

DevOps CookBook

v2.08

Назначение

- Данная книга раскрывает понятие DevOps и его составляющие.
- Книга предназначена для ИТ-специалистов Банка
- ИТ-специалисты должны понимать подходы разработки Sbergile и СПП

Текущая версия книги не является окончательной, так как подразумевается непрерывное улучшение процессов. Конструктивная критика в виде предложений по улучшению ожидается и приветствуется.



Вы всегда можете написать рабочей группе DevOps по адресу devops@sberbank.ru



или задать вопрос по адресу Сигма: <https://sbtatlas.sigma.sbrf.ru/wiki/display/SDLC/DevOps>
Альфа: <http://confluence.ca.sbrf.ru/display/SDLC/DevOps>

а также подписаться на новостную страницу или скачать новые версии книги.

Содержание

Введение в DevOps	3
<u>Continuous Integration</u>	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
<u>Метрики</u>	80

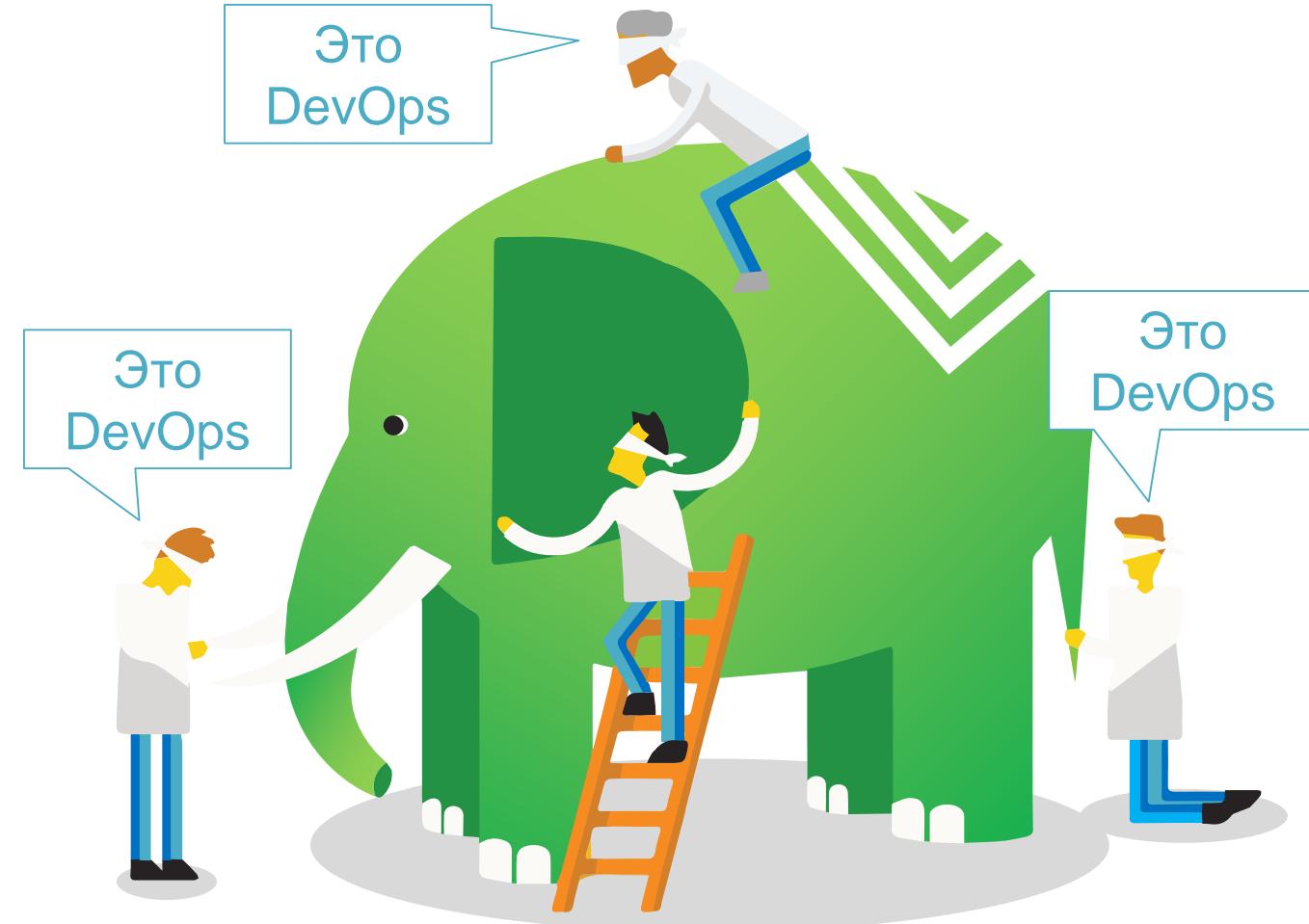
Что такое DevOps

Введение > CI > Среды > CD > CT > Метрики

DevOps* – это инженерная культура, практики и инструменты, направленные на сокращение релизного цикла, повышение эффективности и обеспечение возможности выпуска релиза в любой момент посредством:

- интеграции IT Development и IT Operations;
- создания единой команды;
- автоматизации рутинных операций;
- частой проверки кода

! ЦЕЛЬ: сократить T2M* и повысить гибкость внедрений (release anytime) без потери качества и надежности



DevOps – акроним от англ. *development* и *operations*

T2M (*time-to-market*) – срок от начала разработки продукта до его выхода на рынок

Заблуждения вокруг и около DevOps

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Некто из ИТ: А что, DevOps будет вместо Agile?

Некто из ИТ: DevOps и Waterfall несовместимы!

Менеджер ИТ: Мы в эти выходные не работаем, у вас будет время прийти и установить мне DevOps.

Dev: DevOps – это задача администраторов.

Dev: У меня установка на среды разработки идет по одной кнопке.
Я сделал DevOps!

Dev: Мне DevOps не нужен. Мы все гении разработки, а Вася делает сборку даже во сне.

Вопрос: Что будет, если количество задач увеличится, в вашу команду добавятся новые разработчики, а Вася уйдет в декрет?

Dev: Это проблема новых разработчиков, а Вася ...



Ops: Вы разработайте процесс развертывания и покажите, а мы выставим свои замечания.

Ops: У нас и так все хорошо и надежно.
Не нужны нам новые инструменты.

Вопрос: Что будет,
если не один релиз в
месяц, а два в неделю?

Ops: Срочно
увеличиваем штат!

Ops: Ну хорошо,
внедрили DevOps.
Где моя одна кнопка?

Менеджер ИТ: Надо плотно поработать три месяца,
сделаем DevOps и расслабимся

Вопрос: Вас не смущает слово *Continuous* в названиях практик DevOps?

Ops: Э-э-э ...



Основные компоненты DevOps

Введение > CI > Среды > CD > CT > Метрики



Только изменив **все четыре** составляющих, можно внедрить DevOps!

DevOps принципы

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Culture

- Развитие инженерной культуры – от разработчика к инженеру
- От ответственности за этап – к ответственности за результат

Automation

- Релиз в любое время
- Принятие изменений и уменьшение рисков

Measurement

- Непрерывное совершенствование основанное, на метриках и реальных данных

Sharing

- Коллаборация и открытая коммуникация
- Постоянная обратная связь «справа налево»

Основные практики DevOps, связанные в циклический автоматизированный процесс

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Непрерывное развертывание (Continuous Deployment)

Автоматическое развертывание на средах вплоть до промышленной

Непрерывная поставка (Continuous Delivery)

Автоматическое развертывание и тестирование

Непрерывная интеграция (CI*)

Автоматическая сборка, запуск модульных тестов и статического анализа



Кодирование



Модульное
тестирование



Инспекция кода,
статический
анализ кода
и ИБ



Сборка
и развертывание
на средах
разработки



Развертывание
на СТ



Тестирование
на СТ



Развертывание
на ИФТ, НТ, ИБ



Тестирование
на ИФТ, НТ, ИБ



Регистрация
в хранилище
дистрибутивов
Банка



Развертывание
и проведение
ПСИ /
Hotfix



Развертывание
на ПРОМ

- Повышение качества кода за счет:
 - о практик модульного тестирования (unit testing)
 - о инспекции кода (code review) и статического анализа кода на соответствие лучшим мировым стандартам кодирования и безопасности
- Сокращение сроков разработки за счет:
 - о ускорения цикла обратной связи
 - о увеличения частоты сборок (минимум 1 раз в день)

- Повышение качества тестирования за счет:
 - о внедрения практик автоматизированного тестирования - testing automation
 - о внедрения динамического анализа приложений на уязвимость
- Сокращение времени вывода в ПРОМ за счет:
 - о автоматизации развертывания дистрибутивов на среды
 - о динамического выделения инфраструктуры, infrastructure as code

- Повышение частоты вывода релизов в ПРОМ за счет:
 - о внедрения практик автоматизации развертывания

*Continuous Integration

Практики автоматизированного тестирования

Continuous Testing

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)



Виды автоматизированного тестирования

Что это такое?

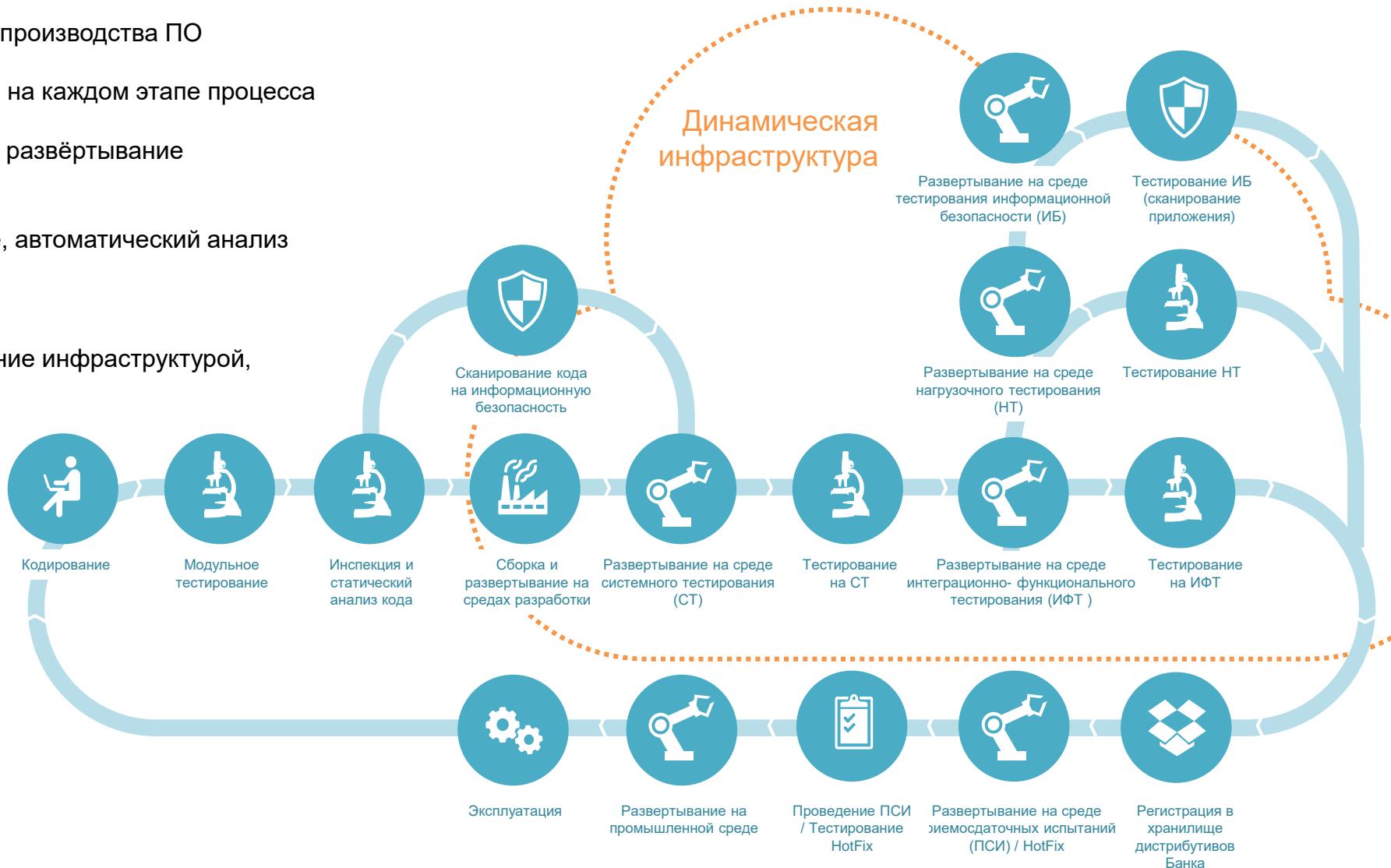
Кто разрабатывает и проводит тесты?

Модульное тестирование – unit testing	Проверка работоспособности части кода (функции, класса и т.д.). Тесты пишутся на языке программирования с использованием специальных фреймворков	Инженер Agile - команды
Статический анализ кода и ИБ	Проверка кода на соответствие международным стандартам языков программирования и написания безопасного кода с помощью специальных инструментов.	
Системное тестирование	Проверка работоспособности изолированной части функционала системы (без интеграции).	
Интеграционно-функциональное тестирование	Проверка работоспособности сквозных бизнес-операций между разными системами для нового и для ранее разработанного функционала.	
Нагрузочное тестирование	Проверка работоспособности системы при одновременной работе большого количества пользователей. Выполняется только для высоконагруженных систем и бизнес-операций	
Динамическое тестирование приложения на безопасность	Проверка системы на уязвимости (повышение привилегий, выполнение произвольного кода, sql injection,...) и «закладки»	Инженер тестирования ДК

Циклический автоматизированный процесс на средах Банка

- Бесшовный сквозной процесс производства ПО
- Максимальная автоматизация на каждом этапе процесса
- Автоматизированная сборка и развёртывание дистрибутива
- Автоматическое тестирование, автоматический анализ качества кода, проверки кода и приложений на ИБ
- Автоматизированное управление инфраструктурой, инфраструктура как код, динамическое выделение инфраструктуры из облака

Введение > CI > Среды > CD > CT > Метрики



Основные практики DevOps

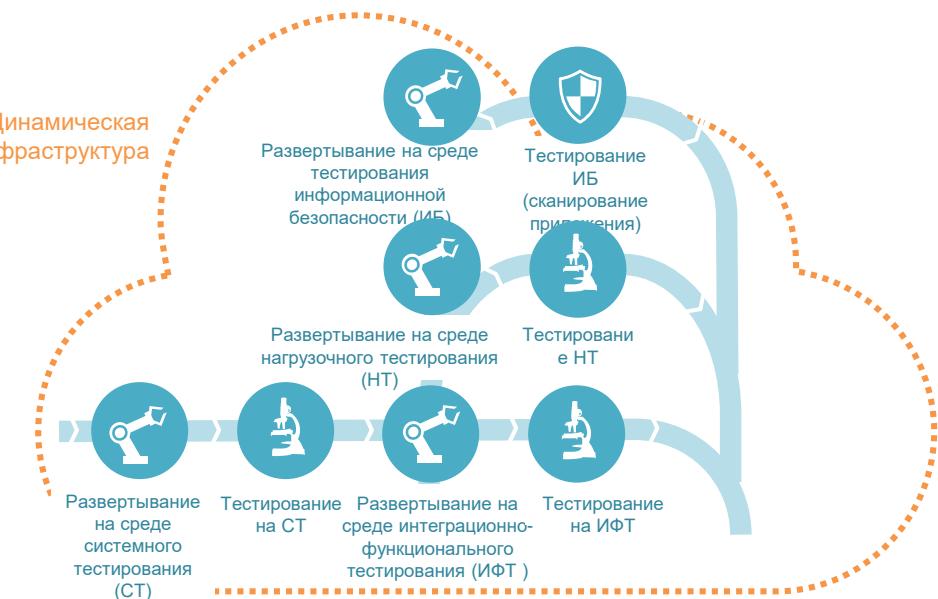
[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)



Непрерывная поставка (CD_L, Continuous delivery) – практика разработки программного обеспечения, гарантирующая то, что программное обеспечение постоянно готово к развертыванию в промышленную эксплуатацию. Практика включает в себя автоматизацию развертывания на тестовые среды, автоматизацию тестирования успешности развертывания на уровне администраторов (технические и функциональные тесты развертывания), а также интеграцию с процессами автоматизированного функционального, нагрузочного тестирования и динамического тестирования на безопасность.



Непрерывная интеграция (CI, Continuous Integration) – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок для скорейшего выявления ошибок. Это позволяет постоянно быть уверенным в том, что код находится в рабочем состоянии. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции изменений кода с существующей кодобазой и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий, сделать процесс разработки прозрачным для всех участников команды.



Непрерывное развертывание (CD_P, Continuous deployment) – практика разработки программного обеспечения, направленная на полную автоматизацию поставки от среды разработки в промышленную среду. Является дальнейшим развитием процесса Непрерывной поставки. В условиях Банка дополнительно затрагивает процесс регистрации в хранилище дистрибутивов Банка и автоматизацию приёмо-сдаточных испытаний. Характеризуется тесным взаимодействием разработки и эксплуатации.

Решаемые проблемы

Введение > CI > Среды > CD > CT > Метрики



Существующие проблемы производства

- 1** Из-за отсутствия общих стандартов ведения разработки (паттерны программирования, нотации и т.д.) и централизованного хранения кода случаются конфликты версий, пишется неоптимальный код, снижается эффективность разработки.
- 2** Разработка не выполняет интеграционные тесты, поэтому ошибки интеграции выявляются на поздних этапах – вплоть до ПСИ.
- 3** Ручное управление средами и развертыванием приводит к длительному ожиданию стендов командами (до нескольких месяцев) и увеличивает число ошибок при настройке стендов.
- 4** Большой объем ручного тестирования, отсутствие модульных и быстрых "smoke" тестов увеличивает время тестирования или сокращает объем тестового покрытия.
- 5** Проверка качества и безопасности кода при сборке не является обязательной. Поэтому в итоге можем получить небезопасный (с уязвимостями) и неоптимальный код (увеличение нагрузки на сопровождение).

Решение с помощью DevOps

 VC	Код хранится централизовано, любое изменение сохраняется с новой версией. Разработка ведется согласно стандартам, адаптированным к команде.
 AT	При каждой сборке выполняются быстрые smoke-тесты, проверяющие интеграцию "на заглушках" и эмуляторах, ошибки выявляются сразу при сборке.
 AD IaaS DI	Выделение среды происходит за минуты из облака Банка. Развертывание и конфигурирование автоматизировано.
 AT	Весь код покрыт модульными тестами, тестирование максимально автоматизировано.
 ACR PrCR IS JaaS	Код и готовое приложение проверяются на уязвимости и на качество – как в Apple и Google все события централизованно журналируются.

На рынке присутствуют десятки инструментов DevOps

[Введение > CI > Среды > CD > CT > Метрики](#)

1	Fm	Gh	Github	Scm	Database Mgmt	Build	Aws	Fm
3	Os	4	Pd	CI	Repo Mgmt	Testing	AmazonWeb Services	Pd
Gt	Git	Dm	DBmaestro	Deployment	Config / Provisioning	Containerization	Az	Az
11	Fm	12	Os	Cloud / IaaS / PaaS	Release Mgmt	Collaboration	Cloud	En
Bb	Bitbucket	Lb	Liquibase	BI / Monitoring	Logging	Security	Google Cloud Platform	En
19	Os	20	En	Maven	Gr	At	Gc	En
Gl	Rg	Redgate	Redgate	Gradle	ANT	FitNesse	Selenium	Os
37	Os	38	En	Gt	Gp	Br	Chef	Os
Sv	Dt	Datical	Datical	Grunt	Gulp	Broccoli	Puppet	En
55	Os	56	En	57	Fr	40	Ansible	Os
Hg	Dp	Delphix	Delphix	sbt	Mk	CMake	Salt	Os
73	En	74	En	75	Os	59	Docker	Os
Cw	Id	Idera	Idera	MSBuild	Rk	Pk	Compose	Os
91	En	92	En	93	En	94	En	En
Xlr	XL Release	Ur	UrbanCode Release	Bm	BMC Release Process	Hp	HP Cedar	En
106	Os	107	Fm	108	Os	109	Os	En
Ki	Kibana	Nr	New Relic	Ni	Nagios	Zb	Zabbix	En
110	En	111	Os	112	Os	113	En	En
Dd	Datadog	EI	Elasticsearch	Ss	StackState	Sp	Splunk	En
114	Fm	115	Fm	116	Fm	117	Os	Os
Le	Logentries	Sl	Sumo Logic	Ls	Logstash	Gr	Graylog	Os
118	Os	119	Os	120	En	Sn	Snort	En
Tr	Tripwire	Ff	Fortify					

PERIODIC TABLE OF DEVOPS TOOLS

Periodic Table of DevOps Tools is a visual representation of the DevOps ecosystem, organized into a grid of tools categorized by their primary function and availability.

Legend:

- Open Source (Os):** Indicated by a grey square.
- Free (Fr):** Indicated by a light blue square.
- Freemium (Fm):** Indicated by a light green square.
- Paid (Pd):** Indicated by an orange square.
- Enterprise (En):** Indicated by a dark grey square.

Category Legend:

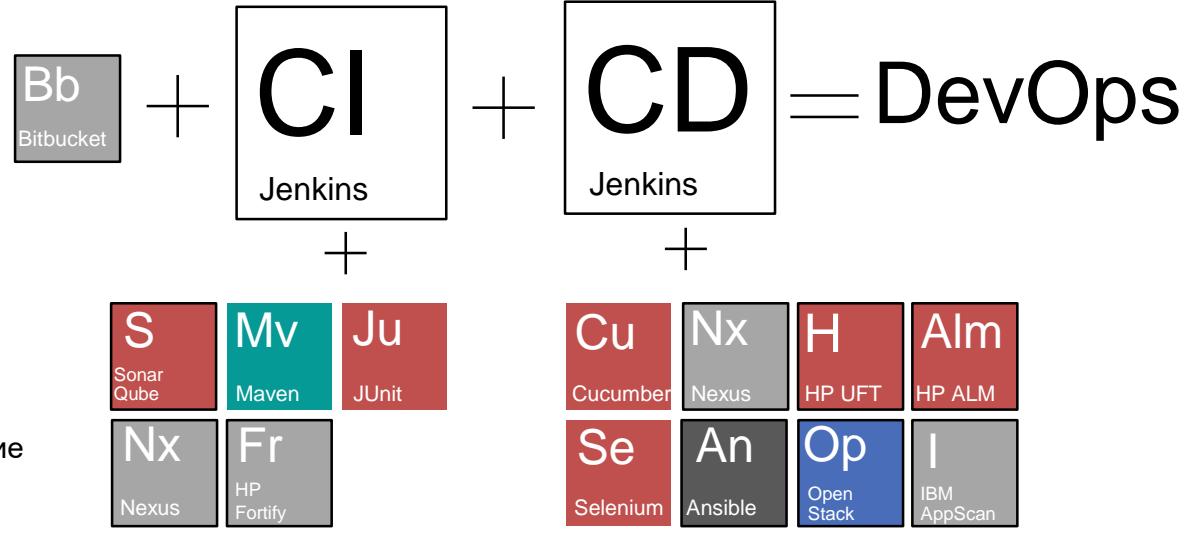
- SCM:** Version Control Management (Yellow)
- Deployment:** Deployment Automation (Green)
- Cloud / IaaS / PaaS:** Cloud Infrastructure as a Service (Dark Blue)
- BI / Monitoring:** Business Intelligence / Monitoring (Grey)
- CI:** Continuous Integration (Dark Blue)
- Repo Mgmt:** Repository Management (Dark Orange)
- Config / Provisioning:** Configuration Management (Grey)
- Release Mgmt:** Release Management (Purple)
- Containerization:** Container Management (Dark Purple)
- Collaboration:** Collaboration Tools (Light Blue)
- Logging:** Logging and Monitoring (Light Green)
- Database Mgmt:** Database Management (Orange)
- Build:** Build Automation (Teal)

Инструменты: утвержденные и выбираемые командами

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Утвержденные инструменты:

- **Atlassian Bitbucket** – система версионного хранения кода
- **Maven** – инструмент авто-сборки
- **JUnit** – библиотека для модульного тестирования Java приложений
- **Jenkins** – инструмент управления процессом сборки, развертывания и настройки
- **Sonar, HP Fortify, IBM AppScan** – проверка кода на качество и уязвимости
- **HP ALM** – инструмент управления процессом тестирования
- **HP UFT, Selenium, Cucumber, HP PC** – автотесты, нагружочное тестирование
- **Ansible** – управление конфигурациями и настройкой сред
- **Nexus** – управление хранением дистрибутивов
- **OpenStack** – виртуализация инфраструктуры, частное облако



- Целевой инструмент, утвержденный / рекомендованный Архсоветом

Инструменты, выбираемые командами:

Однако существуют инструменты, тесно связанные с определенной технологией. Такие инструменты не утверждаются на архитектурном совете, их выбор осуществляется рабочей группой:

- Командой разработки
- ЦЭ процессов производства (ЦЭ DevOps)
- Командой сервиса CI или CD
- Прикладными и системными администраторами

Почему нам важно иметь стандарты в выборе инструментов DevOps?

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

- Централизованное сопровождение инструментов позволяет командам сосредоточиться на разработке
- Предоставление централизованного сервиса 24x7
- Объективный выбор инструментов с учетом критериев стоимости (в т.ч. стоимость эксплуатации), эффективности и надежности и соответствие инструментов принципам прозрачности
- Наличие множества инструментов*, выполняющих одинаковую функцию, создает «бутылочное горлышко» на стороне Ops, что противоречит принципам DevOps.
- К централизованным инструментам прилагается централизованный мониторинг, который позволяет проследить весь путь релиза от разработки до ПРОМа.

*Проблема множества инструментов усугубляется тем, что в эксплуатации могут находиться несколько версий одного и того же инструмента

DevOps Toolchain

Введение > CI > Среды > CD > CT > Метрики

Непрерывная интеграция

1. Кодирование



Разработчик программирует код и пишет модульные тесты к этому коду

Инструменты: Bitbucket JUnit



2. Управление сборкой



После сохранения кода в **Bitbucket** запускается workflow процесса сборки в **Jenkins**. **Jenkins** запускает модульные тесты, статический анализ качества кода и ИБ. **Ansible** производит автоматизированное развертывание дистрибутива на среду разработки.

Инструменты: Nexus Bitbucket

Инструменты сборки
 Jenkins Ansible FORTIFY sonarQube

Непрерывная поставка

3. Подготовка среды



После сборки дистрибутива запускается workflow процесса развертывания в **Jenkins**, происходит (опционально) выделение тестовых сред, подготавливаются тестовые наборы данных, осуществляется автоматическое конфигурирование сред через **Ansible**.

Инструменты: Jenkins ANSIBLE Nexus

4. Управление развертыванием



Ansible развертывает дистрибутив на тестовые среды, выполняются технические и функциональные тесты развертывания

Инструменты: Jenkins ANSIBLE Nexus

5. Управление тестированием



Jenkins вызывает **HP ALM** и запускается набор тестов, как автоматизированных (API, GUI, интеграционных, безопасности и т.д.), так и «ручных»

Инструменты: ALM Unified Functional Testing (UFT) AppScan IBM Security

Мониторинг и обратная связь



Механизм оповещений и мониторинга гарантирует быстрый цикл обратной связи. Доступен для всех этапов

Инструменты: elastic QlikView

Непрерывное развертывание

6. ПСИ



В случае успешного прохождения испытаний принимается решение (человеком) о развертывании дистрибутива на ПРОД среду

7. Развертывание на ПРОД



Происходит автоматизированное развертывание дистрибутива на ПРОД среду. В случае ошибок – выполняется автоматический откат (Rollback)

Инструменты: Jenkins ANSIBLE Nexus

Новая роль – DevOps инженер

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

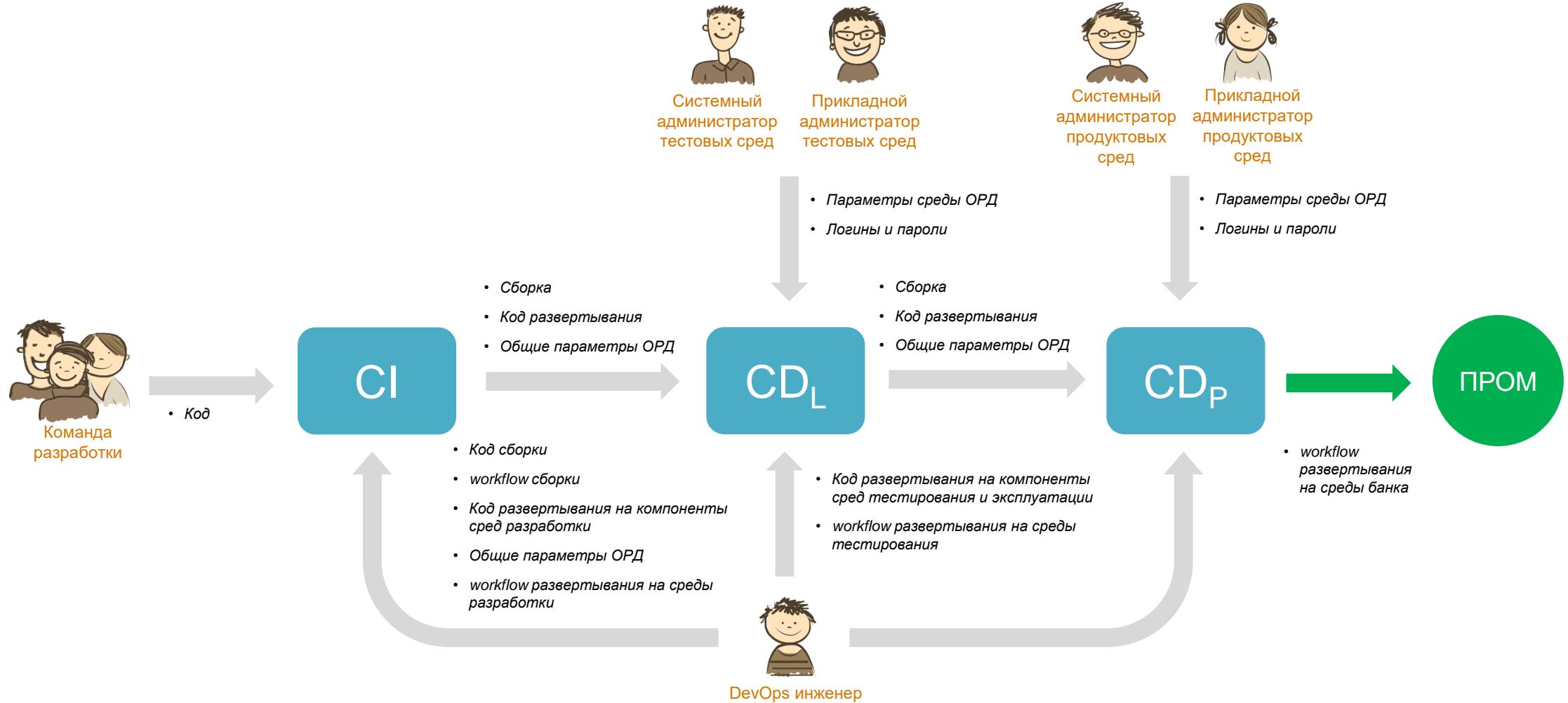


DevOps инженер – является экспертом по практикам DevOps, владеет инструментами CI и CD, выстраивает Pipeline сборки и развертывания.

- Настраивает процесс CI на централизованном сервисе, интегрирует его с GitFlow и кодом развертывания на средах разработки.
- Разрабатывает процесс поставки ПО на целевых инструментах CD для всех сред (разработки, тестирования и эксплуатации).
- Обеспечивает единообразие процессов поставки и выстраивания их в единый бесшовный процесс поставки.
- Управляет и обеспечивает процесс распространения конфигураций тестовых сред для сред разработки и тестирования (за границами обязанностей среды Банка).
- Оказывает сопровождение прикладным и системным администраторам сред Банка
- Владеет инструментами централизованного сервиса CI и CD, а также командными инструментами, в т.ч. по юнит-тестированию и сборки.
- Обладает знаниями технической и интеграционной архитектуры ОРД и механизмов переносов настроек между средами.
- Обладает базовыми навыками администрирования прикладного и системного ПО ОРД.

Работа DevOps Toolchain

Введение > CI > Среды > CD > CT > Метрики



Содержание

<u>Введение в DevOps</u>	3
Continuous Integration	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
<u>Метрики</u>	80

Практика Continuous Integration

Введение > CI > Среды > CD > CT > Метрики

Непрерывная интеграция (CI, Continuous Integration) – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок для скорейшего выявления ошибок. Это позволяет постоянно быть уверенным в том, что код находится в рабочем состоянии. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции изменений кода с существующей кодобазой и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий, сделать процесс разработки прозрачным для всех участников команды.

Некто из ИТ: В DevOps есть практики Continuous Integration. Видимо будут менять корпоративную шину.



Практика включает в себя:

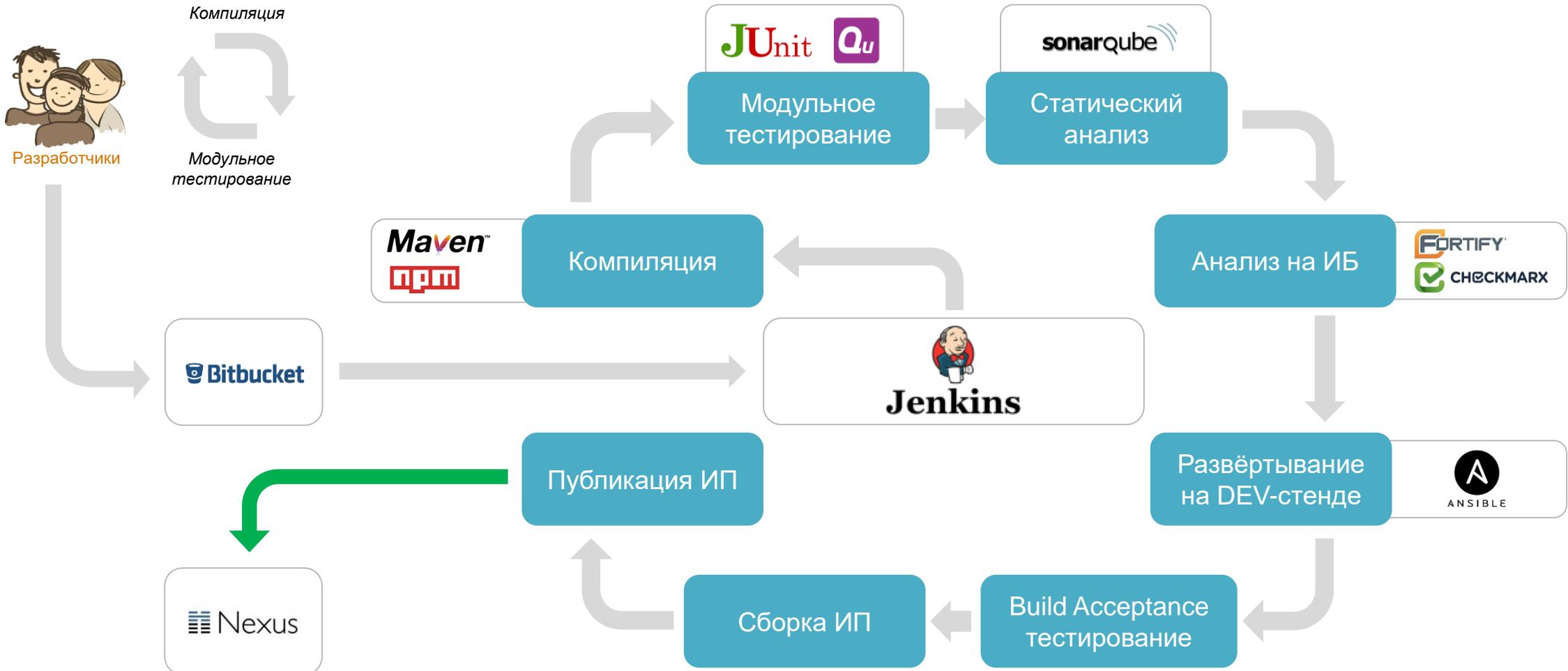
- хранение кода в системе контроля версий
- ручное code review
- частые автоматизированные сборки
- автоматизированное модульное тестирование (unit tests)
- статическую проверку качества кода
- анализ кода на безопасность*
- автоматизированное развертывание на среды разработки
- проверки работоспособности системы после развертывания (Sanity checks, Health checks, Smoke tests)
- автоматизированные Build Acceptance Tests на стендах разработки



*Подключается согласно плану подключения к программе Кибербезопасности

Continuous Integration Workflow

Введение > CI > Среды > CD > CT > Метрики



- При ошибке на любом шаге цикл прерывается с уведомлением
- В зависимости от используемых в проекте технологий порядок выполнения шагов может незначительно меняться

Инструменты CI

Введение > CI > Среды > CD > CT > Метрики



HPE Security Fortify
Static Code Analyzer



Jenkins

Инструменты **модульного тестирования**, генерирующие отчёты о прохождении тестов и покрытии кода тестами в xml-формате. Например, [JUnit](#) для Java, [NUnit](#) для .Net, [QUnit](#) для JavaScript.

Сборка и управление зависимостями. Например, [Maven](#) для Java, [nuget](#) для .Net, [npm](#), [bower](#), [gulp](#) для javascript, css и статики

Статический анализ качества кода – [SonarQube](#)

Статический анализ кода на безопасность – [HPE Security Fortify Static Code Analyzer](#) или [Checkmarx](#)

Хранилище артефактов и ИП – [Sonatype Nexus OSS](#)

Оркестратор процесса CI – [Jenkins](#)

- Запуск планов сборки по событию (коммит, пул-реквест)
- Интеграция со всеми вышеописанными инструментами
- Управление потоком сборки

Система контроля версий

Введение > CI > Среды > CD > CT > Метрики

Система управления версиями (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Для чего нужно: история изменений, откат нежелательных изменений, совместная работа, код не теряется, нерабочие фичи не ломают основной билд

Для чего применять VCS:

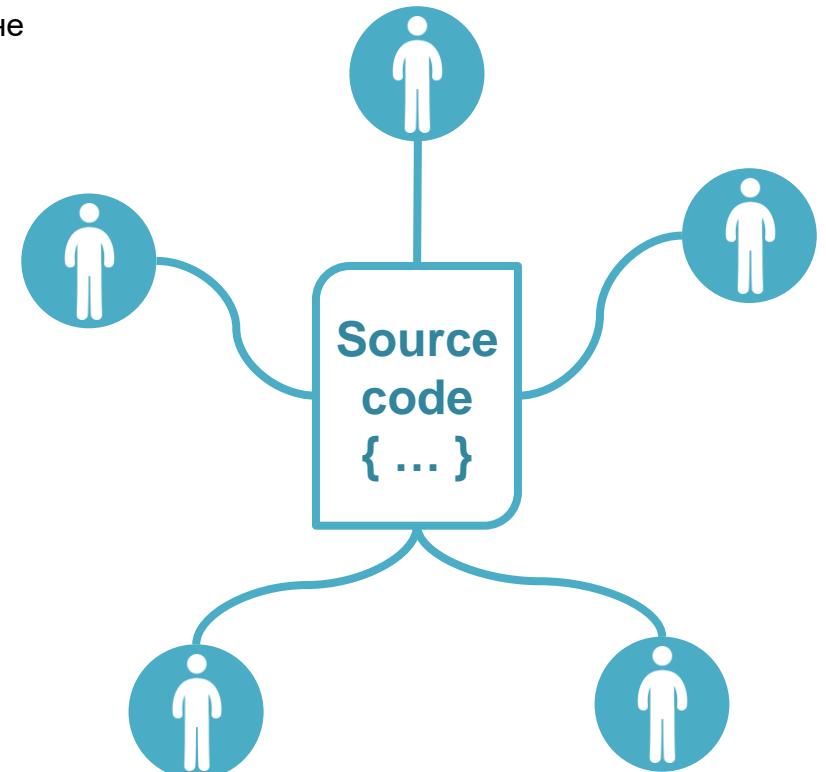
- для любых текстовых файлов: исходники ПО, README (в простом текстовом формате), html и различные скрипты

Для чего не применять VCS:

- для любых двоичных файлов (дистрибутивы, документы в формате doc, xls и прочее, архивы, скомпилированные сущности) - для их версионирования часто имеются специальные инструменты,
- автоматически генерированные файлы - генерированные классы веб-сервисов, файлы проекта (за некоторым исключением) - данные файлы не требуют контроля версий

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pageId=44502656>

<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>



Модели ветвления

Введение > CI > Среды > CD > CT > Метрики

Существуют различные модели ветвления, каждая из которых преследует свои цели.

На данный момент наиболее актуальны и популярные модели ветвления – это:

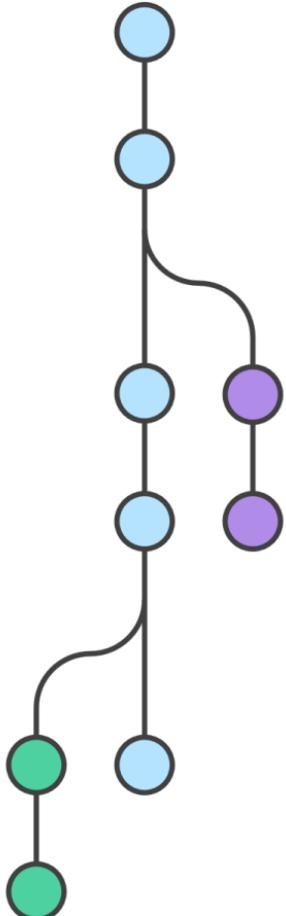
- **Git flow**
- **Github flow**
- **Gitlab flow**



ВАЖНО: Модели ветвления работают только при условии тщательного соблюдении схемы.

Базовые принципы популярных моделей ветвления

- Любое значимое изменение должно оформляться как отдельная ветвь
- Текущая версия главной ветви всегда корректна. В любой момент сборка проекта, проведённая из текущей версии, должна быть успешной
- Версии проекта помечаются тегами. Выделенная и помеченная тегом версия более никогда не изменяется
- Любые рабочие, тестовые или демонстрационные версии проекта собираются только из репозитория системы



Модели ветвления: Git Flow

Введение > CI > Среды > CD > CT > Метрики

Данная модель является практической реализацией основных принципов разработки ПО в VCS с учетом релизных циклов ПО. Подходит как для Scrum, так и для Waterfall.

Принципы

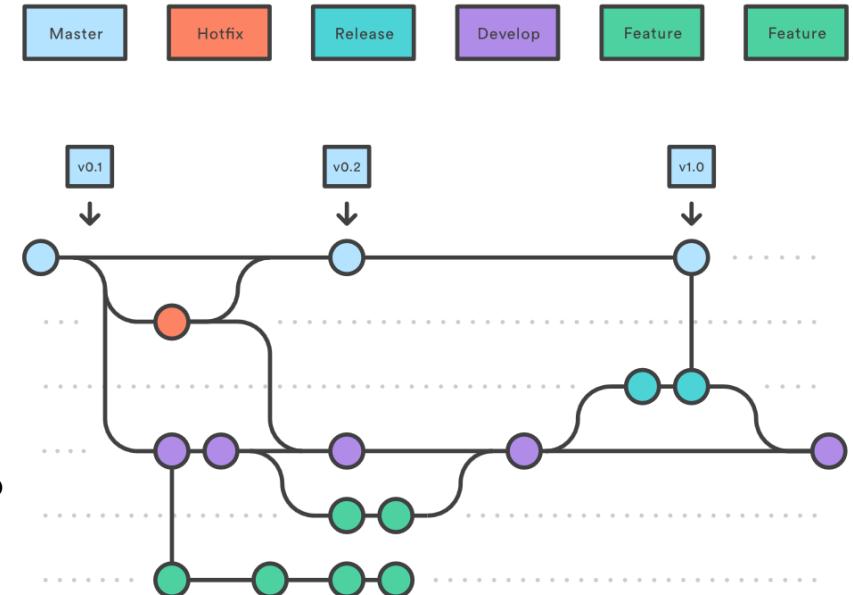
- Две главные ветки:
 - master – текущая стабильная версия, работающая на пром-среде
 - develop – основная ветка разработки
- Вспомогательные ветки:
 - feature – новый функционал или исправление некритичных багов; сливаются в develop
 - release – фиксация и стабилизация релиза (feature-freeze); сливаются в master и develop
 - hotfix – исправление критичных багов; сливаются в master и develop
- Все слияния происходят только через pull-request'ы

Плюсы

- Фиксация и стабилизация функционала релиза до попадания на пром
- Возможность выпускать ночные сборки (nightly build)

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pageId=65257821>

<https://habrahabr.ru/post/106912/>



Минусы

- Не самая простая схема работы; требует досконального понимания от разработчика, какие ветки откуда создаются и куда сливаются
- Готовый функционал попадает на пром с некоторой задержкой

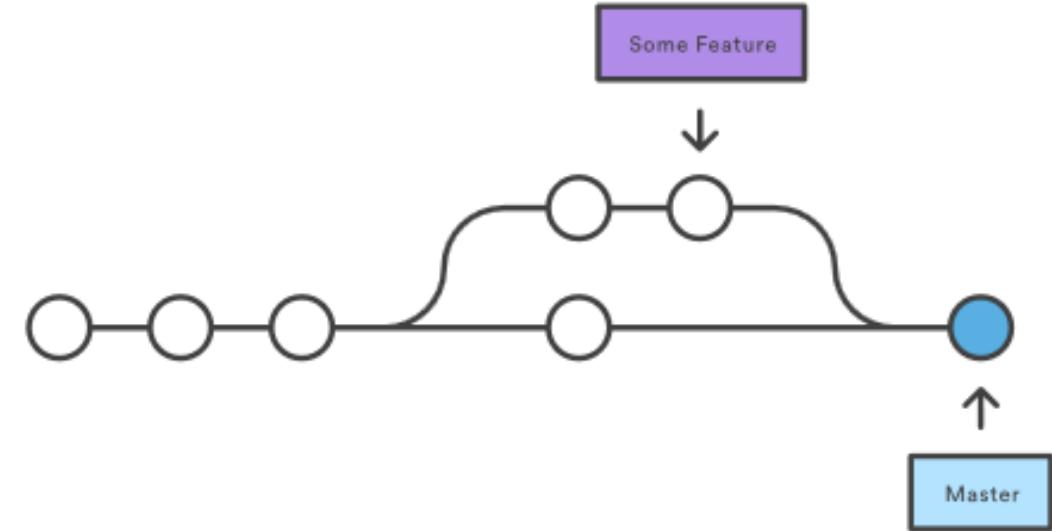
Модели ветвления: Github Flow

Введение > CI > Среды > CD > CT > Метрики

Github Flow – модель ветвления, являющаяся практической реализацией основных принципов разработки ПО в VCS без учета релизных циклов ПО. Подходит для методологии Kanban.

Принципы

- Одна главная ветка **master** – текущая стабильная версия
- Вспомогательные **feature** или **hotfix** ветки ответвляются от **master** и сливаются обратно в неё через pull-request
- Все слияния происходят только через pull-request'ы



Плюсы

- Готовый функционал не «отлёживается» в релизах, а сразу же отправляется на пром
- Простая и прозрачная схема работы для разработчика

Минусы

- Требует крайне высокой культуры разработки и сопровождения
- Требует максимальной автоматизации процессов тестирования и доставки

<https://guides.github.com/introduction/flow/>

<https://habrahabr.ru/post/189046/>

Модели ветвления: Gitlab Flow

Введение > CI > Среды > CD > CT > Метрики

Gitlab Flow – модель, являющаяся некоторым симбиозом Git Flow и Github Flow. Она также проста как и Github Flow, но в то же время позволяет контролировать релиз как Git Flow.

Принципы

- Основная ветка **master** – текущая стабильная версия
- Вспомогательные **feature** или **hotfix** ветки ответвляются от master и сливаются обратно в неё через pull-request
- Стабильная ветка **production** для автоматического деплоя на пром, в которую переносится код из master в тот момент, когда нужно выложить
- Возможно использование дополнительных веток для различных сред
- Все слияния происходят только через pull-request'ы

Плюсы

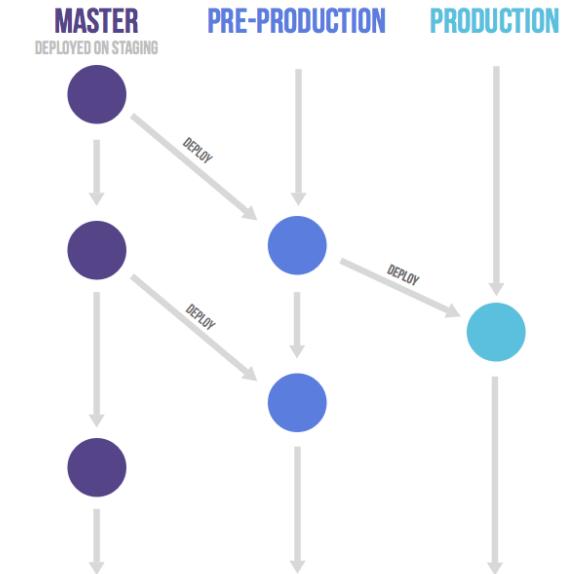
- Простая и прозрачная схема работы для разработчика
- Готовый функционал сразу же готов к отправке на пром

Минусы

- Требует крайне высокой культуры разработки и сопровождения
- Требует максимальной автоматизации процессов тестирования и доставки

<https://habrahabr.ru/company/softmart/blog/316686/>

<https://about.gitlab.com/2014/09/29/gitlab-flow/>



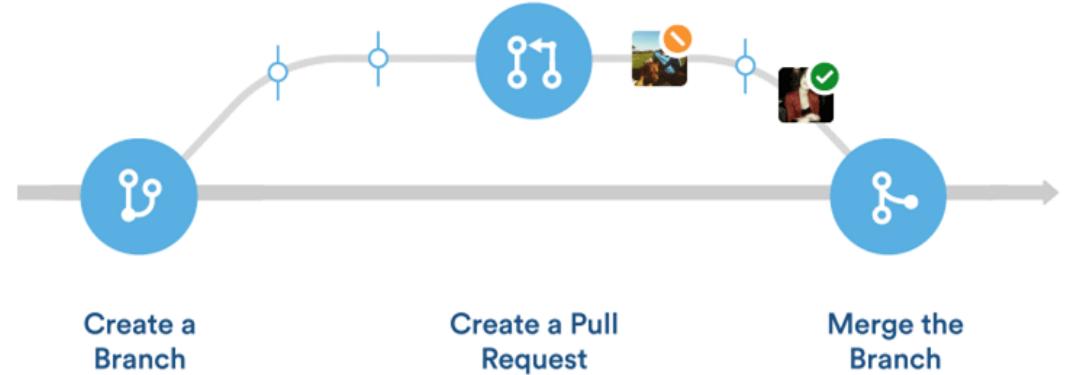
Запросы на слияние (Pull-Request)

Введение > CI > Среды > CD > CT > Метрики

Запрос на слияние (Pull-Request) – механизм системы контроля версий, позволяющий оформить изменения из ветки в виде предложения к слиянию в основную (или какую-то иную) ветку репозитория.

Что даёт

- Описание предлагаемого изменения видно в интерфейсе системы контроля версий всем заинтересованным участникам
- Возможность провести code review и оставить комментарии ещё до включения изменений в целевую ветку
- Возможность не допустить слияния, пока не будут выполнены все необходимые условия. Например:
 - Минимальное количество подтверждений от участников, проводящих реview
 - Успешно прошедшая сборка в системе CI
 - Отсутствие критических замечаний по результатам автоматического статического анализа



Сборка

Введение > CI > Среды > CD > CT > Метрики

Существует два типа сборки:

1. **Сборка ПО** (англ. Software Build) - процесс преобразования файлов с исходным кодом и их компиляция в артефакты, составляющие приложение такие, как бинарные и исполняемые файлы. Для сложных программ после компиляции происходит процесс связывания (линовка), упаковка бинарных/исполняемых файлов и сопутствующих ресурсов, требуемых для работы ПО, в дистрибутив.

2. **Подготовка инсталляционного пакета (ИП)**

Инсталляционный пакет – это совокупность:

- Дистрибутива
- Пользовательской и эксплуатационной документации

ИП создаётся единожды, публикуется в централизованное хранилище Nexus и дальше "путешествует" по трубе DevOps в неизменном виде от сред разработки до сред промышленной эксплуатации.

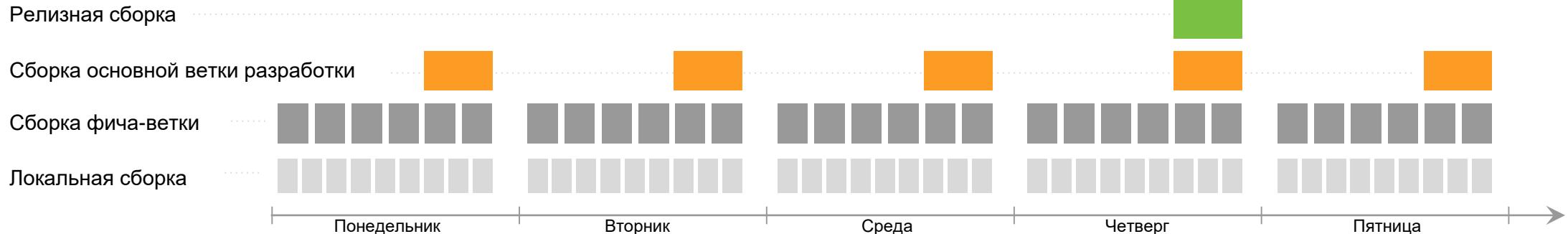
Неизменность ИП гарантируется невозможностью изменения файлов и хранением контрольных сумм всех загруженных ИП в Nexus.

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pageId=50599474>



Виды сборок в процессе CI

Введение > CI > Среды > CD > CT > Метрики



Локальная сборка

Программисты запускают локальные сборки в их локальном окружении для интеграции своих изменений с последней версией исходных кодов из VCS. Локальные сборки позволяют сократить в последующем количество неудачных централизованных сборок. Локальные сборки обязательно должны включать в себя запуск юнит тестов. Если ресурсов достаточно, локальные сборки могут также включать статический анализ кода, тем самым позволяя разработчикам решать разные проблемы ещё до публикации кода в VCS.

Сборка фича-ветки

Сборка фича- и багфикс- веток (или интеграционная сборка), которая запускается на сборочном сервере, интегрирует и тестирует код из ветки с последними изменениями в основной ветке разработки. Такие сборки должны быть частыми, что обеспечит оперативную обратную связь команде разработчиков. Интеграционная сборка выполняет "smoke тесты" (подмножество всех тестов), а также запускает инструменты инспекции кода, что обеспечивает хорошее представление о качестве системы в целом.

Сборка основной ветки разработки

Сборка основной ветки разработки запускает расширенный набор тестов и инспекций кода. Такую сборку нет необходимости запускать сразу непосредственно после попадания изменений в основную ветку разработки, так как она не такая быстрая, как сборка фича-ветки. Её можно запускать регулярно, например, один раз в день. Цель этой сборки – постоянно быть уверенным, что приложение функционирует корректно, и в любой момент можно делать релизную сборку.

Релизная сборка

Релизная сборка выполняется ещё реже. Как правило по требованию. Эта сборка подготавливает приложение для передачи приложения на среды СТ, ИФТ, НТ, ИБ, ПСИ и в конце концов ПРОМ.

<https://sbtatlas.sigma.sbrf.ru/wiki/x/XchHjAw>

Модульное тестирование

Введение > CI > Среды > CD > CT > Метрики

Модульное тестирование, или юнит-тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Модульное тестирование позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже протестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Этот вид тестирования выполняется самими программистами. В процессе написания модульных тестов разработчик:

- делает аудит логики основного кода
- автоматически проверяет модульность своего кода

Модульное тестирование практически невозможно в условиях некачественного исходного кода. Таким образом, оно подталкивает программистов к использованию лучших практик программирования.

Источники

"Goto Fail, Heartbleed, and Unit Testing Culture" by Martin Fowler.

http://ru.wikipedia.org/wiki/Модульное_тестирование

<http://www.protesting.ru/testing/levels/component.html>

http://citforum.ru/SE/testing/unit_testing/

<http://blog.openquality.ru/unit-tests-why>

QA: Unit тесты – это тестирование отдельного модуля АС.

DEV: Я пишу код, а мой падаван пишет к нему юнит-тесты

DEV: Написание тестов отнимает время.

DEV: Это работает, только если писать проект с нуля.



REAL SYSTEM



Green = class in focus
Yellow = dependencies
Grey = other unrelated classes

CLASS IN UNIT TEST



Green = class in focus
Yellow = mocks for the unit test

Модульное тестирование. Ограничение

Введение > CI > Среды > CD > CT > Метрики

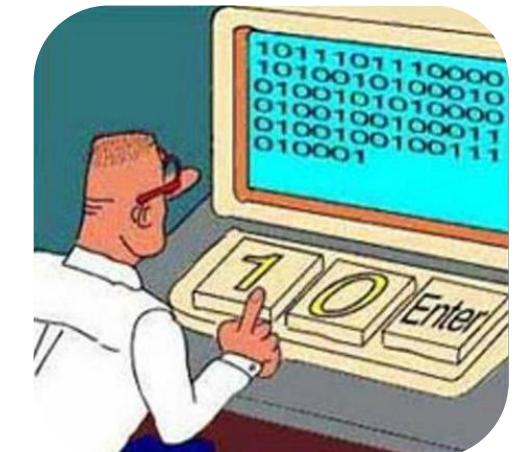
Не нужно писать тесты, если

- Вы пишите тесты на сложный и сильно связный код (об этом вам может подсказать инструмент статического анализа кода). Сначала нужно сделать рефакторинг кода
- Вы делаете прототип для демонстрации концепции. После демонстрации прототипа вы не планируете использовать созданный программный код
- Вы всегда пишете код без ошибок, обладаете идеальной памятью и даром предвидения. Ваш код настолько крут, что изменяет себя сам, вслед за требованиями заказчика. Иногда код объясняет заказчику, что его требования не нужно реализовывать

<http://eax.me/unit-testing/>

<https://habrahabr.ru/post/169381/>

Наличие юнит тестов дополняет, а не отменят другие виды тестирования.



Test-driven development (TDD, Разработка через тестирование)

Введение > CI > Среды > CD > CT > Метрики

Один из наиболее эффективных подходов к модульному тестированию - это **подготовка автоматизированных тестов** до начала основного кодирования (разработки) программного обеспечения (**test first approach**) .

Это называется разработка от тестирования (**test-driven development**) или подход тестирования вначале.

При этом подходе создаются и интегрируются небольшие куски кода, для которых запускаются тесты, написанные до начала кодирования.

Разработка ведётся до тех пор, пока все тесты не будут успешно пройдены.

² ["Monogomous TDD,"](#) 8th Light.

⁴ ["How Do We Learn?"](#) Ron Jeffries.

⁵ ["Gold Plating \(Software Engineering\),"](#) Wikipedia.

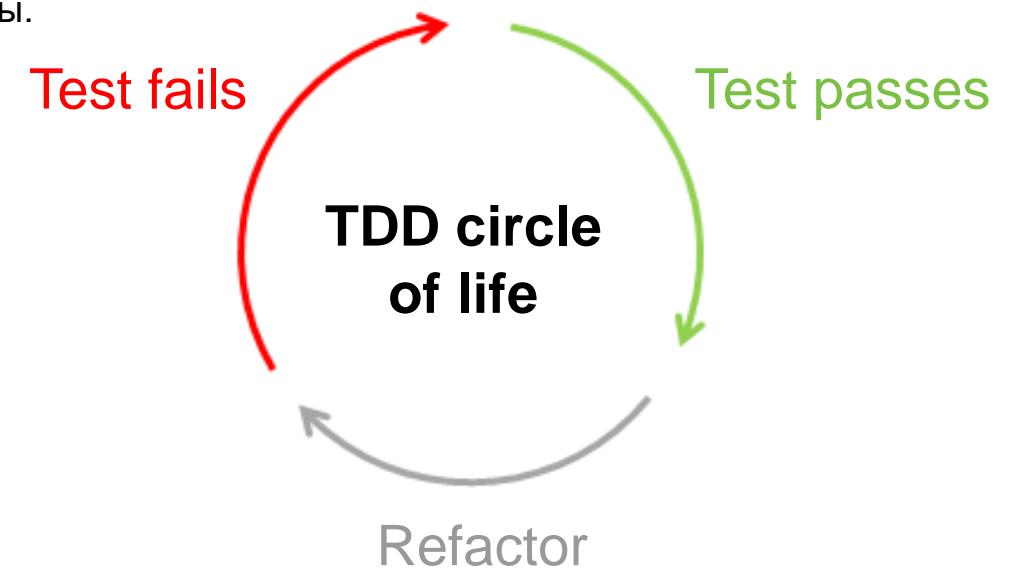
⁶ ["Notes on Structured Programming,"](#) Eindhoven University of Technology.

⁷ ["Top 5 TDD Mistakes,"](#) Telerik by Progress.

⁸ K. Beck. "Extreme Programming Explained: Embrace Change." Addison-Wesley. 2000.

⁹ ["Goto Fail, Heartbleed, and Unit Testing Culture,"](#) Martin Fowler.

¹⁰ ["Refactoring: Improving the Design of Existing Code,"](#) Martin Fowler.



Статический анализ качества кода

Введение > CI > Среды > CD > CT > Метрики

Статический анализ кода - это процесс выявления ошибок и недочётов в исходном коде программ. Статический анализ можно рассматривать как автоматизированный процесс инспекции кода. Он выполняется без исполнения исследуемого кода.

Статический анализ кода решает 3 задачи:

- Выявление **ошибок** и недочётов в программах.
- Проверка кода на соответствие рекомендациям по **оформлению кода** (применение отступов в различных конструкциях, использование пробелов/символов табуляции, регистра букв и так далее).
- **Подсчет метрик** качества кода (связность объектов, дублирование и так далее).

https://ru.wikipedia.org/wiki/Стандарт_оформления_кода

<https://m.habrahabr.ru/company/pvs-studio/blog/327388/>

https://ru.wikipedia.org/wiki/Метрика_программного_обеспечения

<https://msdn.microsoft.com/ru-ru/library/bb385914.aspx>

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pagId=43238082>



Статический анализ качества кода.

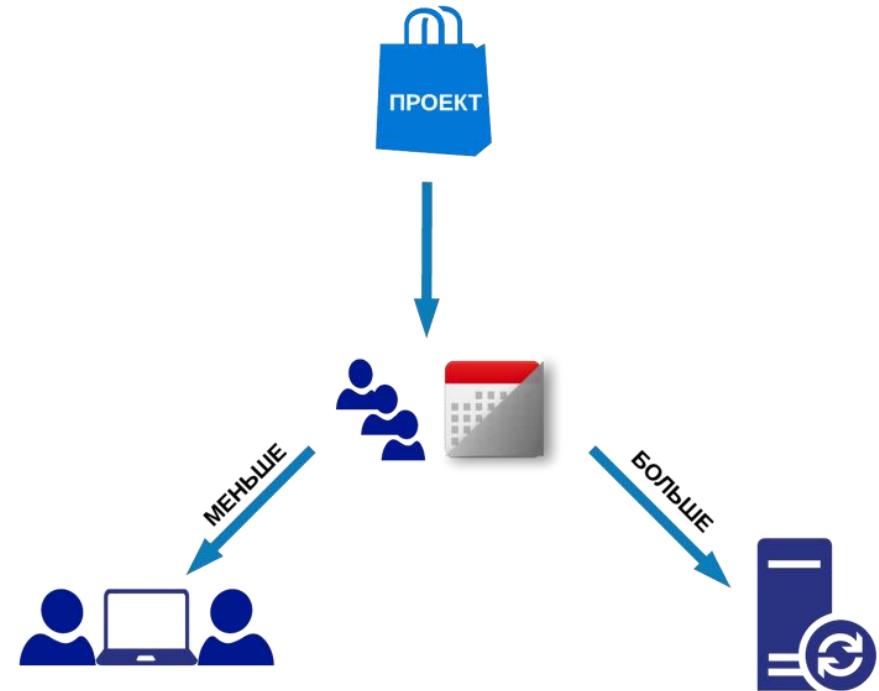
Ограничения

Введение > CI > Среды > CD > CT > Метрики

Результат работы статического анализатора — это список обнаруженных в коде потенциальных проблем с указанием имени файла и конкретной строки. Другими словами, это список ошибок, очень похожий на тот, что выдает компилятор. Термин «потенциальные проблемы» используется здесь не случайно. К сожалению, статический анализатор не может абсолютно точно сказать, является ли эта потенциальная ошибка в коде реальной проблемой. Это может знать только программист.

Статический анализ кода **целесообразно** применять не во всех проектах. Эффект от статического анализа будет очевиден на проектах примерно **от 3х человек и от полугода** **года** длительностью. Если программный проект меньше указанного размера, то вместо использования статического анализа выгоднее иметь в проекте нескольких квалифицированных разработчиков.

<https://habrahabr.ru/company/intel/blog/103776/>



Статический анализ качества кода.

Практика Джона Кармака

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

«Самой важной вещью, которую я сделал как программист за последние годы — это начал агрессивно применять статический анализ кода. Не так важны сотни серьёзных багов, которые удалось предотвратить, сколько смена ментальности и моего отношения к надёжности ПО и качеству кода. Я всегда старался писать хороший код, поскольку одной из моих внутренних мотиваций было то, что как мастер своего дела я постоянно должен совершенствоваться.»

Джон Кармак в течение нескольких месяцев работал, постепенно исправляя найденные баги: сначала свой код, потом другой системный код, потом игровую часть. Эффект был колossalным, нашлись очень важные ошибки, о которых в своё время попросту забыли, а также новые.

В конце концов, Джон Кармак сделал несколько выводов, которые он советует понять и другим разработчикам:

- каждый должен признать, что в его коде существует множество ошибок. Это горькая пилюля, которую требуется проглотить каждому программисту.
- ревизия кода должна быть обязательно автоматизирована. Приятно видеть ошибочные срабатывания автоматических систем на вашем коде, но на каждую ошибку автоматической системы приходится с десяток человеческих ошибок. Советы «писать лучший код», работать парами и так далее, тут не работают, особенно когда десятки программистов находятся в условиях нехватки времени.

Кармак также отмечает, что инструмент после каждого апдейта находит новые ошибки. То есть ошибки в коде вы никогда не сможете устраниТЬ до конца. В большом проекте они всегда будут, как статистические отклонения в свойствах физического материала.

Вы можете только попытаться минимизировать их негативный эффект.

“The first step is fully admitting that the code you write is riddled with errors” (John Carmack)



Статический анализ качества кода.

Метрики качества

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > Метрики

Контроль качества исходного кода предполагает наличие метрик, которые позволяют оценить достижение того или иного уровня качества программного проекта.

Метрика программного обеспечения (software metric) – численная мера, позволяющая оценить определенные свойства конкретного участка программного кода. Для каждой метрики обычно существуют ее эталонные показатели, указывающие, при каких крайних значениях стоит обратить внимание на данный участок кода.

Метрики кода разделяются на категории и могут оценивать совершенно различные аспекты программной системы: сложность и структурированность программного кода, связность компонентов, относительный объем программных компонентов и др.

Основные метрики *:

- Покрытие кода юнит тестами (Code coverage)
- Комментарии (Comments)
- Отсутствие связности внутри методов (LCOM4)
- Индикатор спутанности пакетов (Package tangle index)
- Процент дублирования кода (Duplications)
- Цикломатическая сложность (Complexity)

<https://www.osp.ru/os/2009/08/10748698>

<https://dou.ua/lenta/articles/code-metrics/>

https://ru.wikipedia.org/wiki/Метрика_программного_обеспечения

<https://habrahabr.ru/post/205342/>

<http://cyberleninka.ru/article/n/problemy-izmereniya-kachestva-programmnogo-koda>

* Не все метрики возможно измерить в рамках используемых технологий и средств



Рекомендуемая литература

Введение > CI > Среды > CD > CT > Метрики

- "[Continuous Integration](#)", Martin Fowler, 01 May 2006.
- "[On the Sustained Use of a Test-Driven Development Practice at IBM](#)", IEEE Xplore Digital Library, 2007.
- M. Bland, "[Goto Fail, Heartbleed, and Unit Testing Culture](#)", Martin Fowler, 03 June 2014.
- M. Fowler, "[Test Coverage](#)", Martin Fowler, 17 April 2012.
- "[Toyota Unintended Acceleration and the Big Bowl of 'Spaghetti' Code](#)", Safety Research & Strategies, 07 November 2013.
- "[Bookout v. Toyota: 2005 Camry L4 Software Analysis](#)", M. Barr

Содержание

<u>Введение в DevOps</u>	3
<u>Continuous Integration</u>	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
<u>Метрики</u>	80

Влияние сред на процесс DevOps

Введение > CI > Среды > CD > CT > Метрики

Состав сред напрямую влияет на эффективность автоматизации процесса DevOps.

При автоматизации процесса DevOps необходимо произвести переоценку имеющихся сред разработки и тестирования. Переоценка должна включать следующие проверки:

- Технологическая полнота сред
- Наличие задержек при разработке нескольких версий АС (выпуск хотфиксов для версии ПРОМ, выпуск хотфиксов для версии в тестировании, разработка будущей версии)
- Наличие практики использования разных версий системного ПО и его влияние на процесс сборки и развертывания



Все этапы жизненного цикла программного обеспечения являются обязательными, и каждый этап требует наличия собственной среды. Это необходимо для того, чтобы:

- обеспечить достаточную пропускную способность данного этапа, исключить ожидания в очереди;
- обеспечить изолированность прохождения этапа, исключить возможное влияние различных стадий друг на друга.

Кроме того, среды размещены на различных стендах, которые имеют различные мощности и разный объем интеграций со смежными системами.

Технологическая полнота сред

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Технологическая полнота сред напрямую влияет на процесс разработки и процесс DevOps.

Чем больше технологическая полнота сред сдвинута влево (от среды эксплуатации к среде разработки), тем проще автоматизация процесса DevOps и меньше в нем ошибок. При этом необходимо соблюдать разумный баланс между повышением эффективности от полноты сред и стоимостью реализации полноты.

	Разработки	СТ	ИФТ	ПСИ	ПРОМ
Георезервирование					Есть
Продуктовое резервирование данных				Редко	Есть
Балансировщики нагрузки	Редко	Редко	Есть	Есть	Есть
Прокси сервера	Редко	Частично	Есть	Есть	Есть
СУДИР			Есть	Есть	Есть
Шифрование и электронная подпись	Редко	Частично	Частично	Есть	Есть
Шлюзы к внешним АС (не АС Банка)	Редко	Редко	Частично	Частично	Есть
Интеграция с другими АС Банка	Редко	Редко	Частично	Частично	Есть
Использование протокола HTTPS для интеграции и внутрисистемного взаимодействия		Редко	Есть	Есть	Есть
Контроль траффика через DataPower	Частично	Есть	Есть	Есть	Есть

Версии системного ПО

Введение > CI > Среды > CD > CT > Метрики

- Автоматизация процесса DevOps повышает требования к согласованному обновлению версии системного ПО на средах. Различные версии системного программного обеспечения могут требовать разных конфигурационных параметров, последовательности их применения, дополнительных прав и т.п.
- При автоматизации DevOps необходимо проверить наличие «практики»* использования разных версий системного ПО и её влияние на процесс сборки и развертывания.

Среда
разработки

Oracle
11gR2

Среда СТ

Oracle
11gR1

Среда ИФТ

Oracle
11gR1

Среда ПСИ

Oracle
11gR1

Среда ПРОМ

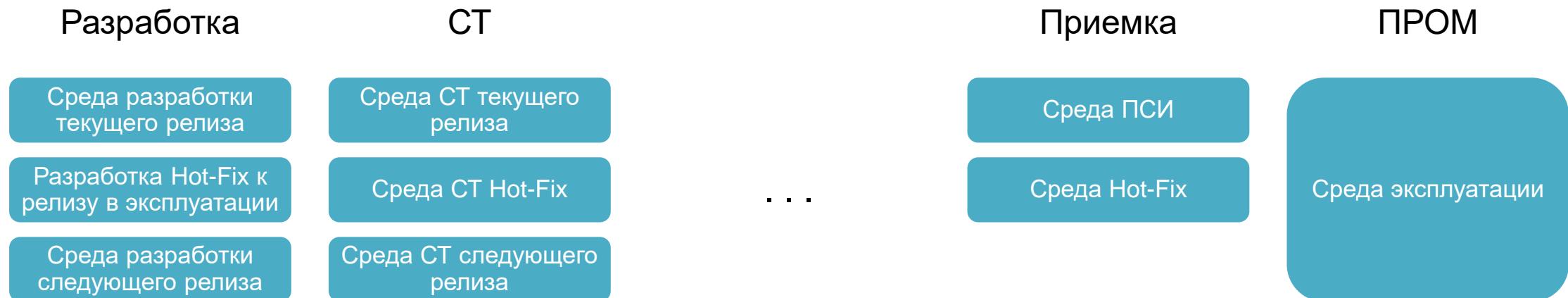
Oracle
11gR1

* «Практика»: На среде разработки у нас уже три релиза как 11R2 версия системного ПО, а на ПРОМ еще только 11R1.

Разработка нескольких версий АС

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

- Если система находится в промышленной эксплуатации, то необходимо иметь отдельные среды разработки, СТ и приемки (HotFix), которые соответствуют версии, установленной в промышленной среде. На этих средах выполняется исправление и тестирование дефектов, найденных на промышленной среде и включаемых в Hot-Fix-ы.
- Также зачастую выполняется параллельная разработка более, чем одного релиза. В этом случае, для исключения взаимного влияния функционала отдельных релизов при тестировании необходимо иметь отдельные среды разработки и СТ следующего релиза (для каждого релиза, если их больше).



Процесс DevOps должен охватывать все среды, включая среды необходимые для разработки нескольких версий ПО.

Обязательный объем сред

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Согласно действующим регламентам Банка обязательный объем сред зависит от критичности системы. В то же время ускорение процесса разработки и автоматизация в рамках процесса DevOps может потребовать дополнительных сред. Данная потребность должна рассматриваться исходя из экономической целесообразности и удовлетворяться в рамках проектов внедрения/доработки конкретных АС.

Не допускается использование одного вида тестовых стендов под различные цели для АС Mission Critical и Business Critical

Тип критичности	Разработка	СТ HotFix	СТ	СТ следующего релиза		MCB	ИФТ	НТ	Обучение	ПСИ	HotFix
				СТ	СТ						
1. Mission Critical	■	○	○	○	○	○	■	○	○	■	■
2. Business Critical	■	○	○	○	○	○	■	○	○	■	○
3. Business Operational	■		○		○	○	■	–	○	■	○
4. Office Productivity	■		○		○	○	■	–	○	■	■

■ Требуется обеспечить (на время простоя мощности процессоров и оперативной памяти могут быть переданы другой среде).

○ Выделяются по запросу

– Не требуется

Динамические среды разработки

Введение > CI > Среды > CD > CT > Метрики

Для выстраивания эффективного процесса CI необходимо иметь возможность одновременного независимого тестирования различных веток кода. Для этого используется функционал динамического выделения стендов разработки, который позволяет:

- при разработке отдельных фич выделять стенд для тестирования наработок по ним;
- при сборке по коммиту автоматически получать стенд для выполнения автотестов с последующим освобождением;
- при работе над исследовательскими задачами проводить испытания на отдельных стендах.

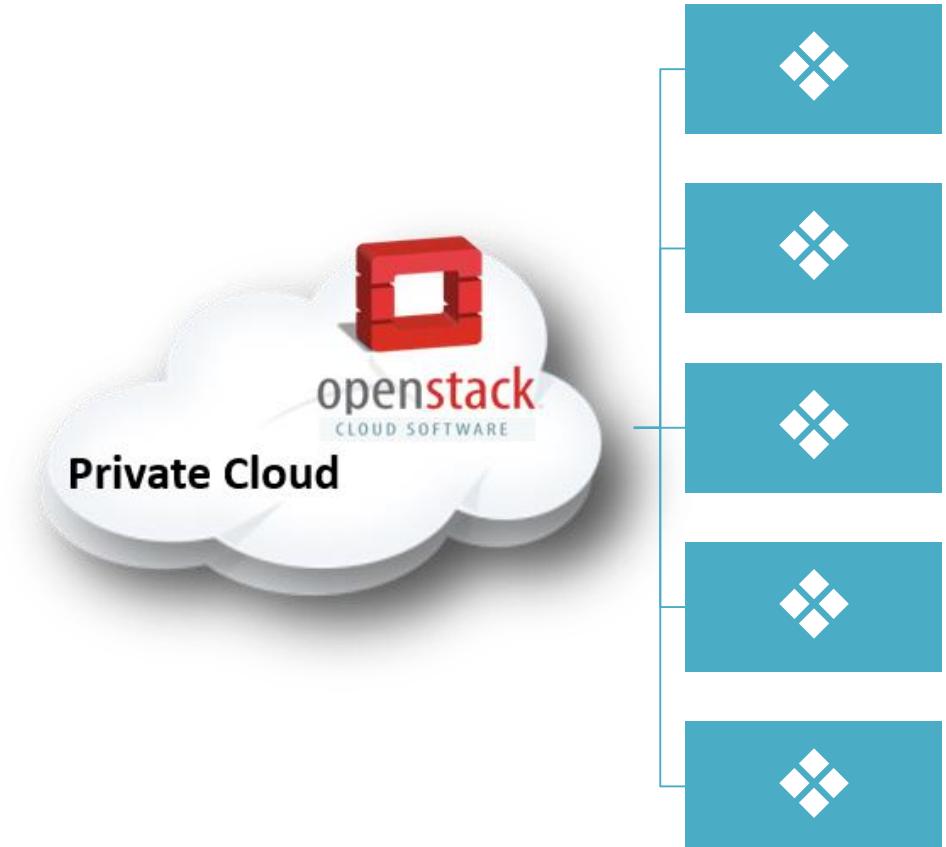
Применение динамических сред дает следующие преимущества:

- отсутствие очереди на автоматическое тестирование сборки;
- минимизацию влияния отдельных задач друг на друга;
- эффективное управление ресурсами.

<https://sbtatlas.sigma.sbrf.ru/wiki/display/CLOUD/CD+AntiCyclon>

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pageId=80483220>

Разработка



Содержание

<u>Введение в DevOps</u>	3
<u>Continuous Integration</u>	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
<u>Метрики</u>	80

Основные практики DevOps.

Continuous Delivery

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

Непрерывная поставка (CD_L, Continuous delivery) – практика разработки программного обеспечения, гарантирующая то, что программное обеспечение постоянно готово к развертыванию в промышленную эксплуатацию. Практика включает в себя автоматизацию развертывания на тестовые среды, автоматизацию тестирования успешности развертывания на уровне администраторов (технические и функциональные тесты развертывания), а также интеграцию с процессами автоматизированного функционального, нагрузочного тестирования и динамического тестирования на безопасность.

Практика включает в себя:

- Хранение инсталляционного пакета (дистрибутива) в централизованном хранилище.
- Гарантирование того, что единый дистрибутив пройдет все циклы тестирования.
- Хранение конфигураций тестовых сред в централизованном версионном хранилище.
- Автоматизацию развертывания приложения на среды тестирования.
- Автоматические проверки успешности процесса развертывания.
- Интеграцию с инструментами тестирования – функционального, нагрузочного и динамического тестирования на безопасность.
- Тесное сотрудничество команды разработки, команды тестирования (ДК) и команды эксплуатации (ЦИ, ЦСПС)



Основные практики DevOps.

Continuous Deployment

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

Непрерывное развертывание (CD_P, Continuous deployment) – практика разработки программного обеспечения, направленная на полную автоматизацию поставки от среды разработки в промышленную среду. Является дальнейшим развитием процесса Непрерывной поставки. В условиях Банка дополнительно затрагивает автоматизацию приемо-сдаточных испытаний. Характеризуется тесным взаимодействием разработки и эксплуатации.

Практика включает в себя:

- Хранение инсталляционного пакета (дистрибутива) в централизованном хранилище.
- Гарантирование того, что единый дистрибутив пройдет все циклы тестирования.
- Хранение конфигураций среды ПСИ и промышленных сред в централизованном версионном хранилище.
- Автоматизация развертывания приложения на среды ПСИ и ПРОМ.
- Автоматические проверки успешности процесса развертывания .
- Обеспечение безопасного автоматического отката до предыдущей версии приложения, либо организацию принципа blue green развертывания.
- Тесное сотрудничество команды разработки, команды тестирования (ДК) и команды эксплуатации (ЦИ, ЦСПС)



Целевые инструменты CD

Введение > CI > Среды > **CD** > СТ > Метрики

В качестве целевого выбран следующий стек технологий:



Nexus OSS – как хранилище всех ИП - артефакт, полученный по окончании процесса CI.



Jenkins

Jenkins + Pipeline plugin – как целевой инструмент оркестрации процессом развертывания на конкретную среду. Сам процесс развертывания создается с помощью языка Groovy с заранее доступными модулями расширениями в Pipeline plugin .



ANSIBLE

Ansible – целевой инструмент развертывания и настройки тестовых стендов. Все элементы развертывания должны быть реализованы в виде ролей Ansible, наиболее типовые роли (например: развертывание WAS, Nginx, и тд) должны быть получены из централизованного репозитория ролей СБТ.



GIT – версионное хранилище. В рамках целевого процесса CD тут должны храниться конфигурации тестовых сред, Groovy код процесса развертывания и сценарии развертывания.



Elasticsearch



logstash

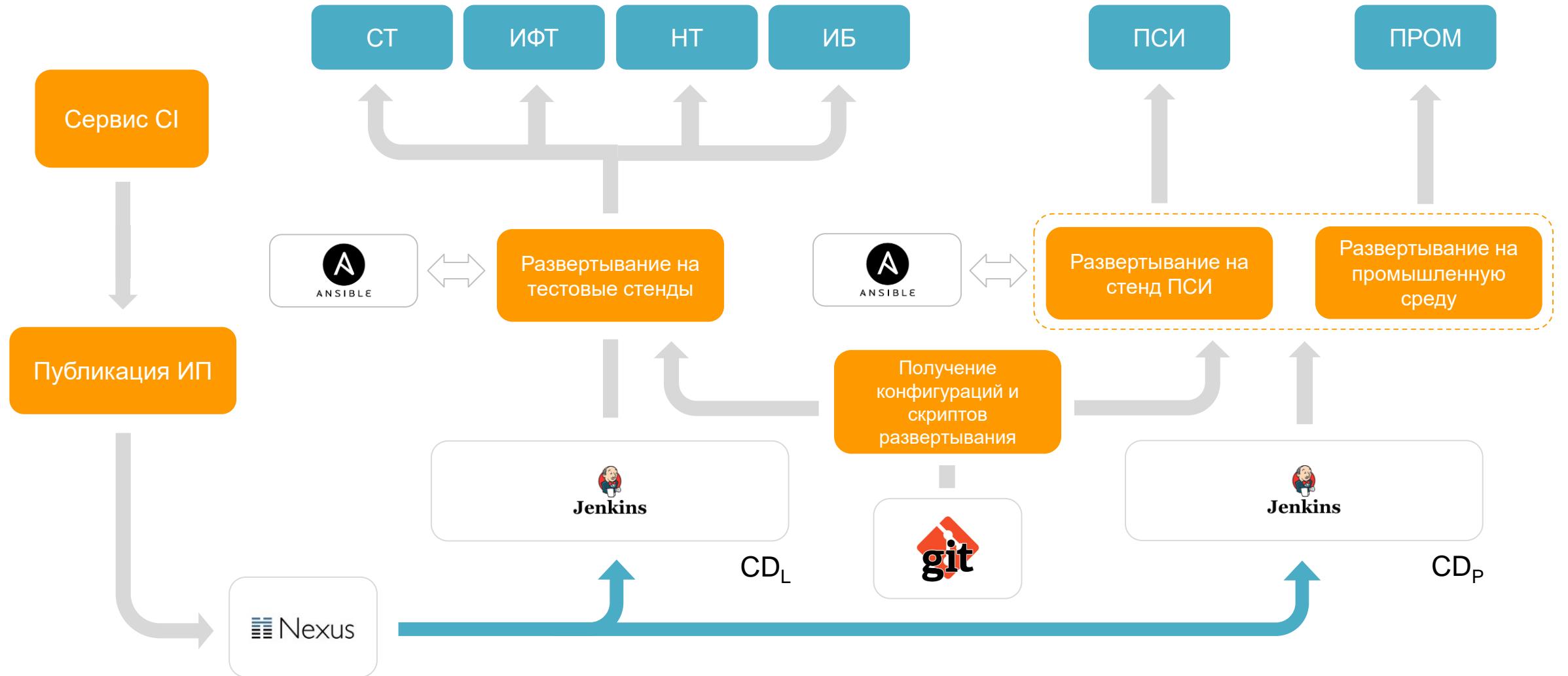


kibana

ELK – стек технологий Elasticsearch, Logstash и Kibana – сбор, агрегация и визуализация журналов развертывания для отслеживания ошибок развертывания.

Continuous Delivery & Deployment Workflow

Введение > CI > Среды > CD > СТ > Метрики



Nexus

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > Метрики



Nexus — это централизованное хранилище Банка. Это хранилище делится на два логических элемента.

Репозиторий разработки Nexus — хранилище содержит системное ПО; различные библиотеки, использующиеся как зависимости; результаты ночных сборок; сборок из feature веток. Также этот репозиторий зеркалит различные общедоступные в Интернете репозитории, которые необходимы для процесса разработки. В общих словах этот репозиторий содержит все, что нужно для процесса **Continuous Integration**. К этому хранилищу применяются обычные требования по отказоустойчивости и по сохранности данных. Сборки в этом репозитории хранятся не более одного месяца.

Nexus Банка или ХИП Nexus — это хранилище инсталляционных пакетов (дистрибутивов) передаваемых в Банк. Этот Nexus служит гарантией того, что тот же самый дистрибутив, собранный один раз под одной версией, разворачивался и тестиировался на всех этапах тестирования. То есть это хранилище служит для обеспечения непрерывной связи двух процессов — **Continuous Delivery** и **Continuous Deployment**. К этому хранилищу применяются повышенные требования по отказоустойчивости и по сохранности данных. То, как долго хранятся дистрибутивы в этом, репозитории определяется требованиями Банка.



Jenkins

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)



Jenkins

Jenkins — это проект с открытым исходным кодом, написанный на Java, созданный решать огромное число задач по оркестрации различных процессов. В нашем случае Jenkins выбран как целевое решение для оркестрации процессов развертывания как на средах тестирования в СБТ, так и на средах приемо-сдаточных испытаний и промышленных средах в самом Банке.

Не предполагается, что **Jenkins** будет оркестрировать весь процесс от среды разработки до промышленной среды. С помощью Pipeline plugin будут оркестрированы отдельные части процесса, например, развертывание ОРД в среды СТ и в среды ИФТ — это два разных Pipeline процесса в рамках одной проектной области в сервисе CD Jenkins .

Pipeline plugin — решение, реализованное в виде плагина к **Jenkins**, позволяющее воплотить принцип «процесс развертывания как код», то есть с помощью этого плагина сам процесс развертывания представлен в виде Groovy кода. Pipeline plugin также предоставляет огромное число уже готовых модулей (функций Groovy) для реализации различных задач, а также имеет подробную документацию по каждому модулю — это обеспечит низкий порог входления, то есть не надо обучаться языку Groovy для того, чтобы начать создавать несложные процессы развертывания.

Используя данный плагин для всех своих развертываний мы решаем несколько задач:

- Версионирование сценария развертывания — версии Groovy кода процесса развертывания будут храниться не в самом Jenkins, а в GIT репозитории, что обеспечит надежное хранение и контроль изменений.
- Задача логично вытекающая из предыдущей — передача процесса развертывания между разными командами и частями процесса — из СБТ (разработчики для ДК) в Банк (для ЦСПС и ЦИ)
- Визуализация процесса - Pipeline plugin имеет очень удобный и понятный интерфейс отображения ранее запущенных сценариев, с возможностью быстрого доступа к логам каждого этапа.

Ansible

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)



Ansible — программное решение с открытым кодом для удаленного управления конфигурациями, реализованное на языке Python. Ansible берет на себя всю работу по приведению удаленных серверов в необходимое состояние посредством ssh протокола, необходимо лишь описать, как достичь этого состояния с помощью сценариев playbooks, выполненных в формате yaml.

С помощью **Ansible** можно решать различные задачи, например:

- Установка/удаление СПО.
- Конфигурирование СПО.
- Создание/удаление пользователей.
- Хранение пользовательских паролей/ключей в защищенном виде (Ansible Vault).
- Развертывание кода вашего ППО.
- Запуск скриптов авто-тестирования в сторонних системах.
- Управление системой создание/удаление контейнеров/виртуальных машин.

Ansible. Преимущества

Введение > CI > Среды > CD > СТ > Метрики



Преимущества Ansible:

- Низкий порог вхождения. Обучиться работе с Ansible можно за очень короткие сроки.
- На управляемые узлы не нужно устанавливать никакого дополнительного ПО. Все работает через SSH.
- Код программы, написанный на Python, очень прост. Разработка дополнительных модулей не составляет особого труда.
- Декларативный язык yaml, на котором пишутся сценарии, также предельно прост.
- Встроенная возможность безопасного хранения чувствительной информации (пароли, ключи и тд) — Ansible Vault.
- Подробная и весьма просто написанная официальная документация. Она регулярно обновляется. Ее можно найти на [официальном сайте](#).
- Ansible работает не только в режиме push, но и pull, как это делают большинство систем управления (Puppet, Chef).
- Имеется возможность последовательного обновления состояния узлов (rolling update).
- Сама структура инструмента позволяет создавать отдельные модули - [роли](#), которые могут быть использованы разными командами в процессе поставки своего ПО. У нас имеется возможность централизованно хранить эти модули и обеспечить их использование разными командами на разных проектах.

Центральный репозиторий базовых ролей

Ansible СБТ

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

Центральный репозиторий базовых ролей Ansible в СБТ содержит **роли Ansible**, которые централизованно разрабатываются и поддерживаются выделенными командами и должны быть использованы разными командами в процессе развертывания каждого проекта в СБТ. По смыслу этот репозиторий – это аналог официального Ansible Galaxy, но отличается тем, что в нем делятся своими наработками исключительно сотрудники СБТ и Банка.

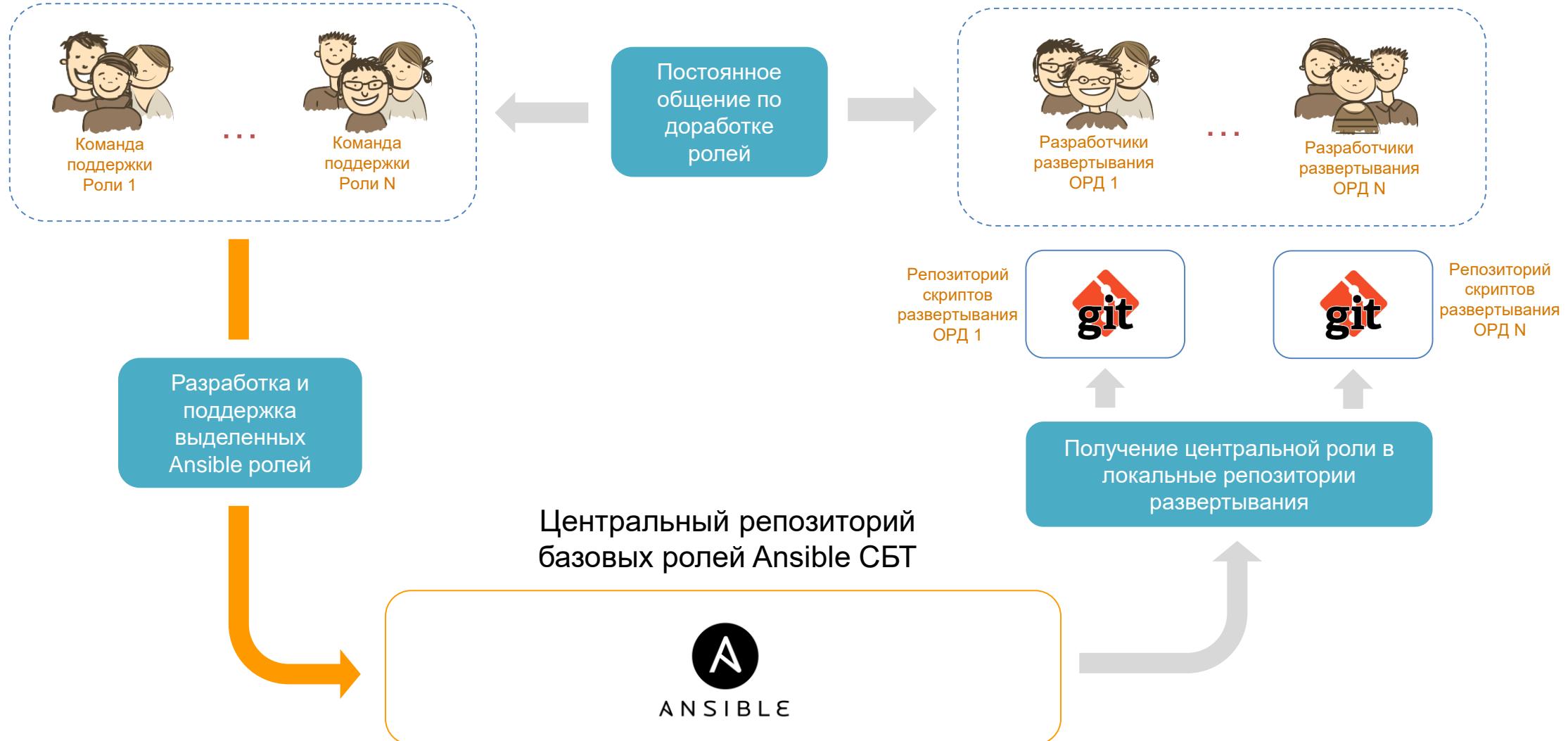
Роль Ansible – совокупность последовательности подзадач и всех связанных с этими подзадачами данных и вспомогательных элементов в одном месте для выполнения одной функциональной задачи. По сути это логические элементы, с помощью которых выстраивается весь процесс настройки или развертывания приложения. То есть будет отдельная роль «Установка WAS», «Установка Nginx», «Установка приложения в WAS» и тд.

Обобщенная структура роли:

Имя роли	files	содержит файлы, которые будут скопированы на настраиваемые стенды. Также может содержать скрипты, которые позже будут запускаться на стендах.
	handlers	обработчики, которые будут использоваться при выполнении операционных задач (таких как перезагрузка сервисов и т. д.)
	meta	мета-данные об авторе роли, контактах для связи. Также описание зависимостей от других ролей.
	tasks	это по сути основная папка роли. Здесь содержится playbook со всеми задачами, которые исполняются в рамках этой роли.
	templates	содержит файлы-шаблоны (.j2) с переменными. Эти шаблоны в основном используются для конфигурационных файлов.
	vars	содержит локальные переменные роли.

Центральный репозиторий базовых ролей Ansible СБТ

Введение > CI > Среды > **CD** > СТ > Метрики



Центральный репозиторий базовых ролей Ansible СБТ. Вопросы

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

Как получить централизованную базовую роль в свой репозиторий Ansible?

Для этого необходимо создать файл requirements.yml в корне репозитория с содержанием типа:

```
- src: git@gitlab.sberbank.ru:mygroup/ansible-example.git
  scm: git
  version: "0.1"
```

Каждая роль будет находиться в отдельном репозитории, то есть для каждой роли, необходимой конкретному проекту, нужно будет сделать отдельное описание (как выше). В итоге у каждого проекта получится свой уникальный requirements.yml, который команда проекта должна будет поддерживать в актуальном виде. После того как requirements.yml подготовлен и актуализирован, на момент создания ИП (финальная часть CI процесса) нужно установить все роли в свой локальный репозиторий из центрального репозитории Ansible СБТ так:

```
ansible-galaxy install -r requirements.yml
```

затем запаковать подготовленную структуру вместе с самим приложением в ИП для дальнейшего развертывания на всех средах.

Что делать, если централизованная роль нам не подходит и нужны изменения?

В таком случае необходимо связаться с владельцем роли и запросить изменения. Если по каким-то причинам у владельца роли сейчас нет ресурсов на внесение изменений, возможен вариант, когда команда сама вносит изменения и создает pull request, который владелец роли рассматривает и одобряет/не одобряет.

Что делать, если необходимая роль вообще отсутствует в центральном репозитории Ansible СБТ ?

Каждая команда может создать свою роль при условии, что она будет добавлена в центральный репозиторий Ansible СБТ и команда возьмет на себя роль владельца. В таком случае команда берет на себя обязанности по доработке, сопровождению этой роли, а также обязуется поддерживать коммуникацию с теми командами, которые возможно захотят использовать эту роль в своем процессе развертывания.

Позволяется ли иметь свои собственные локальные роли?

Если роль очень специфична и с большой долей вероятности никем не может быть переиспользована — да, но при условии, что будет обеспечен read-only доступ к вашему локальному Ansible репозиторию, для того чтобы сотрудники ЦЭПП могли исследовать ваши роли и исключать возможность дублирования функционала с центральными ролями.

Как происходит процесс развертывания?

Введение > CI > Среды > CD > СТ > Метрики

Планируется использование процессов развертывания, основанных на плагине **Pipeline** целевого инструмента **Jenkins**.

Процесс развертывания в идеале должен содержать следующие логические шаги:

1. Получение всего необходимого для развертывания (ИП — приложение и скрипты развертывания, конфигурации)
2. Подготовка к восстановлению уже развернутого приложения (опциональный шаг, зависящий от архитектуры ФП)
3. Подготовка среды для развертывания (все ли папки созданы, права выданы, все ли скрипты на месте и готовы для развертывания)
4. Выполнение развертывания.
5. Проверки после развертывания (проверка успешности развертывания + проверка работоспособности (Smoke testing))
6. Если предыдущий шаг не успешен — восстановление приложения на предыдущую версию (опциональный шаг, зависящий от архитектуры ФП)
7. Информирование заинтересованных лиц.

Некоторые шаги опциональны и могут отключаться в зависимости от среды, на которую происходит развертывание и требований к конкретной ФП.

Например: среда СТ может не требовать создания точки восстановления и возвращения на предыдущую версию.

!

ВАЖНО: это лишь логический процесс развертывания верхнего уровня, которому желательно следовать, чтобы минимизировать возможность ошибок и вреда среде, куда собственно происходит развертывание. Сами сценарии развертывания уникальны для каждого приложения.

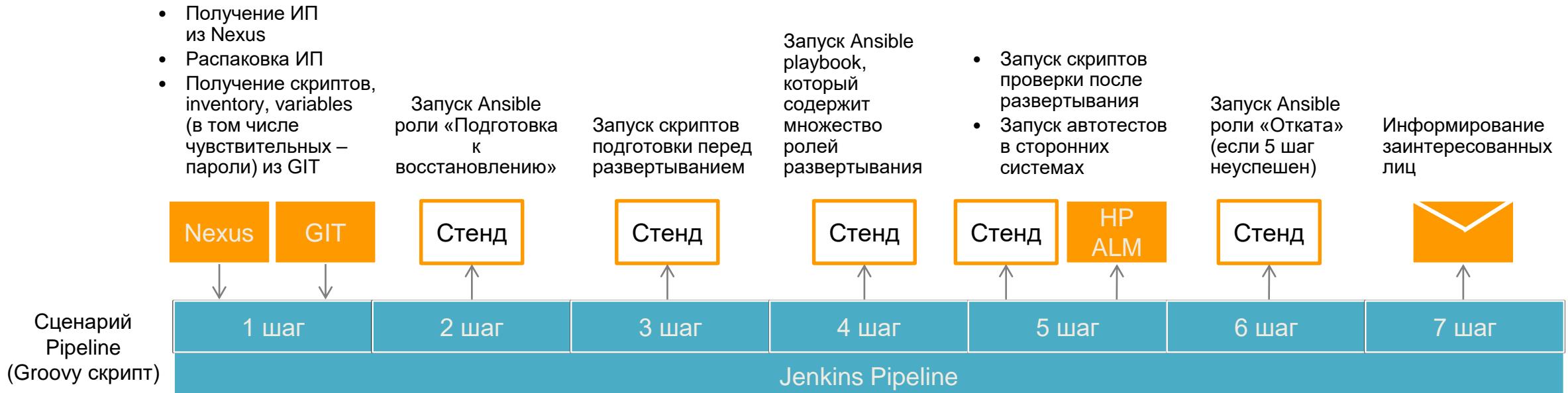
Как реализован процесс развертывания?

Введение > CI > Среды > CD > СТ > Метрики

В целом процесс разделен по следующим инструментам:

- **Jenkins + Pipeline plugin** в виде Groovy кода описывает общий процесс развертывания.
- **Ansible** роли – это логические элементы развертывания.
- **GIT** должен хранить все скрипты развертывания, конфигурации (сервера куда ставить – inventory, различные переменные - variables, включая чувствительные данные – пароли в зашифрованном виде)

На примере процесса изложенного на слайде «Как происходит процесс развертывания?»:

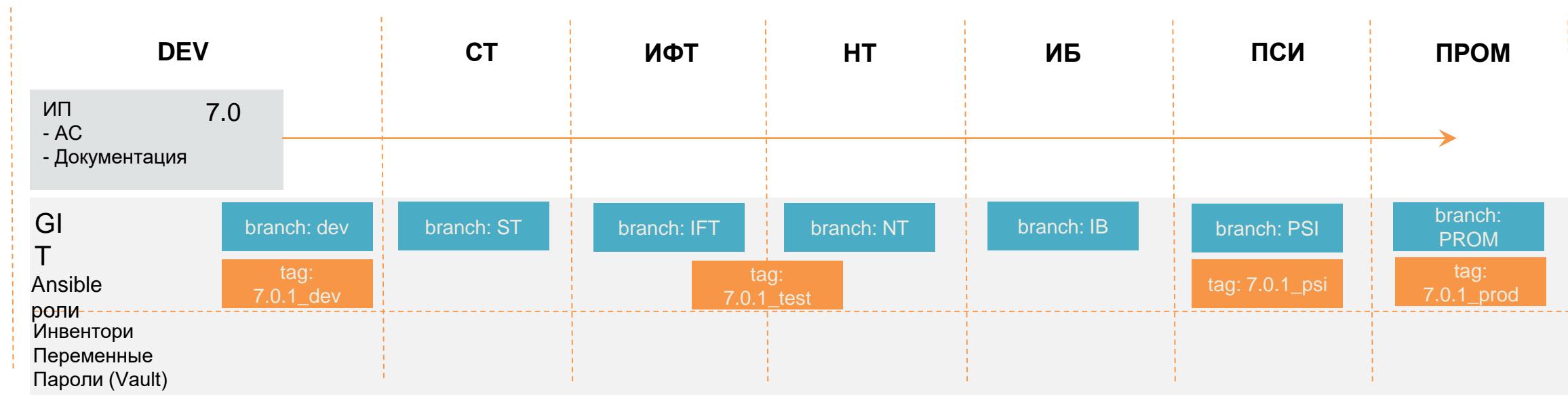


Управление конфигурациями сред

Введение > CI > Среды > CD > СТ > Метрики

Конфигурации сред — это конфигурации, сохраняющие целевое состояние сред как на системном уровне, так и на уровне приложения. Эти конфигурации необходимы для процесса развертывания приложения.

Конфигурации сред хранятся в централизованном **GIT** репозитории. Для каждой ФП создан отдельный репозиторий с разными ветками — каждая среда развертывания имеет свою отдельную ветку и своего ответственного. За ветку «dev» отвечает команда разработки, за ветки «СТ», «ИФТ», «НТ», «ИБ» - инженер развертывания тестовых сред, за ветки «ПСИ» и «ПРОМ» - представитель отдела промышленной эксплуатации. Кроме того, чтобы зафиксировать состояние конфигураций и связать их с конкретной версией ИП, должны создаваться тэги (имя тэга — версия ИП). Таким образом, если появляются изменения в конфигурациях представители разработки создают тэг, который сообщается всем остальным командам, для того чтобы они привели свои конфигурации в соответствующее состояние.



Система управления потоками развертывания

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)



- Управление сквозными потоками развертывания
- Визуализация сквозного маршрута инсталляционного пакета
- Реализация оперативного контроля производства технологических решений
- Сбор информации о процессах и предоставление операционной аналитики
- Сбор фактов по прохождению маршрута инсталляционным пакетом
- Синхронизация данных между сетями

Как это работает

- В конце цикла CI сформированный ИП публикуется в **Nexus** и вместе с этим в DPM отправляется сигнал о появлении новой версии ОРД.
- DPM после получения сигнала запускает поток развертывания, соответствующего ОРД.
- Поток развертывания представляет из себя набор последовательных шагов, в рамках которых DPM запускает задания в системе CD_L , затем в CD_P , получает от них статус исполнения задания и сохраняет соответствующие статусы ИП.
- DPM на специальном дашборде визуализирует процесс выполнения потока развертывания.

Интеграции процессов развертывания и тестирования

Введение > CI > Среды > CD > СТ > Метрики

Процессы **CD_L** и **CD_P** – процессы, отвечающие за все , что касается поставки уже готового собранного приложения по всем необходимым этапам тестирования и в качестве конечной цели - в промышленную эксплуатацию. В рамках этих процессов обеспечивается гарантия целостности, неизменяемости и уникальности дистрибутива, проходящего от разработки до промышленных сред.

Однако **CD_L** и **CD_P** не дают гарантию качества и соответствия политикам безопасности Банка самого дистрибутива. Для реализации этой задачи есть отдельные процессы – Continuous Testing и тестирование информационной безопасности (ТИБ) . Эти процессы глубоко интегрированы между собой, что обеспечивает минимизацию рисков ошибок, сбоев и взломов промышленной среды и как следствие возникновения возможных нежелательных последствий для Банка.



Рекомендованные ссылки и литература

Введение > CI > Среды > **CD** > СТ > Метрики

Внутренняя документация СБТ (в Альфе):

<http://confluence.ca.sbrf.ru/display/OPIR>

Литература:

"[The Phoenix Project](#)", Kim, Behr, Spafford.

"[DevOps Handbook](#)", Kim, Debois, Willis, Humble, Allspaw.

"[Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation](#)", Jez Humble, David Farley .

"[Continuous Delivery and DevOps - A Quickstart Guide Second Edition](#)", Paul Swartout

Содержание

<u>Введение в DevOps</u>	3
<u>Continuous Integration</u>	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
<u>Метрики</u>	80

Continuous Testing

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Непрерывное тестирование (Continuous Testing) – это практика выполнения автоматизированных тестов в рамках конвейера Continuous Delivery для получения немедленной обратной связи о результатах проверки правильности функционирования системы.

Практика включает в себя все виды тестирований на **не промышленных** средах. Это не значит, что все виды тестов выполняются все время, но они запускаются в определенный момент прохождения по конвейеру CD.

Важно правильно управлять временем выполнения тестов

- Важно следить за временем выполнения каждой стадии тестирования. Например, выполняя полный регресс сразу после билда, мы можем затянуть время обратной связи, поскольку полный регресс может выполняться долго

Тесты должны быть атомарными – маленькие и независимые единицы

- Маленькими тестами легче оперировать и организовывать параллельное исполнение
- Тесты не должны быть завязаны друг на друга, в противном случае мы получим трудности с их поддержкой даже при маленьких изменениях, а также это приведет к увеличению времени отладки и разбора ошибок тестирования
- Результат выполнения теста должен определяться самим автотестом

Автоматическая подготовка среды и тестовых данных

- Для возможности постоянного запуска автотестов подготовка среды и тестовых данных должна быть встроена в автоматический процесс конвейера CD

Скоординированный сквозной процесс обеспечения качества

- Для процесса обеспечения качества должны быть вовлечены разработчики, тестировщики и эксплуатация
- Автоматизация тестирования должна идти синхронно с разработкой, поскольку, если тесты не будут сразу правиться при малейшем изменении UI, то AT быстро скатится к простому регрессионному тестированию

Стабильная среда тестирования

- Важно уметь устранить зависимость вашей среды от внешних систем, тем более, если они не обладают высокой доступностью или отсутствуют
- Важно передавать на стадию ИФТ стабильный продукт с оттестированной на заглушках интеграций

Виды тестов

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Тестирование кода

- **Unit tests** – модульные тесты. Осуществляют проверку работоспособности части кода (методов класса). Тесты пишутся на языке программирования с использованием специальных фреймворков. Тесты запускаются во время процесса сборки дистрибутива. Позволяют разработчику получить быструю обратную связь о некорректно работающих методах
- **Статический анализ кода (соответствие правилам разработки)** – проверка на корректность разработки, соответствие принятым правилам и конвенциям разработки, в том числе правилам, принятым в конкретной команде. Проверка может происходить на локальной машине через плагин для IDE, при pull-request'е и на агенте jenkins, при сборке конкретной ветки в git
- **Статический анализ кода ИБ** – проверка кода на уязвимости по безопасности (vulnerability) или на наличие backdoor

Способы тестирования

- **Ручное тестирование** – процесс тестирования, при котором основные функции и шаги теста, выполняются тестировщиком
- **Автоматизированное тестирование** – процесс тестирования, при котором основные функции и шаги теста выполняются автоматически при помощи специальных инструментов, эмулирующих действия пользователей и тестировщиков

Виды тестов

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > Метрики

Тестирование развертывания системы (ОРД, АС, ФП)

- **Технические тесты развертывания (Sanity check, Health Check)** – нефункциональные, технические проверки работоспособности системы, выполняемые после развертывания и перед допуском пользователей. Выполняются для верификации процесса развертывания на ВСЕХ средах путем вызова модуля самодиагностики системы (health check) или выполнения скриптов (sanity checks), которые проверяют доступность портов, доступность сервисов и т.п.
- **Функциональные тесты развертывания (один из видов Smoke Test)** – функциональная проверка системы на работоспособность, выполняемая через визуальный интерфейс системы после развертывания, перед допуском пользователей. Выполняются для верификации процесса развертывания на ВСЕХ средах и эмулируют действия пользователя в системе. На промышленных средах данная проверка проводится под специальными учетными записями, данные которых не участвуют в учете.

Тестирование системы (ОРД, АС, ФП)

- **Build Acceptance Test** – функциональная проверка системы на работоспособность, в неё включаются тесты, покрывающие критичный функционал системы. Занимает достаточно мало времени для возможности предоставления быстрой обратной связи разработчику и тестировщику. Определяет, готов ли билд для дальнейших дорогих этапов тестирования. Состав BAT определяется совместно разработчиками, тестировщиками и эксплуатацией.
- **Функциональное тестирование (UI tests)** – функциональная проверка системы на работоспособность, выполняемая через визуальный интерфейс системы, предназначенная для проверки корректности реализации функциональных требований.
- **Интеграционное тестирование (Integration testing)** - тип тестирования, предназначенный для проверки корректности взаимодействия тестируемой ИТ-системы с другими ИТ-системами, интерфейсами, компонентами, потоками данных.
- **Регрессионное тестирование (Regression testing)** – тип тестирования, предназначенный для проверки работоспособности существующего ранее функционала после внесения изменений в ИТ-систему
- **Нагрузочное тестирование (Load Testing / Performance Testing)** – тип тестирования, предназначенный для оценки соответствия показателей ИТ-системы, установленным требованиям производительности.
- **Динамическое тестирование приложения на безопасность** – проверка системы на уязвимости (повышение привилегий, выполнение произвольного кода, sql injection,...), «закладки» и пр.

Инфраструктура выполнения тестов

Введение > CI > Среды > CD > СТ > Метрики

- Каждый вид тестов должен выполняться на своей инфраструктуре
- Технические и функциональные тесты развертывания выполняются на инфраструктуре CD на средах тестирования и могут быть переиспользованы на промышленных средах



Методы разработки автоматизированных функциональных тестов

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

- **Capture/playback** - запись выполнения действия тестировщика во время выполнения сценариев ручного тестирования. Значительно увеличивает продуктивность и устраняет затраты на ручное повторение однообразных действий во время ручного тестирования. Но любое малое изменение тестируемого ПО требует перезаписи ручных тестов.
- **Scripting** - разработка скриптов тестирования путём программирования на языках, специально разработанных для автоматизации тестирования ПО. Разработка скриптов требует квалификации программирования, изменения в тестируемом ПО требует сложных изменений в соответствующих скриптах, и поддержка все возрастающей библиотеки тестирующих скриптов.
- **Behavior Driven Development** - конечный тест представляет собой не программный код, а описание последовательности действий с их параметрами, написанный на обычном языке, например, «завести в базе данных пользователя с логином XXX и паролем YYY». За непосредственную реализацию ключевых слов (действий) отвечает специализированный фреймворк. Дизайнеру тестов нужно только знать о наборе действий, реализованных во фреймворке. Это даёт возможность создавать тесты людям, не имеющим навыков программирования и позволяет писать тесты, не зная конкретных деталей реализации приложения.
- **Test Driven Development** - разработка через тестирование, это метод разработки модульных тестов, а не тестов проверки функциональности рабочей системы.

Целевые инструменты АТ

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Функциональное тестирование ((АФТ) Регресс) - функциональная проверка АС на предмет работоспособности ранее разработанного функционала АС, возможно полное или частичное выполнение объема регресса, выполняется в рамках системного и интеграционно-функционального тестирования. В СБТ используется следующий стек целевых технологий управления и выполнения автоматизированного тестирования:



HP ALM – инструмент управления тестированием, выполняющий функции управления сценариями тестирования, запусками тестов и управления результатами тестирования. Сценарии тестирования могут быть реализованы в виде тестов выполняемых вручную, так и в виде автоматизированных функциональных тестов.



HP UFT – мощный универсальный инструмент автоматизации тестирования, использующий метод записи Capture/playback при создании тестов с возможностью редактирования созданных VBScript. Основная область применения – десктопные Windows приложения.



Selenium Webdriver

Selenium WebDriver – инструмент автоматизированного управления web-браузерами. Основной областью применения стало тестирование веб-приложений. Технология Selenium поддерживается большинством производителей web-браузеров. Selenium WebDriver имеет библиотеки для различных языков программирования и позволяет писать тесты на этих языках.



Cucumber – инструмент, позволяющий записывать тесты в виде функций и их параметров на обычном языке (существует реализация для русского языка), реализуя Behavior Driven development подход к разработке тестов.



SoapUI – open source web сервис, выполняющий широкий набор функций по разработке, отладке и тестированию SOAP и REST сервисов.

Управление ручным и автоматизированным тестированием

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [СТ](#) > [Метрики](#)

Система управления тестированием HP ALM является единой:

- точкой интеграции с процессом развертывания;
- системой получения отчетов и метрик по тестированию.

Система управления тестированием HP ALM позволяет гибко подходить к процессу тестирования:

- Комбинировать в один workflow автоматизированные и ручные тесты;
- Включать и отключать тесты.



Архитектура функционального тестирования

Введение > CI > Среды > CD > СТ > Метрики



- Сборка и публикация ИП в Nexus
- Получение в Jenkins сообщения о публикации нового ИП
- Запуск установки и настройки приложения с помощью Ansible
- Запуск тест-сетов в HP ALM
- Получение из HP ALM ID автотестов указанных в тест-сете/тест-плане на релиз.

- Получения актуальных версий АТ из версионного хранилища Git
- Сборка АТ
- Выполнение автотестов во фреймворке автотестирования
- Возвращения результатов автотестов из фреймворка автотестирования
- Сохранение результатов выполнения автотестов в HP ALM

Кибербезопасность

Цели и задачи

Введение > CI > Среды > CD > CT > Метрики

Цель проекта:

Минимизация уязвимостей в используемом в Банке программном обеспечении и минимизация затрат на их устранение за счет раннего выявления

Задачи проекта:

- Провести тиражирование существующих практик безопасной разработки ПО на критичные АС Банка
- Внедрить в жизненный цикл процесса разработки новых практик и методов тестирования.
- Распространить среди разработчиков АС правила безопасного программирования, создать роли Security Champion в командах разработки, проведение обучения.
- Определить место и процесс в Agile для роли Security Champion.
- Разработать методики и стандарты безопасной разработки
- Создать библиотеку безопасной реализации критичных функций и функций информационной безопасности
- Проработать возможность создания собственных инструментов анализа кода (Advanced Analytics)
- Разработать единую систему метрик для отслеживания эффективности как подразделения тестирования ИБ, так и производственных команд.
- Организовать экспертную поддержку процесса: проведение программ осведомленности, тренингов, внутренних конференций, разворачивание базы знаний, портала.

Кибербезопасность

Анализ кода на безопасность

Введение > CI > Среды > CD > **СТ** > Метрики

Тестирование безопасности - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Статический анализ - инспекция исходного кода без реального его исполнения. Это может быть:

- Ручное code review
- Анализ исходных кодов специальными инструментами

Процесс тестирования кода на безопасность зачастую достаточно длительный. Поэтому он обязательно должен запускаться на релизных сборках, а при возможности на остальных.

Подключение практики анализа кода на безопасность производится в рамках программы Кибербезопасность.

https://ru.wikipedia.org/wiki/Тестирование_безопасности

<http://www.protesting.ru/testing/types/security.html>

<https://testitquickly.com/2010/11/20/22/>

<https://sbtatlas.sigma.sbrf.ru/wiki/pages/viewpage.action?pageId=81900455>

Кибербезопасность

Анализ приложения на безопасность

Введение > CI > Среды > CD > CT > Метрики

Тестирование безопасности - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Динамический анализ - тестирование на возможность взлома (Penetration testing):

- Fuzz тестирование помогает найти ошибки, на которые программа реагирует неправильно.
- Валидация данных/инъекции (SQL, XSS, XML, XPATH, команды OS, загрузка файлов и т.д.)
- Тестирование аутентификации
- Тестирование авторизации
- Нагрузочное тестирование с модификациями позволяет моделировать DDOS атаки.

Подключение практики динамического анализа АС на безопасность производится в рамках программы Кибербезопасность.

https://ru.wikipedia.org/wiki/Тестирование_безопасности

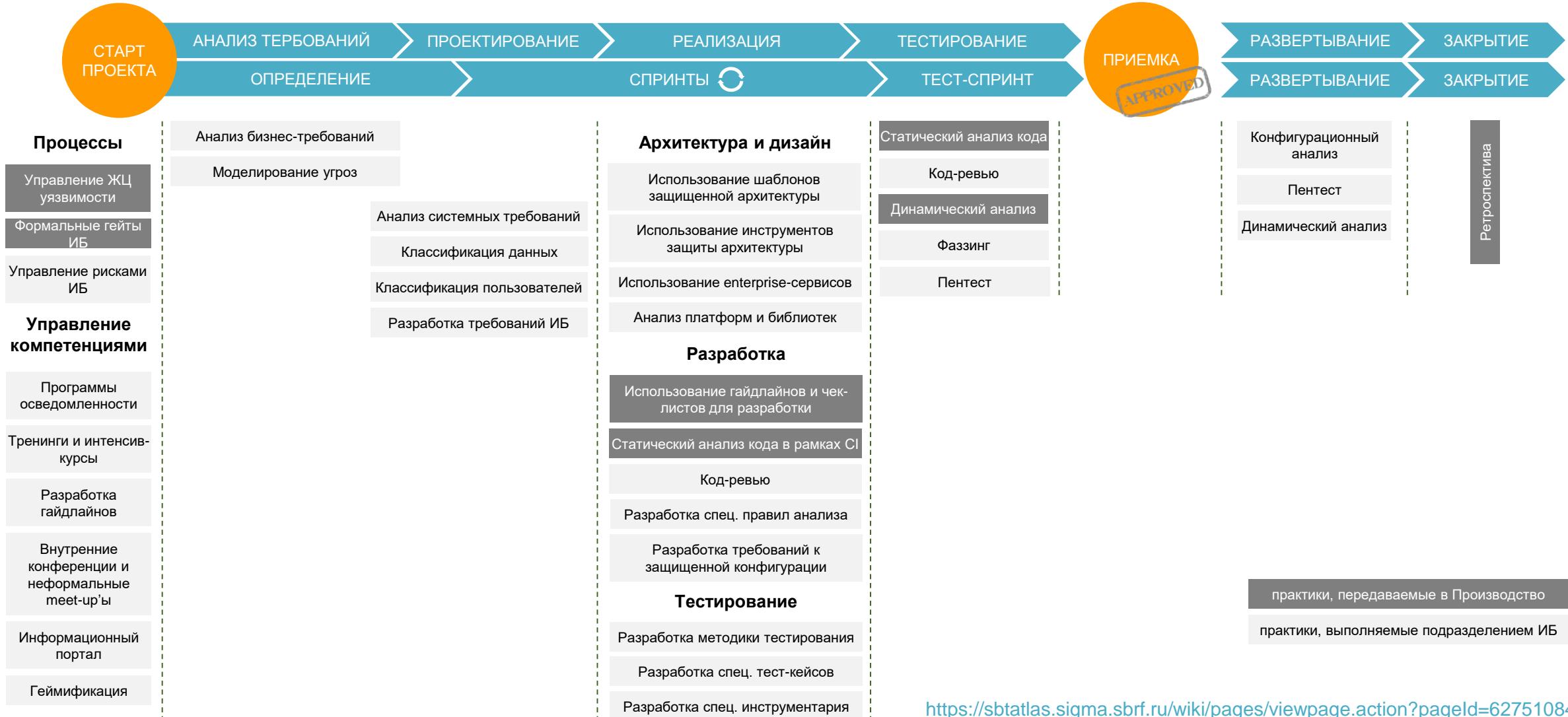
<http://www.protesting.ru/testing/types/security.html>

<https://testitquickly.com/2010/11/20/22/>

Кибербезопасность

Практики разработки защищенных приложений, Secure (SDLC)

Введение > CI > Среды > CD > СТ > Метрики

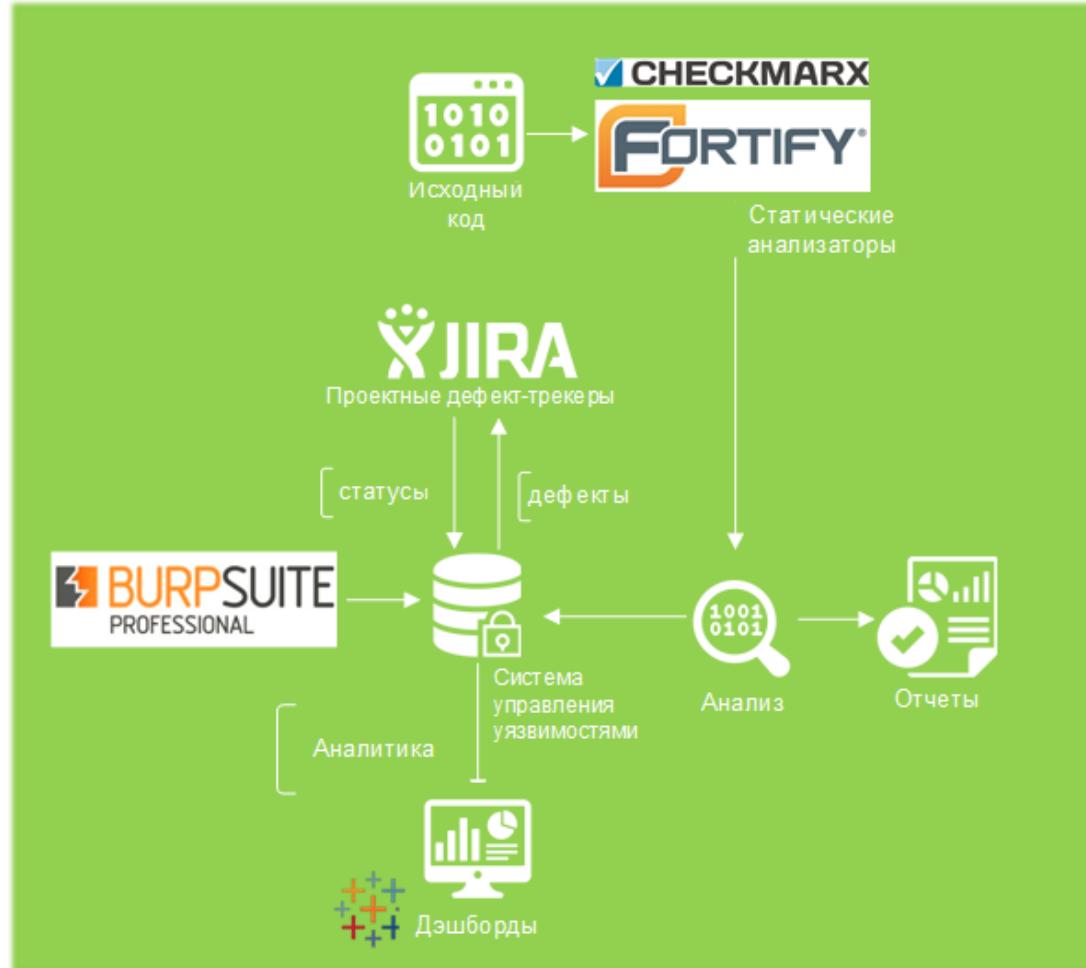


Кибербезопасность

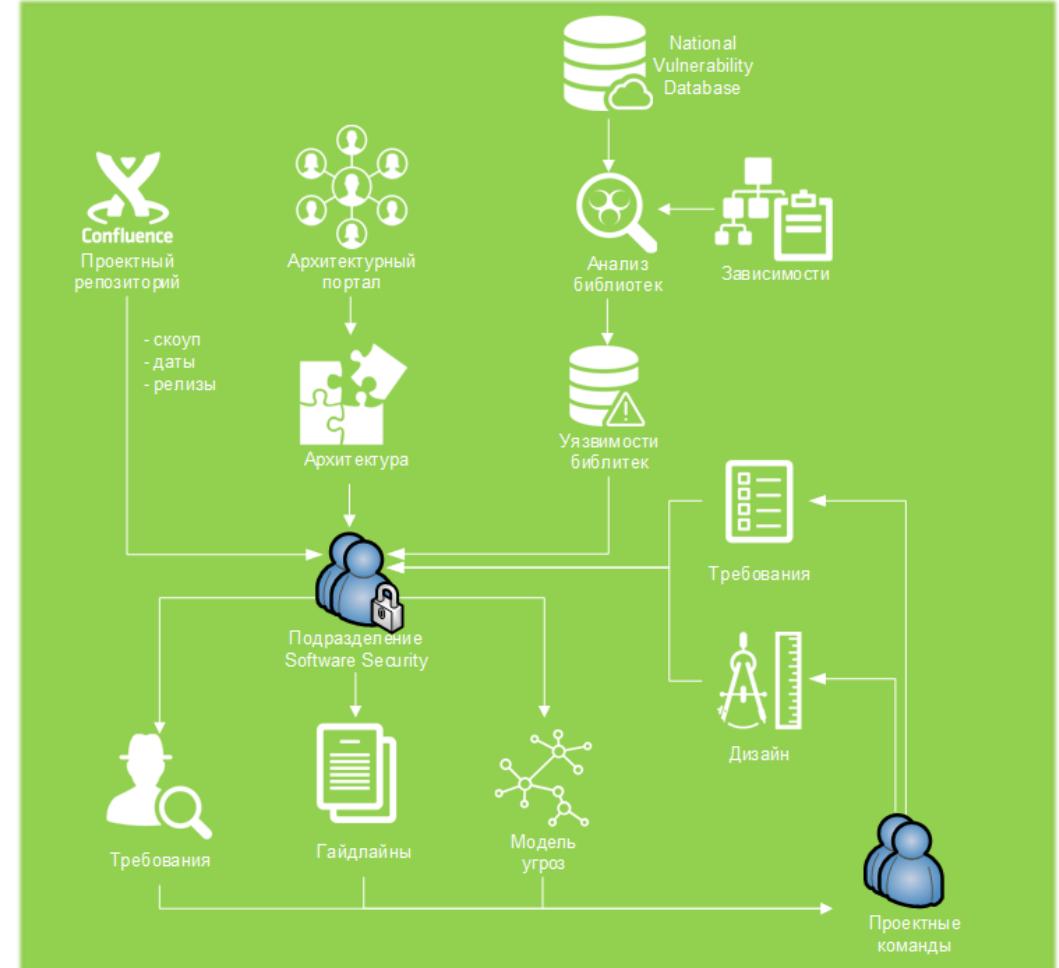
Инструменты и процесс

Введение > CI > Среды > CD > СТ > Метрики

Схема интеграции инструментов



Процесс взаимодействия с проектными командами



Кибербезопасность

Целевая модель фреймворка защищенных приложений

Введение > CI > Среды > CD > СТ > Метрики



Continuous Testing

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

Продолжение следует ...

Содержание

<u>Введение в DevOps</u>	3
<u>Continuous Integration</u>	19
<u>Среды</u>	39
<u>Continuous Delivery и Deployment</u>	46
<u>Continuous Testing</u>	64
Метрики	80

Continuous Measurement

Введение > CI > Среды > CD > CT > **Метрики**

Непрерывное измерение (СМ, Continuous Measurement) – это DevOps практика, которая заключается в постоянном отслеживании показателей всех систем и процессов, составляющих весь DevOps цикл.

Она позволяет всем участникам процессов DevOps оперативно реагировать на любые изменения метрик, чем зачастую помогает выявлять и решать проблемы ещё до того, как ситуация станет критичной.

ДевOps инженерам и командам разработки СМ позволяет:

- Оперативно реагировать на технические инциденты на всех стадиях **от дев до пром**
- Отслеживать состояние технических средств (сервера, сети и прочее)
- Находить и анализировать узкие места процессов разработки, тестирования, развёртывания и эксплуатации

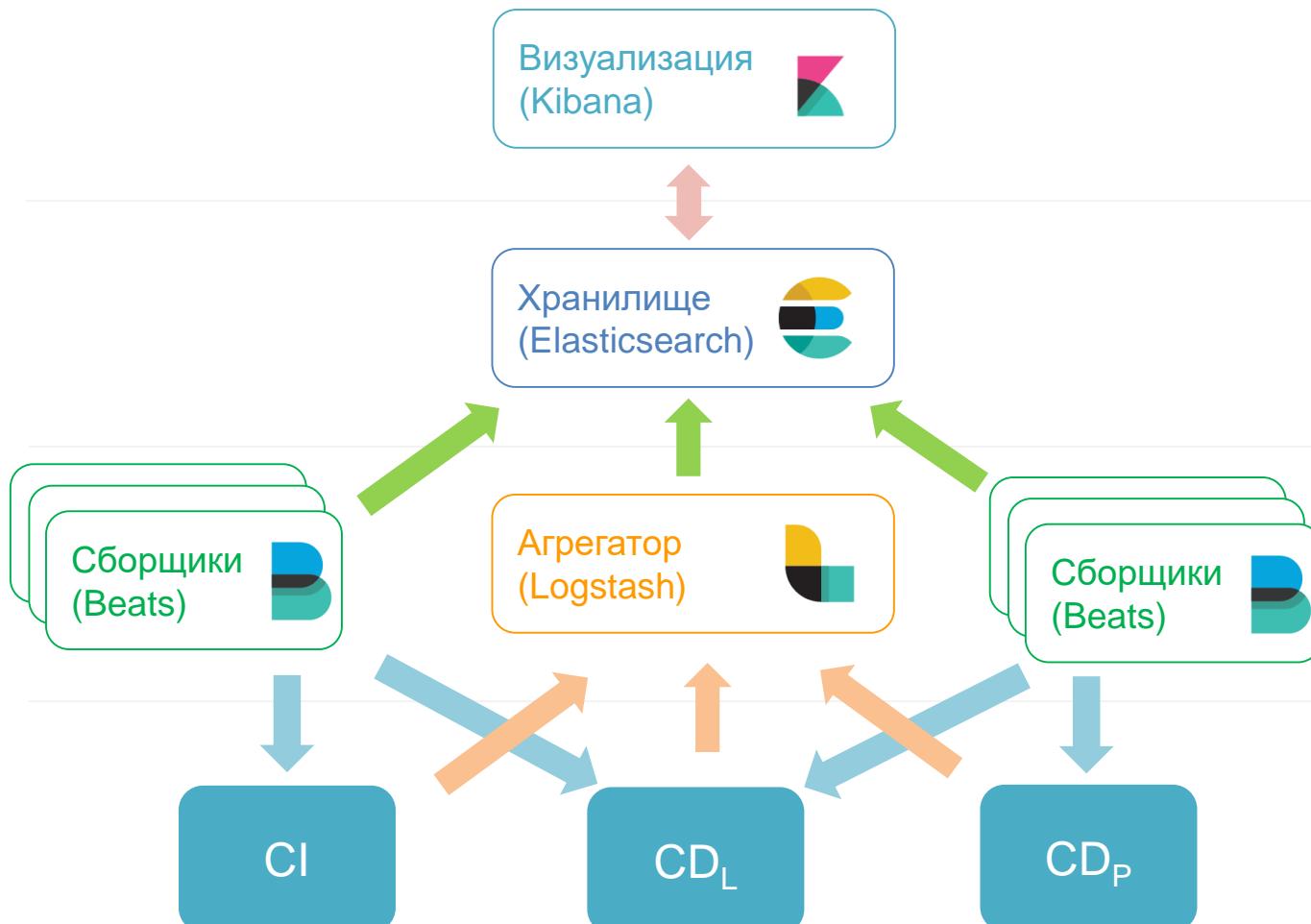
Бизнесу и менеджерам СМ даёт возможность:

- получить вполне обоснованные текущие цифры по Time-To-Market
- прогнозировать ожидания по выходу будущих фичей в пром
- проводить различные анализы для выявления узких мест процессов чтобы в последствии оптимизировать их



Инструменты мониторинга процессов DevOps

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)



- Панель визуализации предоставляет гибко настраиваемые дашборды
- Позволяет налету проводить исследования данных
- Хранилище принимает на хранение данные и строит поисковые индексы
- Хранилище предоставляет удобное апи для получения различных срезов данных
- Агрегатор принимает события и сохраняет их в хранилище
- Сборщики запрашивают данные и сохраняют в хранилище
- Инструменты CI , CD_L , CD_P
 - хранят свои данные
 - отправляют события на агрегатор

Технологические метрики DevOps

[Введение](#) > [CI](#) > [Среды](#) > [CD](#) > [CT](#) > [Метрики](#)

- скорость
- качество
- «здоровье»

Кодирование	
● T принятия pr	● % покрытия кода модульными тестами
● N принятых pr*	
● N коммитов после открытия pr	
● % отклоненных pr	
● % успешных коммитов	
● % билдов	
● Плотность дефектов	

Модульное тестирование

- % покрытия кода модульными тестами

Анализ кода

- T проведения анализа
- N дефектов
- % прохождения ACR (automated code review)

Сборка

- T сборки
- % успешных сборок
- T исправления сборки

Развертывание на CT

- T развертывания
- T исправления сборки
- % успешных развертываний
- N дефектов инфраструктуры
- N дефектов развертывания

Тестирование на CT

- T тестирования
- T устранения дефектов
- % автоматизации тестирования
- N дефектов CT
- Плотность дефектов

Развертывание на ИФТ

- T развертывания
- T исправления сборки
- % успешных развертываний
- N перепоставок
- N дефектов развертывания

Тестирование на ИФТ

- T тестирования
- T устранения дефектов
- % автоматизации тестирования
- N дефектов разработки
- N дефектов регресса
- Плотность дефектов

Эксплуатация

- N инцидентов из-за дефектов ПО
- N инцидентов из-за дефектов развертывания
- N инцидентов не выявленных системой мониторинга

Развертывание на Пром

- T развертывания
- T исправления сборки
- % успешных развертываний
- N перепоставок
- N дефектов развертывания
- Частота поставок

ПСИ

- N дефектов ПСИ

Развертывание на ПСИ

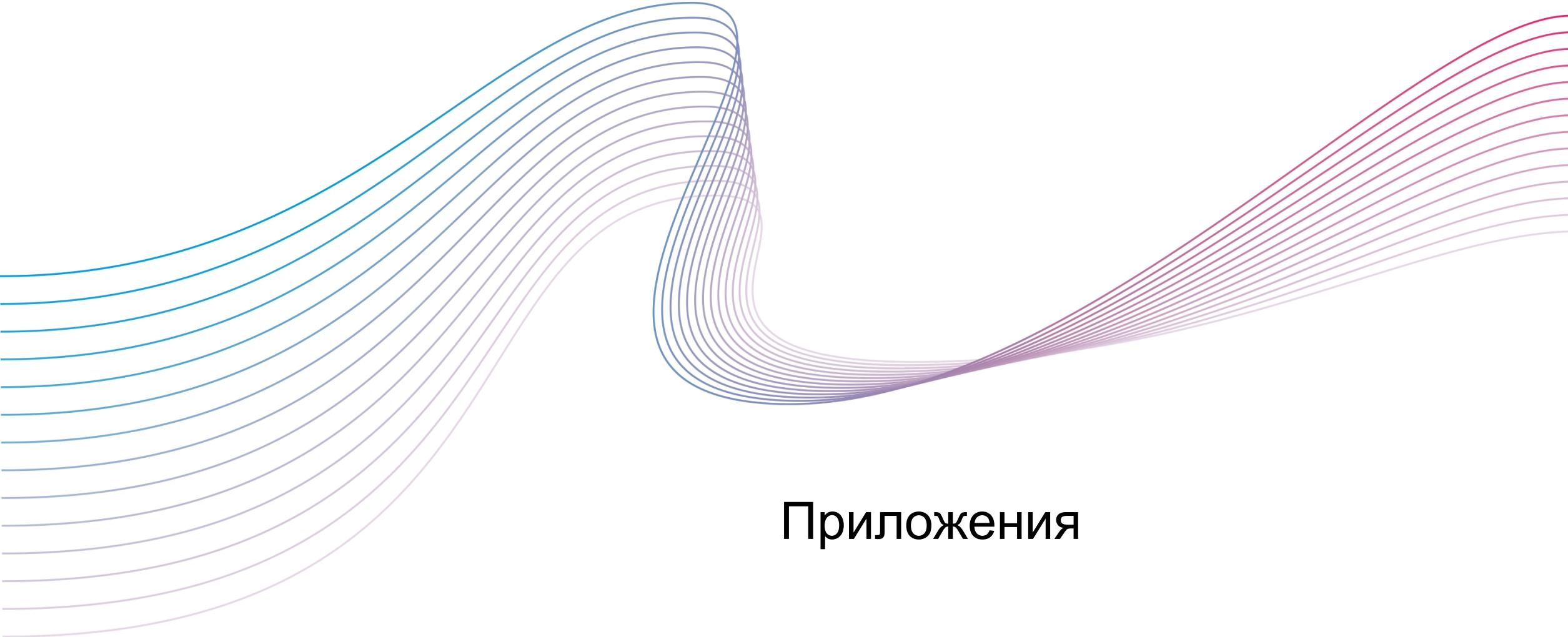
- T развертывания
- T исправления сборки
- % успешных развертываний
- N перепоставок
- N дефектов развертывания

Хранилище дистрибутивов Банка

* Pull request; T – время (скорость); N- количество (шт.)



Спасибо за внимание!



Приложения

Сокращения

API	Application Programming Interface	ИФТ	интеграционно-функциональное тестирование
AT	automation tests	КТС	комплекс технических средств
CD	Continuous Delivery & Deployment	МСФ	межсистемное взаимодействие
CDL	Continuous Delivery	НТ	нагрузочное тестирование
CDP	Continuous Deployment	ОРД	объект релизной деятельности (АС или ФП)
CI	Continuous Integration	ПО	программное обеспечение
CM	Continuous Measurement	ППО	прикладное программное обеспечение
GUI	graphical user interface	ПСИ	приемосдаточные испытания
АС	автоматизированная система	СПО	системно программное обеспечение
ИП	инсталляционный пакет	СТ	системное тестирование
ИТ	информационные технологии	ФП	функциональная подсистема
ИБ	информационная безопасность	ХИП	хранилище инсталляционных пакетов

Модель зрелости DevOps 1/5

Уровень зрелости	Отсутствует (0)	Начальный (1)	Средний (2)	Продвинутый (3)	Экспертный (4)
Кодирование	<ul style="list-style-type: none"> • Код хранится локально • Отсутствуют правила кодирования • Отсутствует практика проведения Code Review 	<ul style="list-style-type: none"> • Код хранится централизовано в VCS системе команды • Приняты Правила кодирования 	<ul style="list-style-type: none"> • CodeReview проводится в ручном режиме • Код документируется в ручном режиме • Документация разрабатывается перед ПСИ • Код хранится в централизованном BitBucket 	<ul style="list-style-type: none"> • Проводится автоматизированный Code Review • Используется автодокументирование кода (практика комментирования кода особым форматом, который дополнительно позволяет генерировать документацию к коду) • Документации разрабатывается перед ИФТ • Merge кода проводится только после тестирования 	<ul style="list-style-type: none"> • Документирование при сборке (встроенная справка и инструкции создаются и генерируются в едином фреймворке)
Модульные (unit) тесты	<ul style="list-style-type: none"> • Отсутствует, развернут, но не используется или носят фиктивный характер 	<ul style="list-style-type: none"> • Тесты вызываются вручную • Покрытие < 20% нового функционала ¹ 	<ul style="list-style-type: none"> • Тесты автоматически вызываются при сборке • Покрытие от 20% до 50% нового функционала ¹ 	<ul style="list-style-type: none"> • Покрытие от 50% до 80% нового функционала ¹ • % покрытия контролируется инструментами (SonarQube) 	<ul style="list-style-type: none"> • % покрытия > 80% нового функционала ¹ • Модульные тесты создаются до написания кода
Статический Анализ Кода	<ul style="list-style-type: none"> • Отсутствует, развернут, но не используется или носят фиктивный характер 	<ul style="list-style-type: none"> • Запускает по требованиям и используется 	<ul style="list-style-type: none"> • Запускается автоматически при сборке. • Использование централизованного инструмента статического анализа кода 	<ul style="list-style-type: none"> • Созданы критерии прохождения тестов при сборке 	<ul style="list-style-type: none"> • Настроенные правила для языков, отличных от Java
Анализ на ИБ	<ul style="list-style-type: none"> • Отсутствует, развернут, но не используется или носят фиктивный характер 	<ul style="list-style-type: none"> • Запускается по требованию и используется 	<ul style="list-style-type: none"> • Результаты проверки – задачи по ИБ в Jira 	<ul style="list-style-type: none"> • Запускается автоматически в конвейере DevOps • Использование централизованного инструмента анализа на ИБ 	<ul style="list-style-type: none"> • Используется «Безопасная разработка (Secure-SDLC)»

Модель зрелости DevOps 2/5

Уровень зрелости	Отсутствует (0)	Начальный (1)	Средний (2)	Продвинутый (3)	Экспертный (4)
Сборка	<ul style="list-style-type: none"> Сборка производится вручную Дистрибутивы хранятся локально Отсутствует версионный контроль дистрибутивов 	<ul style="list-style-type: none"> Автосборка кода из VCS Для каждой среды – своя сборка Централизованное хранение дистрибутивов 	<ul style="list-style-type: none"> Единая сборка для всех сред Проводится регулярное review качества и скорости процедур сборки Сборка пакета для БД на целевом инструменте 	<ul style="list-style-type: none"> Использование централизованного инструмента CI 	<ul style="list-style-type: none"> Результаты unit тестов на сборку влияют на завершение сборки Результаты стат. анализа кода на сборку или последующие этапы DevOps влияют на завершение сборки Результаты тестов ИБ на сборку или последующие этапы DevOps влияют на завершение сборки
Развертывание на СТ / НТ / ИБ / ИФТ	<ul style="list-style-type: none"> Используются статические среды, выделяемые по запросу Развертывание производится в ручном режиме 	<ul style="list-style-type: none"> Автоматическое развертывание для бинарников, конфигураций и настроек Ручные smoke-тесты после развертывания 	<ul style="list-style-type: none"> Автоматическое единообразное (вкл ПРОМ) развертывание для всех сред 	<ul style="list-style-type: none"> Единый оркестратор развертывания (с подтверждением администратора) Infrastructure as Code (конфигурация инфраструктуры хранится в VCS, применяется автоматически с помощью инструмента авторазвертывания) Автопроверка установки через smoke-тесты 	<ul style="list-style-type: none"> Динамические среды по потребностям Полностью автоматическое развертывание вкл Database Автопроверка установки через smoke-тесты (вкл. ПРОМ) Использование инструмента управления релизами и его интеграция в процесс развертывания, с целью управления последовательностью развертывания АС/ФП с учетом совместимости версий релизов и наличием интегрированной процедуры отката. Формирование и развертывание ПИР по одной кнопке

Модель зрелости DevOps 3/5

Уровень зрелости	Отсутствует (0)	Начальный (1)	Средний (2)	Продвинутый (3)	Экспертный (4)
Тестирование на СТ	<ul style="list-style-type: none"> Полностью ручное тестирование или тестирование не выполняется Нет единой системы управления тестированием 	<ul style="list-style-type: none"> Автоматизировано функциональное тестирование Автоматизация тестов API по запускам от 10% до 20% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 10% до 20% общего числа тесткейсов ¹ Мгновенная обратная связь в разработку Все тесты ведутся через единую систему управления тестированием Автоматический запуск тестов инструментом CD 	<ul style="list-style-type: none"> Автоматизация тестов API по запускам от 20% до 40% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 20% до 40% общего числа тесткейсов ¹ 	<ul style="list-style-type: none"> Используются интеграционные «тесты на заглушках» Автоматизация тестов API по запускам от 40% до 60% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 40% до 60% общего числа тесткейсов ¹ 	<ul style="list-style-type: none"> Автоматизация тестов API по запускам от 60% до 80% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 60% до 80% общего числа тесткейсов ¹
Тестирование на ИФТ	<ul style="list-style-type: none"> Полностью ручное тестирование или тестирование не выполняется Нет системы упр. Тестированием 	<ul style="list-style-type: none"> Автоматизировано функциональное и интеграционное тестирование Автоматизация тестов API по запускам от 20% до 40% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 10% до 20% общего числа тесткейсов ¹ Все тесты ведутся в Системе управления тестированием Автоматический запуск тестов инструментом CD 	<ul style="list-style-type: none"> Автоматизация тестов API по запускам от 40% до 60% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 20% до 40% общего числа тесткейсов ¹ 	<ul style="list-style-type: none"> Автоматизация тестов API по запускам от 60% до 80% общего числа тесткейсов ¹ Автоматизация тестов GUI по запускам от 40% до 60% общего числа тесткейсов ¹ 	<ul style="list-style-type: none"> Автоматизация тестов GUI по запускам от 60% до 80% общего числа тесткейсов ¹

Модель зрелости DevOps 4/5

Уровень зрелости	Отсутствует (0)	Начальный (1)	Средний (2)	Продвинутый (3)	Экспертный (4)
Тестирование на НТ	<ul style="list-style-type: none"> Полностью ручное НТ 	<ul style="list-style-type: none"> Автоматическая подготовка среды НТ с помощью централизованного инструмента развертывания 	<ul style="list-style-type: none"> Автоматическая подготовка данных с помощью централизованного инструмента оркестратора процессов НТ Автоматический сбор результатов НТ с помощью централизованного инструмента развертывания 	<ul style="list-style-type: none"> Использование централизованного оркестратора процессов НТ Автоматический запуск НТ с помощью централизованного инструмента развертывания 	<ul style="list-style-type: none"> Автоматический анализ результатов НТ с помощью централизованного инструмента развертывания Автоматическое заключение успешности НТ с помощью централизованного инструмента развертывания
Развертывание на ПСИ / HotFix	<ul style="list-style-type: none"> Используются статические среды, выделяемые по запросу Развертывание ППО производится в ручном режиме ФПД на файловом ресурсе Подпись дистрибутива в ручном режиме 	<ul style="list-style-type: none"> Автоматическая установка ППО скриптом, запускаемым вручную Выгрузка из централизованного хранилища дистрибутивов в файловую систему ФПД по требованию Ручные smoke-тесты после развертывания 	<ul style="list-style-type: none"> Автоматическая дистрибуция бинарных файлов ППО на сервер для установки Автоматическая установка ППО, с применением единообразных (для всех сред) сценариев развертывания Автоматическая настройка СПО скриптом, запускаемым вручную ФПД использует общее централизованное хранилище дистрибутивов Автоматическая передача дистрибутивов в область ФПД 	<ul style="list-style-type: none"> Используются статические среды, расширяемые по запросу, настраиваемые с помощью Infrastructure as Code (конфигурация СПО хранится в VCS, применяется автоматически с помощью инструмента авторазвертывания) Автопроверка установки через smoke-тесты Автоматическое подписание дистрибутива подписью в «Nexus» Единый оркестратор развертывания (с подтверждением администратора) 	<ul style="list-style-type: none"> Используются статические среды, динамически расширяемые по потребностям, настраиваемые с помощью Infrastructure as Code (конфигурация СПО хранится в VCS, применяется автоматически с помощью инструмента авторазвертывания) Полностью автоматическое развертывание ППО вкл Database Использование инструмента управления релизами и его интеграция в процесс развертывания, с целью управления последовательностью развертывания АС/ФП с учетом совместимости версий релизов и наличием интегрированной процедуры отката. Формирование и развертывание ПИР по одной кнопке

Модель зрелости DevOps 5/5

Уровень зрелости	Отсутствует (0)	Начальный (1)	Средний (2)	Продвинутый (3)	Экспертный (4)
ПСИ	<ul style="list-style-type: none"> Формализованная ПМИ отсутствует. ПСИ проводится «по понятиям» 	<ul style="list-style-type: none"> Сценарии ПМИ ведутся и хранятся внесистемно в виде файлов, бумажных и электронных документов 	<ul style="list-style-type: none"> Сценарии ПМИ хранятся в системе управления тестированием 	<ul style="list-style-type: none"> Производится верификация сценариев ПМИ для каждого релиза. Сценарии переиспользуются при смежных изменениях Разработан базовый набор функциональных приемочных тестов применимый на всех средах (в т.ч. СТ/ИФТ, опционально среди разработки) 	<ul style="list-style-type: none"> Сценарии ПМИ адаптируются после каждого их использования и на основании выявленных дефектов ПРОМ.
Развертывание на ПРОМ	<ul style="list-style-type: none"> ПРОМ среда отсутствует или выполняется полностью ручное развертывание 	<ul style="list-style-type: none"> Автоматическая установка ППО скриптом, запускаемым вручную 	<ul style="list-style-type: none"> Автоматическая дистрибуция бинарных файлов ППО на сервер для установки Автоматическая установка ППО, с применением единообразных (для всех сред) сценариев развертывания Автоматическая настройка СПО скриптом, запускаемым вручную Ручные smoke-тесты 	<ul style="list-style-type: none"> Автопроверка установки через smoke-тесты Единый оркестратор развертывания (с подтверждением администратора) Infrastructure as Code (конфигурация инфраструктуры хранится в VCS, применяется автоматически с помощью инструмента авторазвертывания) Используются статические среды, настраиваемые с помощью Infrastructure as Code (конфигурация СПО хранится в VCS, применяется автоматически с помощью инструмента авторазвертывания) 	<ul style="list-style-type: none"> Полностью автоматическая установка ППО на ПРОМ вкл Database Автоматическая дефектовка с передачей дефекта в систему управления дефектами Автоматический откат

Перспективы развития DevOps

- Расширение и концентрация экспертизы по CI и CD
- Контейнеризация приложений (Docker и инструмент управления)
- Интеграция управления релизами с сервисом CD
- Встраивание/интеграция процесс согласования в сервис CD
- Наращивание аналитических возможностей сервиса CI, CD и DPD
- Наращивание объем технологический метрик DevOps (метрики для DevOps-инженеров) DPD
- Использование динамических сред ВЕЗДЕ, в т.ч. на ПРОМе

Технологические ограничения

Следующие технологические ограничения АС препятствуют автоматизации практик DevOps в полном объеме и требуют устранения:

- Имеются параметры, которые зависят от среды, хранятся в коде приложения и требуют пересборки приложения для каждой среды
- Имеются ограничения, препятствующие автосборке
- Имеются архитектурные ограничения, препятствующие разработки Модульных (Unit) тестов
- Имеются архитектурные ограничения, препятствующие выполнению автотестов
- Требуется ручная настройка справочников / бизнес-правил / настроек через интерфейс UI для переноса на новую среду
- Отсутствуют API конфигурирования системного программного обеспечения (серверов приложений, баз данных, адаптеров)