olegchir 17 октября в 10:11

DevOps в Сбербанк-Технологиях. Инструментальный стандарт

Системное администрирование, Серверное администрирование, DevOps, Блог компании JUG.ru Group

В этой статье пойдет речь об организации инструментального стека DevOps на примере Сбербанк-Технологий и ППРБ. Статья предназначена для инженеров по автоматизации инфраструктуры, которым необходима объективная оценка структуры работ по внедрению DevOps — и для всех, кто хочет ознакомиться с их работой.

Не секрет, что в маленькой сплоченной команде организовать DevOps-процесс весьма просто. Более того, у современных грамотн разработчиков и админов, такой процесс может сложиться сам собой — как результат высокой культуры разработки и поддержки. что делать, если твоя маленькая команда из десяти человек начинает расти до конторы из сотен людей? Что делать, если в твоей организации уже несколько тысяч человек, а девопсом и не пахло? С этими вопросами мы обратились к сотрудникам компании Сбербанк-Технологии, имеющей положительный опыт внедрения практик DevOps в крупномасштабных проектах, и кое-что узнали.

Safe Harbor

Во-первых, небольшая оговорка. Многие спрашивают, зачем нам читать статьи о том, как это сделано в других компаниях? Они сд так, а мы сделаем эдак, какая разница-то.

Продукты Сбербанк-Технологий используются, конечно, в Сбербанке, и от стабильности и безопасности их работы зависят наши с деньги. Появление этой статьи в том числе показывает, что компания достигла того момента в своем развитии, когда о внутренних технологических процессах уже можно рассказывать, и это абсолютно нормально. Это не какая-то security through obscurity, а по-настоящему хорошая, качественная отлаженная система — в этом её ценность, и ценность текста, который вы сейчас читаете. Это вещь, которая действительно работает. С этим знанием можно делать все что угодно — например, один в один скопипастить такую инфраструктуру себе. Или пойти работать в Сбербанк-Технологии, если вам вдруг понравилось жить с такой инфраструктурой.

Тем не менее, внедрение девопса в *промышленную* эксплуатацию мы обсуждать все же не станем. Давайте сделаем вид, что это н только из-за безопасности, а еще и потому, что внедрение всё еще идет, и некорректно здесь обсуждать результаты неубитого ме Предполагается, что внедрение девопса ППРБ на пром произойдет уже в конце этого, либо начале следующего года — возможно, расскажут на JBreak/JPoint 2018. Давайте сконцентрируемся на настоящем.

В любом случае, в статье *могут быть* ошибки и недочеты. Данная статья не является официальной позицией компании Сбербанк-Технологии или Сбербанка. Полученная из этой статьи информация не может использоваться как основание для принятия любых коммерческих и других важных решений.

Масштаб проблемы

В чем особенность девопс-проектов СБТ (давайте называть организацию таким неформальным образом)? Конечно, это масштаб. В году количество сотрудников СБТ превысило 9800 человек, расположенных в 17 городах России.

Зачем нужна эта уйма людей, чем они вообще занимаются? (спросит любой нормальный человек). Формально это звучит вот так:

- СБТ создает программное обеспечение для крупнейшего банка в России и Восточной Европе (325 тысяч сотрудников)
- Решает сложные инфраструктурные и сервисные задачи, которые улучшают жизнь 70% россиян
- Обеспечивает удобство и доступность сервисов Сбербанка для 110 миллионов человек в мире

Что это значит с точки зрения инженера-инфраструктурщика, который занимается внедрением девопса? Глядите, у нас обычно ес минимум, следующий набор задач, которые необходимо поставить и решить:

- Повысить общую культуру производства и обслуживания

- Сформулировать и внедрить набор инструментов для типичных задач (в особенности “трубы” CI/CD)
- Улучшить навыки решения повседневных задач (навыки автоматизированного решения рутинных вопросов)

Каждый из этих вопросов резко осложняется от увеличения масштаба. Можно быстро договориться об инженерных практиках вну своей команды, но помирить между собой 8500 человек, имеющих совершенно разные взгляды на разработку — это челленж. Обы галере тут бы и лапки, но СБТ умеет превозмогать.

То, что кажется одной команде свободой (например, свободой использовать стандартизованный пайплайн, в рамках которого мож задумываться о ненужных мелочах), для другой выглядит как существенное ограничение свободы выбора инструментов. На краси маркетологических презентациях по девопсу не любят рассказывать, что если одна команда делает Единую Всеобщую Платформу Всего, то еще 100 команд будут ее ненавидеть за необходимость использования этой стандартной платформы, а не любимой Скал Хаскеля. С другой стороны, если всем дать возможность выбирать разнородные решения, никогда не выйдет общего продукта.

Недостаточно весело. Давайте добавим еще немного сложности! Дело в том, что проекты типа Технологического Ядра ППРБ (один основных проектов в СБТ) — это черезвычайно сложные самописные технологии, специально предназначенные для особого, часто процесса развертывания. (Об этом будет ниже). По сути, это собственная Java-платформа, и как любят шутить разработчики “еще немного — и напишем собственную Java-машину!”. Даже базы данных тут особые — GridGain и набор уникальных ноу-хау на тему, этот GridGain вообще готовить. Эта сложность является частью предметной области, нельзя просто так взять и сказать “а давайте в выбросим и перепишем на Golang в виде трех микросервисов”. Это так не работает!

Таким образом, инструментальный стек является точкой баланса не только между различными культурами, но и различными подхо к проектированию и программированию — бездонная пропасть между высокоспециализированным технологическим стеком и желанием иметь стабильный, десятилетиями оттестированный софт для администрирования.

Поэтому каждый элемент инструментального стандарта CI/CD СБТ — это компромиссное решение. Эти решения, как правила дорожного движения, написаны кровью и годами. Пришло время показать этот инструментальный стандарт!

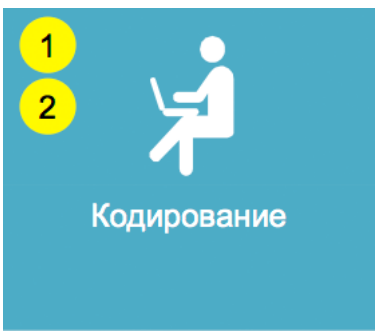
Инструментальный стандарт CI/CD

И вот сейчас вы удивитесь. Удивитесь его *обычности*. Как удивились мы в JUG.ru впервые увидев это, так удивились и архитекторы инфраструктурщики СБТ, когда поняли, что выходит в результате. Вы же привыкли, что СБТ постоянно пишет какие-то невообразим новые фреймворки, используя их странными способами, чтобы потом рассказывать об этом на конференциях? Не в этот раз. Оказывается, самый лучший способ (на данный момент, осень 2017 года) — использовать то, что используют все, и заполировать решение до зеркального блеска.

Для начала, взгляд сверху:

Такая же последовательность шагов есть примерно у всех современных проектов. ПСИ — это Приемо-Сдаточные Испытания.

Заранее извиняемся, что картинка плохо читается на мобильниках. В дальнейшем все эти пункты будут продублированы большими пиктограммами, так что вы ничего не потеряете.



Для многих является шоком то, что кодирование — чуть ли не самая важная и жирная часть стандартов CI/CD. Дело в том, что если исходный софт написан критически неправильно, то ему уже никакой “девопс”, ничего не поможет.

По этому поводу существует два основополагающих элемента, зависящих от роли:

1. *Разработчик*. Обычно это Java/Scala/Cpp стек, работа с которым происходит на локальных машинах.
    - Разработка кода и другого контента
    - Выполнение юнит-тестов перед заливкой в репозиторий
    - Публикация кода в ветку `develop`
    - Привязка к задаче в багтрекере
2. *Главный разработчик*.
    - Code Review
    - Merge в ветку `master`

Разработчики могут использовать то, что нужно для выполнения задачи. В качестве стандарта для Java-разработки, например, луч взять те средства, которые используют во всем мире: Maven (собственно Java), Node Package Manager (для JS-фронта), JUnit и Test (тестирование). СБТ не разрабатывает своих собственных систем сборки не потому, что не может (чем мы хуже Google?), а потому это нарушило бы идею общей точки равновесия. Слишком многие внешние по отношению к компании проекты (попробуйте прожи Spring Framework!) завязаны на стандартные утилиты вроде Maven, и нарушать эту интеграцию бессмысленно.

Хранение детальных требований на этом этапе происходит на Wiki-подобной системе (н-р Confluence). Работа с задачами происхо на основе стандартной аджайл-терминологии (эпик, фича, юзер-стори, итп), нативно реализованных в выбранной системе управле задачами (н-р JIRA). Наличие этих двух систем *критически важно* для получения хорошего результата. Это правило записано кров даже маленькому проекту нужно сразу же выдавать вики и багтрекер.

Важно отметить, что здесь и далее, для разработки используются два различных репозитория:

- один из репозиториев хранит собственно код продуктов,
- другой предназначен для хранения только скриптов сборки

В рамках идеи Infrastructure as Code, сейчас зачастую принято хранить скрипты развертывания вместе с проектом в одном репози В результате длительной практики, в СБТ выбрали иметь отдельный репозиторий для скриптов развертывания, и при необходимос осуществлять сильную связанность программным способом. Это особенность связана исключительно с масштабом и необходимо сложных связанных обобщенных конфигураций, когда один и тот же “скрипт” может работать с разными продуктами. Плюс это позволяет развести команды разработки и администрирования по зонам ответственности и уровням доступа. Конкретные настрой проектов и специализированные скрипты, конечно, хранятся вместе с проектами.

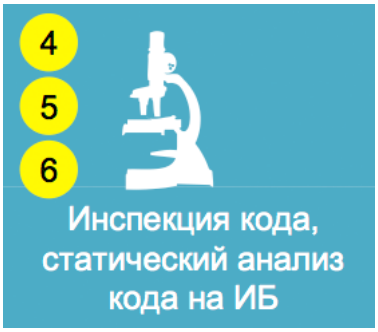
В качестве управляющей системы здесь и далее используется Jenkins. Кому-то он может не нравиться, по причине того, что это др софт, обросший определенным количеством легаси, и имеющим серьезный футпринт по объему оперативной памяти, использован процессора, и так далее. Но помните что мы говорили про точку баланса? Факт остается фактом, что при всех своих недостатках, сейчас (осень 2017 года), Jenkins — это именно та система, которую согласны использовать все. В будущем это, возможно, измени

Кроме Дженкинса используется дополнительная система управления сквозным процессом развертывания, начинающаяся с этапа кодирования, и продолжающаяся вплоть до ПРОМа. Но это специальный внутренний софт, описывать который в данной статье не и смысла не только по причине недоступности в Open Source, но и потому, что он очень заточен под конкретные особенности СБТ. Е предположение, что любая компания с достаточно большим пайплайном *всегда* пишет подобную систему. Для небольших проекто может существовать в рудиментарном виде, в виде пачки скриптов, не зависящих от Дженкинса. При масштабировании на 10+ проектов, идущих в связке, она может превратиться в серьезный софт.

Важно, что репозиторий скриптов развертывания, дженкинс и инструменты сквозного управления доступны сразу же, начиная с эт кодирования. Если кому-то из разработчиков будет нужно пойти на Jenkins и сделать там новый джоб — это нормально. Использов девопс-инструментов — это нечто настолько же свободно доступное и неотъемлемое, как дышать воздухом.

Тут все просто. Используются те инструменты, на которых была сделана изначальная сборка и написаны тесты. Здесь используется из самых расширяемых наборов инструментов, кроме того — сильно зависящий от языка. Для Java это, скорей всего, будет JUnit, д .NET — NUnit, для JS — QUnit. Важно только, чтобы для этих инструментов были соответствующие плагины для Jenkins. В крайнем случае, их можно написать самостоятельно, но это нежелательно. Проверка кода — это критически важный этап, на котором лучш использовать проверенные веками решения.

У многих разработчиков есть собственные DEV-стенды (сервера), на которых они могут проводить простое системное тестирован любые удобные эксперименты вне своего компьютера. Конечно, эти стенды очень ограничены в ресурсах (им экономически бессмысленно выделять терабайт RAM), и на них никак нельзя провести интеграционное тестирование. Но для экспериментов — п



На этом этапе происходит несколько ключевых вещей:

1. Статическая проверка кода
2. Проверка на соответствие правилам информационной безопасности
3. Найденные дефекты сразу же отправляются в работу, потому что это **очень важно**

По поводу ИБ мы тут распространяться не будем по понятной причине. Последнего кто говорил о безопасности забрало НЛО :-) Достаточно сказать, что **все** элементы CI/CD учитывают требования безопасности, и в случае чего там мышь не проскочит. Обычно безопасность понижает удобство использования инструментов CI/CD, поэтому это очень сложная тема, наполненная особыми инженерными компромиссами.

Что касается обычного поверхностного статического анализа, то используется `SonarQube`. Понятно, что для множества разнородны проектов существуют всевозможные конфигурации и наборы правил кодирования, поэтому к каждому из них нужно подходить индивидуально и с пониманием специфики. Конечно, для всех Java backend проектов есть заготовленные наборы стандартных пра чтобы не нужно было забивать голову по-мелочам.

У каждого проекта есть свой собственный стенд (сервер), на котором можно и нужно тестировать готовые решения. Нельзя просто взять и выкатить непроверенное решение на общее интеграционное тестирование. Стенд может выделяться как под весь проект целиком, так и под конкретного разработчика (для тестирования экспериментальных веток). Эти стенды имеют более серьезные ресурсы, чем личные DEV-стенды разработчиков, но все еще ограничены рамками разумного, т.к. полное воссоздание всей инфраструктуры на ресурсах, выделенных проекту, было бы невероятно сложно, долго и экономически бессмысленно. Поэтому нормального интеграционного тестирования тут не получится, да оно и не нужно.

Несколько важных моментов, которые происходят на этом этапе:

1. Финальная сборка и публикация артефактов в хранилище. Например, для Java это будет Nexus, и обратиться к этим артефакта сможет любой проект с помощью Maven или по прямой ссылке.
2. Установка приложений на стенд.
3. Выполнение набора автотестов.

Интересной особенностью этих командных DEV стендов является то, что его уже нужно готовить по всем правилам: устанавливать внутренний софт, создавать тестовых пользователей, и так далее. Обычно стенд — это не один компьютер, а кластер из виртуальны машин, поэтому в этот момент тестируется еще и правильная инициализация кластера. По сути здесь происходит еще и мини-инсталляционное тестирование всего стека, начинающееся прямо с момента развертывания стенда.

На этом этапе обычно появляется `Ansible`. Энсибл в основном используется как средство конфигурирования целевой системы, и запускается из Jenkins. Кроме того, Энсибл можно использовать более комплексно для управления уже развернутыми системами - для этого не предназначен, но для целей на контуре DEV он отлично для этого подходит. Энсибл был выбран в том числе и за то, ч не нужно специального агента (кроме SSH сервера, который и так есть на всех хостовых GNU/Linux системах), поэтому если кому- команде ВНЕЗАПНО потребуется перезапустить что-нибудь на 10 серверах в DEV-кластере — это не требует никаких специальных “тайных админских знаний”. Энсибл очень прост, поэтому и разработчик, и грамотный админ, справляются с изучением Энсибла в самые кратчайшие сроки.

Конечно, в использовании Энсибла существуют свои заморочки. Например, стандартная система ролей может оказаться *излишне* простой для сложного модульного проекта. В самой свежей версии есть экспериментальная поддержка вложенных ролей, но и её не хватить (тем более что она действительно очень сырая и не протестирована на совместимость с другими фичами Энсибла). Есл пытаться использовать Энсибл полномасштабно, как систему не только конфигурирования, но и управления целевыми системами, придется написать много дополнительного обвязочного софта, где сам Энсибл является всего лишь “SSH-ассемблером” для более высокоуровневых команд.

Кроме того, быстро выясняется, что девопсы, плотно работающие с Энсиблом, должны знать Python. Например, у вас есть какой-т самописный софт, который обладает *черезвычайно* сложной процедурой конфигурации, связанной с тонкой настройкой кластера, линукса, прикладного ПО, и так далее. Многие из этих операций требуют работы с малоизвестными и нативными функциями, для которых сообщество не написало готовых решений, да и не напишет никогда. Если описывать это в плейбуке на “чистом ямле”, это заняло бы десятки страниц текста. А лог этих операций был бы трудночитаемой кашей — в, частности это связано с отсутствием нормального управления ветвлением и соответствующей выдачи лога (время от времени вопрос поднимается в сообществе, но во ныне там). Более правильный способ решения этой задачи — написание Action Plugin и Module, весь сложный код в котором инкапсулирован в изящно написанный скрипт на Python. По сути, в самом плейбуке остаются только вызовы плагинов, содержащи высокоуровневые бизнес-параметры — по аналогии с утилитами командной строки GNU/Linux. Из этого напрямую следует, что инженер-инфраструктурщик (девопс, админ, разработчик — кто угодно, кто занимается этой задачей) обязан знать Python на дост уровне. Не обязательно знать его так же хорошо, как специальный Python-программист, но уметь читать и багфиксить готовый код писать простые модули для Энсибла — это рецепт успеха.

Полученный результат зачастую, по сути, является не просто каким-то “скриптом”, а весьма сложным софтом, требующим специал поддержки. Именно отсюда пошло изначальное разделение на пару репозиториев: один для кода продукта, другой для кода скрип развертывания. Для репозитория скриптов развертывания можно вообще использовать какую-то другую, специальную систему управления репозиториями (например, не Atlassian Stash, а GitLab со специфичными настройками), чтобы добавить больше гибкос управлении конфигурациями.

Кроме описанных выше инструментов, существует еще множество мелких. Например, некоторые проекты все еще используют не GridGain, а SQL базы — там можно подключить Liquibase для накатывания миграций. И так далее. Такие вещи уже очень зависят от специфики проекта, и, по возможности, их стоит стандартизировать. Например, если мы выбрали Selenium в качестве фреймворка UI тестирования, именно его и стоит придерживаться, т.к. поддержка таких решений требует большой экспертизы.

На следующем этапе происходит чистое, правильное системное тестирование на выделенном стенде.

Здесь происходит две ключевых вещи:

1. Установка и настройка среды тестирования
2. Выполнение набора автотестов

Установка и настройка среды зачастую сама по себе выявляет множество удивительных вещей, которые сразу же отправляются на доработку. В основном, они касаются простоты и удобства администрирования. Системы имеют тенденцию со временем разраста

сложность настройки увеличивается, и рубить эти проблемы нужно на корню — прямо начиная с этого стенда.

Набор автотестов “чистого” CT тестирования может отличаться от “грязного” DEV-тестирования в сторону увеличения строгости. Д самый незначительный экспериментальный проект обязательно должен иметь тесты, хотя бы базовое smoke-тестирование. Если те нет, на интеграционное тестирование этот проект не пустят.

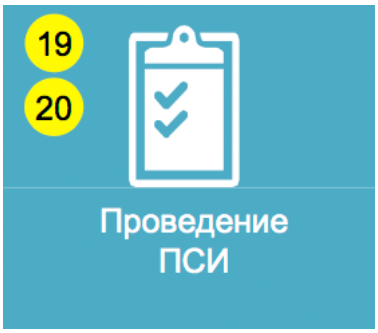
Интеграционное тестирование — самое большое, сложное и важное тестирование. По крайне мере, таким оно кажется разработчи потому что на этом этапе находится самое большое количество сложных багов :-) Провести его нужно максимально аккуратно, пот что любая ошибка затрагивает результаты работы не только конкретного проекта, а еще сотен, если не тысяч людей.

На этом этапе стенды моделируют реальную инфраструктуру, имеют полноценные (весьма большие) ресурсы, и получают полную поддержку системных администраторов.

Происходит несколько обязательных вещей:

1. Тест-менеджер должен получить оповещение о готовности продукта к публикации, и решить — стоит ли это делать. Тест-мене — боец невидимого фронта, но его вклад сложно переоценить.
2. Установка и настройка среды тестирования. Так как среда предназначена для интеграционного тестирования, она общая для в интегрируемых в рамках зонтика проектов, и идет параллельно с приемом заявок на установку продуктов.
3. Автоматическое и ручное тестирование. Самый строгий, сложный и большой набор тестов.
   Прозвенел звоночек на слове “ручное”? Да, в реальных системах *существуют* моменты, слабо поддающиеся автоматизации, и среднем, ничего с этим не сделаешь. Особенно это касается тестов отзывчивости UI (возможно, скоро их будет делать ИИ, но еще — нет). Одна из основных задач всего коллектива разработчиков — сделать так, чтобы этот этап прошел как можно быстр безболезненней.
4. Тест-менеджер получает подтверждение прохождения всех видов тестирования и готовит детальный отчет.
5. Про ИБ мы договорились не писать :-)
6. Особый интерес представляет собой нагрузочное тестирование, результаты которого могут повлечь долговременные последст ведь по сути, это первый момент в пайплайне, когда производительность видно не на формальных расчетах, а по-настоящему.

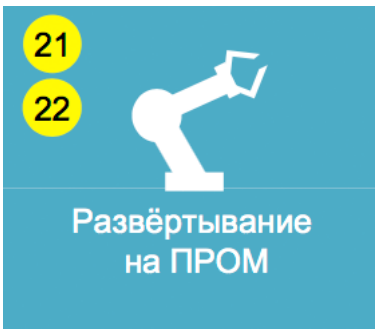
Полностью протестированный готовый продукт выкладывается в специальное централизованное хранилище дистрибутивов. Вмест ним прикладывается вся доступная документация и другие материалы. Полученные файлы замораживаются навсегда. Используют специальные средства для того, чтобы поменять их было невозможно никаким образом.
(Если вы вдруг захотите сами сделать такое хранилище, и пишите на Java, можно просто использовать Nexus).



Проведение приемо-сдаточных испытаний — это другая тема, опасно приближающаяся к тайным знаниям. Но на верхнем уровне т просто, происходит:

1. Проведение специальных автотестов и большого комплекса приемочных испытаний
2. В том числе небывало сложных ручных тестов

После прохождения этого этапа можно считать, что продукт готов к эксплуатации.



Ну и наконец, развертывание в настоящей промышленной эксплуатации, написать про которое, по очевидным причинам, нельзя. Д имеет смысла, т.к. описанный выше пайплайн дотянется до прома только в конце года. В данный момент при развертывании на про используется процедура ручной установки, с использованием инструкций по установке, приложенных к дистрибутиву, которая выполняется специальными экспертами по развертыванию.

В случае автоматизированного развертывания, на этом этапе будет две вещи:

1. Живой человек (условно “devops-инженер”) должен вручную подтвердить готовность к развертыванию
2. Автоматизированное развертывание

ВСЁ. Этого минимального набора инструментов уже достаточно, чтобы куча людей могла ужиться под одной крышей. Важно поним что этот инструментальный стандарт не ограничивает возможности разработчиков, а дает стабильную платформу для дальнейшег развития.

Технологическое Ядро ППРБ

Читатель, знакомый с докладами Сбербанк-Технологий на конференциях JUG.ru (кстати, следующая будет совсем скоро), может заметить, что история девопса вышеописанной схемой не заканчивается. Скорее, она ей только начинается.

Статья начиналась с общего инструментального стека именно потому, что данные решения очень специфичны для самих Сбербанк Технологий, и знать о них, скажем так, нужно не всем. Для тех кто все еще с нами — продолжаем.

Инновационные решения, применяемые в Сбертехе, требуют особенных подходов к разработке. В качестве причин можно было бы назвать разношерстный технологический стек, потребность в аггрегации данных от множества подсистем, невероятное количеств серверов, необходимых для обслуживания такого объема данных, отсутствие “выходных” и “ночи”, когда можно провести обслужи и так далее. Несмотря на необходимость организовать высокую производительность, отказоустойчивость, и обслуживаемость 24x полученные решения должны работать на стандартном железе и не требовать специальных суперкомпьютеров. Результирующее решение должно одновременно обладать признаками и централизованной, и децентрализованной системы. Сложная ситуация.

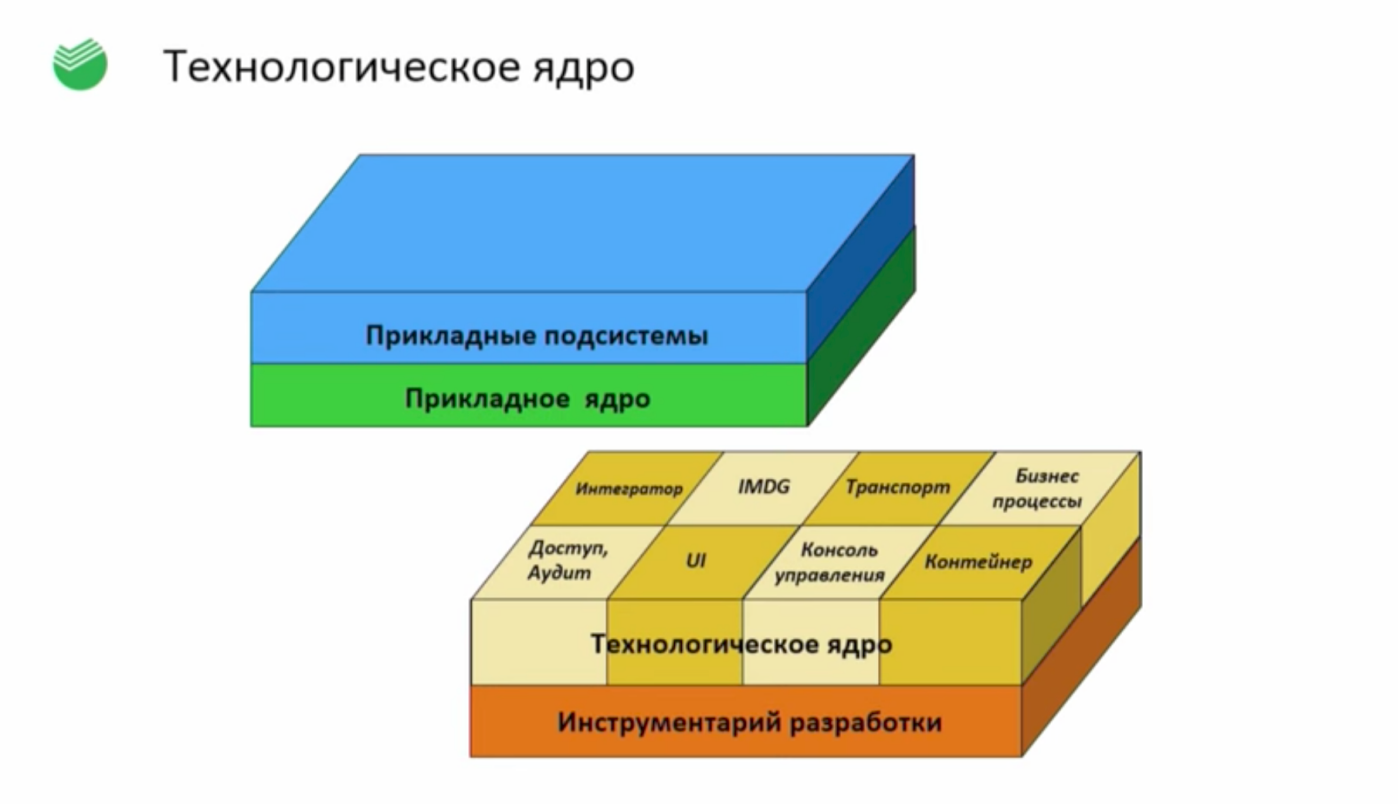
В результате было создано так называемое Единое Информационное Пространство. Это кластерная архитектура, в которой все уз объединены между собой, каждый может общаться с каждым. Нет никакого дублирования и интеграции, все данные существуют в одном экземпляре и доступны всем модулям с любого узла в любое время. В качестве бэкенда этой системы используется in-mem

data grid под названием GridGain. На сегодняшний день, это одна из лучших в мире систем хранения и обработки больших объемов бизнес-данных.

Именно с этой архитектурой администраторам и девопсам приходится сталкиваться при проведении любого интеграционного тестирования. Именно она работает на обслуживаемых ими стендах.

Совершенно очевидно, что управление такой системой ортогонально движению артефактов вдоль CI/CD пайплайна. Jenkins и Ansib можно использовать для инициации инфраструктурных операций, но на них нельзя записать *сами операции*. Это просто другой логический уровень.

Сами операции реализуются в Java-коде специального инфраструктурного уровня, называемого `Технологическое Ядро`.



Для большинства компонентов на этой картинке, имеется как некая “серверная” составляющая (запускаемая на серверах-координаторах), так и прикладные библиотеки, которые обычный прикладной софт использует для работы внутри кластера. По сут Технологическое Ядро дает свое видение на большинство привычных для любого программиста или админа вещей: как хранить настройки приложений, как хранить данные, как организовывать взаимодействие между программами, и так далее. В каком-то см можно думать о нем как о легкой кластерной операционной системе, работающей на основе Java.

Чтобы развеять налет магии, на низком уровне все это работает на основе всем известных и понятных вещей, выложенных в OpenSource.

GridGain можно посмотреть на примере бесплатного Apache Ignite (в чем между ними разница? Недавно, в интервью JUG.ru, об это рассказывал Владимир Озеров). Apache Kafka, ZeroMQ, даже Hyperic — все это находится в Open Source. Они выбраны не только по что писать самостоятельную замену — долго и дорого (СБТ, вероятно, справился бы), но и затем, что для всех этих инструментов у есть сообщество и понимание, как их обслуживание правильно делать в смысле выстраивания процессов администрирования и девопса. Например, если вы вдруг являетесь экспертом по Apache Kafka или OpenStack — быстрей идите в СБТ, вы здесь нужны.

Но также надо понимать, что работа с этими инструментами — это не *администрирование*, а именно **дев**опс. Необходимо колосса количество разработки, чтобы превратить эти компоненты в однородную платформу. **Транспорт** — это не просто Kafka, это специа кластерная среда, которая использует Кафку как бэкенд. Уровень хранения — это не просто Ignite/GridGain, а специальный софт, использующий передовые ноу-хау СБТ, касающиеся работы с IMDG, на который ушло очень много сил лучших разработчиков. Всем компонентам нужно со стороны обслуживания дать однородный интерфейс управления и просмотра диагностики (для администраторов), и однородные Java-интерфейсы (для программистов). Использование этих инструментов требует понимания и культуры использования ото всех участников процесса — и от администраторов, и от программистов, и от специальных инженеров автоматизации инфраструктуры, и так далее.

Главный вывод, который из этого может сделать человек, не являющийся работником СБТ, наверное, в том, какие именно компоненты нужны, чтобы самостоятельно построить подобную систему. И в том, что все эти компоненты железно связаны с дево процессом. Например, рассмотрим сервер хранения конфигурации с графическим интерфейсом на HypericHQ — так называемый Конфигуратор". Вы можете зайти в единый интерфейс и поменять любые настройки системы, и они применятся (или по крайней ме будут доступны для обновления) прямо во время работы кластерного софта, который эти настройки использует. Это что-то типа "реестра Windows", только кластерного, с веб-интерфейсом и нотификациями об обновлении свойств. Как таковой, он не нужен Ja программистам: они могут хранить все настройки в properties-файлах. Как таковой он не нужен и администраторам — вековая пра редактирования конфигов в /etc еще никого не подводила, а старый конь борозды не испортит. Но когда нам нужно выкатить софт действительно быстро, четко и без ошибок, силами обычных администраторов (не подкючая разработчиков конкретного продута) - эти .properties файлы и конфиги в /etc начинают очень сильно мешать. Наличие кластерного реестра моментально решает все эти проблемы.

Как видим, наличие стандартизированного CI/CD пайплайна не ограничивает полет инженерной мысли, а дает ему площадку для беспроблемного результативного развития.

Заключение

В этой статье мы рассмотрели как инструментальный стандарт CI/CD Сбербанк-Технологий (который может адаптировать каждый) специфические для Сбербанк-Технологий решения (которые напротив, могут использовать далеко не все).

Надеемся, что статья оказалась полезной: тем, кто хочет построить что-то подобное у себя; тем, кому просто любопытно; тем, кто пойти на работу в Сбербанк-Технологии и сомневается — будет ли ему там достаточно интересно и приятно работаться. Если у ва нет девопса, можно использовать этот текст как руководство, или показывать друзьям: глядите, вот так у нас все будет хорошо к ко года.

Если хочется узнать больше о девопсе, приходите на нашу конференцию DevOops 2017, которая состоится в Санкт-Петербурге 20 октября 2017. Кстати, Сбербанк-Технологии — золотой спонсор этой конференции.

Благодарности

В подготовке этого текста помогали сотрудники Новосибирского отделения Сбербанк-Технологий: Александр Перковский (замест директора центра компетенции ППРБ), Алексей Курагин и Егор Федоров (архитекторы Технологического Ядра и разработчики сис мониторинга), Олег Чирухин (архитектор и разработчик системы развертывания ППРБ.BPM), Александр Обливальный (инженер по автоматизации инфраструктуры ППРБ) и другие. Мы не можем перечислить здесь всех, но признаем, что ваш вклад был неоценим. Спасибо!

Метки: sbt, сбербанк, сбербанк-технологии, ci/cd, devops, ппрб

Комментарии 30

DexterHD 17.10.17 в 11:47

> Что это значит с точки зрения инженера-инфраструктурщика, который занимается внедрением девопса?

Т.е. внедрением DevOps философии (влияющей абсолютно на все этапы разработки/тестировния/экслуатации и на всех участников процес занимаются какие то отдельные люди в вакууме?

> Именно с этой архитектурой администраторам и девопсам приходится сталкиваться при проведении любого интеграционного тестирова Именно она работает на обслуживаемых ими стендах.

Кто такие эти ДевОпсы? Программисты которые могут задеплоить что угодно куда угодно? Сисадмины которые могут написать достаточно

сложную системную библиотеку? Тестировщики которые понимают систему с высоты в «10000 футов»? А может быть все эти ребята вмест

> Но когда нам нужно выкатить софт действительно быстро, четко и без ошибок, силами обычных администраторов (не подкючая разработчиков конкретного продута)

Когда программисты исключительно программируют, тестировщики тестируют а сисадмины выкатывают, это кажется не DevOps.

Кажется, DevOps — это когда процессы построены таким образом, что даже средний разработчик имеет возможность без проблем выкати свой код какой бы сложной не была система, при этом выкатить быстро, четко и без ошибок.
Потому что во первых, он понимает как оно все работает и что он делает, т.е. для него кусок его кода не является конем в вакууме.
Во вторых, он и его коллеги из эксплуатации, тестирования, ИБ и прочих отделов на славу потрудились все вместе чтобы создать такую сис
В третьих, нет ни каких технических, организационных и бюрократических преград, чтобы это сделать.
В четвертых, если что-то пойдет не так, не будет ни каких негативных последствий, потому что система построена таким образом, что заинтересованные лица сразу же узнают, что оно пошло не так и смогут это оперативно исправить.

Хотя конечно я могу заблуждаться.

olegchir 17.10.17 в 13:32

Внедрением девопса занимаются все. Но есть люди, более ответственные за это, чем все остальные. Наример, архитекторы, местные ан SRE, разработчики девопс-инструментария (в Сбертехе очень много своих внутренних разработок), написатели скриптов для координац CI/CD, и так далее. Есть специальные команды, ответственные за внедрение практик девопса.

В последнее время в России часто употребляют слово «девопс» в смысле должности. Это конечно, звучит несколько неграмотно, но зат позволяет не перечислять каждый раз длинный список этих «специальных особо ответственных». О ком конкретно речь в данном конкре случае — обычно можно понять по контексту.

MasMaX 17.10.17 в 13:51

Все верно — devops это в первую очередь отлаженные процессы. Но разработчики сами врядли будут с нуля разворачивать эти процесс все равно нужен человек который будет заниматься настройкой devops среды.

23derevo 17.10.17 в 11:50

Тут знакомые из СберТеха говорят, что твою статью воткнули во внутренний Confluence, потому что она хорошо саммаризует происходяще них. Мне кажется, парни должны тебе проставиться :)

olegchir 17.10.17 в 13:34

Да, пусть пишут мне в телеграм @olegchir :)

tgz 17.10.17 в 12:20

наверно первый раз читаю про девопс и не испытываю отвращения.

jbaruch 17.10.17 в 15:26

А это потому что про инструменты, а ты технарь. И в этом весь затык с внедренеем ДевОпса — технарей интересуют только инструменты Девопс это про культуру, а не про инструменты.

tgz 17.10.17 в 15:31

Инструменты как раз не интересуют вообще никак и статья не о них.

olegchir 19.10.17 в 11:30

Меня очень нервирует маркетинговый буллшит, поэтому стараюсь писать только по делу. В конце концов, за очковтирательство на Хабре платят, чтобы нести чушь еще и здесь. Вот есть вполне конкретная схема — ее и описал.

tgz 19.10.17 в 12:44

И надо отметить получилось неплохо.

dreka5 17.10.17 в 13:12

Чем больше pipeline. тем дольше путь от заявки до пользователя. с учетом PM etc.
В сентябре я на приложении сбербанк-онлайн обнаружил багу — сбрасывается пароль, приходится перерегистрировать приложение. бага критичная, но крайне не удобная, вызывает крайнее отторжение. И вот я допустим сообщил о ней, через какое время фикс окажется на продакшене?

olegchir 17.10.17 в 13:40

В данной статье рассматривается только девопс внутри ППРБ. ППРБ — это корная платформа банка, дело с которой имеют сотрудники б
А за публичные интерфейсы (и баги в нем) отвечает другая программа — ЕФС. У них даже есть блог на Хабре:
https://habrahabr.ru/company/efs, и они даже уже что-то начали выкладывать на Гитхаб: https://github.com/Sberbank-Technology. С какой
скоростью и как ЕФС отвечает на запросы о багах я, к сожалению, сказать не могу.

404 amarao 17.10.17 в 18:17

Любопытно, но в фейсбуке всего 17 тысяч человек работает, а у фейсбука и пользователей в десяток раз больше, и капитализация в десяток
больше, и девопса у них столько, что непродохнуть.

Почему? Наверное, потому что в фейсбуке пишут код, а сбербанке внедряют аджайл годами.

FyvaOldj 17.10.17 в 19:20

От того, что пользователь Фейсбук увидит на 1 лайк больше или меньше — ничего не произойдет. Ровным счетом ни-че-го. И фиксить эт
можно месяцами — ничего Фейсбуку не будет

404 amarao 17.10.17 в 19:31

Будет. Посмотрите с какой скоростью они реагируют на политические события.

FyvaOldj 19.10.17 в 18:35

А теперь пораскиньте все же мозгами — насколько более критичны ошибки в банках и насколько иная разработка. Принципиально
Даже странно что это приходиться жевать неглупому вроде ИТшнику

404 amarao 19.10.17 в 20:11

Попытался пораскинуть. Сбербанк потерял все базы и бэкапы. Убыток не более 66 ярдов. Facebook потерял все базы и все бэкап
убыток over 500 ярдов.

Получается, база фейсбука ценнее сбербанковской примерно в восемь раз.

FyvaOldj 20.10.17 в 05:01

Фейсбук и должен реагировать на политические события быстро. Это хороший хлеб. Но то что при этом будут сбоить счетчик
лайков — не страшно.

У банков не мржет быть такой избирательности.

olegchir 17.10.17 в 19:34

Хмм, давно у фейсбука ~15 тысяч отделений в 83 субьектах РФ, ~80 тысяч банкоматов, требующих присутствия реальных живых людей?
знал, ай да фейсбук

404 amarao 17.10.17 в 20:43

А это вопрос к вашему бизнесу — почему вам до сих пор нужны люди для обслуживания счетов. Специальный сотрудник, который
показывает как и в какой последовательности на банкомате нажимать кнопки (15 тысяч отделений, 15 тысяч рабочих мест — только чт
не сделать эргономичный интерфейс).

Гордиться числом сотрудников, на занятых автоматизацией, может только компания, которая не считает автоматизацию приоритетом.

olegchir 17.10.17 в 20:55

Вы, батенька, очень тонкий тролль, браво. Но статья не об этом.

ekho 17.10.17 в 21:03

Имхо, какой бы эргономичный интерфейс не был, он не поможет пожилому, зачастую неочень хорошо видящему человеку, к тому ж
теряющемуся при виде всех этих достижений прогресса, оплатить коммунальные платежи. С тем же успехом можно предъявить
претензии к Московскому Метрополитену за то, что они ввели должности помощников на территории метро (это такая бесплатная
услуга, можно позвонить по специальному номеру и к Вам придут один или несколько человек и помогут добраться от входа в метр
выхода на нужной Вам станции).

FyvaOldj 19.10.17 в 18:40

А бабушек и прочих которые с компьютерами не на ты — расстрелять предлагаете?

404 amarao 19.10.17 в 20:12

Я никого не хочу расстреливать, но если бабушка может пользоваться воцапом на телефоне и не может терминалом сбербанка,
это

а) заслуга гугля (эппла) и фейсбука
б) профессиональное несоответствие дизайнера терминала сбербанка

FyvaOldj 20.10.17 в 05:06

У противопоставляемого вами Сберу ФБ интерфейс — далеко не простой, кстати.

А остальные 99% бабушек что и СМС не умеют читать? Их куда?

chemtech 17.10.17 в 18:34

Добрый день! Рассматривали ли вы другие DevOps инструменты: TeamCity, YouTrack, Puppet? Используете ли вы ELK стек?

olegchir 17.10.17 в 19:51

Безусловно рассматривали разное, результат рассмотрения вы видите. В будущем он может измениться. Если вы хотите продать что-то, — точно не тот человек, к которому стоит обращаться с такими вопросами)

Ansible в контексте статьи лучше Puppet а) интеграцией в существующую Python-инфраструктуру б) наличием множества специалистов с хорошим знанием Python в) Отсутствием агента

Про ELK стек ничего не знаю, к сожалению. Это не значит, что какие-то проекты или программы его не используют — Сбт большой :)

Отмечу только, что в ППРБ существует своя собственная система мониторинга, в недрах которой кроме самописных решений используе еще и Splunk. О мониторинге будет одна из следующих статей об Сбт на Хабре, и скорей всего — доклад на JBreak/JPoint 2018. Вы о Splu спрашивали, но вдруг окажется полезным :)

chemtech 17.10.17 в 21:47

Спасибо за ответ. Я такой же DevOps)

986678 25.10.17 в 11:19

Прочитал статью и вопросов осталось куча. Можно я вывалю, а вы уж сами решайте на какие отвечать
1. Что такое ППРБ
2. Статья называется «Инструментальный стандарт», но я так понял, что это стандарт не СБТ, а только некоей системы ППРБ?
3. Можно все таки немного подробностей про «дополнительная система управления сквозным процессом развертывания»? Зачем эта сист Чем обусловлено ее возникновение?
4. Судя по фразе «Финальная сборка и публикация артефактов в хранилище. Например, для Java это будет Nexus,» под каждый тип использ свой инструмент хранения артефактов? Какие?
5. Деплой на стенды выполняется чем? Какой инструментарий используется? Что то свое писали или open source?
6. Тоже про тесты. Пользуетесь стандартными флоу или свои делали?

Вопросов еще много, но пока остановлюсь :)

olegchir 25.10.17 в 11:27

1. ППРБ — Платформа Поддержки Развития Бизнеса. Программа, в ходе которой разрабатывается бэкенд Банка

2. Да, это стандарт ППРБ, но скорей всего — он у всех скоро будет такой. ППРБ бежит впереди планеты всей.

3. Не могу) (на самом деле, потому что сказать особо нечего, по этой части я не спец)

4. Для хранения артефактов для передачи на прод, используется одна система хранения. Я подчеркнул про Nexus и Java, потому что стандартная реакция людей — начать ныть «вот вы банк, у вас много денег, вы можете организовать хранение, а мы — нет». Все это гни отмазки — берете уже существующий у вас Нексус (или Артифактори), суете туда артефакты с хэшсуммами, и уже какое-никакое хранил есть. Напоминаю, что в Нексус можно класть не только с помощью mvn deploy, а вообще любые файлы, в т.ч. с помощью API

5. Деплой на стенды стартуется с Jenkins, который передает управление на управляющую машину Ansible, котоаря запускает плейбуки, в значительной степени состоящие из самописных плагинов на Python. Таким образом, инструментарий: Python, Ansible, Jenkins.

6. Модульные тесты — стандартный флоу. Нагрузочное и UI — сильно специфичное для конкретного проекта.

САМОЕ ЧИТАЕМОЕ

Сутки | Неделя | Месяц

«Чтение на выходных»: 22 независимых блога о разработке, ИБ, тестировании и геймдеве

+11 8,7k 133 1

Как правильно оформить Open Source проект

+56 9,9k 281 37

Клонирование 50Gb базы из Prod в Dev за 1 секунду без потери целостности

+20 10,6k 72 30

Вашим пользователям не нужны пароли

+50 27,9k 162 447

Взлом Bitcoin по телевизору: обфускуй, не обфускуй, все равно получим QR

+59 8,9k 79 15

ИНТЕРЕСНЫЕ ПУБЛИКАЦИИ

«Иногда приходится заглядывать в код Spark»: Александр Морозов (SEMrush) об использовании Scala, Spark и ClickHouse

+7 119 1 0

Чем хорош (и чем плох) Typescript: опыт UI-разработчиков

+15 1,2k 9 4

Пошаговая настройка Graylog2

+7 541 15 3

Новинка от Aftershokz — обзор новой гарнитуры с костной проводимостью Trekz Air GT

+9 742 2 1

Создан первый молекулярный компьютер на синтетических полимерах

+7 1,6k 12 0