



dusterio 13 ноября 2016 в 13:23

Простой API gateway на базе PHP и Lumen

API, Laravel, PHP, Программирование

Tutorial

Термин «микросервисы» сегодня у всех на слуху – внезапно это стало очень модно, и многие компании объявляют переход на этот архитектурный паттерн даже толком не разобравшись в нём. Впрочем, обсуждение полезности микросервисов оставим за пределами этой статьи.

Традиционно перед коллекцией микросервисов предлагается дополнительный слой – так называемый API gateway, который решает сразу несколько проблем (они будут перечислены позже). На момент написания этой статьи open source реализаций таких gateway почти нет, поэтому я решил написать свой на PHP с использованием микрофреймворка Lumen (часть Laravel).

В этой статье я покажу насколько это простая задача для современного PHP!

Что такое API gateway?

Если говорить совсем коротко, то [API gateway](#) – это умный проху-сервер между пользователями и любым количеством сервисов (API), отсюда и название.

Необходимость в этом слое появляется сразу же при переходе на паттерн микросервисов:

- Единый адрес намного удобнее сотни (у Netflix их более 600) индивидуальных адресов API;
- Логично проверять данные пользователя (token) в едином месте, на «входе»;
- Удобно реализовывать ограничения на количество запросов в едином месте;
- Вся система становится более гибкой – можно менять внутреннюю структуру хоть каждый день. Поддержка старых версий API становится тривиальным делом;
- Можно кешировать или мутировать ответы;
- Для удобства пользователя (или разработчиков front end) можно объединять ответы от разных сервисов. Facebook давно предлагает такую возможность.

Преимуществ больше – это просто те, что пришли на ум за 10-20 секунд.

Nginx выпустили [неплохую бесплатную электронную книгу](#) посвященную микросервисам и API gateway – советую почитать всем, кому интересен этот паттерн.

Существующие варианты

- [API Umbrella](#), Lua;
- [Kong](#), Lua;
- [AWS API Gateway](#) – платный сервис от Amazon.

Как я уже сказал выше, вариантов очень мало, да и те появились сравнительно недавно. Многих возможностей в них пока нет.

Почему PHP и Lumen?

С выходом версии 7 PHP стал высокопроизводительным, а с появлением фреймворков вроде Laravel и Symfony – PHP доказал миру, что может быть красивым и функциональным. Lumen, являясь «очищенной» быстрой версией Laravel здесь идеально подходит, ведь нам не нужны будут сессии, шаблоны и прочие возможности full stack приложений.

Кроме того, у меня просто больше опыта с PHP и Lumen, а разворачивая полученное приложение через Docker – будущим пользователям вообще будет не важен язык, на котором оно написано. Это просто слой,

Digital Brand Day 2019: секретные доклады, свежая аналитика, лучшие рекламные кейсы и много-много дискуссионных панелей.

Жми

Реклама

ЧИТАЮТ СЕЙЧАС

Не купили DLC: функцию, которая спасла бы упавшие 737, «Боинг» продавал как опцию

90,6k

753

Компьютер Raspberry Pi встроили в клавиатуру для него

10,1k

39

Результаты Pwn2Own: Tesla Model 3 взломана, на ней поехал домой автор нового метода атаки

2,6k

0

Как я пишу конспекты по математике на LaTeX в Vim

32k

101

В личном чате Telegram можно удалять любые сообщения — даже чужие

17k

79

Игра для любителей и знатоков Linux

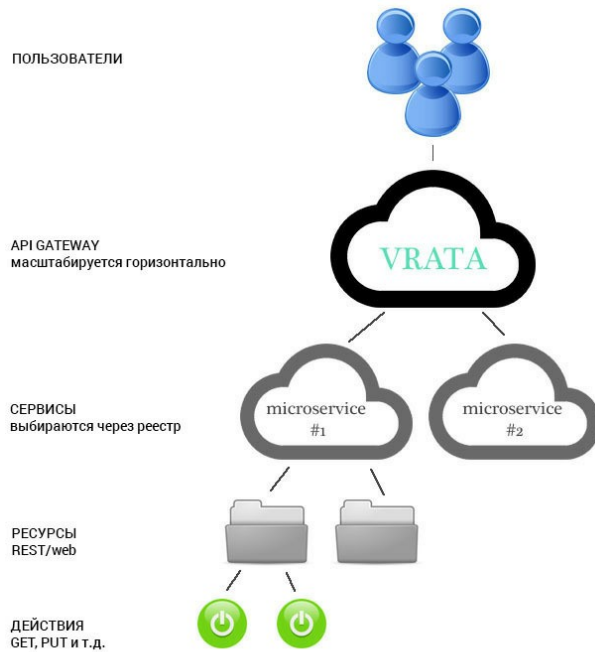
5,8k

4

который выполняет свою роль!

Выбранная терминология

Мною предлагается следующая архитектура и соответствующая ей терминология. В коде буду придерживаться этих терминов, чтобы не запутаться:



Само приложение решил назвать [Vrata](#), потому что «врата» на русском это почти «gateway», а ещё миру не хватает приложений с русскими названиями)

Непосредственно за «вратами» находится количество N микросервисов – API сервисов, способных отвечать на web-запросы. У каждого сервиса может быть любое количество экземпляров, поэтому API gateway будет выбирать конкретный экземпляр через так называемый реестр сервисов.

Каждый сервис предлагает какое-то количество ресурсов (на языке REST), а у каждого ресурса может быть несколько возможных действий. Достаточно простая и логичная структура для любого опытного в REST программиста.

Требования к Vrata

Ещё не приступив к коду, можно сразу определить некоторые требования к будущему приложению:

- Шлюз должен масштабироваться горизонтально, потому что на дворе 2016 год и все хотят масштабировать горизонтально. Следовательно – никакого состояния приложения не должно быть;
- Шлюз должен уметь объединять запросы и вызывать микросервисы асинхронно;
- Шлюз должен уметь ограничивать количество запросов в промежуток времени;
- Шлюз должен уметь проверять достоверность токена аутентификации. Традиционно предлагается, что API gateway выполняет аутентификацию, а скрытые под ним микросервисы выполняют авторизацию на свои ресурсы;
- Шлюз должен уметь автоматически импортировать доступные ресурсы с микросервисов. Для начала выберем формат Swagger, как самый популярный в мире на сегодня;
- Шлюз должен уметь менять (мутировать) ответы микросервисов;
- И напоследок: шлюз должен прекрасно запускаться напрямую из образа Docker и конфигурироваться через переменные окружения. Мы не хотим никаких дополнительных репозиторий, скриптов деплоя и так далее!

Скажу сразу, что большая часть пунктов уже работает, а реализовать их было очень просто. Ведь правду говорят – мы живем в лучшую для программиста эпоху!

Реализация

Аутентификация

В этом направлении почти не пришлось работать – достаточно было поправить Laravel Passport под Lumen и мы получили поддержку всех современных OAuth2 фич, включая JWT. Мой маленький пакет-порт опубликован на [GitHub/Packagist](#) и кто-то его уже устанавливает.

Маршруты и контроллер

Все низлежащие маршруты с микросервисов импортируются в Vrata из конфигурационного файла в формате JSON. В момент запуска в service provider происходит добавление этих маршрутов:

```
// Получаем синглетный класс - база данных всех маршрутов
$registry = $this->app->make(RouteRegistry::class);

// Передаем наш Lumen контейнер этой базе, чтобы она могла зарегистрировать маршруты
$registry->bind($app());
```

А тем временем в базе маршрутов:

```
/**
 * @param Application $app
 */
public function bind(Application $app)
{
    // Очень просто - маршрут за маршрутом добавляем в Lumen
    // Все запросы пойдут в один и тот же служебный контроллер
    // Добавляем middleware для аутентификации OAuth2, а также своего дополнительного по
    // мощника
    $this->getRoutes()->each(function ($route) use ($app) {
        $method = strtolower($route->getMethod());
        $app->{$method}($route->getPath(), [
            'uses' => 'App\Http\Controllers\GatewayController@' . $method,
            'middleware' => [ 'auth', 'helper:' . $route->getId() ]
        ]);
    });
}
```

Теперь каждому публичному (и разрешенному в конфигах) маршруту с микросервисов соответствует маршрут на API gateway. Кроме того, добавлены также синтетические или объединенные запросы, которые существуют только на этом шлюзе. Все запросы уходят в один и тот же контроллер:

Вот так контроллер обрабатывает любой GET-запрос:

```
/**
 * @param Request $request
 * @param RestClient $client
 * @return Response
 */
public function get(Request $request, RestClient $client)
{
    // Это наша баночка с параметрами, подробнее - позже
    $parametersJar = $request->getRouteParams();

    // Соберем финальный ответ из N ответов микросервисов
    $output = $this->actions->reduce(function ($carry, $batch) use (&$parametersJar, $client) {
        // Соберем N ответов полученных асинхронно
        $responses = $client->asyncRequest($batch, $parametersJar);

        // Добавим необходимые новые параметры в баночку параметров
        $parametersJar = array_merge($parametersJar, $responses->exportParameters());

        // Склеим с текущим состоянием - делаем array reduce
        return array_merge($carry, $responses->getResponses()->toArray());
    }, []);

    // Отдаем ответ классу форматирования. Сейчас это только JSON
}
```

```
return $this->presenter->format($this->rearrangeKeys($output), 200);  
}
```

В качестве HTTP-клиента выбран Guzzle, который прекрасно справляется с `async`-запросами, а также имеет готовые средства для `integration`-тестирования.

Составные запросы

Уже работают сложные, составные запросы – это когда одному маршруту на шлюзе соответствует любое количество маршрутов на разных микросервисах. Вот рабочий пример:

```
// Boolean-флаг, обозначающий сложный маршрут  
'aggregate' => true,  
'method' => 'GET',  
// Любой путь на наш вкус, параметры из него сразу попадут в "jar"  
'path' => '/v2/devices/{mac}/extended',  
// Массив с низлежащими маршрутами  
'actions' => [  
  'device' => [  
    // Имя микросервиса из реестра сервисов  
    'service' => 'core',  
    'method' => 'GET',  
    'path' => 'devices/{mac}',  
    // Компоненты с одинаковым порядком будут запущены параллельно  
    'sequence' => 0,  
    // Если в составе есть критичные компоненты и они недоступны - весь маршрут недоступен  
    'critical' => true  
  ],  
  'ping' => [  
    'service' => 'history',  
    // Вывод никак не участвует в нашем финальном ответе  
    'output_key' => false,  
    'method' => 'POST',  
    'path' => 'ping/{mac}',  
    'sequence' => 0,  
    'critical' => false  
  ],  
  'settings' => [  
    'service' => 'core',  
    // Вставляем вывод под альтернативным JSON-ключом  
    'output_key' => 'network.settings',  
    'method' => 'GET',  
    // Используем параметр, добытый ранее в пункте 'device'  
    'path' => 'networks/{device%network_id}',  
    'sequence' => 1,  
    'critical' => false  
  ]  
]
```

Как видим, сложные маршруты уже доступны и обладают неплохим набором фич – можно выделять критически важные из них, можно делать параллельные запросы, можно использовать ответ одного сервиса в запросе к другому и так далее. Помимо всего прочего, на выходе прекрасная производительность – всего 56 миллисекунд на получение суммарного ответа (загрузка Lumen и три фоновых запроса, все микросервисы с базами данных).

Реестр сервисов

Это пока самая слабая часть – реализован только один очень простой метод: DNS. Несмотря на всю его примитивность, он отлично работает в среде вроде Docker Cloud или AWS, где сам провайдер наблюдает за группой сервисов и динамически редактирует DNS-запись.

В настоящий момент Vrata просто берет `hostname` сервиса, не вникая – облако это или один физический компьютер. Самым популярным реестром на сегодня, пожалуй, является [Consul](#), и именно его стоит добавить следующим.

Суть работы реестра очень проста – надо хранить таблицу живых и мертвых экземпляров сервиса, выдавая адреса конкретных экземпляров когда надо. AWS и Docker Cloud (и многие другие) умеют это делать за вас, предоставляя вам один «волшебный» `hostname`, который всегда работает.

Образ Docker

Говоря о микросервисах просто нельзя не упомянуть Docker – одну из самых «горячих» технологий последних пары лет. Микросервисы, как правило, тестируются и деплоятся именно как образы Docker – это стало стандартной практикой, поэтому мы быстро подготовили публичный образ в Docker Hub.

Одна команда, введённая в терминале любой OS X, Windows или Linux машины, и у вас работает мой шлюз Vrata:

```
$ docker run -d -e GATEWAY_SERVICES=... -e GATEWAY_GLOBAL=... -e GATEWAY_ROUTES=... pwred/vrata
```

Всю конфигурацию можно передать в переменных окружения в формате JSON.

Послесловие

Приложение (шлюз) уже используется на практике в компании, где я работаю. Весь код в [репозитории на GitHub](#). Если кто-либо хочет поучаствовать в разработке – милости просим :)

Так как составные запросы как по идее, так и по реализации очень напоминают продвигаемый Facebook формат запросов GraphQL (в противовес REST), то одна из приоритетных будущих фиц – поддержка GraphQL-запросов.

Теги: микросервисы, docker, lumen, laravel, api gateway

+21

119

21,1k

12



22,7

Карма

0,8

Рейтинг

25

Подписчики

Denis Mysenko [@dusterio](#)

Волшебник IT

Поделиться публикацией



ПОХОЖИЕ ПУБЛИКАЦИИ

25 августа 2017 в 21:09

Первые впечатления о Laravel API Resources

+13

14,3k

52

10

23 февраля 2017 в 13:57

Документирование #микросервисов

+11

56,7k

93

13

8 февраля 2015 в 18:13

Разработка микросервисов с использованием Scala, Spray, MongoDB, Docker и Ansible

+29

28,9k

210

8

ВАКАНСИИ

Мой круг



PHP Laravel разработчик

от 800 до 1500 \$




Оки-Токи: Колл-центр в облаках • Возможна удаленная работа



Веб-разработчик Laravel








от 80000 Р

INVITE Transport Software • Тольятти • Возможна удаленная работа

	Laravel/Yii2 разработчик на удаленку Риверстарт · Нижний Новгород · Возможна удаленная работа	от 60000 до 150000 Р
	PHP разработчик Радюшин и Компания · Тольятти	от 50000 до 100000 Р
	PHP-программист в Зеленограде (Yii2, Laravel) Студия Валерия Комягина · Москва	от 70000 до 110000 Р

[Все вакансии](#)

Комментарии 12

	zcasper 13 ноября 2016 в 18:21	0
Мне кажется тут стоило упомянуть о ESB (Enterprise Service Bus)...		
	boodda 13 ноября 2016 в 22:05	0
интересно. у меня вопрос почему в конфигах все значения настроек надо прописывать строками, для чего это. я не хочу помнить их, пишите как константы классов. ide всегда напомнит и дополнит все что я хочу и еще и с комментариями. не пишите строки!		
	dusterio 14 ноября 2016 в 01:49	0
потому что конфиг в формате JSON? да и где там классы в конфигах? :)		
webmoder	14 ноября 2016 в 09:55	0
Не думали реализовать проброс «Request ID» в запросах к микросервисам?		
	dusterio 14 ноября 2016 в 12:30	0
Сейчас пробрасывается User ID из токена и оригинальный IP. В принципе, очень просто добавить любые другие заголовки. А как бы Вы использовали Request ID?		
webmoder	14 ноября 2016 в 13:56	0
Использовал бы для логирования всей цепочки запросов к микросервисам. т.к не всегда понятно на каком этапе/микросервисе произошла ошибка. А имея идентификатор запроса инициированного на API Gateway, можно гараздо проще разобраться с проблемным местом.		
<ul style="list-style-type: none"> • это дело можно визуализировать 		
webmoder	14 ноября 2016 в 15:13	0
Скорее всего не стоит постоянно логировать каждый request. Достаточно иметь возможность проброса, который будет работать в режиме debug. Хотя это уже зависит от задач.		
	ruFog 17 ноября 2016 в 16:45	+1
Спасибо за статью! Интересно как вы авторизуете клиентов в микросервисах. Все микросервисы шарят общую БД юзеров? Или отдельный микросервис, от которого зависят другие микросервисы, которым нужна авторизация?		
	dusterio 18 ноября 2016 в 01:20	0
Да, существует одна общая БД юзеров, на основе которой выдаются OAuth2 токены. В каждом запросе к микросервису есть заголовок X-User с идентификатором пользователя, то есть микросервис всегда знает, что любой запрос к нему — прошел аутентификацию. Микросервису остается сделать авторизацию основываясь на любой своей внутренней логике. Это традиционный способ для этой архитектуры — так советуют делать книги.		
	tzurbaev 18 ноября 2016 в 07:46	0
А как быть с изменениями базы? Если у меня N сервисов, которые напрямую работают с таблицей юзеров, ведь потребуются вносить изменения в каждый из них, если будут проводиться серьезные изменения в БД?		
Сейчас читаю "Создание микросервисов" Сэма Ньюмана, там нет явного запрета на использование		

ЧТО ОБСУЖДАЮТ

Сейчас	Вчера	Неделя
Upwork регистрируется в РФ		
4,2k	14	
Не купили DLC: функцию, которая спасла бы упавшие 737, «Боинг» продавал как опцию		
90,6k	753	
Как я пишу конспекты по математике на LaTeX в Vim		
32k	101	
Польские учёные напечатали первую в мире бионическую поджелудочную железу с сосудами		
7,9k	27	
В личном чате Telegram можно удалять любые сообщения — даже чужие		
17k	79	

такого подхода, но довольно подробно описываются риски при использовании общей БД — в том числе и изменения структуры. Опыта в этой тематике пока что мало, поэтому для начала решил сделать так: есть основная БД приложения, в которой хранятся все данные пользователей. Микросервисы, которые тоже должны работать с юзерами, просто получают необходимые данные при регистрации пользователя и обновляют их при изменении профиля.



dusterio 18 ноября 2016 в 07:49

0

Сервисы не должны работать напрямую с таблицей юзеров — от API gateway передается только сам факт успешной аутентификации и немного дополнительных полей (scopes/права, ID юзера).

Если необходимо реализовать свою (для конкретного сервиса) авторизацию — традиционно нужна _отдельная_ база данных. Это основа философии — один сервис не должен задевать другой, все должно деплоиться по-отдельности без проблем.

Если данные о пользователе используются более чем 1-2 сервисами — их можно хранить в общей базе (которую можно держать «при» API gateway), если данные нужны конкретному сервису — в его местной базе.



tzurbaev 18 ноября 2016 в 09:26

0

Видимо я неправильно вас понял, спасибо.

Только [полноправные пользователи](#) могут оставлять комментарии. [Войдите](#), пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

Как я пишу конспекты по математике на LaTeX в Vim

+163

32k

409

101

Каким был первый айфон?

+43

33,1k

38

9

Не купили DLC: функцию, которая спасла бы упавшие 737, «Боинг» продавал как опцию

+34

90,6k

49

753

На работу на велосипеде. Еще одно мнение

+59

43,4k

117

324

О дисководах и их использовании на современных компьютерах

+48

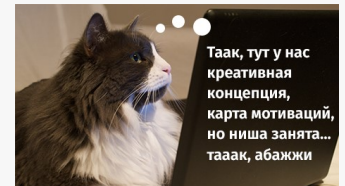
24k

57

83

РЕКОМЕНДУЕМ

[Разместить](#)



Онлайн-курсы школы ИКРА — это практические программы, нужные для работы прямо сейчас

Причины головной боли



Мигрень

Стресс

JAVA

JPoint — международная конференция по Java-технологиям: 5-6 апреля, Москва

Аккаунт

Разделы

Информация

Услуги

Приложения

[Войти](#)

[Регистрация](#)

[Публикации](#)

[Хабы](#)

[Компании](#)

[Пользователи](#)

[Песочница](#)

[Правила](#)

[Помощь](#)

[Документация](#)

[Соглашение](#)

[Конфиденциальность](#)

[Реклама](#)

[Тарифы](#)

[Контент](#)

[Семинары](#)



© 2006 – 2019 «ТМ»

[Настройка языка](#)

[О сайте](#)

[Служба поддержки](#)

[Мобильная версия](#)

