грокаем

continuous delivery

Кристи Уилсон
Вступительное слово: Джез Хамбл и Эрик Брюэр

Кристи Уилсон

Грокаем Continuous Delivery

Серия «Библиотека программиста»

Перевела с английского Н. Григорьева

ББК 32.973.2-018
УДК 004.41

Уилсон Кристи

У36 Грокаем Continuous Delivery. — СПб.: Питер, 2024. — 400 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2372-8

Код должен быть готов к релизу всегда!

Пайплайн Continuous Delivery автоматизирует процессы контроля версий, тестирования и развертывания при минимальном вмешательстве разработчика. Освойте инструменты и методы непрерывной доставки, и вы сможете быстро и последовательно добавлять функции и выпускать обновления.

«Грокаем Continuous Delivery» — это руководство по настройке и работе с пайплайном непрерывной доставки. В каждой главе рассматривается отдельный сценарий, с которым вы столкнетесь при создании системы CD, и приводятся реальные примеры, например автоматическое масштабирование и тестирование унаследованных приложений. Кристи Уилсон сопровождает каждый шаг иллюстрациями, кристально четкими объяснениями и практическими упражнениями для закрепления полученных знаний.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1617298257 англ.

ISBN 978-5-4461-2372-8

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

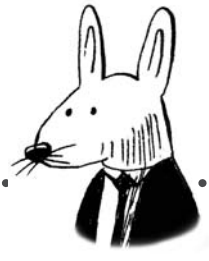
Дата изготовления: 02.2024. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

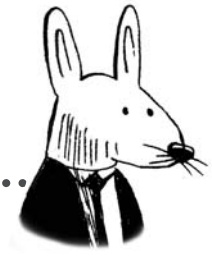
Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 26.01.24. Формат 70×100/16. Бумага офсетная. Усл. п. л. 32,250. Тираж 1500. Заказ 0000.

Оглавление



Вступительное слово



Когда мы с Дэвидом Фарли (David Farley) писали книгу «Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation» (Addison-Wesley, 2010)[1], то благодаря многолетнему применению описанных в ней принципов мы знали, что предлагаем современный, целостный подход к доставке программного обеспечения, дающий существенные преимущества командам и компаниям, которые его используют. Многочисленные исследования (в том числе то, в котором я принимал участие под руководством доктора Николь Форсгрен (Dr. Nicole Forsgren), — о нем рассказывается в главах 8 и 10 этой книги) показали, что применение такого подхода приводит к повышению качества и стабильности разработки ПО, а также к более оперативной его доставке.

Невзирая на то что непрерывная интеграция и непрерывная доставка (continuous integration/continuous delivery, сокращенно CI/CD) сегодня считаются стандартной практикой, их по-прежнему на удивление трудно внедрить и правильно реализовать. Все еще слишком много команд (и пользователей!) сталкиваются с тем, что релизы выходят редко и с высоким риском, по вечерам или в выходные дни; случаются плановые и внеплановые простои, откаты и проблемы с производительностью, работоспособностью и безопасностью. Всего этого можно избежать, но нужны постоянные инвестиции в команды, инструменты и корпоративную культуру.

Важно отметить, что многие новички в нашей сфере не знакомы с основополагающими методиками и способами их применения. Книга «Грокаем Continuous Delivery» отлично восполняет этот пробел. Кристи Уилсон, эксперт в области непрерывной доставки, возглавляющая в компании Google CI/CD-проект Tekton с открытым исходным кодом, написала всеобъемлющее, ясное и подробное руководство, в котором тщательно разобраны нюансы технологии и реализации современной доставки программных продуктов. Кристи не только рассказывает о принципах и их применении, но и наглядно показывает, почему они важны, а также приводит

[1] ФарлиД., Хамбл Дж. «Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий».

пошаговые решения самых сложных проблем, с которыми, по моим наблюдениям, сталкиваются многие команды, например итеративная разработка функций и работа с унаследованным кодом.

Я надеюсь, что эта книга займет достойное место в списке пособий для новичков в каждой команде разработчиков. Она также станет весьма полезным руководством для более опытных инженеров, осваивающих новый для них стиль работы. Я благодарен Кристи за создание источника информации, который, без сомнения, поможет лучше понять, как правильно реализовать современный процесс доставки программного обеспечения на благо всей нашей отрасли и широкой общественности — им, в конечном итоге, мы и служим.

— Джез Хамбл (Jez Humble)
соавтор книг «Continuous Delivery»,
«The DevOps Handbook»[1] *и «Accelerate!»*[2]

Прелесть программ в том, что со временем их можно улучшать. Но в этом и их проклятие — мы, по сути, все время что-то меняем, потому что можем это изменить. Неустанное стремление к внедрению новых функций и другим улучшениям приводит к необходимости максимально быстрой интеграции изменений, их тестирования и доставки пользователям.

Кристи Уилсон живет этим процессом и наблюдает за ним с разных сторон, и она написала книгу о том, как командам разработчиков получить необходимое ускорение. Действительно, команда, способная на ускоренную разработку и внедрившая высокую степень автоматизации, получает конкурентное преимущество. Такие команды не только со временем завоевывают значительную часть рынка — в них выше моральный дух и ниже уровень текучки персонала. Ведь быть продуктивным приятно!

Бытует заблуждение, что более медленные процессы, в которых, возможно, развертыванию что-то препятствует, являются более безопасными или надежными. Многие команды не любят перемены и поэтому выпускают изменения, например, раз в квартал. У такого подхода есть два серьезных недостатка. Во-первых, сложная задача интеграции большого количества изменений в этом случае откладывается до самого конца разработки, но если изменений, которые нужно интегрировать с момента последнего выпуска, накапливается много, их внедрение чревато сбоями и большими задержками. Во-вторых, низкая скорость затрудняет быстрое исправление проблем безопасности, что критически важно для большинства команд. Подход, представленный в этой книге, основан на непрерывных (небольших) интеграциях, позволяющих быстро получать обратную связь по возникающим проблемам и эффективно вносить исправления безопасности.

За последние несколько лет внимание к проблемам безопасности резко возросло, особенно в связи с «атаками через цепочки поставок» (supply chain attacks). В совре-

---

[1] Ким Дж., Дебуа П., Уиллис Дж. и Хамбл Дж. «Руководство по DevOps».

[2] Форсгрен Н., Ким Дж. и Хамбл Дж. «Ускоряйся!»

менное ПО входят компоненты из различных источников — других команд, других компаний — и компоненты с открытым исходным кодом. Оно запросто может состоять из тысячи частей, и все их необходимо интегрировать вместе. Однако это требует совершенно нового уровня автоматизации: нам нужно знать все исходные данные, откуда они взялись и как они совместно использовались. Книга Кристи — одна из первых, в которой освещаются эти вопросы и рассказывается, как повысить уровень безопасности систем.

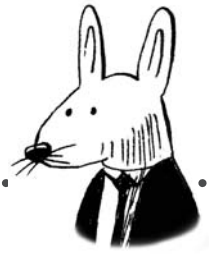
И наконец, несмотря на то что в рассматриваемой области существует огромное количество инструментов и опций, эта книга отлично описывает ее ключевые концепции и цели, одновременно приводя практические примеры и возможные альтернативы.

Для меня эта книга стала настоящим глотком свежего воздуха; надеюсь, что она понравится и вам.

— Эрик Брюэр (Eric Brewer),
вице-президент по инфраструктуре
и fellow-разработчик[1] в компании Google

[1] Fellow — высшая ступень инженерного пути развития карьеры, например заслуженный разработчик и т. д. По степени влиятельности соответствует высшему руководству организации. — *Примеч. ред.*

Предисловие



Программирование увлекло меня с того самого момента, как я узнала о его существовании. Я помню, как (примерно 300 лет назад) друг рассказал мне о программе игры в шахматы, которую он написал; хотя я совершенно не понимала, о чем он говорит, я осознала, что (а) я никогда не задумывалась о том, как работают компьютеры, и (б) теперь мне совершенно необходимо узнать об этом как можно больше. То, что случилось дальше, иногда было удивительно, а иногда приводило в замешательство («переменные подобны почтовому ящику» — это сравнение для меня теперь полно смысла, но сначала оно просто сразу вылетело у меня из головы). А после первого урока по Turbo Pascal в старших классах и множества самоучителей по Java я влипла окончательно.

Хотя программирование было увлекательным само по себе, процессы организации разработки заинтриговали меня едва ли не больше. На протяжении по крайней мере половины своей карьеры я с разочарованием замечала, как мало внимания им уделяется, несмотря на то что они чрезвычайно важны не только для качества создаваемых программ, но и для благополучия и эффективности создателей этих программ. Более того, я была обескуражена, когда поняла, что сами разработчики и менеджеры не считают такое отношение проблемой. Часто оно диктовалось мнением, что выдавать код как можно быстрее — лучший способ окупить вложения по максимуму.

По иронии судьбы время и исследования показали, что скорость действительно определяет успех, но чтобы усилия разработчиков на самом деле были эффективными, а их работа — стабильной, скорость должна сочетаться с безопасностью. Максимальная скорость и безопасность программной разработки составляют основу непрерывной доставки, именно поэтому данная концепция и соответствующие методики оказались мне близки. Тем не менее до недавнего времени я ничего не знала о непрерывной доставке.

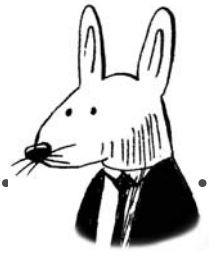
В первую очередь меня привлекли тесты и автоматизация. Я до сих пор помню чувство свободы, которое я испытала, когда познакомилась с тестами, и особенно с разработкой через тестирование; тогда я поняла, что могу проверять програм-

мы, которые пишу, в процессе их создания. Казалось, что у меня с плеч свалился огромный камень, когда я смогла проверять свою работу прямо по ходу дела, — так я избавилась от назойливого внутреннего голоса, который иногда убеждал меня, что я не знаю, что делаю, и что ничего из того, что я понаписала, работать не будет. Инструменты и автоматизация также помогали мне чувствовать себя уверенно: они брали на себя очень ответственные задачи, которые поначалу пугали. Казалось, будто рядом с тобой сидит друг и подсказывает тебе, что делать.

Концепция непрерывной доставки берет все лучшее из тестирования и автоматизации, которые всегда помогали мне в моей работе, и формирует набор практик, дающих возможность любому разработчику усовершенствовать свои методы. Я хочу помочь всем, особенно тем, кто иногда сомневается в себе или борется со страхом сделать что-то не то (и я думаю, подобные чувства хотя бы иногда испытывает большинство из нас), — ощутить ту же свободу, самостоятельность и уверенность в себе, что и я, когда написала свой первый тест.

Спасибо, что нашли время прочитать эту книгу. Я надеюсь, что по крайней мере вы сделаете правильный вывод: большинство багов и ошибок в коде практически не имеют отношения к самому коду (и уж точно не имеют отношения к человеку, который его написал). На самом деле их причины лежат в процессах разработки, которым просто нужно уделить немного больше внимания, и тогда усилия, приложенные к обновлению и исправлению этих процессов, окажутся не напрасны.

О книге



Задача книги — стать недостающим руководством по началу работы с непрерывной доставкой и ее эффективному применению. Книга описывает практики, составляющие непрерывную доставку, и рассказывает о необходимых компонентах автоматизации, которая поддерживает эти практики. Подобные знания приходят постепенно, с годами упорного труда. Надеюсь, что моя книга поможет сократить этот путь и вам не придется получать их на собственном опыте!

Для кого эта книга

Книга «Грокаем Continuous Delivery» предназначена для всех, кто день ото дня занимается созданием программного обеспечения. Чтобы польза от книги была максимальной, вы должны быть знакомы с основами написания shell-скриптов, владеть хотя бы одним языком программирования и иметь опыт тестирования. Вам также понадобится опыт работы с системой контроля версий, HTTP-серверами и контейнерами. Однако глубоко разбираться во всех этих темах не обязательно; вы сможете изучить их по ходу чтения, если понадобится.

Структура и план книги

Книга состоит из 13 глав, разбитых на четыре части. Первые две главы представляют собой введение в концепцию непрерывной доставки и содержат список терминов, которые вам понадобятся на протяжении всей книги:

- В главе 1 дается определение *непрерывной доставки* и объясняется ее связь со смежными понятиями, такими как *непрерывная интеграция* и *непрерывное развертывание*.
- В главе 2 перечислены основные элементы автоматизации непрерывной доставки, а также дана терминология, используемая в остальных частях книги.

Часть 2 посвящена процессам, составляющим непрерывную интеграцию и необходимым для непрерывной доставки:

- Глава 3 поясняет жизненно важную роль системы контроля версий в процессе непрерывной доставки; без этой системы непрерывная доставка невозможна.
- В главе 4 рассматривается эффективный, но редко обсуждаемый элемент непрерывной интеграции: статический анализ, в частности линтинг, и то, как его применять к унаследованному коду.
- Главы 5 и 6 посвящены тестированию — важнейшему элементу проверки в рамках непрерывной интеграции. Мы не будем учить вас тестировать (этому посвящено множество других книг), а сосредоточимся на самых частых проблемах, которые со временем накапливаются в наборах тестов, в частности на нестабильных и медленных тестах.
- В главе 7 рассматривается жизненный цикл изменений кода и исследуются все места, где могут возникнуть баги, а также способы настройки автоматизации для обнаружения и устранения этих багов по мере их появления.

В части 3 мы переходим от верификации изменений, вносимых в продукты с помощью непрерывной интеграции, к выпуску релизов:

- В главе 8 через призму метрик DORA рассматривается, как использование системы контроля версий влияет на скорость выпуска релизов.
- В главе 9 продемонстрировано, как создавать артефакты безопасно, применяя принципы, определенные стандартом SLSA, и объясняется важность версионирования.
- В главе 10 мы возвращаемся к метрикам DORA, основное внимание уделяя метрикам стабильности, и рассматриваем методы развертывания, которые можно использовать для повышения стабильности продукта.

В части 4 мы рассмотрим концепции, относящиеся к автоматизации непрерывной доставки в целом:

- В главе 11 мы вернемся к элементам непрерывной доставки, которые изучались в предыдущих главах, и рассмотрим, как эффективно применять эти элементы в проектах, реализуемых с нуля, и в унаследованных проектах.
- В главе 12 основное внимание уделено «рабочей лошадке», часто лежащей в основе любой автоматизации непрерывной доставки: shell-скриптам. Вы увидите, как применять к скриптам, обеспечивающим безопасную и корректную доставку продукта, те же лучшие практики, что и к остальной части кода.
- В главе 13 представлена общая структура автоматизированных пайплайнов, которые необходимы для поддержки непрерывной доставки, и моделируются функции систем автоматизации непрерывной доставки, обеспечивающие их эффективность.

В приложениях в конце книги описаны особенности наиболее распространенных на момент написания книги систем непрерывной доставки и контроля версий.

Я рекомендую начать с чтения главы 1; такие термины, как *непрерывная доставка*, в реальной жизни используются непоследовательно, и понимание их контекста для настоящей книги поможет вам лучше разобраться в остальных главах.

Удобнее всего осваивать материал, читая части 2 и 3 по порядку, поскольку все последующие главы основываются одна на другой. В частности, предполагается, что при переходе к части 3 вам известны практики непрерывной интеграции, описанные в части 2. Тем не менее вы можете изучать главы по своему усмотрению, так как в каждой главе вы найдете ссылки на соответствующий материал других глав.

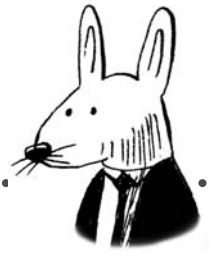
Часть 4 — это продвинутый раздел книги. Каждая глава в ней ссылается на концепции, рассмотренные ранее, а некоторые материалы (например, глава 12) полезнее будет изучить, уже приобретя некоторый опыт работы с системами непрерывной доставки.

Форум liveBook

Приобретая книгу «Грокаем Continuous Delivery», вы получаете бесплатный доступ к веб-форуму издательства Manning (на английском языке), на котором можно оставлять комментарии о книге, задавать технические вопросы и получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, откройте страницу https://livebook.manning.com/book/grokking-continuous-delivery/discussion. Информацию о форумах Manning и правилах поведения на них см. на https://livebook.manning.com/discussion.

В рамках своих обязательств перед читателями издательство Manning предоставляет ресурс для содержательного общения читателей и авторов. Эти обязательства не подразумевают конкретную степень участия автора, которое остается добровольным (и неоплачиваемым). Задавайте автору хорошие вопросы, чтобы он не терял интереса к происходящему! Форум и архивы обсуждений доступны на веб-сайте издательства, пока книга продолжает издаваться.

Часть 1
Что такое непрерывная доставка

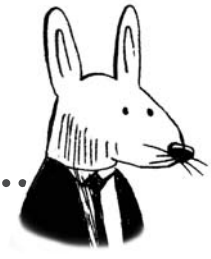


Добро пожаловать в «Грокаем Continuous Delivery»! Первые две главы познакомят вас с принципом непрерывной доставки (CD, Continuous Delivery) и терминологией, необходимой для освоения остальной части книги.

Глава 1 дает определение *непрерывной доставки* и объясняет ее связь со смежными понятиями, такими как *непрерывная интеграция* и *непрерывное развертывание*.

В главе 2 представлены основные элементы автоматизации процесса непрерывной доставки, а также терминология, используемая в остальной части книги.

1 Добро пожаловать в «Грокаем Continuous Delivery»



В этой главе

- ✓ Почему важна непрерывная доставка
- ✓ История развития технологий непрерывной доставки, непрерывной интеграции, непрерывного развертывания и CI/CD
- ✓ К какому ПО можно применять технологию непрерывной доставки и как это правильно делать
- ✓ Условия непрерывной доставки, а именно: поддерживать ПО всегда готовым к доставке и осуществлять его доставку как можно проще

Приветствую вас на страницах моей книги! Я очень рада, что вы решили не только узнать о непрерывной доставке, но и как следует разобраться в ней. Эта книга поможет вам использовать все преимущества непрерывной доставки в своей работе.

Нужна ли вам непрерывная доставка

Первый вопрос, который вы, возможно, себе задаете, — стоит ли тратить время на изучение непрерывной доставки, и даже если да, стоит ли применять ее в проектах. Ответ — *сто́ит*, если:

- вы профессионально занимаетесь разработкой программного обеспечения;
- в вашем проекте участвует больше одного человека.

Если справедливы оба этих условия, вам непременно следует вложиться в непрерывную доставку. *Даже если выполняется только одно из них* (например, вы работаете над проектом с друзьями ради интереса или разрабатываете профессиональное ПО в одиночку), вы не пожалеете об использовании непрерывной доставки.

> *Но ведь вы даже не спросили, чем я занимаюсь. Что, если я работаю над драйверами ядра, прошивкой или микросервисами? Вы уверены, что мне нужна непрерывная доставка? — спросите вы.*

Это неважно! Какой бы продукт вы ни создавали, принципы, изложенные в этой книге, будут вам полезны. В их основе лежат идеи, которые сформировались еще на заре программной разработки. Это не модные тренды, которые со временем исчезают или теряют популярность; это основы, которые останутся неизменными, независимо от того, создаете ли вы микросервисы, монолитные приложения, распределенные сервисы на основе контейнеров или что-то еще.

> ***Но мне не нужно ничего развертывать!***
>
> Совершенно верно! Развертывание и связанная с ним автоматизация применимы не ко всем видам ПО, однако непрерывная доставка — это гораздо больше, чем просто развертывание. Мы поговорим об этом позже в этой главе.

В этой книге рассматриваются основные принципы непрерывной доставки и приводятся примеры их практического использования. Конкретные детали реализации в проекте наверняка будут уникальными, и возможно, вы не найдете здесь их точного описания, но что вы точно найдете, так это компоненты, необходимые для автоматизации непрерывной доставки, и принципы, соблюдая которые вы добьетесь наибольшего успеха.

Зачем нужна непрерывная доставка

Итак, что же это за штука такая? Начну с того, что непрерывная доставка (CD) означает для меня и почему я считаю ее такой важной:

Непрерывная доставка — это один из процессов современной профессиональной программной разработки.

Разберем это определение по частям:

- *Современная* — профессиональная разработка существует гораздо дольше, чем непрерывная доставка, хотя ребята, которые работали с перфокартами, были бы в восторге от CD! Одна из причин, по которой мы можем сейчас использовать непрерывную доставку, а тогда не могли, заключается в том, что CD отнимает много ресурсов процессора. Непрерывная доставка требует выполнения большого объема кода!

> Я даже не могу себе представить, сколько перфокарт потребовалось бы для описания типичного рабочего процесса CD!

- *Профессиональная* — если вы пишете программы для интереса, то вопрос о том, стоит ли вам возиться с непрерывной доставкой, остается открытым. Как правило, ее используют, когда действительно важно, чтобы программный продукт работал. Чем важнее создаваемое ПО, тем тщательнее следует продумывать непрерывную доставку. Кроме того, говоря о профессиональной разработке, мы вряд ли будем иметь в виду одного программиста, самостоятельно пишущего код. Обычно над продуктом работает несколько человек, а иногда даже сотни.
- *Программная разработка* — в других инженерных областях существуют своды стандартов и сертификатов, которые в разаботке, как правило, отсутствуют. Итак, проще говоря, программная разработка — это написание программ. Когда мы добавляем слово «*профессиональная*», мы имеем в виду профессиональное занятие разработкой ПО.
- *Процесс* — профессиональная разработка требует соблюдения определенных подходов, чтобы написанный нами код делал то, что мы задумали. Эти процессы касаются не столько способов, которыми отдельно взятый программист пишет код (хотя и это важно), сколько способности этого человека сотрудничать с другими разработчиками, чтобы создавать продукт профессионального уровня.

Непрерывная доставка — это совокупность процессов, которые необходимы группе профессиональных разработчиков, чтобы создавать программный продукт, соответствующий исходным целям его создателей.

Подождите, вы хотите сказать, что CD означает «непрерывная доставка»? Я думал, что это непрерывное развертывание!

Некоторые действительно расшифровывают эту аббревиатуру именно так (continuous deployment), и тот факт, что оба термина появились примерно в одно и то же время, вносит большую путаницу. В большинстве известных мне публикаций (не говоря уже о сообществе Continuous Delivery Foundation!) авторы предпочитают употреблять аббревиатуру CD для обозначения непрерывной доставки, поэтому в этой книге мы будем делать то же самое.

Непрерывная доставка

Непрерывная доставка — это совокупность процессов, которые необходимы группе профессиональных разработчиков, чтобы создавать программный продукт, соответствующий исходным целям его создателей.

Мое определение отражает то, что я считаю действительно крутым функционалом CD, но оно слишком далеко от стандартной формулировки, которую вы можете встретить. Посмотрим на определение, данное организацией Continuous Delivery Foundation (CDF) (http://mng.bz/YGXN):

Практика программной разработки, при которой команды выпускают релизы изменений ПО для пользователей безопасным, быстрым и устойчивым образом благодаря:

- **способности осуществлять выпуски в любое время;**
- **автоматизации выпуска релизов.**

Заметьте, что CD подразумевает два основных условия. Вы занимаетесь непрерывной доставкой, если:

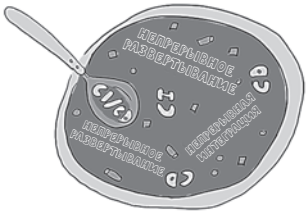
- выпускаете релизы ПО безопасно и в любое время;
- делаете выпуск максимально просто, буквально одним нажатием кнопки.

В этой книге подробно описаны необходимые действия и средства автоматизации, которые помогут вам добиться этих двух целей, а именно:

- Чтобы выпускать изменения безопасно и в любое время, ПО должно всегда находиться в состоянии готовности к релизу (в состоянии

Важным принципиальным изменением, произошедшим в CD по сравнению с CI, является переопределение значения фразы «функция готова». В CD «готова» означает «выпущена» (то есть осуществлен релиз). А процесс перехода от реализации изменений к выпуску автоматизирован, прост и быстр.

История «непрерывных» терминов



- **1994:** *термин «Непрерывная интеграция» (Continuous integration) вводится в книге Грейди Буча (Grady Booch) Object-Oriented Analysis and Design with Applications* [1] *(Addison-Wesley).*
- **1999:** *методика непрерывной интеграции впервые описана в книге Кента Бека (Kent Beck) Extreme Programming Explained*[2] *(Addison-Wesley).*
- **2007:** *дальнейшее развитие методика непрерывной интеграции получила в книге Пола Дюваля (Paul M. Duvall) Continuous Integration*[3] *(Addison-Wesley).*
- **2007:** *термин «непрерывное развертывание» определен в той же книге Дюваля.*
- **2009:** *принципы непрерывного развертывания популярно описаны в блоге Тимоти Фитца (Timothy Fitz) (http://mng.bz/2nmw).*
- **2010:** *методика непрерывной доставки, вдохновленная agile-манифестом*[4]*, описана в книге Continuous delivery*[5] *Джеза Хамбла (Jez Humble) и Дэвида Фарли (David Farley) (Addison-Wesley).*
- **2014:** *первая статья, дающая определение технологии CI/CD, Test Automation and Continuous Integration & Deployment (CI/CD) («Автоматизация тестирования и непрерывная интеграция и развертывание (CI/CD)»), опубликована сообществом Ravello (http://mng.bz/1opR).*
- **2016:** *в Википедию добавлена статья «CI/CD» (http://mng.bz/J2RQ).*

Вы, должно быть, думаете: «О'кей, Кристи, это все хорошо и замечательно, но что же на самом деле означает *доставлять*»? А *непрерывное развертывание*? И что такое *CI/CD*?

Мы действительно вынуждены оперировать множеством терминов! И что еще хуже, разные люди используют их по-разному. В их защиту можно сказать одно: это происходит потому, что некоторые из таких терминов даже не имеют определений!

Остановимся вкратце на эволюции этих терминов, чтобы лучше понять их значение. Непрерывная интеграция, непрерывная доставка и непрерывное развертывание — это термины, которые были созданы намеренно (или, в случае непрерывной интеграции, эволюционировали), а их авторы придавали им совершенно конкретные значения.

CI/CD — особый случай: похоже, этот термин никто не создавал и он появился на свет только потому, что люди, говорившие одновременно обо всех «непрерывных» операциях, нуждались в кратком термине (при этом полная аббревиатура CI/CD/CD почему-то не прижилась!).

Термин CI/CD в том виде, в каком он используется сегодня, означает инструменты и средства автоматизации, применяемые во всех непрерывных операциях — интеграции, доставке и развертывании.

---

[1] Буч Г. и др. «Объектно-ориентированный анализ и проектирование с примерами приложений».

[2] Бек К. «Экстремальное программирование». Санкт-Петербург, издательство «Питер».

[3] Дюваль Пол М. и др. «Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска».

[4] Манифест разработки программного обеспечения по методологии agile. — *Примеч. пер.*

[5] Хамбл Д., Фарли Д. «Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий».

релиза). В этом нам поможет непрерывная интеграция (continuous integration, CI).
- После проверки изменений методами CI процесс выпуска изменений должен проходить автоматически и его должно быть легко повторить.

Прежде чем я начну подробно рассказывать о том, как достичь этих целей, разберемся в терминологии.

Непрерывная доставка — это набор целей, к которым мы стремимся; способ их достижения может меняться от проекта к проекту. Тем не менее достигать этих целей наиболее эффективно помогает определенная последовательность действий, и именно им посвящена моя книга!

Интеграция

Непрерывная интеграция (Continuous Integration, CI) — старейшее из понятий, с которыми мы познакомились, однако это по-прежнему ключевой элемент непрерывной доставки. Начнем с простого — рассмотрим пока только интеграцию.

Что значит *интегрировать* программу? На самом деле часть фразы опущена: интегрировать объект нужно во что-то другое. В разработке интегрируемый объект — это изменение программного кода. Когда мы говорим об интеграции программного обеспечения, фактически мы имеем в виду

интеграцию изменений кода в существующее программное обеспечение.

Это именно то, чем в основном каждый день занимаются разработчики: изменяют код существующих частей программы. Этот процесс особенно интересен в команде: ее члены постоянно вносят изменения в код, и зачастую в одну и ту же часть продукта. Объединение этих изменений в единое целое и есть *интеграция*.

Интеграция программного обеспечения — это объединение изменений кода, сделанных несколькими людьми.

Иногда вам приходится создавать программу с нуля, но после первой успешной компиляции вы раз за разом будете интегрировать новые изменения в уже существующий программный продукт.

Как вы, вероятно, знаете по своему опыту, иногда этот процесс действительно идет криво. Например, когда я изменяю ту же строку кода, что и вы, и мы пытаемся объединить наши изменения, возникает конфликт и нам приходится вручную решать, как их интегрировать.

И еще одна маленькая деталь. Когда мы интегрируем изменения кода, мы не только просто соединяем их вместе; *мы еще и проверяем работоспособность измененного кода.* Можно сказать, что в аббревиатуре CI не хватает буквы *V* (Verification) —

верификация! Верификация уже заложена в процесс интеграции, поэтому, говоря об интеграции ПО, мы имеем в виду следующее:

Интеграция программного обеспечения — это объединение изменений кода, сделанных несколькими людьми, и проверка того, что получившийся код делает именно то, для чего он предназначен.

Кого волнуют все эти определения? Покажите мне уже наконец код!

Трудно выполнять свою работу последовательно и методично, если ей даже нельзя дать четкое определение. Уделить время тому, чтобы прийти к общему пониманию (через определения), а затем вернуться к основным принципам — это самый эффективный способ подняться на новый уровень!

Непрерывная интеграция

Посмотрим, как придать *интеграции* «*непрерывность*», на примере, не связанном с разработкой. Холли, шеф-повар, готовит соус для макарон. Она начинает с подбора продуктов: лука, чеснока, помидоров, специй. Чтобы приготовить соус, ей нужно *интегрировать* эти продукты вместе в правильном порядке и в нужном количестве.

Для этого каждый раз, когда она добавляет новый ингредиент, *она быстро пробует соус на вкус*. Исходя из вкуса, она может добавить больше какого-то продукта либо понимает, что забыла какой-то нужный ингредиент.

Дегустация помогает ей менять блюдо, проводя его через серию интеграций. Интеграция в данном случае подразумевает две вещи:

- объединение ингредиентов;
- проверку для подтверждения результата.

И это именно то, что означает слово «*интеграция*» в словосочетании *непрерывная интеграция*: объединение изменений кода, а также проверка его работоспособности, то есть объединение и верификация.

Холли повторяет эти шаги во время готовки. Если бы она попробовала соус только в конце, она гораздо меньше контролировала бы процесс и было бы уже поздно вносить необходимые изменения. Именно здесь вступает в дело «*непрерывность*». Вы должны интегрировать (объединять и верифицировать) свои изменения так часто, как только можете, и делать это как можно быстрее.

А как часто вы можете объединять и верифицировать программный продукт? Каждый раз, как только вы внесете изменения!

Непрерывная интеграция — это процесс, при котором объединение изменений кода происходит постоянно и каждое изменение проверяется при внесении.

Объединение изменений кода означает, что разработчики, использующие непрерывную интеграцию, фиксируют и переносят код в общую систему контроля версий каждый раз, когда вносят изменения, и проверяют их корректную работу, применяя автоматические средства верификации, такие как тесты и линтинг[1].

Автоматическая верификация? Линтинг? Не волнуйтесь, если вы не знаете, что это, эта книга поможет вам разобраться! Позже мы рассмотрим, как создавать автоматические проверки, благодаря которым работает непрерывная интеграция.

Что мы доставляем

От непрерывной интеграции мы переходим к непрерывной доставке, и для этого нужно вернуться немного назад. Почти в каждом определении, которое мы рассматриваем, упоминается доставка какой-то программы (например, я собираюсь начать рассказ об интеграции и доставке изменений в программу). Неплохо бы убедиться, что мы все имеем в виду одно и то же, когда говорим о программах (software), ведь в зависимости от проекта этот термин может обозначать совершенно разные вещи.

Словарик

Термин «*программное обеспечение*» (software) возник в противоположность термину «*аппаратное обеспечение*» (hardware). Аппаратное обеспечение — это собственно физические части компьютеров. Мы производим действия с этими частями, снабжая их инструкциями. Инструкции могут быть встроены непосредственно в аппаратное обеспечение или переданы ему во время работы с помощью программного обеспечения.

Доставляемые программы могут иметь различные формы (интеграция и доставка для каждой из них тоже будут проходить по-разному):

- *Библиотека* — если программа ничего не делает сама по себе, а предназначена только для использования в составе другого ПО, то это, скорее всего, библиотека.
- *Двоичный файл* — если программа предназначена для выполнения какой-то задачи, вероятно, это двоичный исполняемый файл. Это может быть сервис, или приложение, или инструмент, который выполняет определенные действия и затем завершает работу, или приложение для мобильного устройства, например планшета или телефона.
- *Конфигурация* — это понятие относится к информации, которую можно передать в двоичный файл, чтобы изменить его работу без перекомпилирования.

[1] Линтинг — проверка кода на соответствие стандартам. — *Примеч. пер.*

Как правило, оно обозначает средства, доступные системному администратору для внесения изменений в работающее ПО.

- *Образ* — образы контейнеров представляют собой особый тип двоичных файлов, ставший чрезвычайно популярным форматом совместного использования и распространения сервисов вместе с их конфигурацией, чтобы они могли выполняться независимо от операционной системы.
- *Сервис* — как правило, сервисы — это двоичные файлы, которые постоянно находятся в рабочем состоянии, ожидая запросов, на которые они отвечают, выполняя какие-либо действия или возвращая информацию. Иногда их также называют *приложениями*.

На разных этапах карьеры вы можете работать с отдельными видами программ или со всеми сразу. Но вне зависимости от того, с какой программой вы будете иметь дело, для ее создания вам потребуется осуществлять *интеграцию* и *доставку* вносимых в нее изменений.

Доставка

Что означает *доставить* изменения в программу, зависит от того, какой программный продукт вы создаете, кто его использует и как. Обычно доставка изменений имеет отношение либо только к одному из процессов сборки, релиза и развертывания ПО, либо сразу ко всем:

- *Сборка* — ряд действий, направленных на получение кода (включая его изменения) и преобразование его в пригодную для использования форму. Обычно это означает компиляцию кода, написанного на языке программирования, в машинный язык. Иногда это также означает помещение кода в пакет, например в образ контейнера или похожий, который менеджер пакетов сможет распознать (например, в пакет PyPI для Python).
- *Публикация* — копирование программного обеспечения в репозиторий (место хранения программ), например, путем загрузки образа или библиотеки в реестр пакетов.
- *Развертывание* — копирование ПО в место, где оно должно быть запущено, и приведение его в рабочее состояние.

Сборка осуществляется в рамках процесса интеграции, чтобы проверить, что все внесенные изменения корректно работают вместе.

Можно проводить развертывание без выпуска релиза: например, развернуть новую версию программы, но не направлять на нее трафик. Однако чаще всего развертывание все же предполагает релиз; все зависит от того, где оно происходит. Если вы развертываете ПО в производственной среде (в продакшене), то выпуск релиза происходит одновременно с развертыванием. Подробнее о развертывании см. в главе 10.

- *Выпуск релиза* — предоставление продукта пользователям. Релиз можно выпускать путем загрузки образа или библиотеки в репозиторий или установки параметров конфигурации для направления определенного процента трафика на развернутый экземпляр программы.

Словарик

Мы осуществляем *сборку* программного обеспечения с тех пор, как у нас появились языки программирования. Это настолько распространенный процесс, что первые системы, которые выполняли то, что мы сейчас называем *непрерывной доставкой*, именовались *системами сборки* (build systems). Этот термин стал таким привычным, что даже сегодня вы слышите слово *сборка* (или *билд*), хотя при этом имеется в виду последовательность задач в пайплайне развертывания, которая выполняется для преобразования ПО (подробнее об этом см. в главе 2).

Непрерывная доставка/непрерывное развертывание

Теперь вы знаете, что значит доставлять изменения в программное обеспечение, но что значит делать это непрерывно? В контексте CI мы узнали, что «*непрерывно*» означает «*как можно быстрее*». Так ли это в случае CD? И да и нет. Непрерывность в CD лучше представить в виде диапазона:

Создаваемое вами ПО, должно находиться в состоянии, при котором в любое время можно выполнить его сборку, релиз и/или развертывание. Но как часто вы решите доставлять это ПО, зависит только от вас.

- 2009: статья в блоге, описывающая принципы непрерывного развертывания
- 2010: методика непрерывной доставки описана в книге с аналогичным названием

«А как насчет непрерывного развертывания?» — спросите вы. Отличный вопрос. Если снова обратиться к истории, то можно заметить, что эти два термина, *непрерывная доставка* и *непрерывное развертывание*, получили широкое распространение практически одновременно. Что же происходило в то время, когда эти термины вошли в оборот?

Это был переломный этап в разработке программного обеспечения: старые способы создания программных продуктов, когда все зависело от людей и ручных операций, жесткое разделение разработки и эксплуатации программ (интересно, что термин *DevOps* (development&operations, разработка и эксплуатация) появился примерно в то же время) и четко определенные процессы разработки ПО (например, *этап тестирования*) — все это начало меняться (со сдвигом влево). Непрерывное развертывание и непрерывная доставка совместно легли в основу практик, появившихся в это время. *Непрерывное развертывание* означает, что:

выпуск релиза рабочей версии ПО для пользователей осуществляется автоматически при каждом коммите.

Словарик

Сдвиг влево — это процесс поиска дефектов на начальных этапах создания программного продукта.

Непрерывное развертывание — это дополнительный этап непрерывной доставки. Непрерывная доставка позволяет осуществлять также и непрерывное развертывание. Постоянное пребывание ПО в состоянии готовности к релизу и автоматизация процесса доставки освобождают вас от необходимости выбирать, что будет лучше для проекта.

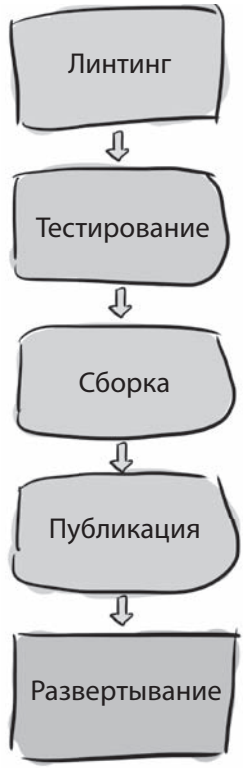
Если непрерывное развертывание на самом деле относится к выпуску релизов, почему бы не назвать его непрерывным релизом?

Отличная мысль! *Непрерывный релиз* — более точное название, и оно прояснило бы, как эту методику можно применять к ПО, которое не нужно развертывать. Однако прижился именно термин «*непрерывное развертывание*». Пример непрерывного релиза см. в главе 9.

Элементы непрерывной доставки

Остальная часть этой книги посвящена описанию основных составляющих процесса CD:

> **Методика программной разработки, при которой релиз рабочего программного продукта для пользователей осуществляется так быстро, как того требует проект, а его сборка производится таким образом, чтобы релизы можно было гарантированно безопасно выпускать в любое время.**



Вы узнаете, как использовать CI, чтобы ваш продукт всегда был готов к релизу, и как сделать доставку автоматической и повторяемой. Это сочетание позволяет вам выбрать, пойти ли на крайние меры и осуществлять релиз продукта при каждом изменении (непрерывное развертывание) или же предпочесть выпуск релизов в другом режиме. В любом случае вы можете быть уверены, что весь процесс автоматизирован, что позволит осуществлять доставку с необходимой частотой.

В основе такой автоматизации будет лежать пайплайн непрерывной доставки. В этой книге я подробно расскажу, что собой представляет каждая из этих задач. Вы обнаружите, что вне зависимости от того, какой программный продукт вы создаете, многие из них будут вам полезны.

> *Пайплайн? Задача? Что это?*
>
> Чтобы узнать, читайте следующую главу!

В следующей таблице перечислены виды программ, которые мы рассмотрели выше, и указано, что включает доставка каждого из них.

	Доставка включает сборку?	**Доставка включает публикацию?**	**Доставка включает развертывание?**	**Доставка включает релиз?**
Библиотека	Зависит от проекта	Да	Нет	Да
Двоичный файл	Да	Обычно	Зависит от проекта	Да
Конфигурация	Нет	Скорее всего, нет	Обычно	Да
Образ контейнера	Да	Да	Зависит от проекта	Да
Сервис	Да	Обычно	Да	Да

Заключение

В сфере непрерывной доставки существует множество терминов и противоречивых определений. В этой книге мы используем аббревиатуру CD для обозначения *непрерывной доставки*, которая включает в себя непрерывную интеграцию (CI), развертывание и выпуск релизов. Основное внимание я уделю автоматизации, которая необходима для использования CD вне зависимости от типа доставляемого программного обеспечения.

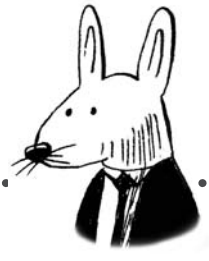
Итоги

- Непрерывная доставка полезна при разработке любого типа программ.
- Непрерывная доставка необходима, чтобы создавать профессиональные продукты, работая в команде.
- Осуществляя непрерывную доставку, используйте непрерывную интеграцию, чтобы гарантировать, что ваш продукт всегда находится в состоянии готовности к доставке.
- Непрерывная интеграция — это процесс, при котором объединение изменений кода происходит постоянно и каждое изменение проверяется при внесении.
- Другим условием непрерывной доставки является автоматизация, позволяющая выпускать релизы максимально просто, буквально одним нажатием кнопки.
- Непрерывное развертывание — это дополнительный этап, который можно добавить по желанию, если это важно для проекта; в этом случае доставка ПО будет происходить автоматически при каждом коммите.

Далее...

Вы познакомитесь с основами автоматизации непрерывной доставки и терминологией, принятой в этой области, и тем самым подготовитесь к дальнейшему изучению материала.

2 Базовый пайплайн



В этой главе

- ✓ Работа с основными составляющими процесса CD: пайплайнами и задачами
- ✓ Элементы базового пайплайна CD: линтинг, тестирование, сборка, публикация и развертывание
- ✓ Роль автоматизации в выполнении пайплайнов: веб-хуки, события и запуск
- ✓ Знакомство с терминологией в области CD

Прежде чем перейти к конкретным деталям создания крутых пайплайнов непрерывной доставки (CD pipelines), узнаем, что вообще такое пайплайн. В этой главе мы в общих чертах рассмотрим несколько пайплайнов и определим основные элементы, которые должны присутствовать в большинстве пайплайнов CD.

Веб-сайт Cat Picture

Чтобы понять, из чего состоят базовые пайплайны CD, рассмотрим сначала пайплайны, используемые для веб-сайта Cat Picture. Cat Picture — это лучший сайт для поиска и обмена фотографиями котиков! Он довольно прост, но, поскольку это популярный ресурс, управляющая им компания (Cat Picture, Inc.) создала его в виде нескольких сервисов.

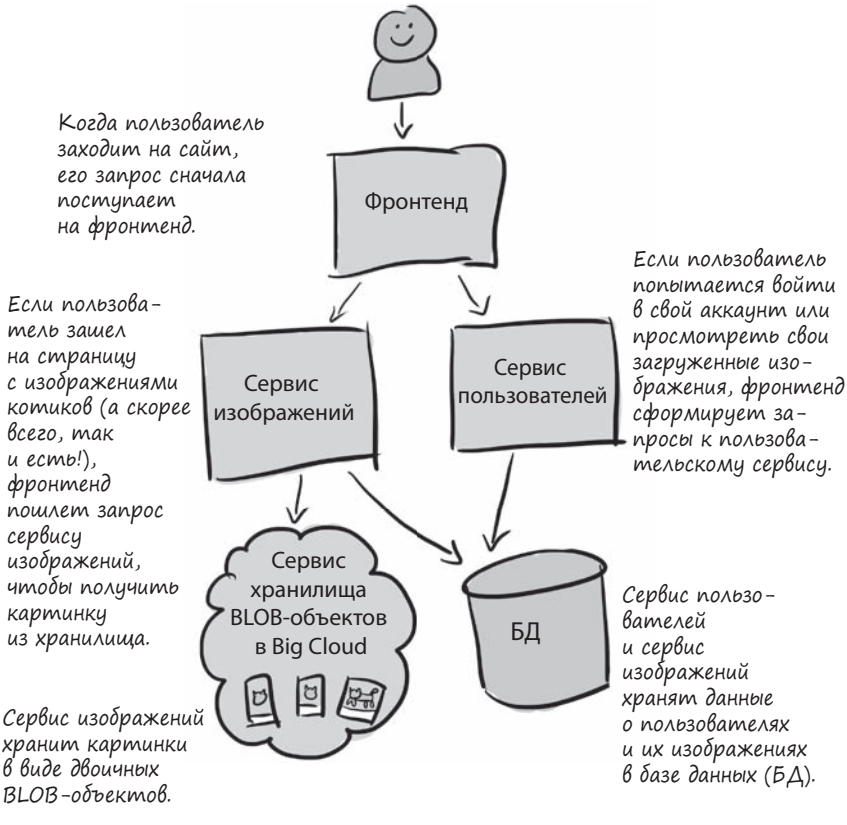
Компания запускает сайт Cat Picture в облаке (ее облачный провайдер называется Big Cloud, Inc.) и использует сервисы Big Cloud, такие как Big Cloud Blob Storage (хранилище BLOB-объектов[1] Big Cloud).

Что это опять за CD?

В этой книге аббревиатурой CD мы обозначаем непрерывную доставку. Подробнее см. главу 1.

Что такое пайплайн?

Не волнуйтесь, об этом мы поговорим через пару страниц!



[1] BLOB (Binary Large Object) — массив двоичных данных. Здесь — специальный тип данных, предназначенный для хранения изображений. Хранилище BLOB-объектов оптимизировано для хранения больших объемов неструктурированных данных. — *Примеч. науч. ред.*

Исходный код сайта Cat Picture

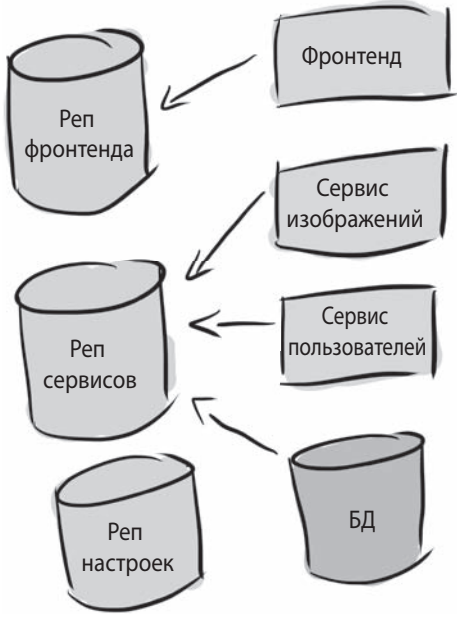
Схема архитектуры показывает, как устроен сайт Cat Picture, но для понимания пайплайна CD необходимо обсудить один важный момент: где находится код?

В главе 1 мы рассмотрели элементы CD, половина из которых связана с использованием непрерывной интеграции (CI) для обеспечения постоянной готовности ПО к релизу. Вспомним определение CI:

> **Непрерывная интеграция — это процесс, при котором объединение изменений кода происходит постоянно и каждое изменение проверяется при внесении.**

Если проанализировать, что мы на самом деле делаем, осуществляя CD, то мы увидим, что в основном CD означает изменение кода. То есть входными данными для пайплайнов CD является исходный код. Фактически именно это и отличает пайплайны CD от других видов автоматизации рабочих процессов: пайплайны CD почти всегда используют исходный код в качестве входных данных.

Прежде чем рассматривать пайплайны CD сайта Cat Picture, нам нужно понять, как организован и где хранится его исходный код. Разработчики сайта Cat Picture хранят свой код в нескольких репозиториях (репах):

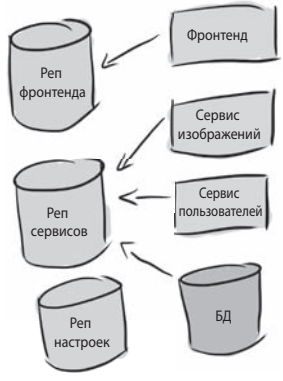


- В репозитории фронтенда хранится код внешнего интерфейса.
- Сервис изображений, сервис пользователей и схемы баз данных хранятся в репозитории сервисов.
- И наконец, сайт Cat Picture использует подход «конфигурация как код» для управления конфигурацией (подробнее об этом в главе 3), храня ее в репозитории настроек.

Организовать код сайта Cat Picture можно множеством других способов, у каждого из которых есть свои плюсы и минусы.

> ***Система контроля версий***
>
> Использование системы контроля версий, такой как Git, является необходимым условием для реализации CD. Без хранения кода вместе с историей и системой обнаружения конфликтов осуществлять CD практически невозможно. Подробнее об этом в главе 3.

Пайплайны сайта Cat Picture



Поскольку сайт Cat Picture состоит из нескольких сервисов, а весь код и конфигурация, необходимые для него, распределены по нескольким репозиториям, он управляется несколькими пайплайнами CD. Мы подробно рассмотрим все эти пайплайны в следующих главах, когда будем изучать более продвинутые их варианты, а пока остановимся на базовом пайплайне, который используется для сервиса пользователей и сервиса изображений.

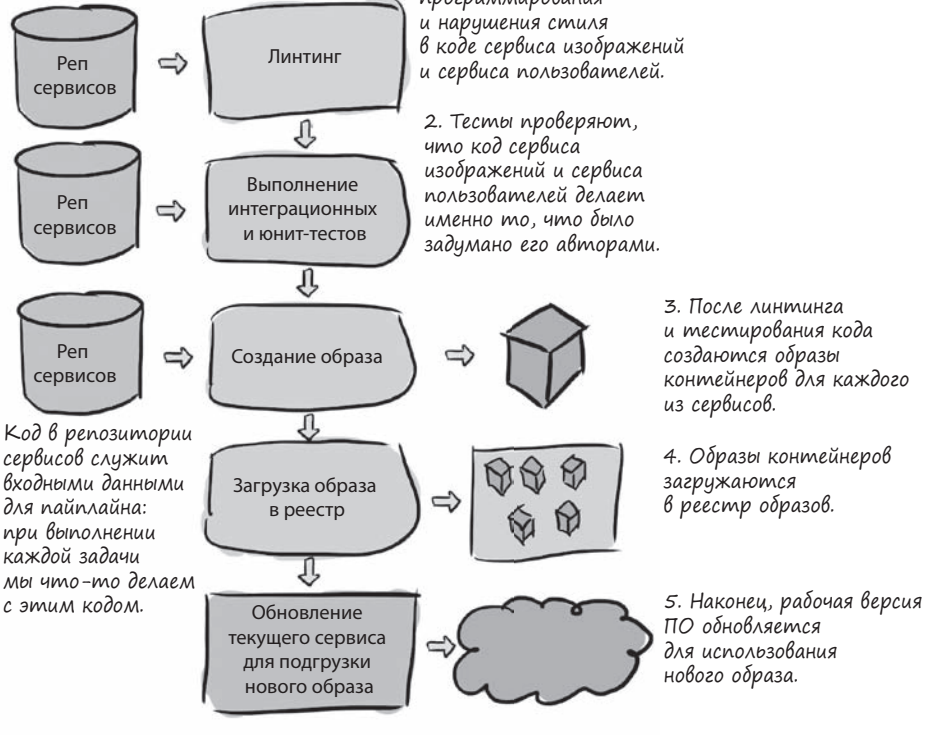
Так как эти два сервиса очень похожи, для них используется один и тот же пайплайн, и он содержит все основные элементы любого пайплайна.

Этот пайплайн не только используется для веб-сайта Cat Picture, но и содержит основные элементы пайплайнов, описанных далее в книге!

> Когда все это на самом деле выполняется? Об этом мы кратко поговорим чуть позже, а более подробно — в главе 10.

Словарик

Образы контейнеров — это исполняемые программные пакеты, содержащие все необходимое для работы программы.

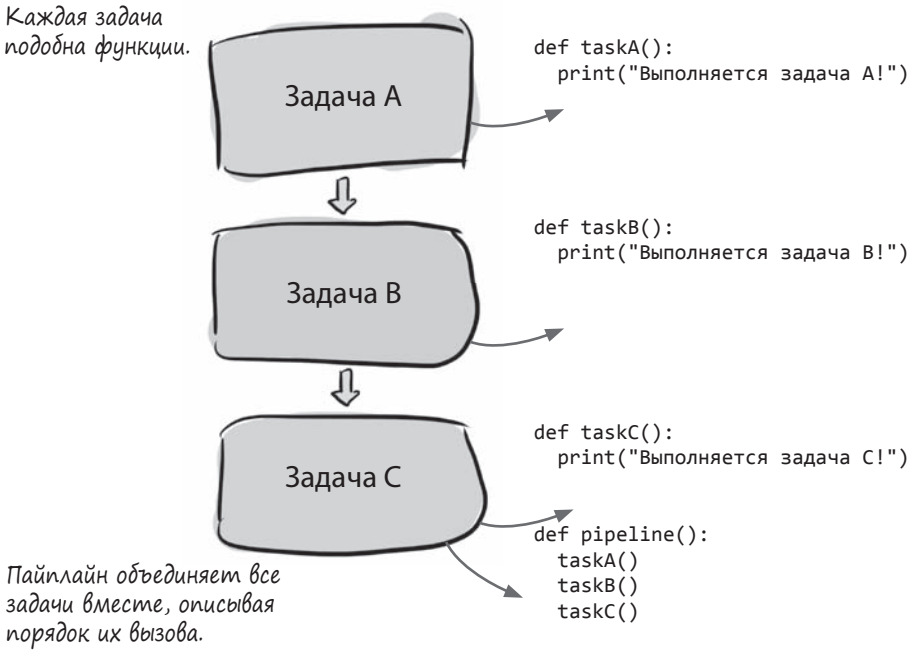


Что же такое пайплайн и что такое задача

На нескольких предыдущих страницах мы рассмотрели пайплайн веб-сайта Cat Picture, но что такое пайплайн вообще? В сфере CD существует множество терминов. Хотя здесь мы используем термин «*пайплайн*» (pipeline, конвейер), иногда используют и другие термины, например «*рабочий процесс*» (workflow). Мы подробнее обсудим терминологию в конце главы, а пока остановимся на пайплайнах и задачах.

Задачи (tasks) — это отдельные действия, которые вы можете выполнять; их можно представить себе как некие функции. А *пайплайн* — это как точка входа, дающая доступ к коду, который вызывает все функции в нужное время и в определенной последовательности.

Ниже приведен пример пайплайна в виде кода на языке Python, состоящий из трех задач: сначала выполняется задача A, затем задача B, а заканчивается пайплайн задачей C.



Пайплайны CD запускаются снова и снова; о том, когда именно происходит их запуск, мы поговорим чуть позже. Если бы мы запустили функцию `pipeline()` (представляющую собой пайплайн, показанный на рисунке), то получили бы следующий результат:

```
Выполняется задача A!
Выполняется задача B!
Выполняется задача C!
```