Пайплайн CI/CD: что это такое, как применяется в разработке

CI/CD-пайплайн (CI/CD pipeline) расшифровывается как «конвейер непрерывной интеграции и непрерывного развертывания» — по-английски «continuous integration and continuous deployment (delivery) pipeline».

Если простыми словами — это особая практика автоматической доставки новых версий ПО пользователю (клиенту) на протяжении всего жизненного цикла разработки.

Продукт не «разбивается» на отдельные строго дифференцированные версии, получение которых требуют долгого ожидания со стороны клиента, а отправляется пользователю порционно по мере итеративного обновления кодовой базы.

И хотя этапы разработки, тестирования, отправки и развертывания ПО можно выполнять вручную, истинная ценность конвейера CI/CD реализуется за счет автоматизации.

Ныряем глубже в CI/CD

Поскольку современные пользовательские приложения (вызов такси, доставка еды, аренда жилья) становятся своего рода «ядром» большого числа компаний, скорость выпуска кода (обновлений приложения) становится конкурентным преимуществом на рынке.

Чтобы организовать максимально быструю доставку цифрового продукта пользователю, необходимы 2 составляющие:

- Непрерывная интеграция (continuous integration). Разработчики как можно чаще сливают изменения в основную ветку, используя систему контроля версий (например, Git). При этом, любые изменения проходят через автоматические тесты. Иными словами, продукт дополняется постепенно, в не лавинообразно. Если представить таймлайн разработки в виде некой линии, а обновления в виде размещенных на ней точек, то все они будут равномерно распределены по всей протяженности таймлайна.

- Непрерывная поставка (continuous deployment). Это дополнительное расширение непрерывной интеграции, которое автоматически развертывает новые изменения кода после этапа сборки в производственной среде клиента. Цель автоматизации очевидна — снизить нагрузку с разработчиков и свести к минимуму человеческий ошибки, при этом поддерживая согласованный процесс выпуска ПО.

Инструменты, «размещенные» в пайплайне, могут включать компиляторы и анализаторы кода, модульные тесты, системы безопасности данных и множество других прикладных компонентов, которые могут быть полезны на всех этапах выпуска продукта.

Надо сказать, что CI/CD — это основа методологии DevOps, которая автоматизирует процесс сборки, настройки и развертывания программного обеспечения.

В этом случае предполагается тесное взаимодействие специалистов по разработки со специалистами по обслуживанию — технологические процессы как бы интегрируются друг в друга. По сути, методология прививает определенную культуру создания и поддержания продукта.

- Частое и порционное тестирование кода сокращает количество ошибок и багов, давая клиенту наилучший опыт.
- Итеративная разработка и поставка ПО ускоряет окупаемость продукта. К тому же, гораздо проще создать MVP. В итоге, вы снижаете затраты на разработку и быстро тестируете гипотезы.
- Написание небольших участков кода вместе с автоматическими тестами снижает когнитивную нагрузку с разработчика.

И да, этапы пайплайна имеют строгую последовательность. Вначале всегда идет разработка, а уже в конце — развертывание в рабочей среде. Ближе к концу проводятся тесты, а ближе к началу — статический анализ кода. Между этапами может работать система уведомлений — сообщения о состоянии пайплайна буквально отправляются в мессенджер или на почту.

И самое главное — все работает автоматически. Пайплайн запускается автоматически при вводе консольной команды, а возможно просто по таймеру — зависит от конкретного инструмента и потребностей разработчика.

Инструменты CI/CD

Множество популярных хостинг-платформ git-репозиториев предоставляют целые системы со скриптовыми сценариями или полноценными интерфейсами, которые облегчают организацию CI/CD — Github CI/CD, GitLab CI/CD, Bitbucket CI/CD.

Впрочем, есть и другие: Jenkins CI/CD, AWS CI/CD, Azure DevOps CI/CD. У всех свои особенности, поэтому выбор какого-то конкретного инструмента — дело «вкусовщины», хотя в каждом из них можно найти и объективные достоинства и недостатки. Вот несколько из тех, которые заточены исключительно под организацию CI/CD-пайплайнов:

Jenkins — бесплатная программная среда с открытым исходным кодом (в виде сервера) созданная специально для выполнения непрерывной интеграции ПО. Продукт написан на Java, поэтому работает в Windows, macOS и других Unix-подобных операционных системах.

CircleCI — инструмент CI/CD, выполненный в виде веб-сервиса, который позволяет полностью автоматизировать весь пайплайн, начиная от создания кода и заканчивая тестированием и развертыванием.

Интегрируется с GitHub, GitHub Enterprise и Bitbucket, запуская построение «билдов» при фиксации новых строк кода в репозитории. Запускает сборки с использованием контейнера или виртуальной машины, автоматически распараллеливая выполнение пайплайна на несколько потоков.

Сервис является платным, но есть и free-функционал с выполнением одного задания без функции распараллеливания. При этом проекты с открытым исходным кодом получают три дополнительных бесплатных контейнера.

TeamCity — сервер управления сборкой и непрерывной интеграцией от JetBrains, ориентированный на DevOps-команды. Работает в среде Java и интегрируется с Visual Studio и IDE. Может быть установлен как на серверах Windows, так и на Linux, а также поддерживает .NET-проекты.

Bamboo — сервер непрерывной интеграции, который автоматизирует управление выпусками приложений. Является одним из продуктов компании Atlassian. Охватывает сборку, функциональное тестирование, назначение версий, пометку выпусков, развертывание и активацию новых версий в рабочей среде.

Этапы CI/CD

Этапы CI/CD пайплайна различаются в зависимости от продукта и конкретной команды разработчиков, но общие черты всегда одинаковые — есть некая стандартная последовательность действий, которая выполняется практически в любом пайплайне. Некоторые этапы могут быть пропущены, либо выполнены вручную, однако это плохая практика.

Чаще всего пайплайн можно представить в виде 7 основных шагов:

Триггер. Пайплайн должен запускаться автоматически каждый раз, когда фиксируется отправка нового кода в репозиторий. Есть много вариантов достичь этого.

Например, инструмент CI/CD (например, Jenkins) может вести опрос («poll») репозитория Git, либо наоборот, определенный «hook» (например, Git Webhooks) будет отправлять push-уведомление инструменту CI/CD каждый раз, когда разработчик делает push в системе контроля версий.

Да, вы все еще можете запускать пайплайн вручную, но человеческий фактор делает это решение не самым лучшим, отчего автоматический запуск внушает больше уверенности.

Проверка кода. Инструмент CI/CD извлекает код из репозитория (через «hook» или «poll») вместе с информацией о том, какой конкретный коммит вызвал триггер пайплайна и как требуется выполнять этапы пайплайна.

На этой стадии могут быть запущены инструменты статического анализа кода, останавливающие выполнения пайплайна в случае выявления ошибки. Если все в порядке, CI/CD процесс продолжается дальше.

Компиляция кода. Очевидно, инструмент CI/CD должен иметь доступ ко всем инструментам сборки, которые необходимы для компиляции кода. Например, если приложение написано на Java, то могут использоваться Maven или Gradle.

Кстати, лучше всего, чтобы сборка происходила в чистой «среде». Для этих целей можно использовать контейнеризацию — Docker, например.

Unit-тестирование. Важнейший элемент пайплайна — модульное тестирование или unit testing. Для этого используются специальные библиотеки для каждого конкретного языка программирования.

Скомпилированное приложения запускается с использованием тестов, если выполнение завершается без ошибки, значит можно переходить к следующему этапу пайплайна.

Важно обеспечить максимальное покрытие тестами всех функций и компонентов приложения. При этом тесты должны поддерживаться и улучшаться по мере роста кодовой базы.

Упаковка кода. Если все тесты пройдены, то перед доставкой пользователю приложение нужно упаковать в конечный «build».

Например, если код на Java, то создается JAR-файл. А если ваше приложение размещено в контейнере Docker, имеет смысл создать образ Docker.

Приемочное тестирование. По сути, это способ убедиться, что ПО соответствует всем необходимым требованиям — либо клиента, либо заявлениям самого разработчика.

Однако, так же, как и модульные, приемочные тесты выполняются автоматически. Иными словами, требования и желаемый результат указывается в формате, который понятен вычислительной система, а значит может быть автоматизирован и неоднократно протестирован — подобно модульным тестам.

Например, можно проверить функциональные части приложения — сможет ли пользователь на сайте интернет-магазина добавить продукт в корзину? Для таких задач можно использовать Selenium, который может имитировать действия пользователя в браузере.

Польза приемочного тестирования в сокращении времени, затрачиваемого на ручное тестирование. Зачем кликать мышкой на кнопку вручную, если это может сделать специальная программа автоматически и по триггеру?

Доставка и развертывание. На этом этапе уже существует готовый к развертыванию программный продукт, которые необходимо доставить в рабочую среду клиента и корректно его установить.

Для непрерывного развертывания нужна производственная среда. Например, это может быть публичное облако со своим собственным API или инструмент Spinnaker, который работает системой оркестрации контейнеров Kubernetes и всеми популярными облаками — Google Cloud Platform, AWS, Microsoft Azure and Oracle Cloud.

Это этап — конец пайплайна. В следующий раз, когда какой-то разработчик отправит новый код в репозиторий, процесс будет запущен снова.

CI/CD на практике

Как правило, ради добавления в продукт новой функции создается отдельная ветка в системе контроля версий (например, Git). В ней пишется код, после чего локально тестируется. Когда новая фича готова, программист делает `pull request` и просит старшего коллегу сделать «ревью», чтобы принять результаты в основную ветку. Далее обновленная кодовая база «деплоится» в dev-окружение. Все делается вручную.

Если вы тратите 25 часов на разработку и 2 часа на деплой — это более-менее нормальное соотношение. Однако, если вы тратите 20 минут на создание «фичи», а на деплой уходят те же 2 часа — это проблема. Время используется не рационально.

У вас есть два пути:

- Вносить изменения в основную ветку реже, накапливая большой `pull request`. Только вот ревьюировать такие объемы кода будет уже сложнее.
- Организовать CI/CD пайплайн для автоматической сборки, тестирования и деплоя.

Во втором случае процесс жестко стандартизирован — любая фича попадает в конечный продукт (основную ветку) только тогда, когда пройдет все этапы пайплайна. Все — без исключений.

И хотя эта статья не призвана помочь в освоении какого-то конкретного CI/CD-инструмента, стоит привести простой пример на примере GitLab CI/CD. Просто для общего понимания, как на практике программируется пайплайн.

Представим, что у вас уже есть репозиторий GitLab с кодом проекта. Вы хотите лишь автоматизировать процесс сборки и деплоя.

В GitLab автоматизированными процессами занимается служба GitLab Runner — изолированная виртуальная машина, выполняющий задания пайплайна.

Раннеры программируются с помощью YAML-скриптов, в которых подробно описываются необходимые инструкции для GitLab CI/CD. В этом файле определяются:

- Структура и порядок заданий, которые выполняются раннером
- Ветки принятие решений бегуна при при возникновении определенных условий

Вот самый примитивный вариант такого файла:

```
build-job:
  stage: build
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"

test-job1:
  stage: test
  script:
    - echo "This job tests something"

test-job2:
  stage: test
  script:
    - echo "This job tests something, but takes more time than test-job1."
    - echo "After the echo commands complete, it runs the sleep command for 20 seconds"
    - echo "which simulates a test that runs 20 seconds longer than test-job1"
    - sleep 20

deploy-prod:
  stage: deploy
  script:
    - echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
  environment: production
```

Здесь есть 4 задания: `build-job`, `test-job1`, `test-job2` и `deploy-prod`. Все, что после `echo` — вывод сообщений в консоль пользовательского интерфейса GitLab.

Есть также предопределенные GitLab-ом переменные `$GITLAB_USER_LOGIN` и `$CI_COMMIT_BRANCH`. Их можно использовать для вывода информации в консоль.

Разумеется, этот пайплайн не выполняет никаких операций. Лишь вывод в консоль. Он служит лишь примером того, в каком формате происходит построение пайплайна.

В данном случае, есть 3 этапа: *build, test, deploy*. При этом на этапе тестирования выполняется 2 задания. Кстати, интерфейс GitLab позволяет увидеть содержание скрипта визуально.

> 13055735-a255-44fd-a2d0-3b242ba18027?width=663&height=219
> Image1

Изображение взято с официального сайта GitLab

Как и у любого инструмента CI/CD, у GitLab есть своя документация. Там можно найти гораздо больше полезных примеров, а также все особенности работы конкретно с этим сервисом.

Например, нечто похожее есть и у GitHub. Для кого-то может показаться полезным использование CI/CD инструмента от той же платформы, которая предоставляет и удаленный репозиторий — своего рода унификация.

Немного полезных советов по CI/CD

- Версионируйте не только кодовую базу вашего продукта, но и скрипты сценариев самого CI/CD пайплайна.
- Не нарушайте порядок выполнения этапов пайплайна. Если вы сперва деплоите продукт в продакшен, а уже потом запускаете тестирование — у вас проблема.
- Не пропускайте этапы пайплайна, даже если соблазн это сделать велик. Все этапы («stages») должны быть последовательно выполнены. Если у вас пара сотен тестов, но несколько из них нарушают пайплайн, останавливая его — исправляйте, а не пропускайте. Любые сбои нужно либо устранить, либо проблемные тесты удалить насовсем.
- Никакой ручной работы! Запуск и переход от предыдущего этапа пайплайна к следующему должны происходить автоматически. Иначе теряется смысл DevOps-методологии.
- Настройте уведомления, регулярно сообщающие о состоянии CI/CD пайплайна в процессе сборки, тестирования и деплоя. Например, они могут приходить вам в мессенджер.

Заключение

В этой статье были рассмотрены лишь общие положения DevOps-методологии в основе которой лежат CI/CD-пайплайны. Были показаны некоторые из популярных инструментов и сервисов, которые используются для автоматизации непрерывной интеграции и развертывания. И хотя функции CI/CD инструментов довольно похожи, каждый из них имеет свои особенности. Любому, кто решил внедрить DevOps подходы в процессы разработку потребуется время, чтобы ознакомиться с каждым конкретным программным решением, понять нюансы его использования и выбрать подходящий.

Источник

Revision #1
Created 2023-11-25 18:00:16 UTC by odiljonov
Updated 2023-11-25 18:00:35 UTC by odiljonov