ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

РЕФЕРАТ

по дисциплине: «**Администрирование информационных систем**»

на тему «**Построение CI/CD пайплайна для автоматического тестирования и развертывания информационной системы**»

Выполнил: студент гр. ПРО-436Б
Думбов Д. С.
Проверил:
доцент каф. ВМиК
Чернышев Е. С.

Уфа 2025 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире разработки программного обеспечения скорость и качество выпуска новых версий продуктов являются критическими факторами конкурентоспособности. Традиционные подходы к разработке, когда код интегрируется и тестируется лишь на финальных этапах проекта, всё чаще уступают место методологиям непрерывной интеграции и непрерывной доставки (CI/CD). Эти практики позволяют командам разработчиков автоматизировать процессы сборки, тестирования и развёртывания, существенно сокращая время от написания кода до его появления в продуктивной среде.

CI/CD-пайплайн представляет собой автоматизированную последовательность этапов, через которые проходит программный код: от момента фиксации изменений в системе контроля версий до развёртывания на целевых серверах. Каждый этап пайплайна выполняет определённую функцию — сборку, статический анализ, модульное тестирование, интеграционное тестирование, развёртывание в тестовые и продуктивные окружения.

Актуальность темы обусловлена тем, что в условиях Agile-разработки и DevOps-культуры автоматизация процессов становится не просто желательной, а необходимой. Согласно отчёту DORA (DevOps Research and Assessment) за 2023 год, организации с развитыми практиками CI/CD демонстрируют в 208 раз более частые развёртывания и в 106 раз более быстрое восстановление после сбоев по сравнению с организациями без таких практик.

Цель данного реферата — рассмотреть принципы построения CI/CD-пайплайнов, изучить основные инструменты и технологии, применяемые для автоматизации процессов тестирования и развёртывания информационных систем, а также проанализировать стратегии развёртывания и вопросы обеспечения безопасности.

1 Понятие CI/CD: определение и ключевые концепции

1.1 Непрерывная интеграция (CI)

CI (Continuous Integration, непрерывная интеграция) — это практика разработки программного обеспечения, при которой разработчики регулярно, часто несколько раз в день, объединяют свои изменения кода в общий репозиторий. Каждое такое объединение автоматически проверяется с помощью сборки проекта и запуска тестов, что позволяет быстро обнаруживать ошибки интеграции и конфликты между изменениями разных разработчиков.

Мартин Фаулер, один из пионеров методологии, определяет непрерывную интеграцию как «практику разработки программного обеспечения, при которой члены команды часто интегрируют свою работу, обычно каждый человек интегрирует хотя бы ежедневно, что приводит к множеству интеграций в день» [Fowler, 2006]. Ключевым принципом является правило: «Если что-то болезненно, делайте это чаще».

Основные принципы непрерывной интеграции:

1) Единый репозиторий исходного кода — весь код проекта хранится в одной системе контроля версий.

2) Автоматизированная сборка — процесс сборки должен выполняться автоматически и быть воспроизводимым.

3) Самотестирующаяся сборка — сборка включает выполнение автоматических тестов.

4) Ежедневные коммиты — каждый разработчик коммитит в основную ветку ежедневно.

5) Быстрая сборка — сборка должна занимать не более 10 минут.

1.2 Непрерывная доставка и развёртывание (CD)

CD может означать как Continuous Delivery (непрерывная доставка), так и Continuous Deployment (непрерывное развёртывание). Несмотря на схожесть названий, эти практики имеют существенные различия.

Continuous Delivery (непрерывная доставка) подразумевает, что код всегда находится в состоянии, готовом к развёртыванию в продуктивную среду. После прохождения всех автоматических проверок артефакт сборки может быть развёрнут в production в любой момент, однако решение о развёртывании принимается вручную командой или бизнесом.

Continuous Deployment (непрерывное развёртывание) идёт дальше — каждое успешно прошедшее все этапы тестирования изменение автоматически разворачивается в продуктивной среде без участия человека. Это требует высокой культуры тестирования и развитых практик мониторинга.

Согласно документации GitLab, «CI/CD — это метод частой доставки приложений клиентам путём внедрения автоматизации на этапах разработки приложений» [GitLab, 2024]. Ключевые преимущества внедрения CI/CD:

1) Сокращение времени выхода продукта на рынок (Time-to-Market) — автоматизация позволяет выпускать обновления в считанные часы вместо недель.

2) Повышение качества кода за счёт автоматического тестирования на каждом этапе разработки.

3) Снижение рисков при развёртывании благодаря частым, небольшим и обратимым обновлениям.

4) Улучшение обратной связи — разработчики узнают о проблемах в течение минут, а не дней.

2 Компоненты CI/CD пайплайна

Эффективный CI/CD-пайплайн состоит из нескольких ключевых компонентов, каждый из которых выполняет определённую роль в процессе автоматизации разработки и развёртывания программного обеспечения.

2.1 Системы контроля версий

Система контроля версий (Version Control System, VCS) является фундаментом любого CI/CD-пайплайна. Она обеспечивает централизованное хранение исходного кода, отслеживание всех изменений и координацию работы команды разработчиков. Наиболее распространённой системой контроля версий является Git, созданный Линусом Торвальдсом в 2005 году для разработки ядра Linux.

Git использует модель распределённого контроля версий, где каждый разработчик имеет полную копию репозитория со всей историей изменений. Это обеспечивает высокую отказоустойчивость и возможность работы в автономном режиме. Популярными платформами для хостинга Git-репозиториев являются GitHub, GitLab и Bitbucket.

Механизм ветвления (branching) позволяет разработчикам работать над различными функциями параллельно, не мешая друг другу. Наиболее популярные стратегии ветвления:

1) Git Flow — использует отдельные ветки для разработки (develop), релизов (release) и исправлений (hotfix).

2) GitHub Flow — упрощённая модель: main-ветка и feature-ветки с pull request.

3) Trunk-Based Development — все разработчики работают в одной ветке с короткоживущими feature-ветками.

2.2 Серверы непрерывной интеграции

CI-сервер — это центральный компонент пайплайна, который автоматически реагирует на изменения в репозитории и запускает определённые задачи. К наиболее популярным CI-серверам относятся Jenkins, GitLab CI/CD, GitHub Actions, CircleCI и TeamCity.

Jenkins — open-source сервер автоматизации с богатейшей экосистемой плагинов (более 1800). Jenkins позволяет создавать сложные пайплайны с использованием декларативного или скриптового синтаксиса в файле Jenkinsfile. Благодаря своей гибкости и зрелости, Jenkins остаётся одним из самых популярных инструментов в enterprise-среде.

GitLab CI/CD — встроенный в платформу GitLab инструмент непрерывной интеграции, конфигурируемый через файл .gitlab-ci.yml. Отличается тесной интеграцией с репозиторием, встроенным реестром контейнеров и простотой настройки. Поддерживает Auto DevOps для автоматической настройки пайплайна.

GitHub Actions — сервис автоматизации от GitHub, использующий YAML-файлы для определения рабочих процессов (workflows). Предоставляет бесплатные минуты выполнения для публичных репозиториев и обширный маркетплейс готовых действий (более 15000 actions).

2.3 Инструменты автоматического тестирования

Автоматическое тестирование — краеугольный камень CI/CD. Без надёжного набора тестов невозможно гарантировать качество кода при частых релизах. Концепция «пирамиды тестирования» Майка Кона определяет оптимальное соотношение тестов разных уровней.

Модульные тесты (Unit Tests) — составляют основание пирамиды. Проверяют работу отдельных компонентов системы изолированно.

Выполняются быстро (миллисекунды) и должны составлять 70-80% всех тестов. Фреймворки: JUnit (Java), pytest (Python), Jest (JavaScript), NUnit (.NET).

Интеграционные тесты — проверяют взаимодействие между компонентами системы, включая работу с базами данных, внешними API и другими сервисами. Требуют более сложной инфраструктуры, часто используют тестовые контейнеры (Testcontainers).

End-to-End тесты (E2E) — вершина пирамиды. Имитируют действия реального пользователя и проверяют работу системы целиком. Инструменты: Selenium, Cypress, Playwright, Puppeteer. Должны составлять не более 10% тестов из-за высокой стоимости поддержки.

2.4 Средства контейнеризации и оркестрации

Контейнеризация обеспечивает изоляцию приложений и их зависимостей, гарантируя идентичное поведение в различных средах — от ноутбука разработчика до продуктивного сервера. Docker, выпущенный в 2013 году, стал стандартом де-факто для контейнеризации.

Dockerfile описывает процесс сборки образа контейнера как последовательность слоёв. Каждая инструкция создаёт новый слой, который кэшируется для ускорения последующих сборок. Лучшие практики включают: использование многоэтапной сборки (multi-stage builds), минимизацию размера образа, запуск от непривилегированного пользователя.

Kubernetes — система оркестрации контейнеров, разработанная Google на основе внутренней системы Borg и переданная в Cloud Native Computing Foundation в 2015 году. Kubernetes управляет развёртыванием, масштабированием и обеспечением отказоустойчивости контейнеризованных приложений. Ключевые абстракции: Pod, Deployment, Service, ConfigMap, Secret.

3 Построение CI/CD пайплайна на практике

3.1 Пайплайн для веб-приложения

Типичный пайплайн для веб-приложения на базе GitLab CI включает следующие этапы (stages):

1) Build — сборка приложения: установка зависимостей, компиляция TypeScript, сборка фронтенда (webpack/Vite). Результат — артефакт сборки.

2) Lint — статический анализ кода (ESLint, Prettier, SonarQube) для проверки стиля кодирования и выявления потенциальных ошибок.

3) Test — запуск модульных и интеграционных тестов с генерацией отчёта о покрытии кода (code coverage). Минимальный порог покрытия обычно 80%.

4) Security Scan — проверка зависимостей на уязвимости (npm audit, Snyk, OWASP Dependency-Check), статический анализ безопасности (SAST).

5) Docker Build — сборка Docker-образа и публикация в реестр контейнеров (Docker Hub, GitLab Container Registry, Amazon ECR).

6) Deploy Staging — автоматическое развёртывание в тестовую среду для ручного тестирования, UAT и приёмки.

7) Deploy Production — развёртывание в продуктивную среду с ручным подтверждением (manual approval) или автоматически.

3.2 Пайплайн для микросервисной архитектуры

Микросервисная архитектура предъявляет особые требования к CI/CD. Каждый микросервис должен иметь независимый пайплайн, при этом необходимо обеспечить согласованность версий при развёртывании. Ключевые особенности:

1) Monorepo vs Polyrepo — выбор между единым репозиторием для всех сервисов (monorepo) или отдельными репозиториями (polyrepo). Monorepo упрощает атомарные изменения и рефакторинг, но требует специальных инструментов (Nx, Turborepo, Bazel) для определения затронутых сервисов.

2) Контрактное тестирование — инструменты типа Pact позволяют проверять совместимость API между сервисами без необходимости поднимать все сервисы одновременно. Consumer-driven contracts обеспечивают уверенность в совместимости.

3) GitOps — современный подход, при котором желаемое состояние инфраструктуры описывается в Git-репозитории, а специальный оператор (ArgoCD, Flux) обеспечивает синхронизацию реального состояния кластера с декларативным описанием. Преимущества: аудит изменений, откат через git revert, единый источник истины.

4 Стратегии развёртывания и обеспечение безопасности

4.1 Стратегии развёртывания

Выбор стратегии развёртывания определяет, как новая версия приложения заменит текущую. Основные стратегии:

1) Rolling Update (постепенное обновление) — экземпляры приложения обновляются последовательно. В любой момент часть экземпляров работает на старой версии, часть — на новой. Это стратегия по умолчанию в Kubernetes.

2) Blue-Green Deployment — поддерживаются два идентичных окружения (blue и green). Новая версия развёртывается в неактивное окружение, после тестирования трафик переключается. Преимущество — мгновенный откат путём переключения обратно.

3) Canary Deployment (канареечное развёртывание) — новая версия развёртывается на небольшой процент трафика (1-5%). При отсутствии

проблем процент постепенно увеличивается до 100%. Позволяет выявить проблемы до полного развёртывания.

4) Feature Flags (флаги функций) — код новой функциональности развёртывается, но скрыт за флагом. Функция включается для определённых пользователей или процента трафика без повторного развёртывания. Инструменты: LaunchDarkly, Unleash, Flagsmith.

4.2 Безопасность в CI/CD пайплайне

Безопасность должна быть встроена в каждый этап пайплайна (подход «Shift Left Security»). Ключевые практики:

1) SAST (Static Application Security Testing) — анализ исходного кода на уязвимости без выполнения. Инструменты: SonarQube, Checkmarx, Fortify.

2) DAST (Dynamic Application Security Testing) — тестирование работающего приложения на уязвимости. Инструменты: OWASP ZAP, Burp Suite.

3) SCA (Software Composition Analysis) — проверка зависимостей на известные уязвимости (CVE). Инструменты: Snyk, WhiteSource, Dependabot.

4) Secret Management — хранение секретов (API-ключей, паролей) в специализированных хранилищах (HashiCorp Vault, AWS Secrets Manager), а не в коде или переменных окружения.

5) Container Security — сканирование Docker-образов на уязвимости (Trivy, Clair, Anchore), использование минимальных базовых образов (Alpine, Distroless).

4.3 Мониторинг и наблюдаемость

Наблюдаемость (Observability) — способность понять внутреннее состояние системы на основе её внешних выходов. Три столпа наблюдаемости:

1) Метрики (Metrics) — числовые показатели состояния системы (CPU, память, latency, error rate). Prometheus — стандарт де-факто для сбора метрик в Kubernetes. Grafana используется для визуализации.

2) Логи (Logs) — записи о событиях в системе. ELK Stack (Elasticsearch, Logstash, Kibana) или Loki + Grafana для централизованного сбора и анализа логов.

3) Трассировки (Traces) — отслеживание запроса через все сервисы в распределённой системе. Jaeger, Zipkin, OpenTelemetry для распределённой трассировки.

ЗАКЛЮЧЕНИЕ

В ходе написания реферата были рассмотрены ключевые аспекты построения CI/CD-пайплайнов для автоматизации тестирования и развёртывания информационных систем. Показано, что внедрение практик непрерывной интеграции и непрерывной доставки позволяет существенно повысить качество программного обеспечения, сократить время выпуска новых версий и снизить риски, связанные с развёртыванием.

Были детально проанализированы основные компоненты CI/CD-пайплайна: системы контроля версий с различными стратегиями ветвления, серверы непрерывной интеграции (Jenkins, GitLab CI/CD, GitHub Actions), инструменты автоматического тестирования в соответствии с концепцией пирамиды тестирования, а также средства контейнеризации и оркестрации (Docker, Kubernetes).

Особое внимание уделено стратегиям развёртывания (Rolling Update, Blue-Green, Canary), позволяющим минимизировать простои и риски при обновлении приложений. Рассмотрены вопросы обеспечения безопасности в пайплайне с применением подхода «Shift Left Security» и инструментов SAST, DAST, SCA.

Проанализированы современные подходы к наблюдаемости систем на основе трёх столпов: метрик, логов и трассировок. Показано, что комплексный мониторинг является необходимым условием для успешного применения практик непрерывного развёртывания.

Таким образом, CI/CD — это не просто набор инструментов, а целая культура разработки, направленная на повышение эффективности и качества создания программного обеспечения. Грамотное применение рассмотренных практик позволяет организациям быстрее реагировать на изменения рынка, обеспечивая при этом высокое качество и надёжность информационных систем.

СПИСОК ЛИТЕРАТУРЫ

1. Fowler, M. Continuous Integration [Электронный ресурс] / M. Fowler. — 2006. — URL: https://martinfowler.com/articles/continuousIntegration.html (дата обращения: 15.12.2025).
2. GitLab. What is CI/CD? [Электронный ресурс]. — URL: https://about.gitlab.com/topics/ci-cd/ (дата обращения: 15.12.2025).
3. GitHub. GitHub Actions Documentation [Электронный ресурс]. — URL: https://docs.github.com/en/actions (дата обращения: 15.12.2025).
4. Jenkins. Jenkins User Documentation [Электронный ресурс]. — URL: https://www.jenkins.io/doc/ (дата обращения: 15.12.2025).
5. Docker Inc. Docker Documentation [Электронный ресурс]. — URL: https://docs.docker.com/ (дата обращения: 15.12.2025).
6. Kubernetes. Kubernetes Documentation [Электронный ресурс]. — URL: https://kubernetes.io/docs/ (дата обращения: 15.12.2025).
7. Humble, J. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. — Addison-Wesley Professional, 2010. — 512 p. ISBN 978-0321601919.
8. Kim, G. The DevOps Handbook / G. Kim, J. Humble, P. Debois, J. Willis. — IT Revolution Press, 2016. — 480 p. ISBN 978-1942788003.
9. DORA. Accelerate State of DevOps Report [Электронный ресурс]. — URL: https://dora.dev/ (дата обращения: 15.12.2025).
10. ArgoCD. Argo CD — Declarative GitOps CD for Kubernetes [Электронный ресурс]. — URL: https://argo-cd.readthedocs.io/ (дата обращения: 15.12.2025).
11. HashiCorp. Terraform Documentation [Электронный ресурс]. — URL: https://developer.hashicorp.com/terraform/docs (дата обращения: 15.12.2025).
12. Prometheus. Prometheus Documentation [Электронный ресурс]. — URL: https://prometheus.io/docs/ (дата обращения: 15.12.2025).

Ссылка на Github: **https://github.com/AIS-436/Dumbov_Danil_Sergeevich_25**