Лекция 10. CI/CD

10.1. Зачем всё это нужно

Когда в организации проектов и задач становилось больше, и выполнение однотипных задач начинает занимать больше времени, необходимо его сокращение. Рассмотрим классический процесс решения задачи, подходящий для большинства компаний:

- Берем задачу из списка/получаем от начальства;
- Создаем новую ветку в git;
- Пишем программный код;
- Лично или с помощью коллеги выполняем код-ревью (*code review* — обзор/проверку кода);
- Запускаем тесты;
- Сливаем ветку в develop, затем в master, когда будет готов новый релиз;
- Выполняем сборку проекта;
- Публикуем новую сборку;

Этот процесс повторяется для каждой задачи, если вы 10 дней писали код и на сборку/развертывание потратили 1 час, то это выглядит разумно и не трудозатратно. Но что если вы поправили мелкий баг за 1 минуту, но на развертывание потратите тот же час? В этой ситуации это выглядит довольно расточительно. А если вам нужно выполнять в день 10 — 20 небольших исправлений?

Первый путь, укрупнять запросы на изменение и делать объединение в мастер как можно реже. Второй путь настроить CI чтобы процесс тестирования/построения/развертывания выполнялся автоматически. Делать ревью больших пул реквестов неудобно, поэтому реальным видится только второй путь решения проблемы. Здесь нам на помощь и приходят CI/CD.

10.2. Непрерывная интеграция

Continuous Integration (CI, непрерывная интеграция/доставка) — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления потенциальных дефектов и решения интеграционных проблем.

В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет раннего обнаружения и устранения ошибок и противоречий. Основным преимуществом является сокращение стоимости исправления дефекта, за счёт раннего его выявления.

Непрерывная интеграция впервые названа и предложена Гради Бучем в 1991 году.

Требования к проекту, в который необходимо встроить CI:

- Исходный код и всё, что необходимо для сборки и тестирования проекта, хранится в репозитории системы управления версиями (Git, Subversion, Mercurial и т.д.);
- Операции копирования из репозитория, сборки и тестирования всего проекта автоматизированы и легко вызываются из внешней программы.

Организация CI

На выделенном сервере организуется служба, в задачи которой входят:

- получение исходного кода из репозитория;
- сборка проекта (компиляция);
- выполнение тестов;
- развёртывание готового проекта (иногда эту часть рассматриваю уже как Continuous Delivery, о нём мы поговорим немного позже);
- отправка отчетов (на электронную почту).

Сборка может осуществляться:

- по внешнему запросу;
- по расписанию;
- по факту обновления репозитория и по другим критериям.

Сборка по расписанию

В случае сборки по расписанию (*daily build, ежедневная сборка*), они, как правило, проводятся каждой ночью в автоматическом режиме — *ночные сборки*.

Для различия дополнительно вводится система нумерации сборок — обычно, каждая сборка нумеруется натуральным числом, которое увеличивается с каждой новой сборкой. Исходные тексты и другие исходные данные при взятии их из хранилища системы контроля версий помечаются номером сборки. Благодаря этому, точно такая же сборка может быть точно воспроизведена в будущем — достаточно взять исходные данные по нужной метке и запустить процесс снова. Это даёт возможность повторно выпускать даже очень старые версии программы с небольшими исправлениями.

Преимущества

- проблемы интеграции выявляются и исправляются быстро, что оказывается дешевле;
- немедленный прогон модульных тестов для свежих изменений;
- постоянное наличие текущей стабильной версии вместе с продуктами сборок — для тестирования, демонстрации, и т. п.;

- немедленный эффект от неполного или неработающего кода приучает разработчиков к работе в итеративном режиме с более коротким циклом.

Недостатки

- затраты на поддержку работы непрерывной интеграции;
- потенциальная необходимость в выделенном сервере под нужды непрерывной интеграции;
- обратите внимание на скорость CI: разработчики должны получать результат CI максимум за 10 минут, иначе продуктивность падает из-за потери концентрации и частого переключения между задачами.

10.3. Непрерывной доставка

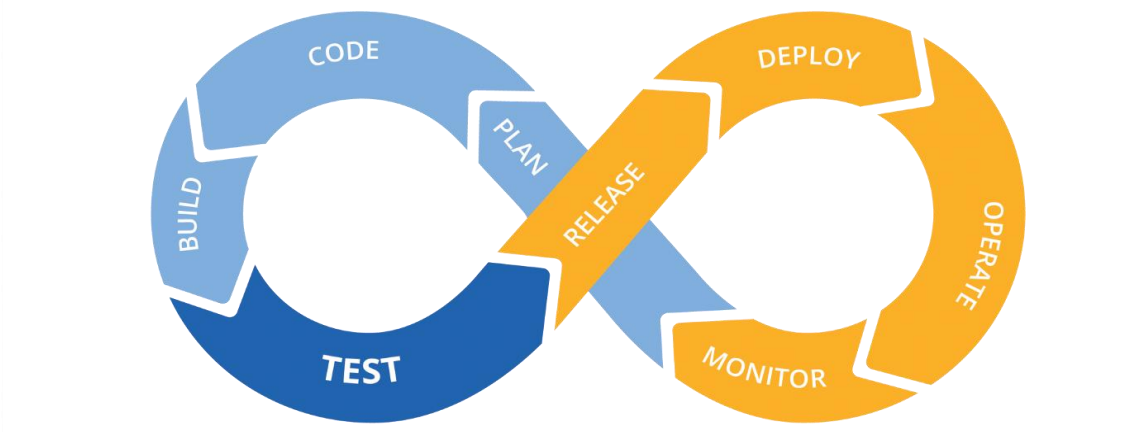
Непрерывная доставка (Continuous Delivery, CD) – это практика автоматизации всего процесса релиза ПО. Идея заключается в том, чтобы выполнять CI, плюс автоматически готовить и вести релиз к промышленной среде. Программист, избавившись практически от всей ручной работы, трудится продуктивнее.

Как правило, в процессе непрерывной доставки требуется выполнять вручную как минимум один этап: одобрить развертывание в промышленную среду и запустить его.

Непрерывная доставка располагается «на уровень выше» непрерывного развертывания (CI). В данном случае все изменения, вносимые в исходный код, автоматически развертываются в продакшен. Как правило, задача разработчика сводится к проверке запроса на включение (pull request) от коллеги и к информированию команды о результатах всех важных событий.

Непрерывное развертывание требует, чтобы в команде существовала отлаженная культура мониторинга (через различные трекеры задач, например Jira).

Разработчики, практикующие CI и желающие перейти к непрерывному развертыванию, для начала автоматизируют развертывание в тестовую среду, чтобы ничего не сломать а развертывание в промышленную среду делают, когда уже настроен и отлажен процесс поставки в тестовую.



Общая схема CI + CD

10.4. CI/CD: принципы, внедрение, инструменты

Концепция непрерывной интеграции и доставки (CI/CD)—концепция, которая реализуется как конвейер, облегчая слияние только что закомиченного кода в основную кодовую базу, так и установку приложения в промышленную/тестовую среду.

Мы рассмотрим принципы CI/CD.

- Первый принцип CI/CD: разделение ответственности заинтересованных сторон;
- Второй принцип CI/CD: снижение риска;
- Третий принцип CI/CD: короткий цикл обратной связи;
- Реализации среды в CI/CD;
- Инструменты для CI/CD.

Рассмотрим их подробнее:

1) Сегрегация ответственности заинтересованных сторон

Одним из основных преимуществ CI/CD является своевременное участие различных заинтересованных сторон в любом проекте.

Разработчики и дизайнеры (Developers) создают опыт и логику продукта, представленные в рассказах пользователей, и несут ответственность за создание работающий функций, доказывая это через модульные тесты.

Инженеры по качеству (Quality Engineers) отвечают за поддержание качества продукции, просмотр функций по мере их выполнения и вводят сквозные (E2E – end to end) функции / приемочные тесты, чтобы установить, что клиент получит продукт, работающий без ошибок.

Бизнес-аналитики (Business Analysts) и владельцы продуктов (Product Owners) несут ответственность за приемлемость (например, стоимость бизнеса), что подтверждается взаимодействием с фактическими пользователями и созданием пользовательских историй. Они координируют и анализируют результаты приемочных тестов для проверки рассказов пользователей и, возможно, создания новых, основанных на отзывах.

Оперативный отдел (DevOps-инженеры) несут ответственность за доступность продукта пользователям. Работая в области CI/CD, они масштабируют концепции по мере необходимости и разрабатывают логистику кода, чтобы код от разработчиков мог перейти в производственную среду и пользователи могли получать доступ.

Пользователи несут ответственность за использование продукта. Подразумевается, что продукт должен быть полезен и оправдывать усилия.

Каждый этап конвейерной обработки CI/CD создает среду, настроенную на то, чтобы *группы брали ответственность за соответствующую стадию тестирования*, обеспечивая целостность процесса.

2) Снижение риска

Каждый этап конвейерной обработки CI/CD создается для снижения риска в определенном аспекте. Разработчики отвечают за логические и письменные тесты, чтобы снизить риск нарушения логики. QE отвечают за целостность потока пользователей и записывают тесты для снижения риска сломанных историй пользователей. BAs и POs отвечают за удобство использования и принимают участие в приемочных тестах пользователей, чтобы снизить риск создания непригодных / нежелательных функций. Ops/ DevOps несут участвуют в обслуживании CI/CD, связанные с развертыванием операций (анализ схемы данных/миграции данных) и масштабирование, чтобы снизить риск недоступности продукта.

3) Короткий цикл обратной связи

Причина внедрения конвейерной обработки CI/CD—использование машин для работы с людьми. Это позволяет сократить время, затрачиваемое на обратную связь по разрабатываемым функциям.

4) Реализации среды в CI/CD

Чаще всего данный подход реализуется отдельными ветками в репозитории (для стабильного кода, тестов, QE, и т.д.).

5) Инструменты для CI/CD

Локальные

GitLab CI, TeamCity, Bamboo, GoCD Jenkins, Circle CI.

Облачные

BitBucket Pipelines, Heroku CI, Travis, Codeship, Buddy CI, AWS CodeBuild.

Если у вас нет потребности в локальном хостинге, существует множество других облачных инструментов CI, таких как Codeship, Buddy CI и Travis. Для личных проектов все большее число репозиториев кода предоставляют решение «все-в-одном», которое дает доступ к инструментам CI. Недавно BitBucket запустил свой собственный бесплатный CI под названием Pipelines, ограниченный вычислительным временем, как и Heroku. GitHub также имеет бесплатную интеграцию с Travis, на случай если ваш проект—с открытым исходным кодом.

10.5. DevOps

В настоящее время всё чаще слышно про такую должность, как DevOps инженер. Разберёмся, что вкладывается в это понятие.

Что значит непрерывный?

Прежде, чем мы начнём разбираться в различных концепциях DevOps, следует понять, что значит “*непрерывный*” в области программного

обеспечения. Термин “непрерывности” относится к изменениям программного обеспечения, которые происходят в течении всего процесса разработки ПО.

Изначально DevOps не имел ничего общего с конкретной должностью в организации. Многие по-прежнему заявляют, что *DevOps* - это культура, а не профессия, согласно которой коммуникация между разработчиками и системными администраторами должна быть налажена максимально тесно.

Естественным путём DevOps из разряда “культуры” и “идеологии” переместился в разряд “профессии”. Рост вакансий с этим словом внутри стремительно растёт. Но что же ждут от DevOps-инженера рекрутёры и компании? Зачастую от него ждут смеси таких навыков как *системное администрирование, программирование, использование облачных технологий и автоматизация крупной инфраструктуры.*

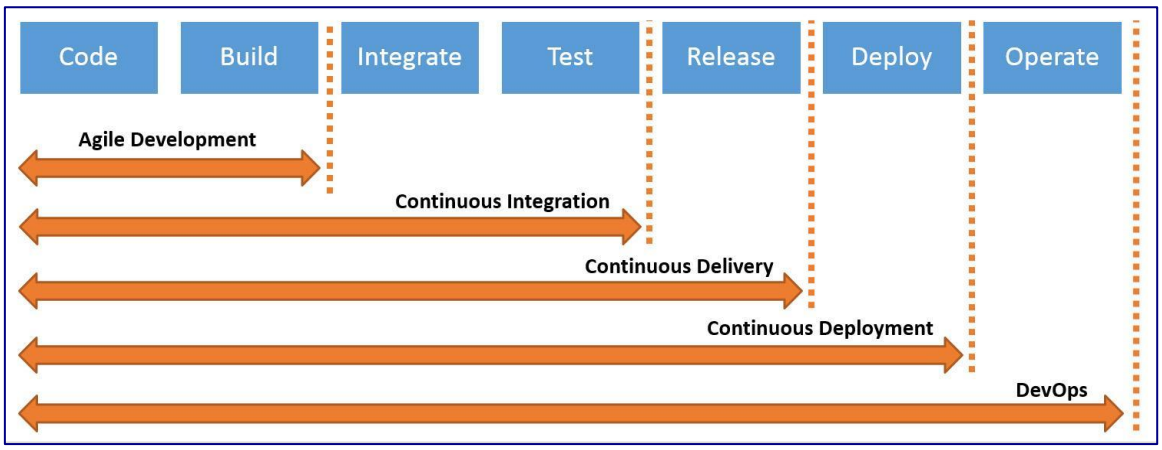
Это означает, что нужно не только быть хорошим программистом, но так же идеально разбираться в том, как работают сети, операционные системы, виртуализация, как обеспечить безопасность и отказоустойчивость, а так же несколько десятков различных технологий, начиная от основных и проверенных временем вещей как iptables и SELinux и заканчивая более свежими и модными ранее упомянутыми технологиями как Chef, Puppet или Ansible.

На можно сказать: настоящий инженер должен уметь не писать на языках Ruby, Go, Bash или “настраивать сеть”, а уметь строить сложные, красивые, автоматизированные и безопасные системы, и понимать как они работают от самого низкого уровня вплоть до генерации и отдачи HTML-страничек в браузер.

Конечно, нельзя быть абсолютном профессионалом во всех сферах IT в каждый конкретный момент времени. Но ведь и DevOps – это не только о людях, умеющих всё делать хорошо. Это так же о максимальной ликвидации безграмотности по обе стороны баррикад (на самом деле являющихся одной командой), будь ты уставшим от ручной работы сисадмином или молящимся на C++ разработчиком.

10.6. Как это всё работает вместе?

Перед тем как вы дойдёте до непрерывного развёртывания вам предстоит длинный путь. Сначала вы должны будете многое автоматизировать и пройти фазы непрерывной интеграции и доставки.



Взаимодействие CI/CD/DevOps

Конкретные *цели* DevOps охватывают весь процесс поставки программного обеспечения. Они включают:

1. Сокращение времени для выхода на рынок;
2. Снижение частоты отказов новых релизов;
3. Сокращение времени выполнения исправлений;
4. Уменьшение количества времени на восстановления (в случае сбоя новой версии или иного отключения текущей системы).

Методики DevOps делают простые процессы более программируемыми и динамическими. С помощью DevOps можно максимизировать предсказуемость, эффективность, безопасность и ремонтопригодность операционных процессов.

Интеграция DevOps предназначена для доставки продукта, непрерывного тестирования, тестирования качества, разработки функций и обновлений обслуживания для повышения надежности и безопасности и обеспечения более быстрого цикла разработки и развертывания.

DevOps дает преимущества в управлении выпуском программного обеспечения для организации путем стандартизации среды разработки. События можно более легко отслеживать, а также разрешать документированные процессы управления и подробные отчеты. Подход DevOps предоставляет разработчикам больше контроля над средой, предоставляя инфраструктуре более ориентированное на приложения понимание.

Преимущества DevOps

Компании, которые используют DevOps, сообщили о значительных преимуществах, в том числе:

- значительном сокращении времени выхода на рынок;
- улучшении удовлетворенности клиентов;
- улучшении качества продукции;
- более надежных выпусках;

- повышении производительности и эффективности;
- увеличении способности создавать правильный продукт путем быстрого экспериментирования.

Однако, исследование, выпущенное в январе 2017 года компанией "F5 Labs", на основе опроса почти 2200 ИТ-руководителей и специалистов отрасли, показало, что только один из пяти опрошенных полагает, что DevOps оказывает стратегическое влияние на их организацию, несмотря на рост использования. В том же исследовании было установлено, что только 17 % определили DevOps как ключевой инструмент.

DevOps и архитектура

Чтобы эффективно использовать DevOps, прикладные программы должны соответствовать набору архитектурно значимых требований, таких как: возможность развертывания, изменяемость, тестируемость и возможность мониторинга (система постоянного наблюдения за явлениями и процессами, проходящими в приложении).

Хотя в принципе можно использовать DevOps с любым архитектурным стилем, архитектурный стиль микросервисов (вариант сервис-ориентированной архитектуры программного обеспечения, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практики гибкой разработки и DevOps) становится стандартом для построения постоянно развернутых систем. Поскольку размер каждой услуги невелик, он позволяет создавать архитектуру отдельного сервиса посредством непрерывного рефакторинга, что уменьшает необходимость в большом предварительном дизайне и позволяет выпускать программное обеспечение на ранней стадии непрерывно.

В конце концов, всё это способствует устранению накладных расходов процесса разработки. Однако, не стоит забывать о целесообразности процессов. Возможно, для вашей ситуации это будет излишним.

Использованные источники:

- https://medium.com/southbridge/ci-cd;
- Википедия,CI;
- https://onagile.ru/industries/software-development/continuous-delivery;
- https://habr.com/company/piter/blog/343270/;
- http://qaat.ru/kakaya-raznica-mezhdu-continuous-delivery-continuous-deployment-i-continuous-integration/;
- https://mkdev.me/posts/chto-i-kto-takoe-devops.