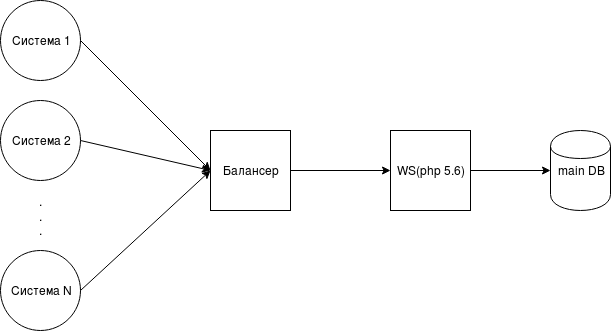
**Расширенные тезисы на DevFestSiberia**

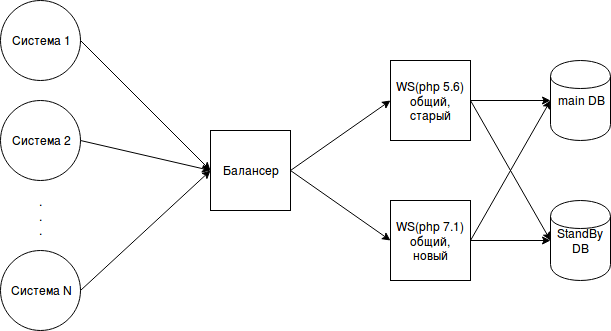
Есть достаточно нагруженный веб-api(WS), который обрабатывает порядка миллиона запросов в сутки. На него ходит около десятка разных систем: jira, астериск, антифрод, парсер документов и другие внутренние сервисы банка. WS же много что может: и данные из ЕГРЮЛ получить, и платежки запилить, и СМСку отправить — в общем, большая многофункциональная система-комбайнер. Вся эта радость была написана на php5.6 и ZendFramework1. Но все технологии со временем устаревают, не исключением является и этот WS.

Изображение 1- Как было

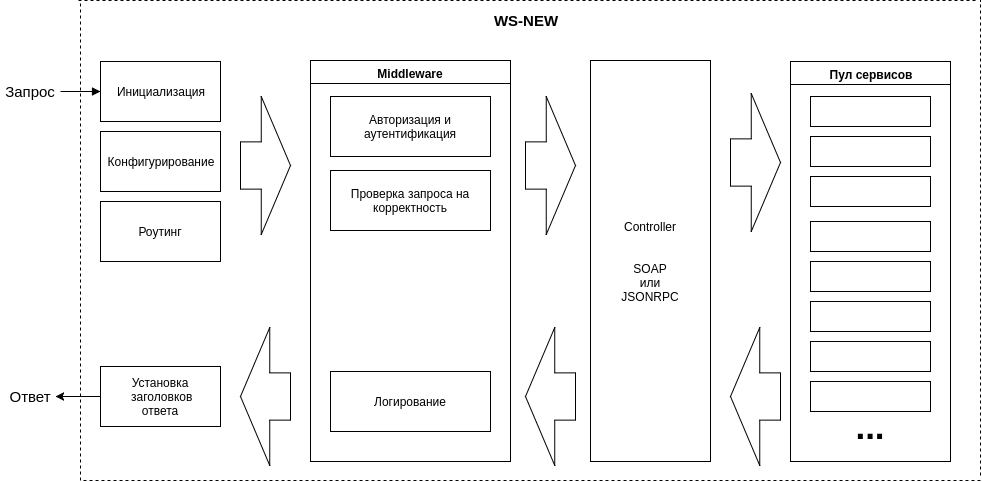
Сначала с выходом php7 дальнейшей поддержки лишился фреймворк ZF1 (он поддерживает лишь php5.6). Поскольку php7 по сравнению с php5.6 имеет прирост производительности 100% - нельзя так просто не внедрять его. В качестве базы выбран современный и популярный фреймворк Lumen. Переход на довольно новую технологическую 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.

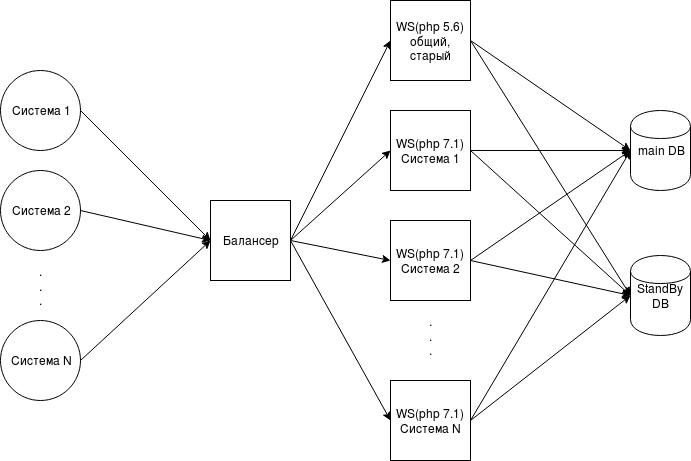
На самом деле, получившийся результат нас вполне удовлетворял, поскольку получившаяся конфигурация сервисов хорошо справлялась с нагрузкой, а также ее было достаточно для роста нагрузки ориентировочно в 5 раз. Однако, в дальнейшем произошел конфликт интересов в рамках новой парадигмы организации работы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