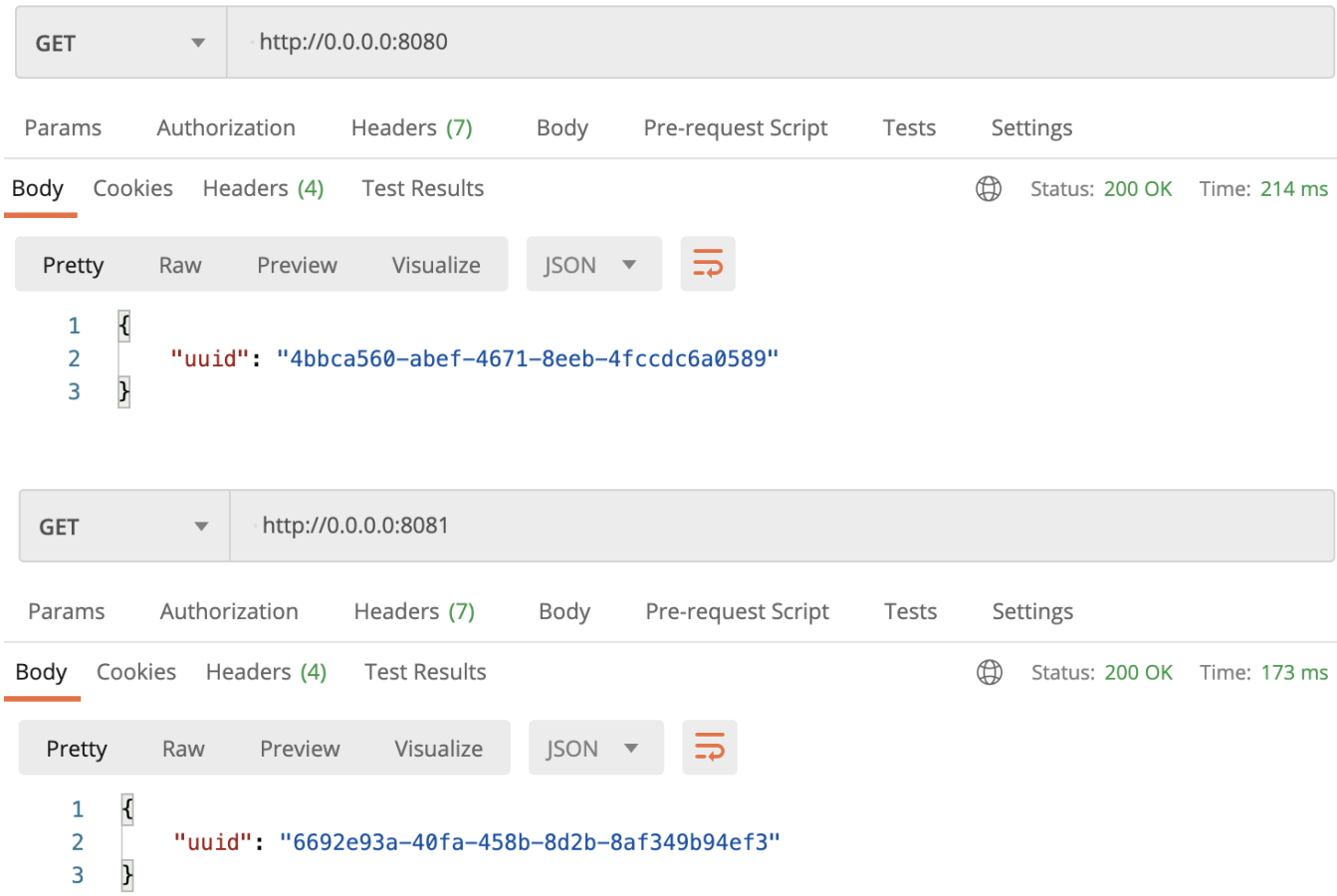
Данная инструкция запустит в *docker-compose* приложение **web**. В **build** указывается место расположения образа (в нашем случае *Dockerfile* лежит в корне директории) на основе которого собирается контейнер. В **ports** открываем порты `8080` и `8081` которые будут связаны с портом `8080` внутри контейнера. Запись в виде диапазона нужна при запуске приложения в несколько инстансов (реплик). Приложения сами займут свободные порты из предоставленного пула. Только единственное условие: количество портов в пуле должно быть больше либо равно количеству запускаемых реплик, иначе возникнет ошибка.

Запустить несколько контейнеров разом можно командой:

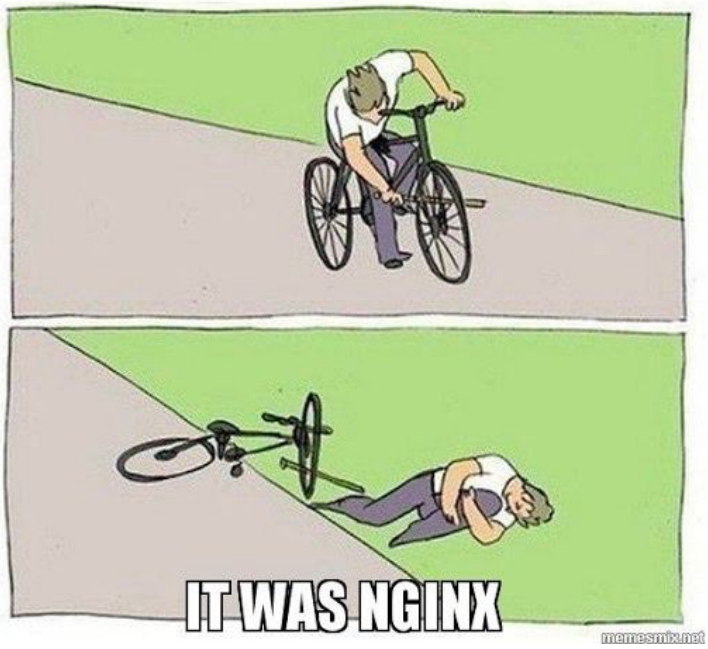
```
docker-compose up --build --scale web=2
```

```
web_1  | INFO:     Started server process [1]
web_1  | INFO:     Waiting for application startup.
web_1  | INFO:     Application startup complete.
web_1  | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
web_2  | INFO:     Started server process [1]
web_2  | INFO:     Waiting for application startup.
web_2  | INFO:     Application startup complete.
web_2  | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

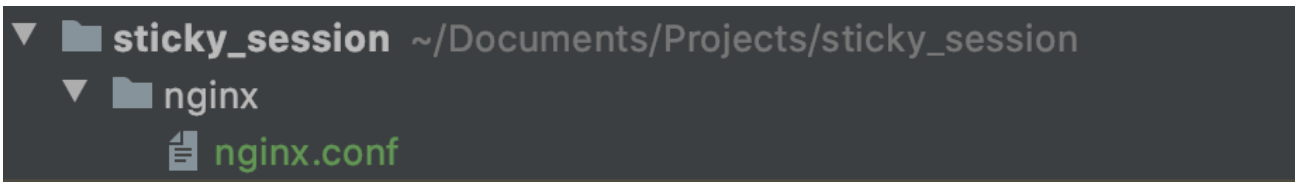
В *Postman* проверим работу контейнеров `0.0.0.0:8080` и `0.0.0.0:8081` – отвечают оба сервера:



Теперь можно приступить к настройке **Nginx**!



Создадим папку *nginx*, а в нем файлик **nginx.conf**.



Внутри *nginx.conf* опишем следующее:

```
worker_processes auto;

events {

}

http {
    upstream sticky-app {
        server web:8080;
        server web:8081;

    }

    server {
        listen 80;
        location / {
            proxy_pass http://sticky-app;
        }
    }
}
```

Данный конфиг указывает *nginx* слушать порт 80 и весь трафик проксировать на сервера перечисленные в **upstream**. В данной конфигурации происходит балансировка типа round-robin, липкие сессии добавим попозже.

Теперь поднимем *nginx* вместе с приложением *web*. В **docker-compose-yaml** добавим следующее:

```
    nginx:
        image: nginx:latest
        container_name: my-nginx
        depends_on:
          - web
        volumes:
            - ./nginx/nginx.conf:/etc/nginx/nginx.conf
        ports:
            - 80:80
            - 443:443
```

Тут все просто. В **image** указывается какой официальный образ забрать при сборке. В **Volumes** мы монтируем наш конфиг из рабочей директории прямиком в контейнер. Это например, позволит нам изменять конфиг *nginx* и не пересобирать заново *docker-compose*, достаточно будет только перезапустить *nginx* контейнер.

Финальный штрих: подключим приложение web и nginx к новой сети **my-app**. Благодаря этому в конфиге *nginx* можно указывать DNS сервиса (web), который определен в *docker-compose*.

Полный **docker-compose.yaml** выглядит так:

```
version: '3.4'

services:

    web:
        build:
            context: .
        ports:
            - "8080-8081:8080"
        networks:
            - app-net

    nginx:
        image: nginx:latest
        container_name: my-nginx
        depends_on:
          - web
        volumes:
            - ./nginx/nginx.conf:/etc/nginx/nginx.conf
        ports:
            - 80:80
            - 443:443
        networks:
            - app-net

networks:
  app-net:
     driver: bridge
```

Удалим запущенные контейнеры:

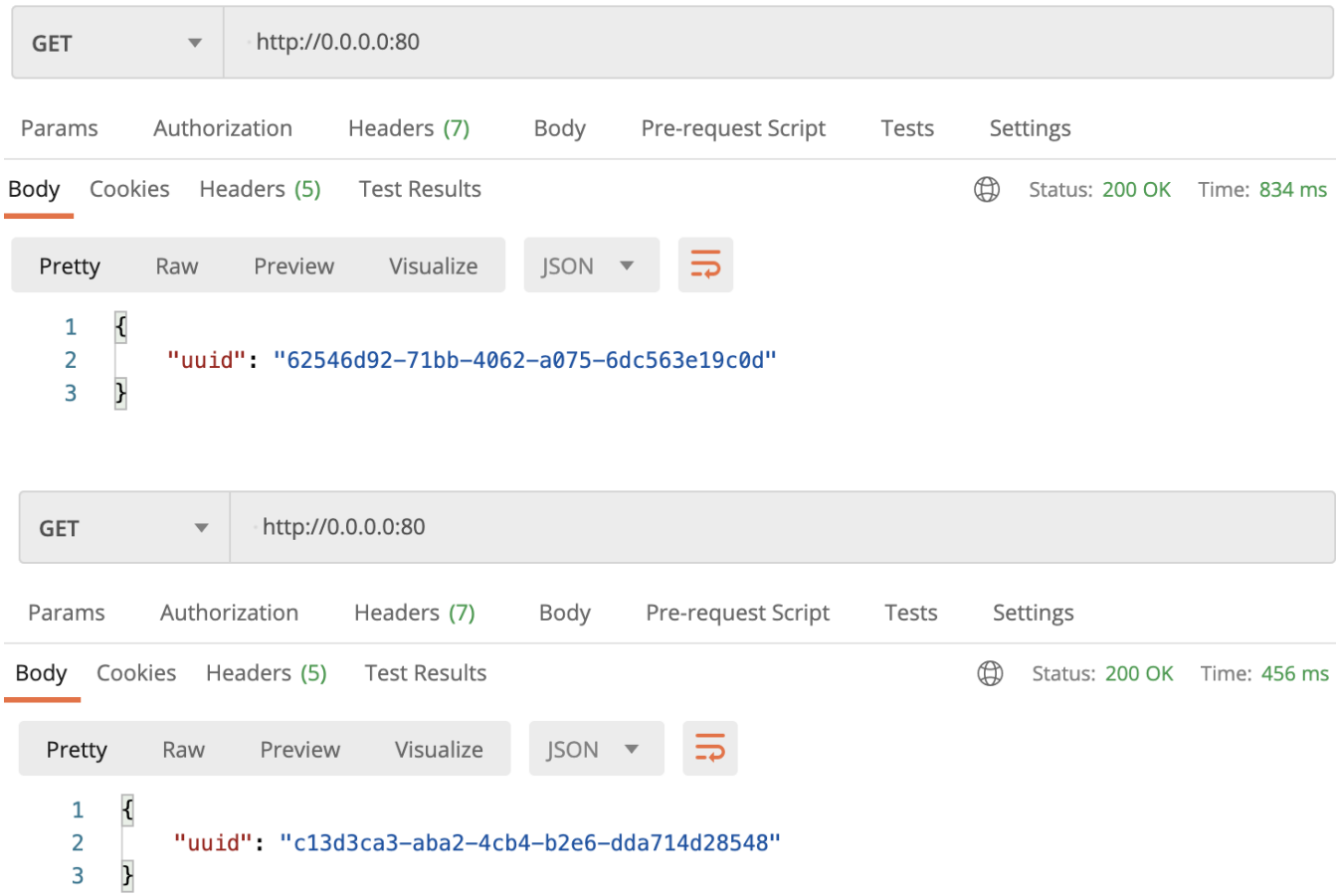
```
docker-compose down
```

И запустим сервис в двух экземплярах вместе с *nginx*:

```
docker-compose up --build --scale web=2
```

```
Attaching to sticky_session_web_1, sticky_session_web_2, my-nginx
web_2    | INFO:     Started server process [1]
web_2    | INFO:     Waiting for application startup.
web_1    | INFO:     Started server process [1]
web_2    | INFO:     Application startup complete.
web_2    | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
web_1    | INFO:     Waiting for application startup.
web_1    | INFO:     Application startup complete.
web_1    | INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

В *Postman* отправляем запросы на порт 80 и видим, что балансировка работает – ответы приходят разные.



Теперь сделаем наконец липкие сессии! Обновим конфиг *nginx* добавив одну строчку в блок **upstream**:

```
    upstream sticky-app {
        hash $cookie_key;
        server web:8080;
        server web:8081;
    }
```

▾ Полный конфиг nginx

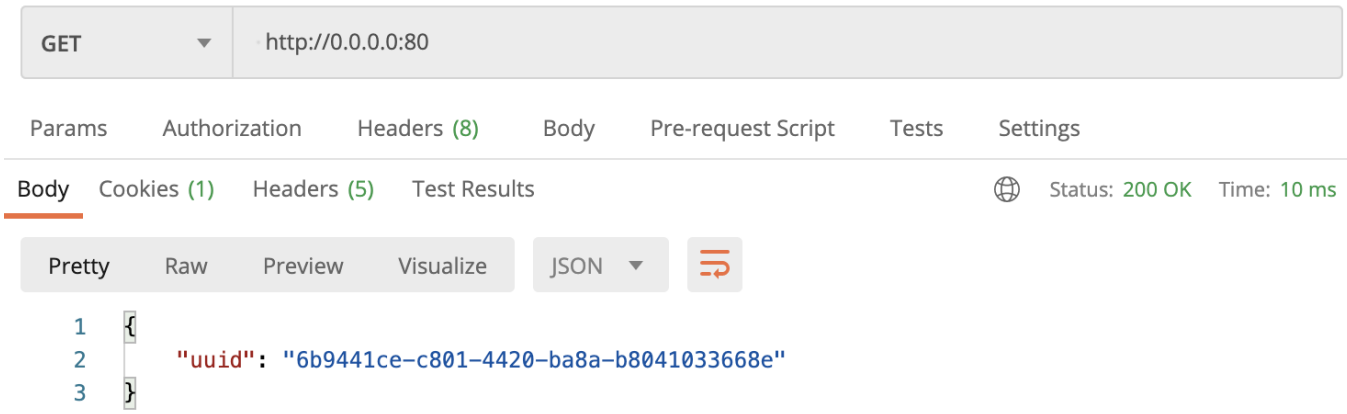
```
worker_processes auto;

events {

}

http {
    upstream sticky-app {
        hash $cookie_key;
        server web:8080;
        server web:8081;
    }

    server {
        listen 80;
        location / {
            proxy_pass http://sticky-app;
        }
    }
}
```

Nginx будет брать хэш от *cookie* с именем **key** и если такой хэш уже был - направит трафик на тот же сервер. Перезапустим *docker-compose*:

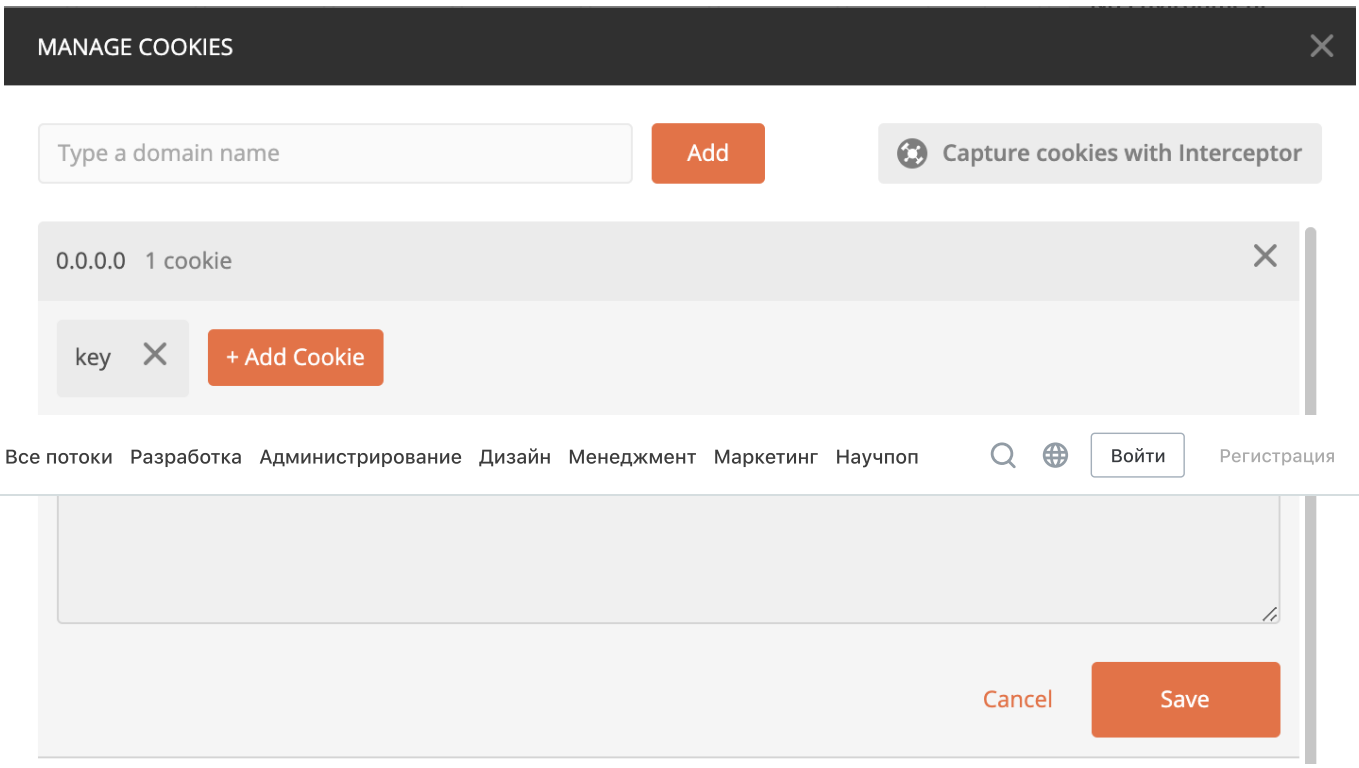
```
docker-compose down
docker-compose up --build --scale web=2
```

После сборки создадим в *Postman* cookie с именем ***key*** для адреса `0.0.0.0` :

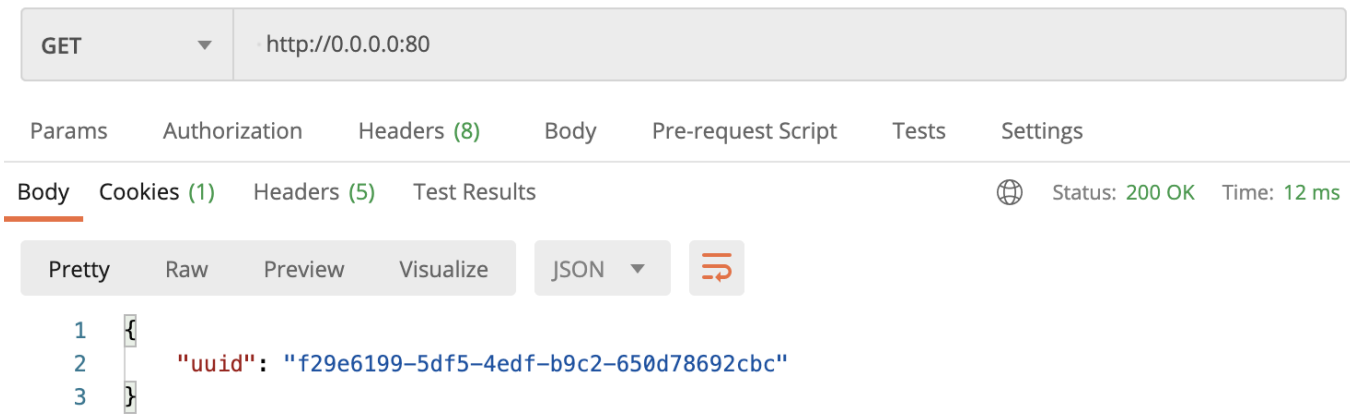
Проверим `0.0.0.0:80`



Сколько бы я не отправлял запросы - ответ не меняется. Теперь изменим значение ***key:***



И отправим запрос по тому же адресу:



Ответ изменился и больше не меняется с отправкой новых запросов. **Готово!** Таким простым способом можно реализовать липкие сессии в *nginx*.

В следующей части разберем создание липкой сессии в ***kubernetes***.

Теги: nginx, python, sticky sessions, microservices

Хабы: Блог компании ДомКлик, Python, Nginx

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

Уважаемые рекрутеры, а вы не офигели?

+144 58,5k 65 236

Два с половиной странных правила английского языка, которых не учат в школе

+41 56,2k 122 107

Windows 95 — как она выглядит сегодня?

+77 28,3k 53 201

Zoom запретил пользоваться сервисом властям России и госкомпаниям

+54 40,1k 11 55

Летай как бабочка, жаль как пчела, учись на Learn

Подборка