**Раскадровка по слайдам**

Слайд 1

Приветствую собравшихся здесь коллег.

Сегодня я расскажу вам о достаточно распространенной задаче — рефакторинге, а конкретно рефакторинге банковского веб-сервиса, что вносит свои коррективы. Расскажу о том, как мы пришли к выделению отдельных микросервисов из монолита, к сегодняшнему мейнстриму. Но для начала я вкратце расскажу о себе и о компании, в которой работаю.

Слайд 2

Итак, меня зовут Иванов Сергей, окончил магистратуру по вычислительной технике в томском политехе в 2013м году. Работал на языках Java, Javascript, но больше всего по нраву пришелся PHP, с которым я знаком, уже как лет пять, однако плотно работаю на нем буквально 2.5 года. За это время успел поработать в двух стартапах, ну и в одном банке.

Слайд 3

Точка же является продолжительницей дела банка для деловых людей Банк24.ру. Костяк управленцев, разработчиков, да и рядовых сотрудников перешел из Банк24.ру в Точку.

Слайд 4

Нас отличает то, что мы обслуживаем преимущественно малые и средние предприятия, не предоставляем кредиты(ну нет у нас такой услуги), а также высочайший сервис обслуживания наших клиентов. Это подтверждается множеством наград: из последних получили первое место в нескольких номинациях в рейтинге Business Internet Banking Rank 2017 от 2го августа, опередив продукты от Тинькоф банка, Всяких там Сберов и Альфачей.

Слайд 5

В конце 2016 года, когда я пришел в Точку, мне достался на поддержку, разработку и сопровождение веб-API(WS), пользователями которого были порядка 10 различных внутренних сервисов банка. Функционально же WS из себя представлял большой многофункциональный комбайн, основной функционал которого:

* Управление расчетными счетами и картами клиентов;
* Управление информацией о клиентах;
* Управление банковскими операциями;
* Отправка СМС;
* Платежи, безналичные переводы;
* Парсинг документов;
* Предоставление информации из ЕГРЮЛ, КЛАДР;
* Отправка голосовых сообщений на телефон;
* Проверка паспортов;
* и многое другое.

Самостоятельно веб-сервис этим всем не занимался, во многих случаях он просто предоставлял API, а все запросы проксировал на соответствующие специализированные системы.

Слайд 6

И вся эта радость была написана на языке php версии 5.6. с использованием фреймворка ZF1. К слову, осенью 2015го года вышла новая версия языка php — 7, а поддержка фреймворка ZF1 закончилась в сентябре 2016. Поэтому факт, - мне досталась устаревшая система, поддерживать и развивать которую не было никакого желания.

Слайд 7

…

Слайд 8

Эта ситуация вносила определенные сложности и риски при обслуживании данной системы. Логирование было примитивное: nginx-логи, а также логи приложения в текстовом файле. Например, чтобы посмотреть лог конкретного запроса, приходилось лезть в файл и искать нужный запрос grep-ом. Доходило до банального — нельзя было точно установить всех потребителей веб-сервиса, поэтому бывали ситуации, когда какая-то одна система в короткий промежуток времени отправляла такое большое количество запросов, что грузило ЦПУ веб-сервиса под чистую, что иногда, приводило к отказам другим пользователям в обслуживании. А что за система это делает — мы искали анализируя текущие логи nginx. В общем, это достаточно грустный опыт.

Слайд 9

Было принято решение все это безобразие рефакторить…

Слайд 10

…и преобразить с использованием более современной версии языка php 7.1, а также иного фреймворка Lumen, но по-прежнему с использованием компонентов из ZF. Сказано — сделано, и уже через месяц появилась новая версия веб-сервиса, на котором был доступен лишь один метод получения выписки из ЕГРЮЛ.

Слайд 11

В синтетических тестах, проведенных с помощью утилиты apache-benchmark, производительность нового веб-сервиса(WS-NEW) была практически в два раза выше, что было для нас ожидаемо.

Слайд 12

Здесь видно различие в коде двух методов, выполняющих одно и то же. Если приглядеться, код справа может показаться более лаконичным.

Все хорошо, новый веб-сервис есть, осталась одна маленькая деталь: нужно наполнить его функциональностью, идентичной WS. Нельзя так просто перенести функционал на php7.

Дело в том, что большинство методов получает информацию из БД Oracle, и там не простые выборки, а целые процедуры с двойным биндингом переменных. С такой крутой штукой я сталкивался только в Oracle, и она доставляет очень много неудобств потому, что готовых абстракций, реализующих такой функционал практически нет, а те что есть, совершенно унылы. В общем, на WS использовался один пакет для взаимодействия с БД из ZF1, а на WS-NEW другой, из ZF3. Соответственно, синтаксис совершенно другой, значит, надо все переписывать, значит новый функционал надо проверять.

Слайд 13

Проверять желательно не руками, а тестами. Функциональными тестами, если мы говорим о взаимодействии с БД. Результатом данного этапа жизни проекта стало появление функциональных и unit-тестов на код, а также использование системы непрерывной интеграции — jenkins. Конечно же, главное преимущество системы непрерывной интеграции — это автоматический прогон тестов, всякие статические анализаторы кода, конечно хорошо, ими можно поиграться, но не столь полезно в реальной разработке. Наиболее сложные участки кода проверялись простым мануальным тестированием, куда уж без него.

Вот, собственно, на такой инфраструктуре потихоньку перетаскивался фунционал WS на WS-NEW, и она оказалось вполне жизнеспособной.

Слайд 14

Почему нельзя переписать метод и перенаправить его вызов с WS на WS-NEW, наверное, промелькнуло в головах у некоторых. Все просто, API не полностью совместимы. Там, где старое API выдает 500ю ошибку и просто пустой экран, новое оборачивает ее в необходимый формат и предоставляет удобный для обработки ответ. Если где-то выдавался трейс ошибки, то теперь такого нет. Мало ли, кто мог завязаться на такое поведение.

Unit-тестам все методы не покрыть, их более 300, а это упороться. Функциональные тесты тоже могут покрыть порядка 50% запросов. Ну и кроме как вручную не протестить функционал, связанный с банковскими картами, поскольку для карт разработан специальный регламент с их работами, и их данные постоянно меняются.

Слайд 15

Поэтому здесь необходимо было использовать именно разные подходы. Но стек сервиса разросся до систем тестирования Jenkins и codeception.

Слайд 16

Стоит сказать, что одно дело переносить функциональность на WS-NEW, и совсем другое дело — заставить потребителей веб-сервиса использовать его. Даже если для этого надо в URL поменять один символ. Риски все-таки присутствуют, вдруг новое api не полностью совместимо со старым, а ведь на веб-сервис обращаются банковские системы, где неполная совместимость чревата денежными потерями. Поэтому выявить пользователей, их заставить перейти на WS-NEW отдельная и весьма длительная задача.

Где-то к началу весны мы имели рабочий WS-NEW с портированной покрытой тестами функциональностью и минимальной нагрузкой на него, а также WS, на который приходило подавляющее количество запросов. Тем временем бизнес точки активно рос вместе с нагрузкой на WS, и к середине весны она выросла вдвое по отношению к началу года, и составила 800000 запросов. Стало ясно, что если мы никого не пересадим на новый веб-сервис, придется масштабировать старый, и тогда мы с ним совсем надолго. Благо один крупный потребитель WS — система по открытию расчетных счетов( ОРС ), генерирующий половину запросов, согласился перенаправить свой трафик на WS-NEW. И после месяца совместного тестирования ОРС полностью пересел на WS-NEW. В итоге получилось следующее:

Изображение 4 — Конфигурация веб-сервисов

Таким образом, в начале мая этого года нагрузка приблизительно поровну распределилась между WS и WS-NEW и составила ~1000000 запросов в сутки.

Изображение 5 — CPU load из Zabbix по WS

На графике видно, как полегчало WS после того, как половину запросов перекинули на WS-NEW, а еще стало понятно, что WS может обработать без проблем около 1-1.2 млн. запросов.

Отдельного внимания стоит описание взаимодействия всех сервисов с БД. Взаимодействие происходит поверх tls-соединения, что накладывает некоторую специфику. Установка соединения, его шифрование производится процессором. Таким образом, когда процессор сильно занят, то и установка соединения с БД длится дольше(30-50мс против 800-5000мс). Соответственно, критично сделать так, чтобы процессор веб-сервиса не был занят на 100%. Эта проблема снимается использованием технологии постоянного соединение с БД( persistent connection ), однако, в php7, к сожалению, oci8 драйвер имеет багу при использовании этой технологии поверх tls. Также поскольку для обработки некоторых банковских запросов требуется достаточно много времени(счет может идти на десятки секунд), очень остро встает вопрос о производительности одной БД. В таком случае целесообразно поднять дополнительную stand-by БД, которая является полным клоном основной БД, наладить синхронизацию, и перенаправить на нее запросы, выполняющие операции чтения. Это простой и одновременно действенный способ.

На самом деле, получившийся результат нас вполне удовлетворял, поскольку получившаяся конфигурация сервисов хорошо справлялась с нагрузкой, а также ее было достаточно для роста нагрузки ориентировочно до 4 млн. запросов. Однако, в дальнейшем произошел конфликт интересов в рамках новой парадигмы организации работы в компании.

Изображение 6 — Добавлен StandBy DB

Изображение 7 — Архитектура WS

Новая парадигма организации работы в компании называется холакратия. Впервые она была внедрена в америнканской компании Zappos. Так вот, в ней вместо привычного большинству подхода к организации структуры компании по отделам( отдел состоит из сотрудников, выполняемых схожие или одинаковые функции ), состоялся переход к организации структуры на основе холакратии — это когда компания делится на круги, организованные по работе над одними бизнес-задачами( чаще это называют доменом круга). Эта кардинально иная структура, требует соответствующего подхода к разработке — каждому кругу свои системы, обслуживающие свой домен. В рамках холакратии целесообразнее, чтобы каждый круг был самодостаточен в технологическом плане и не зависел от сервисов других кругов. На эту структуру хорошо ложится подход в разработке, когда большие системы распиливаются на сервисы и микросервисы, каждый из которых отвечает за узко-специализированную область. Таким образом, к микросервисам нас подвигла эволюция, а не тренд в разработке. В контексте этого подхода наш большой и неповоротливый WS распиливается на отдельные части, и обслуживает свой круг. Как видно из изображения 7, архитектура WSа позволяет сделать это с минимальным рефакторингом кода — достаточно лишь каждому из своих кругов оставить собственные сервисы, которыми они пользуются. Этот процесс уже запустился — система ОРС уже имеет собственный веб-сервис. Дальше несколько сложнее, поскольку многие оставшиеся операции используются разными системами. Также стоит отметить, что на данный момент разделению повергся только веб-сервис, БД стоит на очереди, и сделать это будет также достаточно просто, поскольку разрабатывалась модульно, с разнесением ответственности за бизнес логику.

Изображение 8 - К чему стремимся

Процесс выделения отдельных сервисов является болезненным и длительным. В некоторых случаях целесообразно провести дополнительный рефакторинг и забрать по немного функционала из WS и БД и выделить получившееся в отдельный сервис. В ближайшем будущем у нас в планах два таких сервиса: шлюз по приему платежей на счета клиентов, а также сервис формирования выписки.

Также при переходе на новую технологическую платформу была решена еще и задача по увеличению доступности логов веб-сервисов. Поскольку внутри банка уже имелся опыт использования ELK(elasticsearch, logstash, kibana)-кластера, а также он уже был развернут, выбор стал очевиден. Интеграция предельно проста — формируешь логи в определенном формате, устанавливаешь на стенд агент filebeat, конфигурируешь и наслаждаешься. Логи в elastic хранятся в формате json-объекта, также elastic позволяет производить поиск по логам с использованием фильтров по полям плюс полнотекстовый поиск по индексируемым значениям. Logstash — парсер логов из любого пользовательского формата в формат elastic, kibana — интерфейс для поиска логов.

Невероятно просто теперь искать какой-то определенный запрос, ведь у Kibana есть возможность фильтровать запрос по тем или иным полям передаваемого json-объекта. Также, если в ELK пишутся логи как приложения, так и nginx, то не составляет труда связать nginx-лог с логом приложения. Такие кейсы обычно необходимы для мониторинга скорости ответа веб-сервиса или поиска запроса, ответившего 500й ошибкой.

Список используемых технологий сейчас:

nginx

Lumen

Zend Framework 3

Codeception

PHPUnit

Jenkins

Zabbix

ELK

consul

Для себя я открыл большую мощь данного софта в качестве базы для построения мониторинговой системы. Несколько скриншотов мониторинговых столов:

Количество запросов к веб-сервису в течение дня

Распределение нагрузки на веб-сервис по пользователям системы

Конечно же, разбив один WS на несколько сервисов, мы столкнемся с увеличением сложности администрирования. Однако, уже сейчас мы рассматриваем возможность использовать систему Consul и Consul Tempate для автоконфигурирования как отдельных сервисов, так и балансера.

Подытоживая, выполняемая работа активно снижает две сложности — технологическую, связанную с устаревшей платформой, и организационную, связанную с выделением каждому кругу своих сервисов, повышает прозрачность и отказоустойчивость работы получившихся сервисов. В общем, в банке все довольны:)