КАК СТАТЬ АВТОРОМ Sapphire снова впереди Поехали в гик-трип по Кал...

75.51
Рейтинг

MOEX

Инвестиции начинаются здесь

Подписан Я работаю здесь

Manassian
22 фев в 16:03

Реальная инженерная трансформация: от команд и метрик до культуры, конвейеров и инфраструктуры

Сложный 25 мин 3.8K

Блог компании MOEX, Управление разработкой*, Agile*, DevOps*

Кейс

+4 26 2

Привет, Хабр! На этапе выбора темы статьи было много идей: написать про DevOps или про платформы, а может про продуктовые команды или про практики SRE? Но пришли к выводу, что нет ничего интереснее, чем реальная увлекательная история трансформации. Мы, команды платформы разработки MOEX и экосистемы Финуслуги.ру, в лице Карапета Манасяна, Александра Барыкова, Антона Квашёнкина и Юлии Лутковской, расскажем практически про весь путь изменений и про их подводные камни. Важно отметить, что в статье затрагивается довольно много тем, местами даже будут блоки со скриптами 😊 . Поехалиии!

Оглавление:

> Про Финуслуги.ру и портфель продуктов
>
> 1. Как было до изменений?
>
> 1.1 Предпосылки
>
> 1.2 Состояние “до”: метрики
>
> 1.3 Состояние “до”: конвейер производства и CI/CD-процесс
>
> 2. Выбор траектории трансформации
>
> 2.1 Целевая топология и как менялись команды
>
> 2.2 Как создавали сообщества
>
> 2.3 Создание модифицирующей или change-команды
>
> 2.4 Целевой конвейер и какую стратегию выбрали
>
> Описание самого CI/CD-процесса
>
> a) Почему мигрировали с Jenkins на Gitlab CI в части CD
>
> b) Про структуру CI
>
> c) Про версионирование в целевом подходе
>
> d) CD: почему выбрали ArgoCD
>
> 2.5 Что изменили в архитектуре продуктов

> 3. Планы на развитие
>
> 4. Выводы
>
> Полезные ссылки и выступления

Про Финуслуги.ру и портфель продуктов

Финуслуги — первая платформа личных финансов. Мы создали нашу платформу, чтобы сделать финансовые продукты доступными для любого жителя нашей страны вне зависимости от региона проживания. Одна регистрация на нашей платформе открывает широкие возможности по выбору и оформлению онлайн банковских и страховых продуктов. Без визита в офис, в удобное для вас время. Без каких-либо комиссий.

- **Вклады** — один из самых простых инструментов, который поможет сохранить и приумножить сбережения. Когда открываешь вклад, важно найти лучшую ставку по депозиту, чтобы получить максимальную доходность. Но отследить изменение процентной ставки по вкладам вручную бывает сложно, потому что условия в разных банках постоянно меняются. Чтобы вы могли на одной странице видеть актуальные ставки по депозитам, мы создали индекс доходности вкладов, в котором сравниваем условия вкладов крупнейших банков. Наши специалисты регулярно мониторят предложения в топ-50 банков и размещают актуальные предложения. Благодаря этому вам не нужно самостоятельно идти в банки или на их сайты в поисках самой высокой ставки по депозитам. На нашей платформе можно сравнить предложения и перейти к открытию вклада. Оформить депозит на Финуслугах можно полностью онлайн без посещения офиса банка. Для этого нужно только авторизоваться через Госуслуги и один раз встретиться с представителем платформы в удобное время.
- **Облигации** — эксклюзивно на нашей платформе мы предлагаем новый простой инструмент с ежедневным начислением дохода. Народные облигации — инвестиции без рыночного риска. Они не торгуются на бирже, а продать их можно в любой момент по цене покупки плюс накопленный доход. При покупке и продаже облигаций вы не платите никаких комиссий.
- **Страхование** — на нашей платформе автолюбители могут оформить электронный полис ОСАГО с выгодой до 78%. Без комиссии, без посещения офиса и без встречи с курьером. Наша платформа рассчитывает цены в более чем 15 страховых компаний и находит самые выгодные предложения для вас. У нас удобный онлайн-калькулятор, который позволит быстро рассчитать стоимость полиса. Оформленный полис мы направляем по email и отображаем в нашем мобильном приложении, чтобы полис всегда был под рукой. Также вы можете подобрать полис КАСКО. А если у вас оформлена ипотека для вас у нас онлайн-оформление страховки.

- **Кредиты** — отправьте одну заявку сразу в несколько банков, выберите самое выгодное предложение и получите кредит. У вас есть возможность получить кредит как без визита в банк, так и в офисе банка. А бесплатный кредитный рейтинг поможет вам понять вероятность одобрения кредита. Чем выше рейтинг, тем больше предложений вы получите.

1. Как было до изменений?

До начала трансформации мы жили в компонентном подходе. Использовали, так называемую, модель "водопад", при которой время от старта разработки до вывода в прод занимала в среднем 8 недель. Организационная структура в компонентном подходе предполагала четкие зоны ответственности по направлениям: аналитика, тестирование, архитектура, DevOps, а разработка делилась на фронт и бэк**.** Бизнес в данной модели был практически изолирован. Задача от него ставилась при старте разработки и по истечению 8 недель осуществлялась бизнес-приемка функционала.

В организационной структуре, приведенной ниже, глава поставок отвечал за скоуп и даты релизов, а также за все quality gates в рамках релиза (приемочное тестирование, регресс, нагрузка и тд). Фактически он был релиз-менеджером на большую программу (всех команд). Он являлся в некотором смысле арбитром между ИТ и бизнесом, что в свою очередь влияло на эффективность поставки и создавало "bus factor". Под главой поставок были все команды, как компонентные, так и сервисные (аналитика, архитекторы, DevOps, тестирование). При этом нужно подчеркнуть, что существовала отдельная линия сопровождения (команда Ops), которая структурно была в другом подразделении. Фактически, были команды разработки, DevOps и 2 линия (Ops) обособленные друг от друга, что создавало в свою очередь фактор "перекидывания через стенку". Каждый отвечал в рамках своей зоны ответственности.

1.1 Предпосылки

Любые трансформации должны иметь предпосылки. Во всех остальных случаях – это проявление фактора “слепых улучшений”. Все предпосылки были определены в коллаборации разных направлений: бизнеса, разработки, тестирования, конвейера доставки, инфраструктуры, HR, коммуникаций и тд. Это позволило добиться максимально объективной картины текущего состояния.

Можно выделить следующие основные предпосылки:

- мы медленно поставляли функционал до пользователей: пользователи и бизнес требовали быстрой доставки изменений. Ввиду того, что скорость поставки была медленной мы теряли клиентов. А с другой стороны конвейер поставки был устроен настолько сложно, что для доставки изменений подключались много специалистов, которое увеличивало сильно стоимость поставки;
- страдало качество поставок: этому тоже были четкие объяснения. Не была описана методология управления качеством, отсутствовали явные quality gates в конвейере. Да, формально в нем был статический анализ, но это был всего лишь информационный этап, без явных блокировок;
- была несогласованность между бизнес и ИТ на всех этапах, и мы часто не оправдывали ожидания друг друга;
- блокирующие проблемы возникали на поздних этапах, что регулярно приводило к сдвигу релизов;
- несмотря на достаточное количество ресурсов и высокую экспертность, конвейер не удовлетворял потребностям команд: команды всегда хотели понимать и управлять конвейером самостоятельно, но ввиду того, что он был сложным и непонятным, они по любой проблеме, даже самой незначительной, обращались к "devops-команде";
- необходимость в ускоренной реакции на потребности рынка или уменьшение TTM: в экосистеме Финуслуги.ру существуют много гипотез и они требовали быстрых проверок. Из-за высокого TTM возникали сложности в их проверках;
- отсутствовала ответственность за продукт: часто звучали такие фразы как "это все девопсы", "тестировщики не дотестили" и тд. Простой инженер и разработчик продукта не брали на себя ответственности за конечный результат целиком. Причина была, конечно же, не в конкретных ребятах, а в самой инженерной культуре и формулировках миссий продуктов;
- большое количество коммуникаций;
- потребность в быстром масштабировании с повышением качества, безопасности и доступности сервисов для клиентов;

- отсутствовала культура практик проверки гипотез: делали-делали что-то большое, выкатывали в прод, но потом оказывалось, что это уже не нужно рынку;
- релизы ставились ночью. Это очень сильно выматывало команды и приводило к выгоранию.

1.2 Состояние “до”: метрики

Чтобы понять куда идти, надо понять, где мы сейчас. Здесь нам как раз и помогают правильно определенные метрики, исходящие из предпосылок и целей бизнеса. Правильно определенные – это значит метрики, которые помогают находить проблемы, а не только рисует красивые мигающие лампочки. Мы сразу четко обозначили, что метрики не предназначены для того, чтобы хвалить или наказывать людей.

Некоторые квадранты радара метрик приведены ниже:

Технологические метрики и метрики потока:
Time to Market
Время от создания (open) до релиза (lead time), дни, истории
Частота релизов (release frequency) в месяц, без хотфиксов
Время от принятия в работу до релизов (cycle time), дни, программа
Качество
Процент внеплановых Hotfix-релизов (Change Failure rate)
Кумулятивный поток дефектов
Качество заведенных дефектов (% rejected)
Среднее время жизни дефекта, дни
Дефекты конвейера, к-во
Долг по автоматизации тестирования
Процент дефектов, найденных до продуктива, от общего числа дефектов
Эффективность
Эффективность потока (время кодирования к общему TTM)
Rework (время на исправление дефектов ко времени разработки)

Метрики потока
TTM в управлении продуктом, дни
TTM User stories в ИТ, дни
Эффективность потока (время написания кода), %
Затраты на переделывание
Доступность, %
Change failure rate, %
Метрики разработки
Технический долг (часы)
Покрытие unit-тестами (%)
Покрытие код-ревью (%)
Метрики Scrum
Velocity в story points
План / факт, %
Scope creep / рост задач во время спринта

Клиентские метрики
Индекс удовлетворённости клиентов, NPS
TTM, от идеи до прода, дни
% user stories, по которым высказана и проверена гипотеза (получена обратная связь на ценность)

Бизнес-метрики
Новые продукты, к-во
Новые партнеры, к-во
Product-Market Fit

Выручка, руб.
Продажи фин. продуктов
Время открытия и пополнения вклада
Доступность, %

Организационно-культурные метрики
Количество коммуникаций и встреч
Качество информации и задач, %
Текучка
Переключение контекста
Открытость к инновациям
Лояльность к ошибкам и неудачам

Мы также "померили" сервисные команды, которые, в некотором смысле, являлись "серой зоной", т.е. сервисные команды не измеряли качество предоставляемого сервиса и удовлетворенность своих клиентов. Это не давало возможности ретроспективно посмотреть на свои процессы и внести туда корректировки и улучшения.

Нужно подчеркнуть, что мы рассматривали каждый показатель только в определенном контексте, в связке друг с другом, а также с другими метриками, исходя из бизнеса. Более того, мы связывали эффективность работы с обратной связью от конечных пользователей и их удовлетворенностью. Это давало возможность принимать максимально верные управленческие решения по изменениям и ходе трансформации.

1.3 Состояние “до”: конвейер производства и CI/CD-процесс

До всех изменений, можно сказать, что конвейер и связанные работы с ним были непрозрачны, а команды вообще не понимали как он устроен. Для них была доступна только минимальная информация — в какой репозиторий залить код и кому написать, если возникнет красная лампочка.

Структура конвейера была следующей:

- CI-процесс был построен на базе Jenkins и плагина Jenkins templating-engine. Под каждый стек были написаны библиотеки для тестов, сборки и тд.

- Стандартная сборка артефактов в виде докер-контейнера и Helm-чарта с последующей публикацией в хранилище артефактов Nexus.

Прежде чем рассказывать про процесс CD, стоит рассказать про окружения или, как у нас принято называть, "полигоны".

Полигоны разделяются по функционалу:

1. **Командные полигоны** — полигоны, которые используются только продуктовыми командами. Под каждую команду – отдельный полигон, на котором развёрнуты все необходимые сервисы работы конкретного продукта. Технически, командный полигон представляет из себя отдельный namespace в Kubernetes-кластерах dev-окружения.
2. **Полигон для запуска тестов** — отдельный полигон, который используется исключительно для запуска автоматизированных API и UI-тестов. Используется в основном командой QA. Также, как и командные полигоны, располагается в отдельном namespace в Kubernetes-кластерах dev-окружения.
3. **Интеграционный полигон** — полигон, который подключен к реальным интеграциям с партнёрами, банками и т.д. На данном полигоне также проводится регрессионное тестирование. Аналогично другим полигонам в отдельном namespace.
4. **Полигон для проведения нагрузочного тестирования** — отдельные Kubernetes-кластеры, на которых команда НТ запускает нагрузочное тестирование. Максимально приближен к конфигурации с продом.
5. **Хотфикс —** через этот полигон прокатывались хотфиксы на прод.
6. **Препрод-полигон** — окружение, на котором проводятся финальные проверки перед публикацией в прод. Также максимально похож на продакшен в части конфигурации.
7. **Продакшен Полигон** — прод).

Теперь вернёмся к CD-части нашего рассказа. Доставка также была построена на базе Jenkins, но самое интересное в том, что процесс отличался в зависимости от полигона. И это порождало немало проблем. Пробежимся по каждому из полигонов.

- Командные полигоны: деплой на командные полигоны выполнялся в Jenkins в последнем шаге пайплайна. По сути, шаг представлял из себя выполнение команды helm install с определённым набором параметров в виде версии артефакта и т.д.

Креды доступа в виде kubeconfig-файлов к кубер кластерам dev-окружения были добавлены в настройки Jenkins.

- Полигон для запуска тестов: деплой выполнялся по шедулеру несколько раз в день запуском аналогичных команд, что и на командных полигонах.
- Интеграционный полигон: деплой выполнялся только после прохождения основной джобы API/UI-тестов на QA-полигоне, в которой запускались тесты по всем продуктам. После успешного завершения тестов деплою давался зелёный свет.
- Препрод-полигон: вот как раз здесь крылось основное различие в процессах по сравнению с описанными выше полигонами. После прохождения регрессионного тестирование на интеграционном полигоне в игру вступала отдельная джоба в Jenkins, в которой запускался специально написанный тулинг (на Go) для снятия так называемой "релизной карты". Релизная карта представляла из себя готовый helmfile.yaml (helmfile) с версиями сервисов, которые были продеплоены на интеграционный полигон в момент снятия карты. Далее запускался процесс "переноса релизной карты" — джоба по переносу артефактов из DEV Nexus в TEST Nexus согласно слепку версий сервисов в ранее снятой релизной карте. И финальный этап - джоба деплоя, которая под капотом запускала утилиту helmfile с ранее подготовленным файлом.
- Продакшен Полигон: процесс аналогичен TEST. Различия в том, что для переноса артефактов осуществлялся из TEST Nexus в Prod Nexus.

За процесс деплоя на TEST/PROD была ответственна отдельная команда сопровождения прод-окружения (ops-ы, devops-ы). Они, в том числе, занимались поддержкой/доработками всевозможного тулинга, описанного выше. Продуктовые команды в процессе деплоя никакого участия не принимали. Сейчас, смотря на это ретроспективно, видно, что это был ориентированный, в первую очередь, на команду Ops инструментарий, а не клиентоориентированный набор платформенных сервисов продуктовым командам.

2. Выбор траектории трансформации

2.1 Целевая топология и как менялись команды

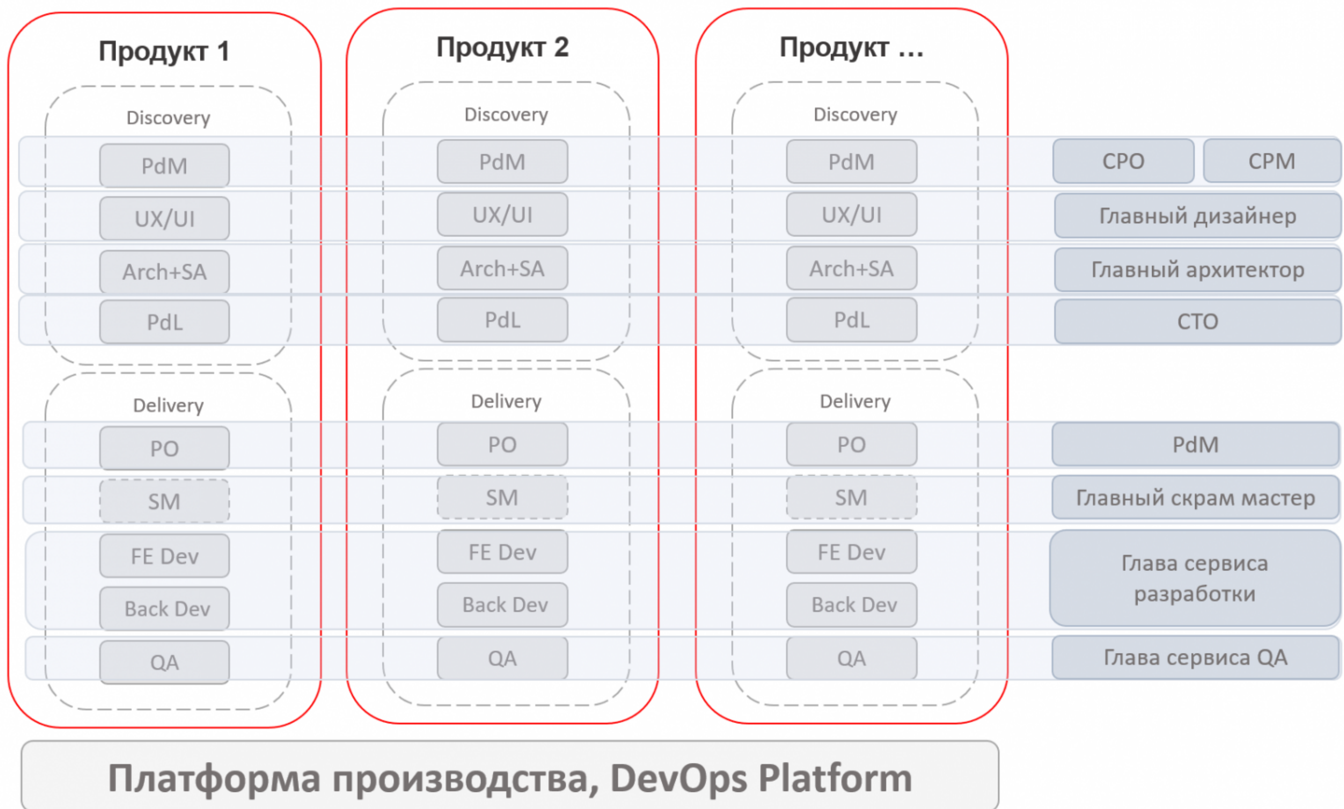
Как было сказано выше, первым делом мы определили метрики и померили их. Основное, что мы поняли — многие проблемы исходили из организационной структуры. Именно с нее начался наш первый шаг к трансформации, где мы сделали акцент на потокоцентричные команды. Команды, главная миссия которых — это создание ценностей для клиентов, посредством сосредоточения вокруг регулирования скорости потока, а также качества, безопасности, доступности и удобстве для пользователей. А их цель — привлечение аудитории, которая согласится купить предлагаемое решение ее проблемы, посредством постоянного совершенствования потока создания ценностей. Формула непростая, но все по порядку.

Организационная структура была разделена на три вертикали:

- уровень программы (CPO, CPM, главный Scrum Master (SM), CTO, главный архитектор, глава интеграционной команды). У каждой роли своя зона ответственности и фокус:
  - SM — повышение эффективности команды, процессы и их оптимизация;
  - CTO — как обеспечить выполнение бизнес=задач, опираясь на текущие технологии и современные тренды;
  - Глава интеграционной команды — быстрый и надежный конвейер, доставка релизов в прод;
  - Главный архитектор — выбор оптимальных технологий для решения задач бизнеса;
  - CPO/CPM — какой продукт востребован на рынке, что принесет нам прибыль и т. д.
- уровень продукта;
- уровень команды.

Мы детально проработали новые роли и зоны ответственности, что позже было отражено в виде плейбука. **Плейбук** — это инструмент, который позволяет получить исчерпывающие ответы на вопросы по процессам, ролям и инструментам. В тоже время, это некоторая их стандартизация и спецификация команд.

Центром вселенной стала продуктовая/потокоцентричная команда, а роль менеджера изменилась — от позиции командования и контроля к лидеру и «слуге». Такие команды мы укомплектовали всеми необходимыми ролями, чтобы они могли независимо работать. Это был кардинально новый подход к работе, который требовал от членов команд также терпения и готовности измениться. В свою очередь, платформа производства или DevOps-платформа MOEX аккумулировала в себя инструменты, компетенции и платформенные решения, которые предоставлялись командам в виде сервисов. Важной особенностью платформы являлась ее независимость от потокоцентричных команд. Она никак не участвует в производственных процессах самих команд, а только лишь отвечает за свою «платформу как продукт». Конечно, эта прекрасная и идеальная картина случилась не сразу. Команда платформы усердно работала над сбором потребностей команд, их обратной связи и developer experience (опыт разработчика). Именно это позволило нам создать полностью удовлетворяющую потребности разработчиков платформу.



2.2 Как создавали сообщества

Продуктовые команды стали более изолированы и появилась потребность сохранять и выращивать экспертизу, делиться опытом между командами. Сначала мы определили, что есть "сообщество". Это объединение людей, имеющих общие интересы для обмена опытом, создания комфортной среды и решения текущих проблем. Концепция ИТ-сообществ в MOEX – это поиск и внедрение практик по развитию производственных процессов и инструментов для ускорения принятия решений и сокращения времени создания ценностей.

На тему сообществ и их создания существуют много докладов во внешней среде, поэтому мы сделали акцент на то, какие сообщества создавались в рамках Финуслуги.ру, и с какими проблемами мы столкнулись.

Было принято решение создать сообщества по каждому направлению:

- архитектуры,
- тестирования,
- разработки,
- DevOps и процессов разработки.

Сложности, с которыми столкнулись:

- Не всегда легко найти лидера сообщества. Без лидера/ов сообщества, как правило, не функционируют. Поэтому наша задача была найти мотивированных ребят, готовых лидировать и собирать вокруг идеи других инженеров. Признание стало главной составляющей мотивации лидера сообщества. Их драйвила причастность к созданию крутого сообщества и того, что они стоят у истоков этих событий. На уровне всей компании лидеры стали известными и признанными экспертами. Казалось бы, мелочь, но нам этот фактор очень сильно помог.
- Конфликт с бизнес-задачами. Эта проблема была ярко выраженной в начале пути. Когда процессы работы сообщества устаканились, задачи сообщества стали для команд обычным делом.
- Выделять человека только под задачи сообщества слишком дорого.
- Сопротивления нововведениям. Эта сложность решалась не принуждением или "спусканием сверху". Мы создавали модифицирующие/change-команды, которые помогали другим осознать свои проблемы, создавать прототипы, воспринимать явную пользу от нововведений. Далее эти процессы и технологии масштабировались на другие команды.

Давайте рассмотрим работу сообщества на примере **Finuslugi QA Community**:

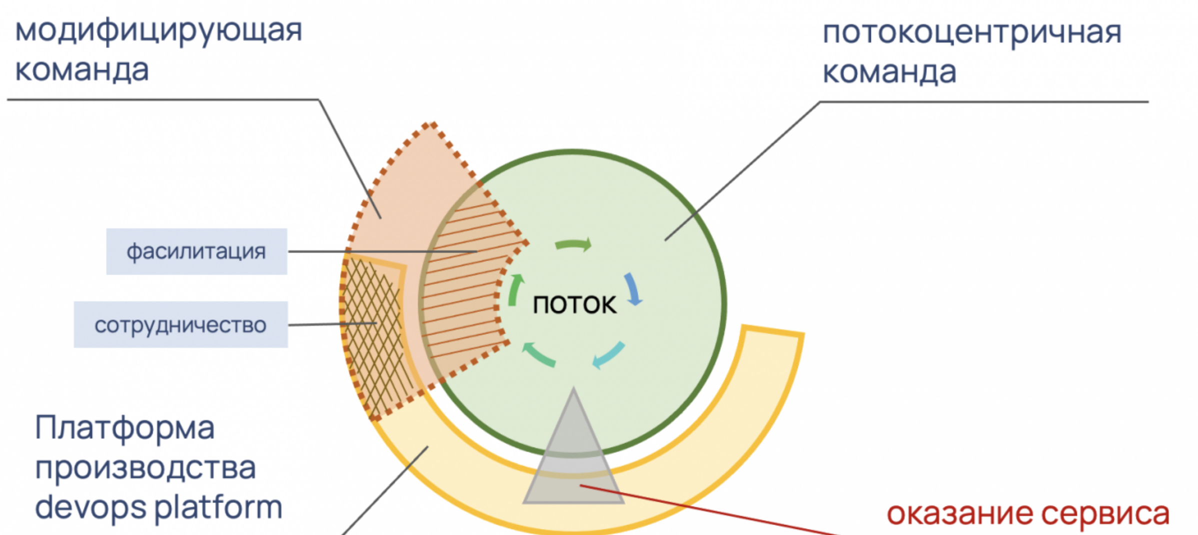
- встреча проходит раз в 2 недели, открытое мероприятие, любой участник команды может подключиться;
- повестка встречи определяется заранее с помощью голосования и насущных проблем;
- отдельное пространство в таск-трекере, куда можно создавать свои пожелания и потребности.

Ответственность лидера сообщества:

- анализ ключевых метрик сообщества и проработка мер в случае отклонения;
- помощь с поиском новых сотрудников (собеседование, онбординг, матрица компетенций и т.д.);
- проведение обучающих сессий и митапов;
- формирование стратегии тестирования, работы по её развитию;
- проведение встреч гильдии QA;
- создание общих регламентов и стандартов;
- участие в проведении post-mortem-ов;
- содействие развитию инструментов тестирования.

2.3 Создание модифицирующей или change-команды

Серьезные инженерные трансформации не заканчиваются лишь внедрением CI/CD и k8s. Основные проблемы изменений кроются в инженерной зрелости команд, которая банально может стать преградой для внедрения модных-молодежных практик. Поэтому наряду с разработкой стандартов, целевых архитектур и конвейеров, мы начали процесс создания модифицирующей команды. Задача такой команды заключалась в максимальной помощи и содействии процессам трансформации и, в первую очередь, командам, которые хотят измениться, но им сложно или они не знают с чего начать, куда идти. Модифицирующую команду мы создали из максимально лояльных, сеньёрных и проактивных ребят, которые четко понимали свою миссию и цели (разумеется, они были представлены всем командам). Она создавалась как временная команда и состояла, как из ребят команды DevOps-платформы, так и Финуслуги. Работа была организована по классическому скраму: планирование, дейли, ретро, ревью и тд. Здесь мы ничего нового не придумывали. Но успех был неизбежен, за счет удаления всех стен между членами команд и создания атмосферы доверительной работы *без руководителей, но с сильными лидерами* 😊 . Никаких ограничений на межкомандные коммуникации, никаких искусственных согласований с "непосредственным руководителем" и, конечно, никаких формальных "работаю только по таскам с детальным описанием и ТЗ". Каждый инженер понимал, что, находясь в модифицирующей команде, ему нужно включать голову и, засучив рукава, разобраться, описать и решить общую задачу, чтобы нанести непоправимую пользу.



В успех модифицирующей команды многие не верили. Но мы уж слишком были уверены в своих силах. Во-первых, мы не только занимались проектированием светлого будущего, но и решали задачи, стоящие здесь и сейчас у команд, и, во-вторых, любые

изменения и модификации представляли из себя сначала прототип, который в "сыром" виде обсуждался и тестировался с командами. Так мы уменьшали риски дальнейших переделываний, а конечные внедрения имели высокий уровень качества. Кстати, после каждого спринт-ревью модифицирующей команды, мы собирали обратную связь от команд, на горячую. Как правило, это были вопросы на измерение основных метрик: удовлетворённость модифицирующей командой и их решениями, насколько понятны объясняемые кейсы и решения, чего не хватает в рассказе, какие сложности возникают и т.д. Вопросы зависили от конкретного контекста спринта.

2.4 Целевой конвейер и какую стратегию выбрали

Главной целью целевого потока и конвейера являлось создание возможности независимой доставки компонентов продуктов до конечных клиентов. Мы хотели создать такой процесс, где разработчик сможет сам доставить изменения до прода. Да-да, вы не ослышались, именно сам и до прода. И у нас это получилось сделать. Самая основная преграда, существовавшая на пути к этому – была жесткая разделенность между Dev и Ops. Разработчики самостоятельно ничего не могли доставлять в прод без обращения к Ops-ам. Эта позиция аргументировалась всегда тем, что "разработка не должна иметь доступа в прод". Мы, как модифицирующая команда, с этим полностью были согласны. И предложили абсолютно новый концепт конвейера и CI/CD, который был ориентирован на четко определенных Quality Gates (QGs) и Security Gates (SGs), полное прохождение которых давало возможность разработчику и команде довести свои изменения до прода, не обращаясь ни к кому. Здесь важно еще раз подчеркнуть тот фактор, что мы создали процесс, при котором разработка прямого доступа до прода не имеет (доступ до серверов, k8s и тд),но, ввиду полной прозрачности процесса доставки, видимости и наблюдаемости окружений, разработчик фактически получил все необходимые ему инструменты, чтобы "девопсить по фэн-шую" и нести ответственность за свой продукт. У него фактически пропала возможность говорить "виноваты все, кроме меня, локально все работает". Здесь нам также помогла созданная доверительная атмосфера в командах, где за неудачи не ругают, а видят в них зоны роста.

Чтобы вы лучше поняли наш процесс, для начала нужно разобраться с терминами и что мы под ними понимаем:

- **QGs и SGs (quality and security gates)** - Проверки и тесты на различных этапах. Такие как SAST, юнит, UI, API, регрессионные, контрактные, ручные, интеграционные, агентские и т.д. Несоответствие значения QGs/SGs эталонному блокирует конвейер до исправления ошибок.
- **PROMOTE ARTIFACTS** - Перенос артефактов из нижестоящего (с разработческого контура в интеграционный. Это разделенные сегменты) NEXUS в вышестоящий. На самом полигоне при этом ничего не меняется и не запускается. Это всего лишь

декларирование факта, что артефакт соответствует всем QGs и SGs и тем самым получил визу, чтобы перейти в другой NEXUS.

- **DEPLOY** - Выкатка нового кода на полигон. При использовании feature toggle новый функционал не включен, включение происходит отдельным процессом. То есть deploy только выводит новый код на полигон, но пользователь никаких изменений не замечает.
- **PROMOTE VERSION** - Коммит версии приложения в репозиторий с deploy-map (это репозиторий gitops с состоянием полигона. Более подробное описание см. в п4 "Описание самого CI/CD-процесса"), для PROD открывается MR в защищенную ветку.
- **RELEASE** - Включение нового функционала с помощью feature toggle. Именно в этом случае пользователь видит новую фичу или изменения.

Чтобы наша концепция работала, мы определили, что:

- команда должна быть готова выкатывать каждый сервис отдельно и независимо;
- должно быть реальное покрытие тестами кода на уровне не менее 80%;
- должна быть возможность запуска API и UI тестов для отдельного сервиса.

Описание самого CI/CD-процесса

a) Почему мигрировали с Jenkins на Gitlab CI в части CD

В качестве CI-инструмента в новом конвейере стали использовать GitLabCI. Он был выбран как платформенный инструмент в MOEX. Про опыт выбора инструментов можно говорить долго - мы напишем отдельно про это статью. Тут хочется лишь описать основные моменты.

- Единая точка хранения кода ПО и кода деплоя.
- Хорошая интеграция с различными системами (Jira, Teams).
- YAML для описания пайплайнов, не нужно использовать полноценные ЯП.
- Большая популярность для поиска и онбординга новых сотрудников.
- Возможность использования шаблонов (широкие возможности по переиспользованию и распространению новых фич).

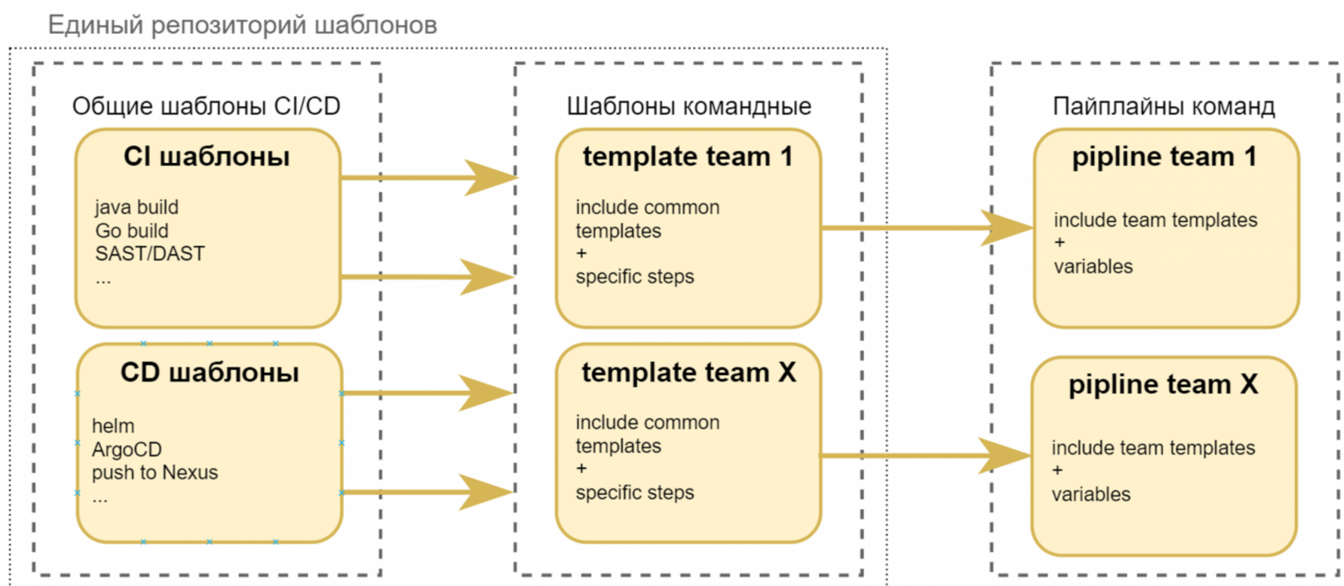
b) Про структуру CI

Шаблонизация легла в основу всех пайплайнов. В GitLab есть замечательный функционал, который позволяет включить в описание вашего пайплайна внешние YAML-

файлы. Выглядит это так:

```
include:
  - project: platform/templates
    file: ci/templates/java/java-gitlab-ci.yml
variables:
  VAR1: "var1"
  VAR2: "var2"
```

Мы сделали единый репозиторий с шаблонами, доступный всем командам. Он состоит из общих шаблонов CI и шаблонов CD. В них содержатся отдельные "кирпичики", из которых строятся пайплайны. Также в репозитории есть директория с командными шаблонами. В ней содержатся шаблоны команд, в которой объединены общие шаблоны +специфика команд.



Такая концепция позволяет разработчикам почти моментально ставить свои сервисы на общие рельсы. Для этого добавляем в наш gitlab-ci.yml include шаблон и пару переменных – все готово. Также мы можем легко изменять процесс одновременно для всех команд, сделав необходимые правки в командном шаблоне. Последним мы сделали шаблон проверки docker image с помощью инструмента trivy, протестировали, добавили в общий командный шаблон, и теперь все разработчики могут просмотреть уязвимости в своих сборках, при необходимости их устранить.

В новом процессе особую роль мы отвели понятию QGs. Так как мы предоставили разработчикам возможность доставки своего кода до прода, нам нужно было их подстраховать и дать инструмент контроля качества изменений. QG стоят на разных

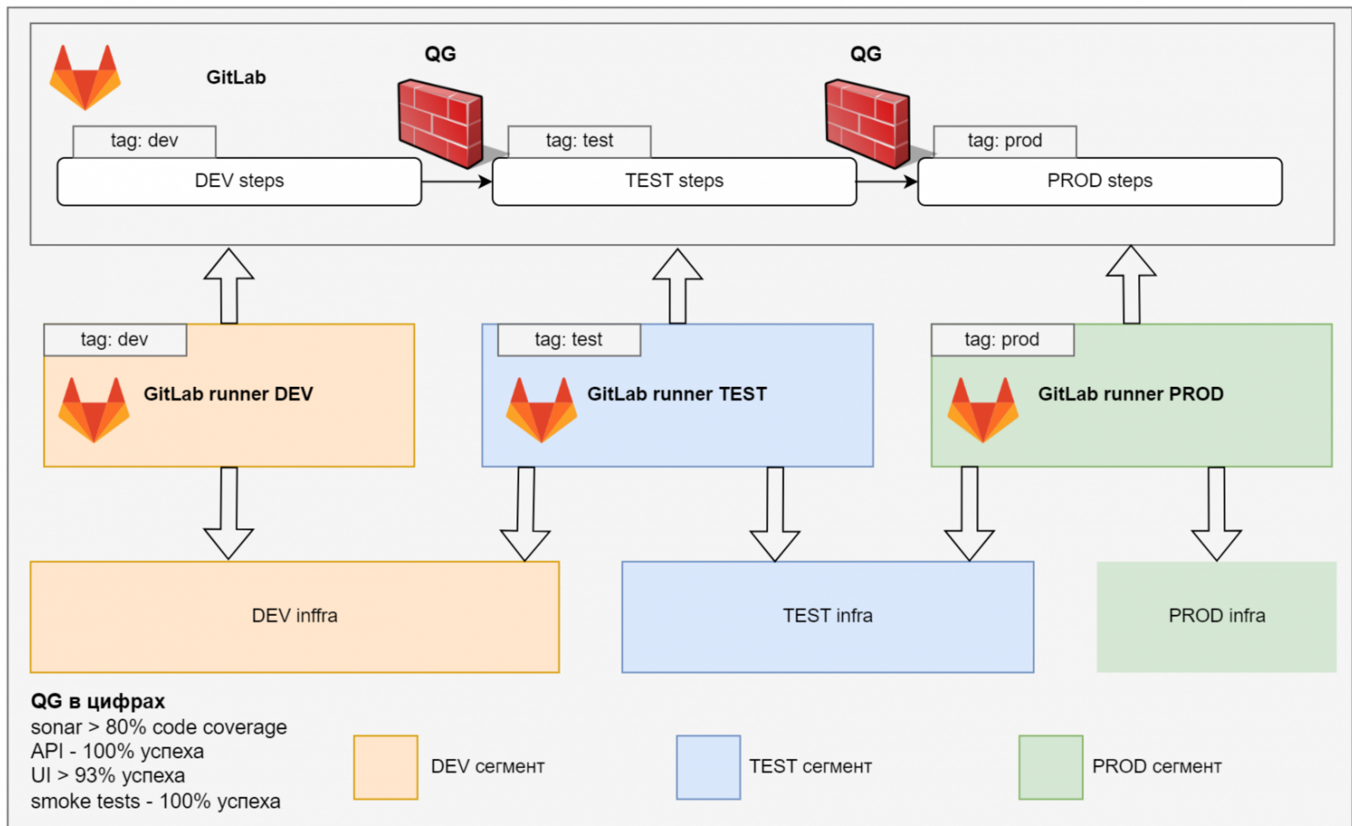
этапах пайплайна и, если не достигнут целевой уровень, то он блокирует дальнейшее продвижение пайплайна.

Что в себя включает Quality Gates (QGs) и Security Gates (SGs):

- Ручной CodeReview
- Синтаксический анализ кода (code-style/lint)
- Статический анализ кода
- Статическое тестирование безопасности приложения (SAST)
- Запуск unit-тестов
- Динамическое тестирование безопасности приложения (DAST, при необходимости)
- Запуск API, UI, manual, smoke=тестов
- Запуск интеграционных, нагрузочных, smoke-тестов
- Проверка артефактов на уязвимости и лицензии.

Для всех Gates с командами разработки были согласованы уровни прохождения. На переходный период были выставлены уровни меньше целевых и смотрели на новый код, чтобы команды успели подтянуть свои сервисы до общего целевого уровня.

Как мы уже писали, разработчики не могут иметь доступа в прод. Но как же им тогда доставить свои изменения? Мы дали возможность разработчикам, при условии прохождения всех Quality Gates (QGs) и Security Gates (SGs) переместить артефакты, полученные в результате сборки, с нижестоящего полигона на вышестоящий. Таким образом, команда разработки может полностью контролировать, когда и на каком полигоне появятся их изменения. И останется только их задеплоить. Также нам приходилось решать определенные трудности изолированности сред и возможности доступов. По нашей концепции мы можем иметь доступ только с вышестоящих полигонов в нижестоящие. То есть прод → тест можно, наоборот – нет. В выбранном нами в качестве CI инструменте GitLab всю полезною работу выполняют runners. При этом они постоянно отслеживают, есть ли для них задание на GitLab, основываясь на заданном для них теге. Соответственно, в каждом полигоне развернуты runner-ы для выполнения шагов в соответствующем полигоне.



c) Про версионирование в целевом подходе

При проектировании целевого версионирования были выработаны следующие подходы:

- Версионирование должно быть автоматическим в рамках конвейера. Избавиться от ручного инкремента минорных версий.
- Для версионирования сервисов используется SemVer строго формата X.Y.Z без добавления beta и т.д. Пример: 1.4.0.
- Источником правды для версий служат git tags как более универсальное решение в отличие от билд-номеров CI-систем.

Major версия берется для каждого языка из разных мест и полностью контролируется разработчиками:

1. NodeJS – файл package.json
2. maven – pom.xml

Далее во время сборки осуществляется автоматическая корректировка **minor** и **patch** версии:

- Для **trunk** (**develop** branch) изменяется **minor** версия;
- Для **hotfix** изменяется **patch** версия;

- Для **feature** веток **не генерируется** версия и **не создаются** артефакты.

Правила изменения версий компонентов:

1. Для **major** версии нумерация **начинается с 1**, изменяется на **+1** командой разработки, ответственной за компонент, только в случае, когда невозможно поддержать обратную совместимость с предыдущей версией компонента. Происходит по итогам разработки story перед вливанием **feature** branch в **develop** branch (trunk). Через **последний** commit в feature branch.
2. Для **minor** версии нумерация **начинается с 0**, изменяется на **+1 автоматически при каждом push в trunk**. Сбрасывается в 0 в случае изменения major версии.
3. Для **patch** версии нумерация также **начинается с 0**, изменяется на **+1 автоматически при каждом push в hotfix**. Сбрасывается в 0 в случае изменения minor или major версии.

Техническая реализация

Для реализации версионирования, согласно принятым правилам, был выбран инструмент GitVersion.

GitVersion позволяет выполнять bump версии без написания дополнительной логики. Необходим только **GitVersion.yml** конфиг с описанием логики:

```
mode: ContinuousDelivery
no-bump-message: ^\b$
continuous-delivery-fallback-tag: ''
commit-message-incrementing: Disabled
branches:
  trunk:
    regex: ^master$|^main$|^develop$
    tag: ''
    increment: Minor
    track-merge-target: true
    source-branches: []
  hotfix:
    regex: ^hotfix[/-]?
    tag: ''
    increment: Patch
    track-merge-target: true
    source-branches:
    - develop
    - main
    - master
```

```
    tracks-release-branches: false
  main:
    regex: ^\b$
  develop:
    regex: ^\b$
  feature:
    regex: ^\b$
  release:
    regex: ^\b$
  pull-request:
    regex: ^\b$
  support:
    regex: ^\b
```

Основная логика:

1. Считывание Major версии из файлов проекта (в зависимости от каждого языка) и сохранение в переменную окружения **$NEXT_MAJOR**
2. **Запуск утилиты** для определения новой версии:

```
/tools/dotnet-gitversion /config /tools/GitVersion.yml /nofetch /nocache /showv
```

1. Сохранение результата в build.env, для использования в последующих шагах в пайплайне
2. После успешной сборки создание git tag с соответствующей версией и пушем тэга в транк

Пример Set Version в пайплайне:

```
Set version:
  image:
    name: <private repo>/cicd/gitversion:${GITVERSION_IMAGE_TAG}
    entrypoint: [""]
  stage: version
  cache: []
  variables:
    GIT_DEPTH: "0"
  before_script:
    - |
```

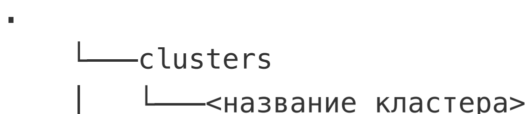
```
      NEXT_MAJOR=$(grep '<version.major>' pom.xml | cut -d'>' -f2 | cut -d'<
      echo "INFO - Next major: ${NEXT_MAJOR}"
  script:
    # run gitversion for additional information
    - /tools/dotnet-gitversion /config /tools/GitVersion.yml
    - APP_VERSION=$(/tools/dotnet-gitversion /config /tools/GitVersion.yml /r
    - 'echo "INFO - Next version: ${APP_VERSION}"'
    - 'echo "INFO - Check if a git tag with version ${APP_VERSION} does not e
    - |
      if [ $(git tag -l "${APP_VERSION}") ]; then
        echo -e "ERROR - git tag for ${APP_VERSION} already exists in reposit
      fi
    - echo "APP_VERSION=${APP_VERSION}" >> build.env
  artifacts:
    reports:
      dotenv: build.env
```

d) CD: почему выбрали ArgoCD

В качестве инструмента CD мы используем ArgoCD, который является также платформенным инструментом в MOEX. Основным плюсом ArgoCD для нас является декларативный подход, следования подходу GitOps и красивый UI, который очень нравится разработчикам 😊 . Для создания приложений мы выбрали паттерн app-of-apps - создавать приложения и проекты в ручном режиме у нас запрещено. Под каждое окружение dev/test/prod у нас развёрнуты отдельные инстансы ArgoCD, которые располагаются также в отдельных, так называемых, management-кластерах под каждое окружение (в этих кластерах только инфраструктурный тулинг). Для авторизации в ArgoCD используется интеграция с корпоративным Keycloak+LDAP. Также для каждого окружения созданы отдельные гит-репозитории, в которых декларативно описано состояние приложений и проектов для данного окружения. У себя мы также называем эти репозитории deploy-map.

Deploy-map, по сути, представляет собой репозиторий, в котором находится Helm-чарт и набор values-файлов. Для удобства поддержки нескольких репозиториев deploy-map под каждое окружение мы написали Library Chart, с помощью которого можно гибко создавать приложения в ArgoCD.

Структура размещения values-файлов имеет следующий вид:



```
.
    └──clusters
    |   └──<название кластера>
```

```
    |        ├──<название полигона>.yaml
    └...
```

Пример файла описания дев-полигона:

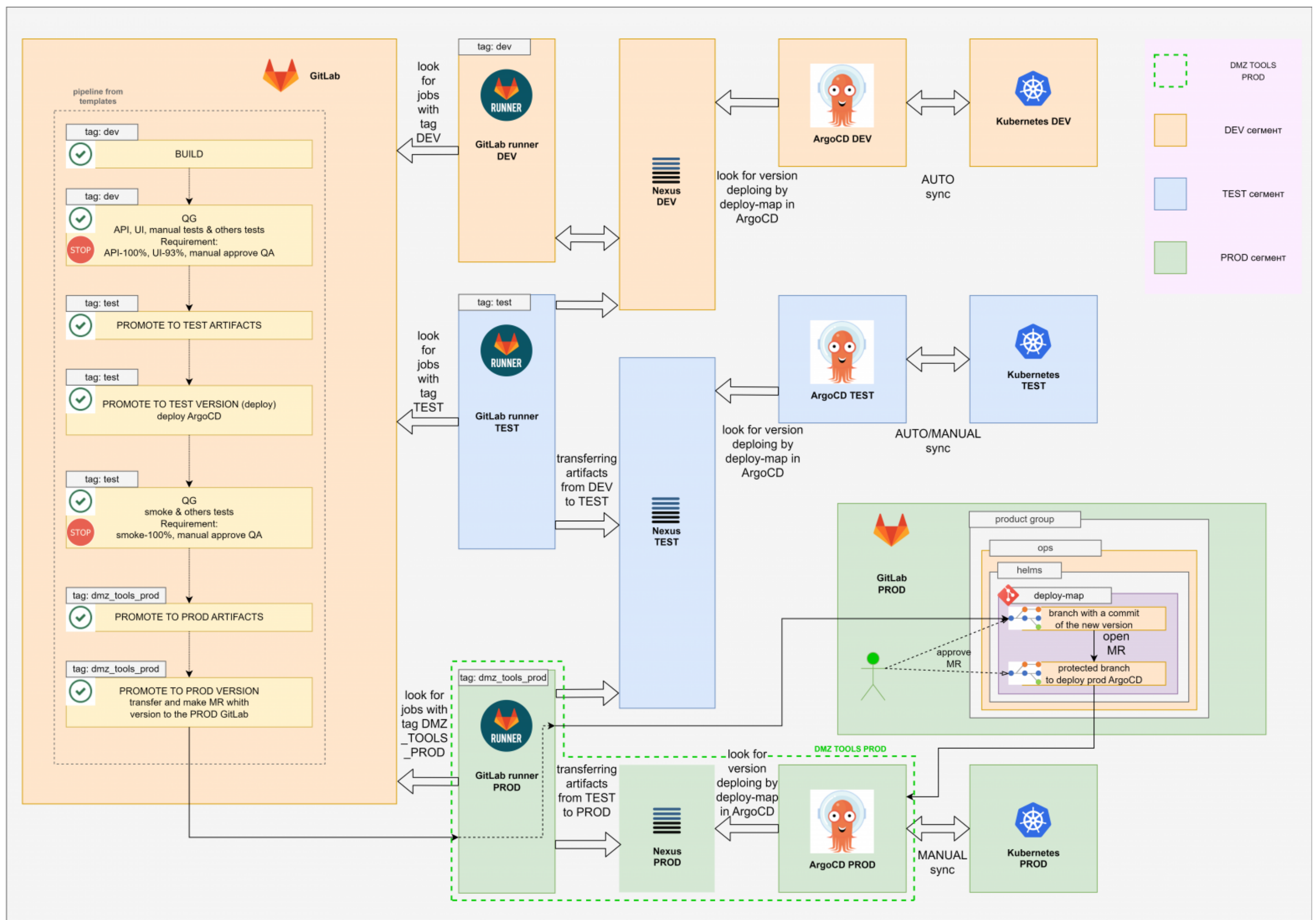
```
autoSync: false
cluster: dev
environment: dev
namespace: dev
project: dev
selfHeal: false
valueFile: values-dev.yaml
valueFileCommon: values-common-dev.yaml

applications:
- name: foo-1
- name: foo-2
- name: foo-3
- name: foo-4
- name: foo-5
```

Далее в ArgoCD создается приложение в парадигме app-of-apps, которое смотрит на гит-репозиторий и создаёт новое приложение автоматически. Поскольку deploy-map находится в гите, то добавление приложения сводится к созданию MR, код-ревью и успешному прохождению пайплайна с линтингом. Продуктовые команды активно добавляют приложения на окружения именно таким образом.

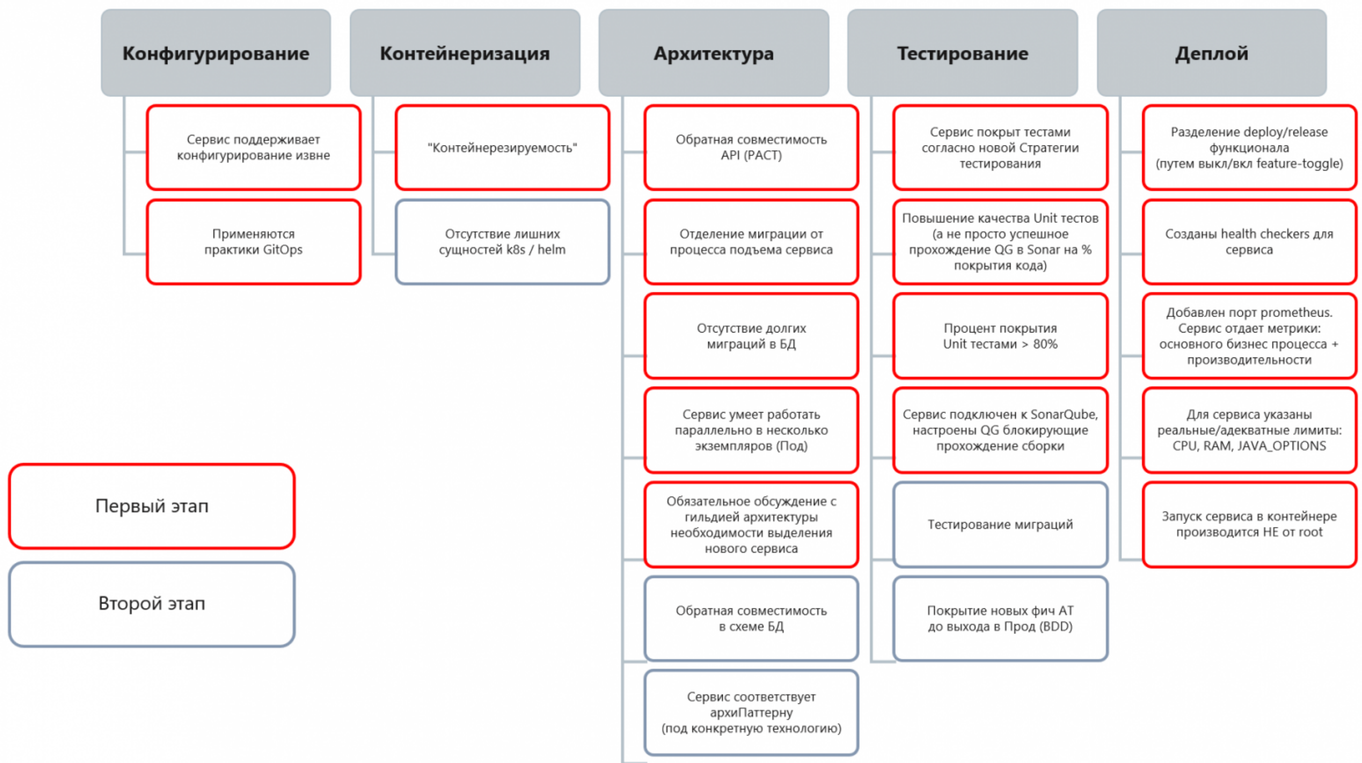
Все это позволило нам деплоить сервисы на наши полигоны одинаково и единообразно, чего раньше не было.

Общая архитектура конвейера:



2.5 Что изменили в архитектуре продуктов

Важной частью инженерных изменений была архитектура продуктов и их связанность. Ввиду того, как было сказано ранее, что наша цель была обеспечить доставку микросервисов обособленно, мы начали активно работать над созданием стандарта к микросервисам и формировать план приведения всей архитектуры к этому стандарту. Все полученные стандарты были заложены в функционал платформы разработки - "Микросервис по кнопке". Это фича в портале разработчика, которая позволяет за несколько кликов получить готовый микросервис по стандарту, подключенный к единому CI/CD, инструментам мониторинга и логирования, внутренним репозиториям и т.д.



Наряду с этим, был организован переход на trunk-based подход. Ранее у нас были проблемы с ветками- под каждый релиз отводились новые ветки, что-то забывали подливать, теряли изменения...

Кратко, про сам подход:

- у нас одна ветка develop, т.е. все живут в единой стратегии ветвления;
- разработка ведется по принципу branching by abstraction;
- подход поддерживает фича-флаги, что дает нам возможность гибко управлять изменениями и отключать функционал, если он не готов или нашлась ошибка в последний момент.

Однако у данного подхода, наряду с плюсами, есть свои особенности:

- удаление старых фича-флагов и, в целом, управление ими;
- quality gates и security gates должны быть на очень высоком уровне;
- необходимо инвестировать ресурсы в автотестирование, и этот процесс должен быть прозрачен бизнесу;
- требуется обучение новому подходу разработки branching by abstraction.

Для перехода были организованы технические спринты. Это были итерации, направленные только на технологические задачи, связанные с изменением ветвления и конвейера. В этих спринтах мы сначала наращивали покрытие тестами. После

проставляли quality gates (QGs) в нужных местах конвейера, неспеша переходили на новые рельсы и стабилизировали все последующие процессы.

3. Планы на развитие

Работа над улучшениями – непрерывный процесс, поэтому хочется поделиться нашими некоторыми планами на 2023 г. Наряду с описанной выше трансформацией, мы решили из линии сопровождения сделать полноценную SRE-команду со всеми необходимыми компетенциями. Что уже успели сделать:

- команда активно работает над повышение отказоустойчивости и надежности системы,
- на порядок увеличена наблюдаемость прод-окружения,
- перешли на спринты,
- расширен функционал инфраструктурной платформы с фокусом на потребителей, то есть на продуктовые команды. Инфраструктурную платформу начали рассматривать как полноценный продукт.

Что предстоит:

- способность системы автоматически восстанавливаться (применение паттернов при построении архитектуры),
- усовершенствование конвейера доставки новыми проверками,
- добавление новых фич на платформу разработки для продуктовых команд,
- развитие сообществ и увеличение количества совместных активностей.

4. Выводы

Мы хотели показать, что изменения и трансформации охватывают много различных аспектов: как культурного характера, так и команды, технологии и инструменты. Поэтому статья получилось большой, но, надеемся очень интересной. Хочется подытожить рассказ следующими тезисами:

1. Для трансформации должны быть определены инвестиции. Такие истории невозможно реализовывать только на голом энтузиазме, и в этот процесс должны быть вовлечены все стейкхолдеры.
2. Подходов к трансформации у нас было несколько, но самый последний и масштабный, который затронул все вышеперечисленные аспекты, занял около 6 месяцев. Нужно подчеркнуть, что дальше следовали процессы постоянного

усовершенствования, основные задачи трансформации служили толчком для последующих положительных перемен.

3. Переходы на новые практики и процессы осуществлялись постепенно. Модифицирующая команда брала один микросервис одной продуктовой команды, переводила на новые рельсы в виде прототипов, делала демо для самой команды, и,когда пилот был успешным, команда самостоятельно переходила на целевые процессы. Модифицирующая команда всего лишь была рядом и могла сориентировать или подсказать.

4. Если продуктовая команда четко понимает, зачем ей нужны изменения и какие возможны бенефиты, то она сама приложит максимальные усилия для реализации изменений. Но для этого сервисным и модифицирующим командам нужно включить навыки продажников и маркетологов, чтобы продать новую концепцию. Никакие принуждения к изменениям не работают. Да, возможно, инструмент и процесс вы поменяете, но команда не поймет, зачем это нужно было...

5. До трансформации публикации в прод были только в ночное время с плашкой о технических работах. Трансформация позволила реализовать стратегию бесшовной публикации в любое время без простоя и влияния на пользовательский путь.

6. Очень важно не увлекаться сверхулучшениями. Инженерам это нужно хорошо осознавать, в противном случае инженерные процессы превращаются в "полигон для оттачивания навыков" о новомодные инструменты, не нужные бизнесу.

7. Конвейер производства должен быть построен на нативных и простых инструментах. Нужно обращать внимание на порог входа в инструмент и саму структуру пайплайнов. Важно понимать, чем занята инженерная команда, отвечающая за построение инфраструктуры производства и насколько принятые там решения/паттерны соответствуют целям продуктов.

8. Описанная выше трансформация позволила решить все обозначенные проблемы, исходящие из предпосылок. Залог успеха – правильно сформулированные миссии и конечные образы результатов.

9. Процессу трансформации Финуслуги.ру дала сильный импульс общегрупповая стратегия DevOps-трансформации. Многие аспекты инженерной культуры, паттернов и инструментов там уже были описаны и отлично себя зарекомендовали к тому моменту.

10. Идеальный DevOps случается тогда, когда тот, кто разработал продукт, его доставляет и поддерживает.

В конце, хочется поблагодарить всю нашу крутую команду, которая реализовывала процессы трансформации и продолжает совершенствовать продукты. Мы уверены, что люди - это главное, а инвестиции в развитие команд, лидеров и культуры могут изменить мир к лучшему.

Полезные ссылки и выступления:

- Доклад Карапета Манасяна про стратегию DevOps-трансформации;
- Первое сообщество про platform engineering – Platen, который ведет Карапет. Подписка, колокольчик и пальчик вверх приветствуются 😊

Теги: devops, трансформация, platform engineering, коммуникации, топологии, лидерство, команда, разработка, ci/cd, devsecops

Хабы: Блог компании MOEX, Управление разработкой, Agile, DevOps

MOEX

Инвестиции начинаются здесь

Подписан ×

Сайт Facebook Twitter ВКонтакте Instagram

5 Карма 0 Рейтинг

Каро Манасян @Manassian

Пользователь

Комментарии 2

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

AlexeyNadezhin

9 часов назад

Тест самых выгодных батареек AA и AAA

Простой 3 мин 7.1K

+64 37 24