



ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ЛЕКЦИЯ № 12

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.



ЧТО ТАКОЕ НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ (CI)?

- Непрерывная интеграция (CI — Continuous integration) предполагает, что все, кто участвует в проекте разработки ПО, регулярно публикуют сделанные изменения в кодовой базе в центральном репозитории. Обычно в проекте задействовано несколько разработчиков, поэтому очень важно хранить весь код, над которым идет работа, централизованно. В идеале объединение должно выполняться автоматически несколько раз в день. Цель непрерывной интеграции — обеспечить стабильность разработки и выпуска ПО за счет совместной работы, автоматизации и быстрой обратной связи.
- Внедрение непрерывной интеграции начинается с регулярной отправки изменений в систему управления версиями / исходным кодом, чтобы все участники проекта работали с одинаковым кодом. Каждый коммит становится триггером для сборки и серии автоматизированных тестов, чтобы проверить поведение кода и убедиться, что изменение ничего не сломало. Непрерывная интеграция полезна и сама по себе, а еще она становится первым шагом к реализации CI/CD-пайплайна.



ОСНОВНЫЕ ЭЛЕМЕНТЫ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ:

- источник или система контроля версий с единой кодовой базой, включая файлы исходного кода, библиотеки, файлы конфигурации и скрипты;
- автоматизированные билд-скрипты;
- автоматизированные тесты;
- инфраструктура для выполнения сборки и тестов.

ПРАКТИЧЕСКОЕ ИСПОЛЬЗОВАНИЕ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ

- Чтобы все участники проекта работали с одинаковым кодом, они должны использовать один репозиторий и регулярно публиковать свои изменения. Практика показывает, что каждый участник должен отправлять изменения в основную ветку не реже раза в день.
- Следующий шаг после отправки изменений — сборка решения и проведение автоматизированных тестов, чтобы проверить поведение кода. Автоматизация этого процесса — неотъемлемая часть непрерывной интеграции. Если сборка и тестирование осуществляются вручную, это отнимает много времени и может вести к ошибкам. В результате ежедневная интеграция изменений теряет смысл. Конкретные инструменты сборки и фреймворки тестирования зависят от рабочего языка программирования.

- После настройки скриптов и тестов необходимо контролировать и при необходимости обновлять процесс. Сюда входит добавление автоматизированных тестов при появлении любых новых функций, устранение сбоев и отслеживание производительности.
- Если вы добавите сервер непрерывной интеграции, который осуществляет мониторинг репозитория, запускает сборку, выполняет автоматизированные тесты и создает отчеты о результатах, это поможет собрать вместе все фрагменты и сэкономит время на написание собственной логики автоматизации. Кроме того, вы получите дополнительную информацию, например, метрики покрытия кода и историю сборок.
- Но, несмотря на важную роль инструментов и процессов, максимальную пользу из непрерывной интеграции удастся извлечь, только если все люди будут на практике следовать ее принципам. Необходимо пересмотреть рабочие процессы команды, включив в них регулярную отправку изменению в основную ветку, добавление автоматизированных тестов для всех новых функций и приоритетное устранение проблем со сборкой, когда что-то идет не так. Чтобы преодолеть организационную разобщенность, нужно не забывать о совместной работе с тестировщиками для приоритетной подготовки и бесперебойного выполнения автоматизированных тестов, а также о сотрудничестве со специалистами, отвечающими за инфраструктуру, — они помогут получить машины для выполнения сборок и тестов.

ПРЕИМУЩЕСТВА НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ

Внедрение непрерывной интеграции позволяет сократить цикл разработки без ущерба для качества кода. Основная задача непрерывной интеграции — снизить возможные риски, связанные с развертыванием ПО, и сократить цикл обратной связи.

Вот основные преимущества непрерывной интеграции:

- **Минимум риска при развертывании.** Благодаря непрерывному объединению изменений по мере написания кода любые возможные ошибки удастся обнаружить на ранних этапах работы.
- **Повышение качества.** Автоматизация значительной части ручных тестов позволяет разработчикам сосредоточиться на тестировании более высокого уровня.
- **Снижение расходов.** Внедряя непрерывную интеграцию с последующей доставкой ПО пользователям небольшими пакетами, а также автоматизируя значительную часть работы, компании могут значительно снизить расходы на доставку ПО.

ЧТО ТАКОЕ НЕПРЕРЫВНОЕ РАЗВЕРТЫВАНИЕ (CD)?

- Именно в непрерывном развертывании DevOps-технология автоматизации сборки, тестирования и развертывания получила свое наибольшее логическое развитие. Если изменение успешно проходит все предыдущие стадии пайплайна, оно автоматически (без ручных вмешательств) попадает в продакшн. Непрерывное развертывание позволяет быстро доставлять пользователю новую функциональность и не жертвовать при этом качеством.
- Непрерывному развертыванию предшествует стадия непрерывной интеграции с тщательным тестированием и стадия непрерывной доставки. Разработчики делают регулярные коммиты небольших изменений в основную ветку (master), после чего выполняется автоматическая сборка и тестирование, развертывание на препродакшн-окружения, и, если проблем не обнаруживается, изменения наконец попадают в продакшн. С устойчивым и надежным пайплайном непрерывного развертывания релизы становятся обычным делом, вы сможете выполнять их по несколько раз в день.
- И хотя автоматизация финального выпуска в продакшн подойдет не всем проектам, вам могут пригодиться отдельные шаги, реализующие непрерывное развертывание. В этой статье мы рассмотрим, из чего состоит этот процесс и обсудим моменты, которые следует учесть, если вы решаете сделать финальный шаг в мир полной непрерывности.




РЕАЛИЗАЦИЯ

- Если ваши процессы интеграции и развертывания выполняются вручную, вы периодически выполняете заморозку кода, тестирование проводится коллективно, а день релиза вся компания переживает затаив дыхание — ежечасное едва заметное развертывание может показаться фантастикой.

- Однако на практике оказывается, что к этому подходу прибегает множество организаций, от гигантов вроде Netflix, Etsy и Amazon до небольших компаний, старающихся идти в ногу с рынком. Реализовав непрерывное развертывание, эти компании смогли ускорить процесс релиза: вместо недель (и даже месяцев) он теперь занимает всего несколько часов. Возможность быстро выпускать новую функциональность и оперативно реагировать на обратную связь начинает играть большое значение во многих отраслях.
- Продолжая процедуры непрерывной интеграции и доставки, непрерывное развертывание основывается на полностью автоматизированном процессе сборки, тестирования и развертывания — именно это позволит вам быть уверенными, что вы не жертвуете ради скорости качеством. Однако это лишь фундамент для построения эффективной реализации непрерывного развертывания.
- Ключевым вопросом при планировании реализации непрерывного развертывания является то, как именно будут выпускаться изменения. Помимо использования плавающих релизов (вместо того, чтобы выключать серверы, чтобы не допустить перебоев в работе онлайн-сервисов), вы также можете сделать выпуск своего рода расширением процедуры автоматизированного тестирования.

- Канареечное развертывание делает изменения доступными лишь для небольшой доли пользователей, делая их таким образом невольными тестировщиками системы в продакшне. Проследив их поведение и метрики использования, вы сможете убедиться, что ваш релиз не привнес новых ошибок, после чего можно будет выпустить обновление для остальных пользователей.
- Некоторые компании, продолжая развивать автоматизацию, ввели для канареечного релиза доверительный показатель, который автоматически сравнивает множество метрик с их базисными значениями. Выпуск продолжит выполняться автоматически, пока показатель превышает определенный порог, а анализ метрик будет создавать отправные точки для дальнейшего исследования возможных проблем.
- Сине-зеленое развертывание распространено в организациях, использующих непрерывное развертывание, поскольку оно упрощает процедуру отката к предыдущей версии, необходимого на случай возникновения проблем: для этого старый код продолжает храниться в продакшне, пока вы не убедитесь, что все изменения работают должным образом. Если нужно, вы можете выполнить канареечное развертывание с последующим сине-зеленым выпуском.

- 
- Независимо от того, делаете ли вы сине-зеленое развертывание или выпускаете версии на замену, если вы хотите иметь возможность быстро реагировать на ошибки, проскользнувшие через процесс релиза, вам необходимо следить за состоянием системы в продакшне.
 - Следя за метриками, отражающими состояние вашей системы (начиная с дискового пространства и использования процессора, заканчивая количеством запросов и транзакций), и сравнивая их со стандартом, вы будете раньше узнавать о проблемах в поведении ПО. Далее вы сможете выбрать между откатом к предыдущей версии и исправлением проблемы (с прохождением новой версии через пайплайн).

НА ЧТО ОБРАТИТЬ ВНИМАНИЕ

- Приступая к реализации непрерывного развертывания, вы также должны знать о тех проблемах, которые могут возникнуть.
- Процесс разработки включает не только изменения в коде. Команды, занимающиеся исследованием пользовательского поведения, маркетингом, дизайном взаимодействия, составлением документации, поддержкой, а также коммерческий и юридический отделы — все они тоже принимают участие в процессе разработки.
- Если вы не подготовили почву вместе с коллегами и не поинтересовались, какие у них требования к процессу релиза, у них может возникнуть ощущение, что переход на непрерывное развертывание выведет разработку из-под контроля. В ответ они могут ввести ручные проверки и стадии ревью, которые замедлят процесс, либо и вовсе посчитать непрерывное развертывание неудачным экспериментом и отменить его.

- Важно создать культуру сотрудничества. Вовлечение других команд в процесс разработки, использование их вклада (в проектирование, вопросы безопасности, терминологию или соответствие требованиям) на ранних стадиях процесса — еще один пример того, как короткие циклы обратной связи делают процесс разработки более эффективным. Помимо привлечения команд к участию в процессе, важно обеспечить им обозримость процесса (что выпускается и когда). Информировать коллег можно автоматически с помощью CI-сервера — инструмента непрерывного развертывания, который будет распространять информацию через панели мониторинга и уведомления.
- Иногда обозримости процесса может быть недостаточно. Если вы работаете над большим объемом новой функциональности или вам нужно контролировать сроки релиза, выполнять развертывание в продакшн каждого коммита, прошедшего все тесты, — не совсем идеальный вариант.
- Эту проблему решают флаги функций, позволяющие контролировать, будет ли пользователя доступен тот или иной фрагмент кода или нет: при этом код будет находиться в продакшне, и вы сможете следить за его поведением. Другой подход заключается в использовании специальных веток, развертывание которых выполнялось бы в отдельных пайплайнах, не публикующих изменения в продакшн, — своего рода объединение технологий непрерывной доставки и непрерывного развертывания.

ЛУЧШИЕ ПРАКТИКИ CI/CD

Непрерывная интеграция, доставка и развертывание позволяют компаниям снизить расходы и существенно сократить цикл доставки программного обеспечения. При правильной реализации они значительно повышают эффективность разработки, тестирования и выпуска ПО. В число лучших практик CI/CD входят:

- **Делать коммиты как можно раньше и чаще.** Частая публикация небольших фрагментов кода позволяет получать и отслеживать обратную связь по каждому изменению. По данным отчета «State of DevOps», команды, показывающие высокую производительность, делают в 417 раз больше развертываний, чем те, у кого производительность низкая.
- **Сборки должны быть зелеными.** Внедрение в рамках CI/CD-пайплайна автоматизированных тестов, которые запускаются при каждом коммите нового кода, обеспечивает разработчикам быструю обратную связь по всем изменениям.
- **Пайплайн — единственный путь к развертыванию в продакшн.** Если команде нужно быстро выпустить ПО, очень хочется пропустить некоторые этапы продуманного CI/CD-процесса. Однако следование утвержденному порядку поможет избежать ошибок, которые в противном случае могут попасть в кодовую базу.

ЛУЧШИЕ ПРАКТИКИ НЕПРЕРЫВНОГО РАЗВЕРТЫВАНИЯ

- При правильном подходе непрерывное развертывание может помочь командам автоматизировать развертывание программного обеспечения. Следование современным практикам непрерывного развертывания позволит вам оптимизировать процесс и добиться лучших результатов. Например, важно регулярно следить за работой пайплайна и использовать метрики для своевременного обнаружения проблем. Также имеет смысл привлекать к построению эффективного CI/CD-пайплайна всю команду.

ДЕЛАЙТЕ КОММИТЫ РАНО И ЧАСТО

- Первое, чего требует реализация непрерывной интеграции, — это размещение всего исходного кода, конфигурационных файлов, скриптов, библиотек и исполняемых файлов в системе контроля версий/ Это позволит вам отслеживать изменения.
- Однако недостаточно просто завести инструмент — важно то, как вы будете им пользоваться. Чтобы упростить процесс интеграции изменений от нескольких контрибьюторов, непрерывная интеграция предлагает публиковать небольшие изменения, но зато делать это чаще.
- Каждый коммит запускает набор автоматизированных тестов, которые быстро дают вам обратную связь. При регулярных коммитах вся команда будет работать с одними и теми же исходными данными, а значит вам будет легче сотрудничать и реже придется разрешать конфликты при слиянии крупных и комплексных изменений.

- Чтобы получить максимум пользы от непрерывной интеграции, важно, чтобы все разработчики публиковали свои изменения в основную ветку (master) и обновляли свою рабочую версию, подгружая изменения остальных. Общее правило: старайтесь делать коммит в master минимум раз в день.
- Такие частые публикации в основную ветку могут показаться неудобными, если ваша команда привыкла работать с долгосрочными ветками. Люди могут бояться преждевременной оценки коллег, а объем некоторых задач может не укладываться в один день.
- Важно создать в команде культуру сотрудничества, а не осуждения. Полезно обсудить, как именно будет происходить работа команды. Вместе разбивая задачи на более мелкие и дискретные, ваша команда сможет быстрее усвоить эту практику.
- Долгосрочные ветки используются для хранения новой функциональности, которая пока не готова к релизу. Для этой цели можно также использовать флаги функций. Они позволяют контролировать видимость той или иной функциональности в разных окружениях. Вы сможете включать изменения в основную ветку и в сборки с тестами, при этом скрывая соответствующую функциональность от пользователя.

ПОДДЕРЖИВАЙТЕ СБОРКИ ЗЕЛЕНЫМИ

- Собирая решение и запуская набор автоматизированных тестов для каждого коммита, CI/CD-пайплайн дает разработчикам быструю обратную связь по их изменениям.
- Цель — постоянно держать код в состоянии, пригодном для релиза. Решать проблемы сразу по возникновении — это не только более эффективный подход, но он также позволит вам быстро выпускать изменения в случае, если возникнет проблема в продакшне.
- Если сборка по какой-либо причине падает, команда должна сразу заняться решением проблемы. Возникает желание обвинить того, кто внес последнее изменение в код, и оставить этого человека разбираться с проблемой. Однако обвиняя таким образом коллег, вы вряд ли создадите в команде культуру созидания, при этом причины проблем могут так и оставаться невыясненными. Возлагая ответственность за исправление сборки и выяснение причин падения на всю команду, вы сможете улучшить CI/CD-процесс в целом. Конечно, на деле, когда испытывается высокое давление и напряжение, это может оказаться не так просто. Развитие DevOps-культуры — это упражнение, которое тоже требует постоянного совершенствования.


- Представьте, что вы отвлекаетесь от своей работы, принимаетесь искать причину падения и в конце концов выясняете, что оно было вызвано чем-нибудь совсем тривиальным — синтаксической ошибкой или пропущенной зависимостью. Такое может раздражать. Чтобы таких ситуаций не возникало, можно поручить членам команды выполнять сборку и базовый набор тестов локально и только после этого публиковать свои изменения. В идеале, у всех должна быть возможность использовать в качестве CI/CD одни и те же скрипты — так никому не придется делать лишнюю работу.

СОБИРАЙТЕ ОДИН РАЗ

- Типичной ошибкой является создание новой сборки для каждого шага CI/CD.
- Пересобирая приложение для разных окружений, вы рискуете нарушить консистентность и не будете знать наверняка, было ли тестирование на предыдущих шагах успешным. Поэтому на протяжении всех шагов CI/CD-пайплайна (включая конечный релиз в продакшн) необходимо использовать один и тот же артефакт.
- Чтобы реализовать это, нужно сделать сборки независимыми от окружения. Любые переменные, параметры аутентификации, конфигурационные файлы и скрипты должны вызываться скриптом развертывания и не быть частью самой сборки. Это позволит делать развертывание одного и того же артефакта в каждом тестовом окружении. Тогда прохождение каждой стадии будет повышать уверенность команды в этом артефакте.
- В отличие от скриптов сборки, конфигурационных файлов и скриптов развертывания, хранить которые лучше в единой системе контроля версий, артефакты сборки должны храниться отдельно. Все эти данные являются входными по отношению к процессу сборки, и конечный продукт не должен принадлежать системе контроля версий. Вместо этого сборке должна быть присвоена версия, после чего она сохраняется в центральный репозиторий артефактов, например Nexus, откуда ее всегда можно достать, чтобы выполнить развертывание.

ОПТИМИЗИРУЙТЕ ТЕСТЫ

- CI/CD в значительной мере опирается на автоматизированное тестирование — оно дает уверенность в качестве разрабатываемого ПО. Однако это не значит, что вам нужно стремиться протестировать каждый возможный сценарий.
- Цель CI/CD — обеспечить вам быструю обратную связь и доставлять программное обеспечение пользователям быстрее, чем это возможно с традиционными методами. А это значит, что нужно соблюдать баланс между тестовым покрытием и производительностью. Если тестирование выполняется слишком долго, люди будут искать возможность обойти эту процедуру.
- Вначале запускайте те тесты, которые выполняются быстрее всего, чтобы как можно скорее получить первую порцию обратной связи. Более длительные тесты можно будет выполнить тогда, когда вы уже будете достаточно уверены в своей сборке. Что касается ручных тестов, учитывая то, что они выполняются долго и требуют привлечения коллег, лучше выполнять эту фазу тестирования после того, как у вас будут зеленые авто-тесты.

- 
- Первой прослойкой обычно выступают юнит-тесты. Ими можно обеспечить широкое покрытие, и они смогут указать вам на очевидные проблемы во вносимых изменениях. Вслед за юнит-тестами у вас может быть прослойка автоматизированных интеграционных или компонентных тестов, проверяющих взаимодействие между различными частями вашего кода.
 - Также вы можете вложиться в создание более сложных автоматизированных тестов (например, тестов GUI, производительности или нагрузки). После этого можно заниматься ручным исследовательским и/или приемочным тестированием. Все эти виды тестов (будь они автоматизированными или ручными) более длительны. Чтобы достичь эффективности, вам нужно сосредоточиться на вещах, которые представляют наибольший риск для вашего продукта и пользователей.

ЧИСТИТЕ ВАШИ ОКРУЖЕНИЯ

- Чтобы получить максимум пользы от тестирования, стоит уделять время чистке пре-продакшн окружений перед каждым развертыванием.
- Когда среды работают слишком долго, отслеживать изменения конфигураций становится сложнее.
- Со временем окружения отклоняются от первоначальных настроек и начинают отличаться друг от друга. А это значит, что тесты, запущенные в разных окружениях, могут выдавать разные результаты. Статические окружения требуют поддержки — это может замедлять тестирование и задерживать процедуру релиза.
- Для создания окружений и запуска в них тестов можно использовать контейнеры. Они позволяют с легкостью настраивать и сбрасывать окружения каждый раз, когда вам необходимо выполнить развертывание: для этого используется скрипт, фиксирующий все необходимые шаги (подход Infrastructure as Code). Создавая новый контейнер под каждое развертывание, вы будете соблюдать консистентность. Также с контейнерами легче масштабировать окружения, поэтому при необходимости вы сможете тестировать несколько сборок параллельно.

НЕ ПОЗВОЛЯЙТЕ ДЕЛАТЬ РАЗВЕРТЫВАНИЕ ОБХОДНЫМИ ПУТЯМИ

- Допустим, вы построили надежный, быстрый и безопасный CI/CD-пайплайн, который действительно позволяет вам быть уверенными в качестве сборок. Но ваша работа легко обесценится, если вы будете разрешать людям идти в обход процессу по каким бы то ни было причинам.
- Как правило, просьбы обойти процесс релиза поступают тогда, когда изменения небольшие либо срочные (иногда и то, и другое), но соглашаясь на них, вы оказываете медвежью услугу.
- Пропуская авто-тесты, вы рискуете остаться с проблемами, которые вполне можно было бы отследить. А воспроизводить ошибки и делать отладку намного сложнее, поскольку нет возможности взять готовую сборку и развернуть ее в тестовом окружении.
- Вероятно, в какой-то момент вас попросят обойти эту процедуру («ну только в этот раз!»). Вы же в этот момент, скорее всего, будете, как пожарный, всех спасать. Однако в будущем полезно поднять этот вопрос на ретроспективе или на post-mortem и разобраться, почему так произошло. Слишком ли долгий процесс? Возможно, нужно поработать над улучшением производительности. Есть ли непонимание относительно того, когда использовать CI/CD? Убедив ваших коллег в преимуществах CI/CD-пайплайнов, вы уже не столкнетесь с такого рода просьбами, если снова случится форс-мажор.

ОТСЛЕЖИВАЙТЕ И АНАЛИЗИРУЙТЕ ВАШ ПАЙПЛАЙН

- Создавая CI/CD-пайплайн, вы скорее всего будете реализовывать способ мониторинга продакшн-окружения, чтобы иметь возможность как можно раньше отслеживать проблемы.
- Вашему CI/CD-пайплайну, как и вашему ПО, необходим цикл обратной связи.
- Анализируя метрики, собранные вашим CI/CD-инструментом, вы сможете выявлять потенциальные проблемы и области, требующие улучшения.
- Сравнивая количество сборок, запускаемых в неделю, день или час, вы поймете, каким образом используется инфраструктура пайплайна, нужно ли ее масштабировать и когда случается пик нагрузки.
- Следя за скоростью развертывания (проверяя, не падает ли она), вы будете знать, пора ли заняться оптимизацией производительности или еще нет.
- Статистика по автоматизированным тестам поможет определить, имеет ли смысл что-то выполнять параллельно.
- Пересматривая результаты тестов и находя те, которые систематически пропускаются, вы будете знать, как оптимизировать тестовое покрытие.

РАБОТАЙТЕ ВСЕЙ КОМАНДОЙ


- Создание эффективного CI/CD-пайплайна требует не только подходящих процессов и инструментов, но и командной и организационной культуры.
- Непрерывная интеграция, доставка и развертывание — это DevOps-практики. Они устраняют традиционную разобщенность между разработкой, тестированием и операционной деятельностью и способствуют их коллаборации между специалистами.
- Устранение разобщенности помогает командам лучше обзирать процесс, дает им возможность сотрудничать и объединять разные области знаний. Поддержкой пайплайна не должен заниматься один человек.
- Ощущая общую ответственность за доставку продукта, свой вклад смогут внести все члены команды: кто-то починит сборку, кто-то переведет окружения в контейнеры, кто-то автоматизирует ручную задачу, чтобы ее можно было выполнять чаще, и т.д.
- Культура доверия, при которой члены команды могут экспериментировать и делиться идеями, поможет не только сотрудникам, но и всей организации, и окажет положительный эффект на ваш продукт. Если что-то идет не так, не нужно обвинять в этом членов вашей команды; вместо этого стремитесь извлекать уроки из произошедшего, разбираться в причинах проблем и в том, как их избежать в будущем.
- Не упускайте возможность улучшить свои CI/CD-процессы, сделать их устойчивее и эффективнее. Позволяя членам команды экспериментировать и создавать новое, не опасаясь ничьих упреков, вы поощрите культуру созидания и непрерывного совершенствования.

ЧТО ТАКОЕ СЕРВЕР НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ?

- Реализация CI/CD требует особого подхода к культуре труда для налаживания взаимодействия между различными командами и внедрения новых рабочих процессов. Помимо этого, необходимы инструменты для автоматизации различных этапов и обеспечения эффективной работы пайплайна.
- Ключевую роль в этом играет сервер непрерывной интеграции (или билд-сервер). Это связующее звено, которое на основе вашей бизнес-логики объединяет все этапы пайплайна и координирует выполнение автоматизированных заданий, а также собирает данные и дает обратную связь.

ИНТЕГРАЦИЯ С СИСТЕМОЙ КОНТРОЛЯ ВЕРСИЙ

- Любой CI/CD-пайплайн начинается с интеграции с системой контроля версий.
- В рамках базовой реализации сервер непрерывной интеграции настраивается на отслеживание коммитов, которые попадают в ветку master. В случае наличия изменения происходит запуск пайплайна.
- И хотя при этом каждый коммит проверяется и тестируется, остается высокая вероятность, что кто-то отправит код, который сломает сборку, весь процесс прервется и другие изменения не будут проверяться, до тех пор пока проблемный код не будет удален или исправлен.
- Настройка сервера непрерывной интеграции для выполнения сборки и тестирования изменений перед их отправкой помогает предотвратить такого рода проблемы и обеспечивает дополнительный цикл обратной связи для каждого разработчика.

- 
- Важно отметить, что билд-сервер не только управляет выполнением сборки и тестов на удаленной машине и передает их результаты разработчику, но и делает это условием для отправки изменений в основную или функциональную ветку.
 - Кроме того, вы можете рассмотреть возможность интеграции сервера непрерывной интеграции с вашим инструментом для код-ревью, чтобы перед отправкой каждый коммит обязательно проходил код-ревью и отправлялся только в случае успешного выполнения сборки и тестирования.
 - Благодаря реализации этих дополнительных уровней бизнес-логики в начале процесса, поддерживается чистота кодовой базы и обеспечивается ее готовность к выпуску, при этом минимизируются прерывания и задержки в пайплайне.

УПРАВЛЕНИЕ СБОРКАМИ

- Когда дело доходит до запуска сборки и тестирования в рамках CI/CD-пайплайна, ваш CI-сервер становится центральным звеном, координирующим выполнение заданий и распределяющим работу по билд-агентам на основе различных критериев.
- Ваши билд-агенты берут на себя работу по запуску сборок и выполнению тестов в соответствии с инструкциями, полученными от сервера непрерывной интеграции.
- Рекомендуется отделять сервер непрерывной интеграции от билд-агентов, которые запускают сборки и выполняют тесты, по меньшей мере в продакшн-конфигурации, во избежание возникновения конфликта доступа к ресурсам и проблем с производительностью.
- При использовании сервера непрерывной интеграции для организации логики этапа вашего пайплайна есть возможность задать особые условия и правила. Например, можно запускать определенные тесты коммитов в основную ветку, но не во время выполнения предкоммитной сборки на ветке разработки, или, скажем, контролировать количество сборок, которые могут одновременно обращаться к одной базе данных.

- Одновременный запуск определенных заданий с использованием разных билд-агентов может повысить эффективность вашего пайплайна. Это полезно, когда необходимо запустить тесты на разных операционных системах или когда вы работаете с огромной кодовой базой, а тесты исчисляются сотнями тысяч, в этом случае единственный рациональный выход — параллельное выполнение. Во втором случае использование композитной сборки позволит объединить результаты, чтобы можно было обрабатывать задания за один шаг сборки.
- CI-сервер, интегрированный с облачной инфраструктурой, такой как AWS, позволяет воспользоваться преимуществами гибких, масштабируемых ресурсов для выполнения сборок и тестов. Если у вас повышенные требования к инфраструктуре, поддержка контейнерных билд-агентов и интеграция с Kubernetes позволит эффективно распорядиться ресурсами сборки, независимо от их расположения — в облаке или в локальной инфраструктуре.

ОПРЕДЕЛЕНИЕ НЕУДАЧНОГО ЗАВЕРШЕНИЯ СБОРКИ

- Важнейшей составляющей вашей бизнес-логики является определение того, что считать неудачным завершением сборки на каждом этапе CI/CD-пайплайна.
- CI-сервер позволяет задавать различные условия неудачного завершения сборки, которые он затем применит и будет использовать для определения состояния того или иного этапа и принимать решение, можно ли перейти к следующему этапу пайплайна.
- Помимо очевидных неудачных завершений сборки (например, когда возвращается код ошибки или не выполняются тесты), можно задать и другие типы неудачных завершений на основе данных, собранных CI-сервером.
- Скажем, это может быть уменьшение покрытия кода тестами по сравнению с предыдущей сборкой (а значит, для очередных изменений кода не были добавлены тесты) или увеличение количества проигнорированных тестов по сравнению с последней успешной сборкой.
- На основании этих данных можно сделать вывод о возможном снижении качества кода. Задав эти метрики в качестве триггера для неудачного завершения сборки и ограничив число пользователей, которым разрешено игнорировать эти сбои, можно добиться нужного сценария выполнения сборки.

ИСПОЛЬЗОВАНИЕ НЕПРЕРЫВНОЙ ДОСТАВКИ

- Хотя название «сервер непрерывной интеграции» подразумевает, что его использование ограничивается лишь непрерывной интеграцией, большинство CI-серверов также поддерживают непрерывную доставку и развертывание.
- После формирования артефактов сборки и выполнения исходного набора тестов во время этапа непрерывной интеграции следующим шагом является развертывание этих артефактов в тестовых средах для дальнейшего тестирования. После этого следует проверка рабочей версии всеми заинтересованными сторонами. Если на этом этапе нареканий нет, то за ним следует релиз.
- CI-сервер не только обеспечивает репозиторий артефактов для хранения результатов каждой сборки, чтобы при необходимости их можно было развернуть, но и может хранить и регулировать параметры для каждой среды в пайплайне. В дальнейшем можно указать, должны ли ваши скрипты развертывания автоматически срабатывать в зависимости от результатов, полученных на предыдущем этапе.

НАБЛЮДЕНИЕ ЗА ХОДОМ ПРОЦЕССА

- Ключевой составляющей CI/CD-пайплайна является быстрая обратная связь по каждому этапу.
- Сервер непрерывной интеграции может обеспечивать информацию об очереди заданий, выдавать отчеты о сборках и тестах в режиме реального времени (прямо во время их выполнения), а также отображать состояние выполненных шагов сборки.
- Настроив уведомления, вы и ваша команда будете оперативно узнавать о любых событиях, при этом интеграция с баг-трекерами дает возможность видеть исправления, которые содержит коммит, и оперативно выяснять причину неудачного завершения сборки. Из статистических данных можно получить полезные сведения для оптимизации пайплайна, а также определения условий в логике пайплайна.

НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ ИЗ МИРА AGILE


- Гибкая методология разработки (Agile-разработка) была предшественником DevOps, а следовательно, и непрерывной интеграции, доставки и развертывания, которые вместе обозначаются сокращением CI/CD. Поэтому она тесно связана с этими понятиями. Если вы разберетесь с основными принципами Agile, это поможет эффективнее использовать CI/CD и создать CI/CD-пайплайн, реализующий методологию Agile.
- К сожалению, появившиеся за последние годы многочисленные методологии, стратегии и консультанты искажают основные положения Agile и сводят все к бездумному следованию набору правил. Без понимания принципов гораздо сложнее их эффективно применять.


ПРИНЦИПЫ AGILE-РАЗРАБОТКИ


- Agile — это в первую очередь мировоззрение, то есть общий взгляд на то, как следует разрабатывать программное обеспечение.
- Agile видит главную цель в том, чтобы создавать работающее ПО, и утверждает, что стимулирование изменений и сотрудничества между отдельными людьми позволяет эффективнее достигать этой цели, чем жесткое следование плану в соответствии с заданным набором требований.
- Принципы, изложенные в манифесте Agile, отражают эти ценности и предлагают способы их воплощения. Сюда входит создание профессиональных и готовых к сотрудничеству команд и доставка ПО на основе коротких циклов, позволяющих команде быстро реагировать на изменения.
- Логика проста: если мы раз и навсегда устанавливаем жесткие требования и строго следуем плану, то теряем гибкость и не можем адаптировать продукт к новым обстоятельствам по мере того, как лучше узнаем ситуацию, а также при изменении требований пользователей. Agile-подход предполагает наличие конечной цели и гибкий выбор путей для ее поэтапного достижения.

РЕАЛИЗАЦИЯ ПРИНЦИПОВ AGILE-РАЗРАБОТКИ С ПОМОЩЬЮ CI/CD

- Принципы Agile-разработки стали основой для формирования методологии DevOps, из которой в свою очередь выросла CI/CD.
- В центре внимания методологии Agile-разработки, по крайней мере, на первом этапе ее существования, была деятельность групп разработчиков. DevOps идет дальше и охватывает также последующие рабочие процессы — от написания кода до релиза.
- DevOps подчеркивает важность преодоления разобщенности и сотрудничества между разными командами для достижения единой цели: доставки пользователями удобного работающего ПО. Непрерывная интеграция, доставка и развертывание представляют собой практическую реализацию методологии DevOps для ускорения релиза ПО без ущерба для качества.

- 
- За счет автоматизации максимального числа шагов CI/CD позволяет быстро получать обратную связь по новым сборкам, сокращая время доставки программного обеспечения пользователям.
 - С учетом истории Agile и DevOps неудивительно, что многие элементы CI/CD-пайплайна помогут вам придерживаться принципов Agile-разработки. Начнем с того, что рекомендация «делать коммиты быстро и часто» стимулирует разделение работы на небольшие отрезки.
 - Разделение функций на небольшие части необходимо для цикличного процесса сборки и выпуска ПО. Цель заключается в том, чтобы после каждого коммита сборка могла пройти пайплайн и быть готова к релизу. Таким образом, каждый коммит должен приводить к созданию работающего продукта. Реализация такого подхода поможет сосредоточиться на создании работающего ПО.


- 
- И Agile, и DevOps подчеркивают важность совместной работы и коммуникации. Хотя первоначально в центре внимания DevOps была совместная работа разработчиков и операционных команд, преимущества этой методологии гораздо шире.
 - Включив предпроизводственные среды в CI/CD-пайплайн и контролируя изменения в сборках с помощью панелей мониторинга, можно делиться с другими командами и отделами — например, маркетологами, дизайнерами или специалистами по безопасности — информацией о ходе работе и запрашивать у них обратную связь.
 - Центральное место в любом CI/CD-пайплайне занимают автоматические тесты, которые позволяют быстро получить обратную связь по изменениям кода и гарантировать качество сборки. Для доставки работающего продукта очень важно выполнять автоматические тесты для каждого коммита.

- 
- Среди принципов Agile одно из первых мест занимает частая доставка ПО пользователям. CI/CD-пайплайн играет ключевую роль в реализации этого принципа. Автоматизация этапов процесса выпуска ПО позволила разработчикам ускорить работу и доставлять изменения каждый день или даже каждый час: гораздо чаще, чем можно было предположить в то время, когда писался манифест.
 - Наконец, непрерывные циклы получения обратной связи, которые и составляют суть CI/CD-пайплайна, обеспечивают непрерывное улучшение не только разрабатываемого ПО, но и самого рабочего процесса. Тем самым реализуется принцип Agile, который гласит: команда должна регулярно анализировать свою работу и при необходимости корректировать ее. Циклическое создание сборок и получение обратной связи помогает поддерживать стабильный темп работы, о чем говорится в манифесте.

ПОСТРОЕНИЕ AGILE-ОРГАНИЗАЦИИ

- Хотя создание CI/CD-пайплайна может способствовать развитию образа мышления в духе Agile, это тоже не панацея, как и основанные на принципах Agile методологии и решения, возникшие за время существования Agile.
- При этом в CI/CD регулярно встречаются такие схемы работы, которые очень далеки от настоящего Agile.
- Частым препятствием на пути эффективной работы CI/CD-пайплайна и внедрения принципов Agile становится использование различных ручных этапов в процессе подготовки релиза. Это может быть утверждение изменений экспертной группой или требование представить подробное описание изменений и оценки рисков перед развертыванием новой сборки в производственной (или даже тестовой) среде.
- Обычно такие процедуры внедряются для дополнительного контроля релизов. Однако они значительно тормозят процесс и не учитывают задачи автоматического тестирования, которое должно гарантировать качество сборки.

- Чтобы рассеять опасения сомневающихся, можно продемонстрировать надежность CI/CD-пайплайна с помощью метрик. В то же время панели мониторинга и автоматические уведомления могут избавить вас от большого объема ручной работы, информируя все заинтересованные стороны о том, какие изменения проходят пайплайн.
- Еще один частый признак неблагополучия — появление запросов на замедление процесса и объединение изменений для уменьшения частоты релизов.
- Объединение изменений и выпуск релизов раз в неделю или в две недели — разумный вариант для некоторых продуктов. Однако при более длительных интервалах вы рискуете потерять преимущества, которые дает развертывание изменений в производственной среде, поскольку полученную обратную связь можно использовать при дальнейшей разработке.
- Объяснив принципы, на которых построены методологии Agile и DevOps, а также CI/CD, и получив согласие от всех заинтересованных сторон, можно упростить переход к их использованию.

- 
- Одно из главных препятствий для внедрения как CI/CD, так и Agile — недоверие, и как следствие — отсутствие у команд полномочий на выполнение необходимых действий. Если для принятия решений или внесения изменений нужно пройти многоступенчатую процедуру утверждения, это замедляет пайплайн и уменьшает пользу от быстрого фидбэка.
 - Если же команда получает необходимые полномочия, разработчики должны доставлять работающий продукт, а руководители — предоставить им необходимые инструменты и среды для выполнения работы.

CI и CD

- CI и CD не противоречат друг другу, а представляют собой две части процесса разработки, и, объединив их преимущества, вам будет легче доставлять пользователям ПО, не содержащее ошибок.
- Непрерывная интеграция — это процесс слияния изменений кода с основной веткой. В основе непрерывной доставки лежит автоматизация сборки и тестирования, реализованная на этапе непрерывной интеграции. Непрерывное развертывание — последний этап CI/CD-процесса, когда новая версия ПО доставляется пользователям после выполнения всех требований.
- [Подробнее об особенностях CI и CD](#)

ЗАПУСК СЕРВЕРА СБОРКИ

Пошаговое руководство по разворачиванию Build-сервера на:

- Jenkins: <https://habr.com/ru/post/695978/>
- GitHub pipeline: <https://habr.com/ru/company/jugru/blog/505994/>
- GitLab pipeline: <https://habr.com/ru/company/flant/blog/332712/>