КОНСПЕКТ ЛЕКЦИИ · LECTA

CI/CD, стратегии ветвления Git и варианты деплоя

Длительность: 58 мин 29 с
Подготовлено: 2026-09-15 23:02

Кратко

> Лекция посвящена доставке кода в продакшн через CI/CD и связанным с этим практикам работы с Git. Лектор объясняет разницу между Continuous Integration, Continuous Delivery и Continuous Deployment, а также показывает, как эти подходы влияют на сборку, тестирование и выкладку артефактов. Отдельный большой блок посвящён стратегиям ветвления в Git: GitFlow, GitHub Flow, GitLab Flow, trunk-based и fork-based workflow. В конце разбираются стратегии деплоя — простой, Blue-Green, Canary, feature flags и rolling update — с их плюсами, минусами и сценариями применения.

Оглавление

Конспект

ГЛАВА 1

Введение в CI/CD и зачем он нужен

00:08 — 05:00

Лектор вводит тему доставки кода в продакшн и объясняет, что CI/CD нужен для автоматизации пути от разработки до выкладки. Поясняется, как раньше релиз был долгим, ручным и рискованным процессом с простоями и высоким шансом ошибки. Затем формулируется, что CI/CD повышает скорость, повторяемость и качество за счёт автоматизированных сборок и тестов.

ГЛАВА 2

CI, CD и различие между Delivery и Deployment

05:00 — 11:40

Разбирается расшифровка CI/CD и смысл каждой части: Continuous Integration отвечает за сборку и первичное тестирование, а CD — за доставку артефакта дальше по средам. Лектор отдельно подчёркивает разницу между Continuous Delivery и Continuous Deployment: в первом случае финальный деплой на продакшн ручной, во втором — автоматический. Также объясняется, почему в критичных системах автоматический деплой на прод используется осторожно.

ГЛАВА 3

Пайплайны, тесты и примеры CI/CD-схем

11:40 — 18:20

На примерах GitLab и типовых схем показывается, как выглядит пайплайн: коммит, сборка, юнит- и интеграционные тесты, стейджинг и выкладка в продакшн. Лектор отмечает, что pipeline — это цепочка автоматических действий, а тестирование может выполняться на разных этапах. В качестве иллюстрации приводятся и простые, и чрезмерно сложные реальные схемы CI/CD.

ГЛАВА 4

CI/CD в других областях и роль Git

18:20 — 23:20

Лекция расширяет идею DevOps на другие домены, прежде всего на MLOps, где модели и данные тоже рассматриваются как артефакты. Далее объясняется, почему CI/CD тесно связан с Git: код и описание пайплайнов обычно хранятся в репозитории, а значит, важны правила ветвления и работы с историей. Подчёркиваются практики аккуратной работы с коммитами, gitignore, code review и rebase.

ГЛАВА 5

Стратегии ветвления Git: обзор и сравнение

23:20 — 31:40

Лектор вводит понятие стратегии ветвления и объясняет, что она определяет способ организации разработки и доставки кода. Сравниваются несколько моделей: OneBranchFlow, GitFlow, GitHub Flow, GitLab Flow, trunk-based и fork-based workflow. Для каждой подчёркивается компромисс между простотой, управляемостью, историей релизов и удобством интеграции с процессами Ops.

ГЛАВА 6

Практические замечания по Git и выбор стратегии

31:40 — 39:10

На вопрос из аудитории лектор поясняет разницу между ветками, тегами и релизными ветками в GitFlow и GitLab Flow. Отдельно обсуждается, зачем сохранять релизные ветки и когда это полезно для истории, сравнения версий и поиска потерянных изменений. В ответе также даётся практический совет: для новых рабочих проектов проще всего стартовать с GitHub Flow.

ГЛАВА 7

Варианты деплоя: от простого до Blue-Green и Canary

39:10 — 48:20

После разговора о ветвлении лекция переходит к стратегиям деплоя как отдельному слою поверх CI/CD. Сначала разбирается простой деплой и частичное обновление как базовые, но рискованные подходы. Затем подробно объясняются Blue-Green Deploy и Canary Deploy: их механика, преимущества, стоимость и способы переключения трафика.

ГЛАВА 8

Feature flags и rolling update

48:20 — 58:29

Лектор показывает, что feature flags позволяют выкатывать код с включением и выключением функционала внутри приложения, не полагаясь на несколько копий среды. Затем объясняется rolling update как постепенное обновление реплик с проверкой здоровья через пробы, при котором трафик переводится только после готовности всех экземпляров. Обсуждаются ограничения: необходимость stateless-приложений и риск при высокой нагрузке или неудачном совпадении с апдейтом.

Ключевые цитаты

> «И уже потом это попадает в продакшн. И вот как раз за это отвечает процесс CI/CD.»
>
> 04:00

> «Непрерывная интеграция — это у нас сборка и первичное тестирование.»
>
> 06:20

> «Continuous Delivery предполагает только ручной деплой на продакшен.»
>
> 08:40

> «Continuous Deployment — это как раз про то, что у нас и выкатка в прод тоже автоматизирована.»
>
> 10:00

> «Мы стараемся коммитить более часто и более мелкими.»
>
> 33:00

> «Не забывать делать ребейс, потому что если вы отвели ветку, и пока вы там работали, основная ветка убежала, то всегда хорошим тоном считается ребейзинг.»
>
> 35:20

> «У нас есть ветка Main, которая содержит самый актуальный код. Но этот Main мы никак не используем вообще.»
>
> 25:00

> «Самый простой деплой, который существует, это, в общем-то, просто деплой.»
>
> 40:00

> «Blue-Green Deploy. То есть, у нас есть две идентичные среды.»
>
> 42:20

> «Можно мгновенно откатить, если вы переключили, что-то пошло не так — быстренько откатили обратно на плечо, которое ещё не обновилось.»
>
> 43:40

> «Идея в чём? Мы можем просто поднять рядом вторую версию сервиса и переключать трафик.»
>
> 45:50

> «Сервисы состоят из нескольких реплик, мы их обновляем постепенно.»
>
> 53:20

Глоссарий

CI/CD

Подход к автоматизации сборки, тестирования и доставки кода в продакшн; включает Continuous Integration и Continuous Delivery/Deployment.

Continuous Integration

Часть CI/CD, где код автоматически собирается и проходит первичное тестирование.

Continuous Delivery

Режим, в котором артефакт доводится до готовности к продакшн-деплою, но финальный выкладочный шаг остаётся ручным.

Continuous Deployment

Режим, где выкладка в продакшн тоже автоматизирована и происходит без ручного подтверждения.

Артефакт

Результат сборки, который дальше тестируется и доставляется в другие среды, вплоть до продакшна.

Пайплайн

Последовательность автоматических шагов, по которым артефакт проходит сборку, тестирование и деплой.

Стейджинг

Промежуточная среда, где проверяют собранный артефакт перед продакшном.

GitFlow

Стратегия ветвления с отдельными develop, feature, release и hotfix ветками и релизами через main.

GitHub Flow

Простая стратегия, где есть основная ветка и короткоживущие feature-ветки, которые вмерживаются обратно.

GitLab Flow

Вариант ветвления, где релизная ветка создаётся от стабильной версии, проходит проверку и потом сливается обратно.

Trunk-based development

Стратегия, при которой разработка ведётся в одной основной ветке с короткими ветками и частыми вливаниями.

Blue-Green Deploy

Стратегия деплоя с двумя идентичными средами, между которыми переключают трафик после проверки обновлённой копии.

Canary Deploy

Стратегия, где новую версию выкатывают на небольшую долю трафика и постепенно увеличивают охват.

Feature flags

Флаги внутри приложения, которые позволяют включать и выключать функциональность без отдельного деплоя разных версий.

Rolling update

Постепенное обновление реплик сервиса, при котором новая версия вводится поэтапно, а трафик переводится после успешных проверок.

Hotfix

Срочная короткая ветка для микроскопического исправления, которую создают прямо от main и быстро возвращают обратно.

Вопросы для самопроверки

1. **Что такое CI/CD и какую проблему он решает?**

   ОТВЕТ. CI/CD автоматизирует сборку, тестирование и доставку кода, сокращая ручной труд, риск ошибок и простои при релизе.

2. **В чём разница между Continuous Delivery и Continuous Deployment?**

   ОТВЕТ. Delivery заканчивается ручным нажатием кнопки на этапе выкладки в прод, а Deployment доводит до автомата и сам финальный деплой.

3. **Почему CI/CD особенно полезен при наличии нескольких стендов?**

   ОТВЕТ. Потому что одинаковый пайплайн позволяет повторяемо проверять артефакт на разных средах и заранее ловить ошибки.

4. **Зачем нужна стратегия ветвления в Git?**

   ОТВЕТ. Она задаёт правила организации разработки, вливания изменений и подготовки релизов, что упрощает интеграцию с CI/CD.

5. **Чем GitFlow отличается от GitHub Flow?**

   ОТВЕТ. GitFlow использует develop, feature, release и hotfix ветки и больше ориентирован на релизы, а GitHub Flow проще: основная ветка и короткие feature-ветки.

6. **Когда уместен trunk-based development?**

   ОТВЕТ. Когда команда умеет быстро и аккуратно интегрировать изменения в одну основную ветку и хочет минимизировать ветвление.

7. **В чём идея Blue-Green Deploy?**

   ОТВЕТ. Есть две идентичные среды: одна активна, другая обновляется и проверяется, после чего трафик переключается на новую.

8. **Как работает Canary Deploy?**

   ОТВЕТ. Новая версия получает лишь часть трафика, затем доля увеличивается, если метрики и поведение в норме.

9. **Что дают feature flags по сравнению с отдельными средами?**

   ОТВЕТ. Они позволяют включать и выключать функционал внутри одной версии кода без необходимости держать несколько копий среды.

10. **Почему rolling update требует stateless-приложений?**

    ОТВЕТ. Потому что старые и новые реплики должны сосуществовать одновременно, не ломая состояние и совместную работу сервиса.

Карточки для повторения

#	ВОПРОС	ОТВЕТ
1	Как расшифровывается CI/CD?	Continuous Integration и Continuous Delivery или Continuous Deployment. CI отвечает за автоматическую интеграцию изменений: сборку кода и первичное тестирование.
2	Чем Continuous Delivery отличается от Continuous Deployment?	При Continuous Delivery весь путь автоматизирован, но финальный деплой на production делается вручную. При Continuous Deployment выкат в продакшн полностью автоматический.
3	Почему для критичных систем осторожнее подходят к Continuous Deployment?	Полностью автоматический выкат в production менее безопасен: чем критичнее система, тем осторожнее с автоматизацией прода.
4	Из каких типовых этапов состоит CI/CD-пайплайн?	Код коммитится, начинается билд, прогоняются юнит- и интеграционные тесты, артефакт деплоится в стейджинг и потом в production.
5	Как идея CI/CD переносится в MLOps?	Вместо обычного артефакта рассматривается модель машинного обучения: её циклически обучают, упаковывают как артефакт (образ или вольюм) и автоматически деплоят.
6	Что такое стратегия ветвления в Git?	Это набор правил о том, как команда разрабатывает ПО, как пушит изменения, как интегрируется в общую ветку и как готовится релиз.
7	Как устроена ветка Main в GitFlow и зачем нужен hotfix?	Main хранит самый актуальный стабильный код, но активно не используется — только для релизов. Для срочных мелких изменений создаётся ветка hotfix прямо от main и быстро возвращается обратно.
8	Чем Blue-Green Deploy отличается от Canary-деплоя по стоимости и гибкости?	Blue-Green требует двух идентичных сред («плечей»), одно из которых постоянно простаивает, поэтому он дорогой, но даёт мгновенный откат. Canary дешевле и гибче: новая версия поднимается рядом со старой, и трафик переводится постепенно (например, 25% на новую версию).