**Расширенные тезисы на DevFestSiberia**

В конце 2016 года, когда я пришел в Точку, мне поддержку, разработку и сопровождение достался веб-API(WS), пользователями которого были порядка 10 различных внутренних сервисов банка. Функционально же WS из себя представлял большй 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. 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, ну а осенью 2015го. Поэтому факт, - мне досталась устаревшая система, поддерживать и развивать которую не было никакого желания.

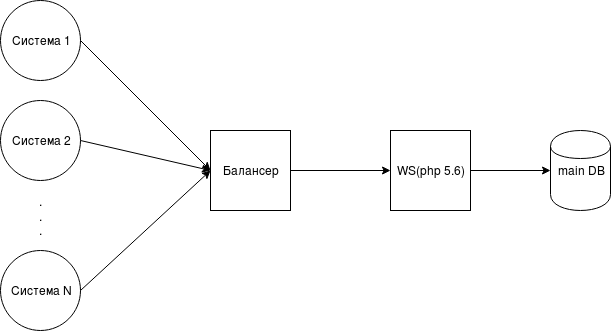
Эта ситуация вносила определенные сложности и риски при обслуживании данной системы. Логирование было примитивное: nginx-логи, а также логи приложения в текстовом файле. Например, чтобы посмотреть лог конкретного запроса, приходилось лезть в файл и искать нужный запрос grep-ом. Доходило до банального — 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, соответственно, приводило к недовольствам других пользователей. А что за система это делает — мы искали анализируя текущие логи nginx. В общем, это достаточно грустный опыт. Список используемых технологий:

nginx

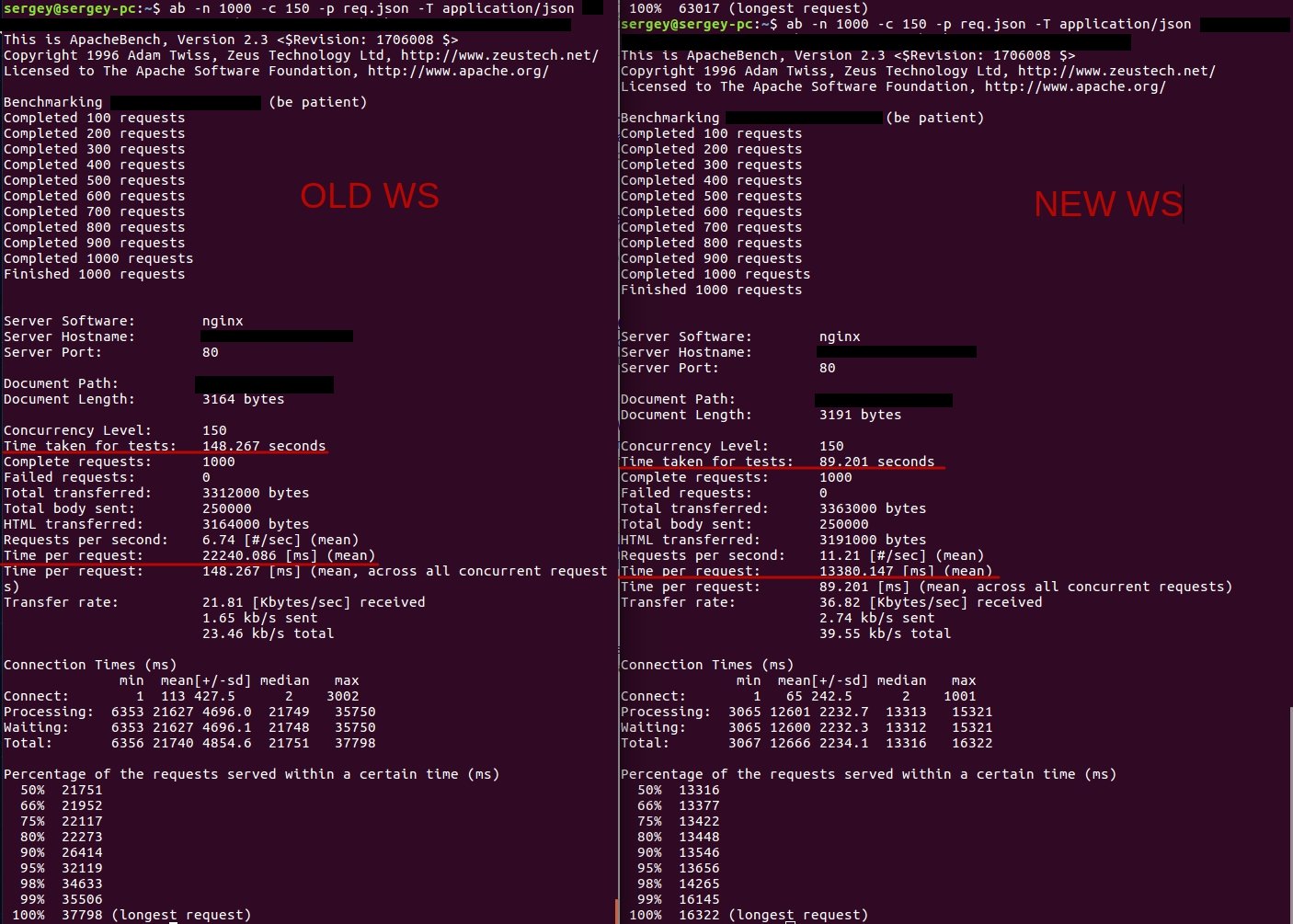
Zend Framework 1

php 5.6

Zabbix

Изображение 1- Как было

Было принято решение все это безобразие преобразить с использованием более современной версии языка php 7.1, а также иного фреймворка Lumen, а также 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. 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.

Изображение 2 — Производительность WS-NEW

Список технологий на данном этапе незначительно вырос:

nginx

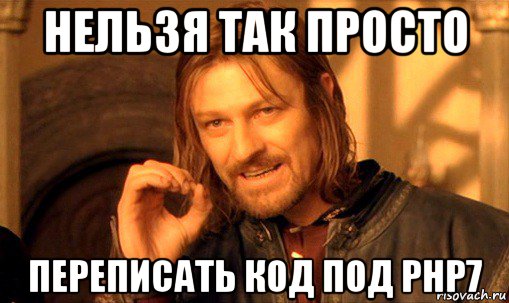
Lumen

Zend Framework 3

php 7.1.

Zabbix

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 абсолютно все методы получают 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 надо все проверять. Проверять желательно не руками, а тестами. Функциональными тестами, если мы говорим о взаимодействии с БД. Результатом данного этапа жизни проекта стало появление функциональных 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, но не столь полезно.

С таким набором уже можно нормально и вполне себе быстро, а главное — спокойно, переносить методы из старого WS.

Список технологий подрос:

nginx

Lumen

Zend Framework 3

Codeception

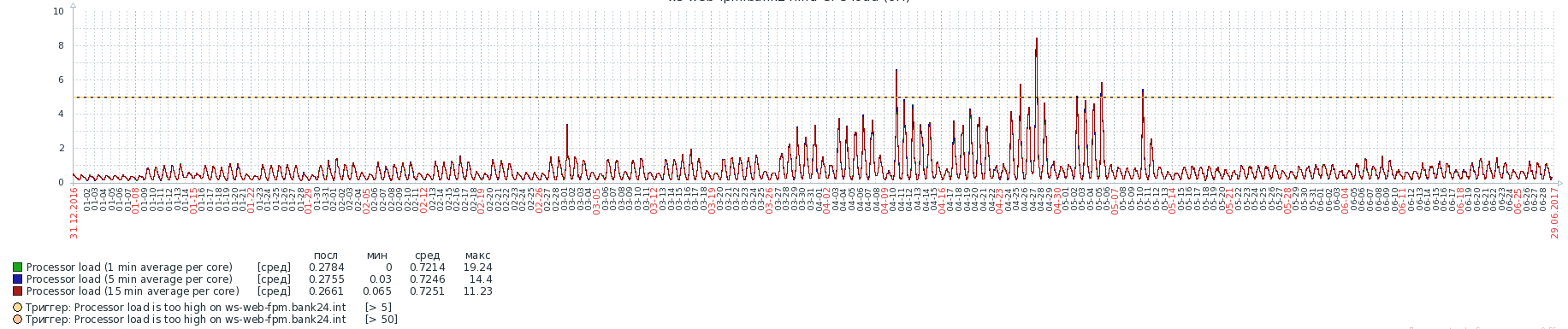
PHPUnit

Jenkins

Zabbix

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 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, генерирующий половину запросов, согласился перенаправить запросы на WS-NEW. И через месяц это событие состоялось. Таким образом, в начале мая этого года нагрузка приблизительно поровну распределилась между WS и WS-NEW и составила ~1000000 запросов в сутки.

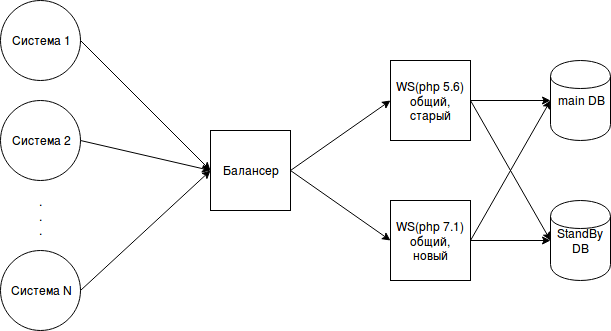
Изображение 4 — Скрин из мониторинга Zabbix по WS

Дальше, внезапно нам пришлось столкнуться с проблемой увеличивающейся нагрузки. Стало ясно, нужно активно договариваться с владельцами систем, которые используют WS на то, чтобы они пересели на WS-NEW.

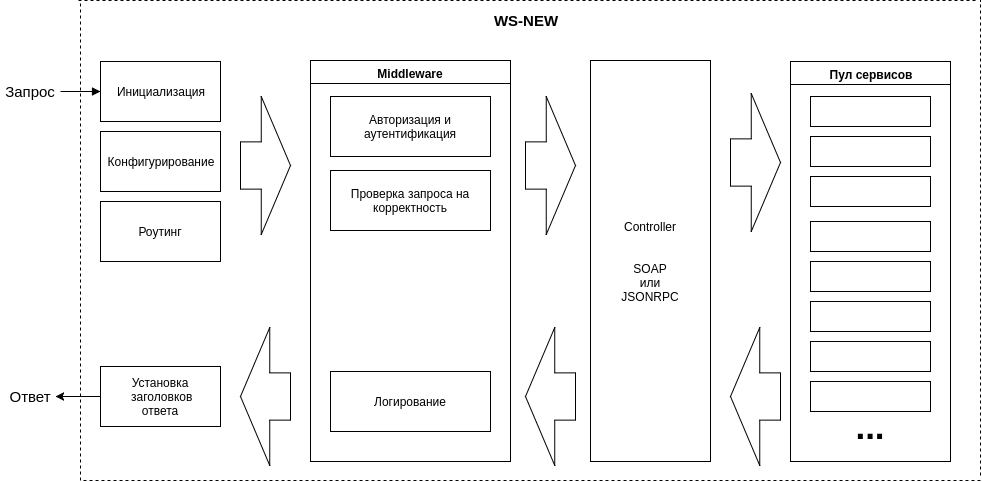
Сначала с выходом php7 дальнейшей поддержки лишился фреймворк ZF1 (он поддерживает лишь php5.6). Поскольку php7 по сравнению с php5.6 имеет прирост производительности 100% - нельзя так просто не внедрять его. В качестве базы выбран современный и популярный фреймворк Lumen. Переход на довольно новую технологическую платформу чреват сложностями и болью — пришлось набить несколько шишек о баги драйвера работы с БД Oracle - oci8 на php7, о чем есть баг-репорты на официальном трекере.

Отдельного внимания стоит описание взаимодействия всех сервисов с БД. Взаимодействие происходит поверх tls-соединения, что накладывает некоторую специфику на взаимодействие. 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. Этот момент у нас уже выполнен, и результат нас удовлетворяет.

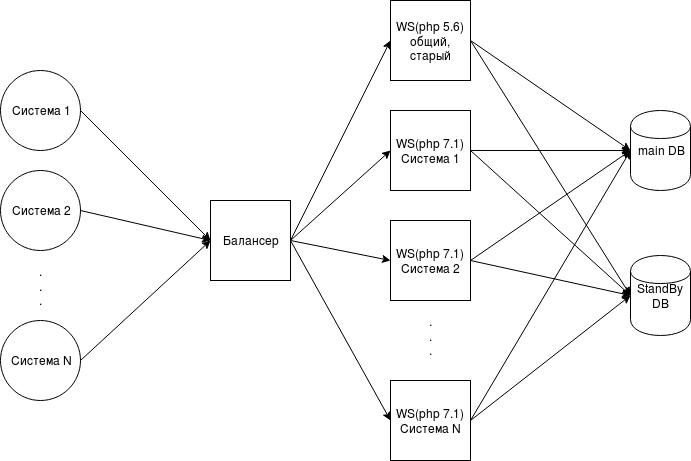
На самом деле, получившийся результат нас вполне удовлетворял, поскольку получившаяся конфигурация сервисов хорошо справлялась с нагрузкой, а также ее было достаточно для роста нагрузки ориентировочно в 5 раз. Однако, в дальнейшем произошел конфликт интересов в рамках новой парадигмы организации работы в компании.



Изображение 2 — Что получилось

Изображение 3 — 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 3, 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 — первым делом были выделены две явно определяемые по своим доменам системы — система, занимающаяся открытием расчетных счетов в банке и банк, который без офисов. Они уже выделены в отдельные сервисы. 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 4 - К чему стремимся

Процесс выделения отдельных сервисов является болезненным и длительным. Дело в том, что мало перенести функциональность, ее необходимо протестировать. И здесь хорошо задуматься над тем, как это делать. Одними тестами тут не отделаешься, поскольку имеется множество функциональности, формирующих настоящие бизнес-процессы, которая взаимозависима. Как тестировать такие вещи — это вопрос открытый, мы же часто используем мануальное тестирование. Для всего остального налажен процесс непрерывной интеграции, на основе уже ставшей классической платформы jenkins с выполнением классического набора действий: api и unit-тестов, а также статического анализа кода. На основе этого процесса принимается решение, выгружать ли новый код на тестовые стенды или нет. Таким образом, достигается два положительных эффекта — программисты более спокойны за свой код, а также тестовый слой ПО всегда находится в условно-рабочем состоянии(по крайней мере по версии роботов).

Выгода от переноса функциональности api на новую платформу, однако, перевешивает все трудности, связанные с данным процессом. Прирост производительности составляет более 100%, не говоря уже о большей отказоустойчивости и лучшей масштабируемости получившегося решения.

После того, как функциональность перенесена на новую технологическую платформу, за ней следует следить. Именно здесь имеет место использования разных мониторинговых систем и систем логирования — от банального заббикса, до, в нашем случае, инструмента pinba и стека ELK.

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.