ПОДХОДЫ К РАЗРАБОТКЕ МНОГОКОМПОНЕНТНОГО ПРОГРАММНОГО СРЕДСТВА НА ПРИМЕРЕ СЕРВИСА ТАКСИ

Сапраньков Ф.А.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Мелких Е.Г. – ст. преподаватель

На сегодняшний день разработка программных средств с использованием клиент-серверной архитектуры, таких как сервис такси, сталкивается с проблемой обеспечения быстрого и надёжного выпуска обновлений в условиях наличия сложных взаимосвязей между программными компонентами разрабатываемого программного средства такими как микросервисы и веб-клиенты. В данной статье рассматривается решение на основе внедрения CI/CD-процессов с использованием GitLab CI/CD и Kubernetes, позволяющее автоматизировать сборку, тестирование и развёртывание компонентов системы.

Современные клиент-серверные приложения сталкиваются с рядом сложностей, связанных с их архитектурой и процессами разработки. Как правило, такие системы включают серверную часть (например, Java с использованием фреймворка Spring) и клиентскую часть (например, веб-интерфейс на React), что создаёт высокую нагрузку на координацию команд, тестирование и развёртывание. Традиционные подходы, основанные на ручных операциях, часто приводят к ошибкам, увеличению времени выпуска релизов и конфликтам между компонентами. Например, обновление API сервера без проверки совместимости с клиентской частью может нарушить работу приложения, а ручная сборка Docker-контейнеров может оказаться длительной в случае большого числа микросервисов, используемых в приложении.

Цель данной статьи – рассмотреть способ адаптации программного средства сервиса такси с клиент-серверной архитектурой (на основе микросервисов Java с использованием Spring и веб-клиента на React) к production-среде через внедрение CI/CD-процессов.

В данной статье рассматриваются 4 этапа по разработке многокомпонентной системы сервиса такси: CI/CD-процессы, управление окружением, использование мониторинга, применение канареечный релизов.

CI/CD (Continuous Integration/Continuous Deployment) представляет собой методологию, направленную на автоматизацию ключевых этапов жизненного цикла программного обеспечения, включая интеграцию кода, его тестирование и доставку в production. Данный подход минимизирует риски, связанные с ручными операциями, и обеспечивает высокую скорость выпуска обновлений, что особенно критично для распределённых систем, таких как клиент-серверные приложения сервисов такси. Графическое представление основных этапов CI/CD отражено на рисунке 1.

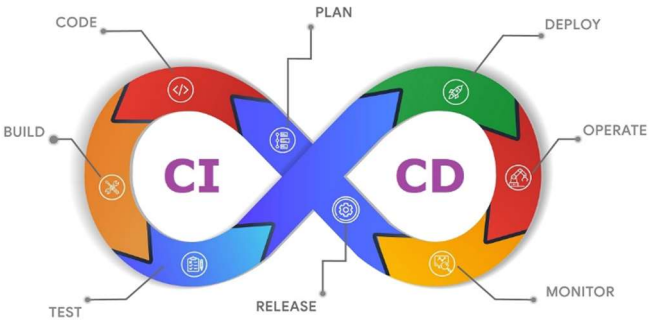

Рисунок 1 – Основные этапы CI/CD-процессов

Далее приводятся три основных вида окружений, используемых в CI/CD-процессах. Окружение представляет собой совокупность конфигураций, используемых в инструментах и платформах для разработки программного обеспечения. Dev-окружение (Development Environment) предназначено для разработки и первичного тестирования кода. Здесь программисты пишут и отлаживают новые функции, интегрируют модули и проверяют их в изолированной среде, недоступной внешним пользователям. В dev-окружении часто используются упрощённые конфигурации, а версии кода могут быть нестабильными, так как они находятся в активной фазе разработки. Stage-окружение (Staging Environment) имитирует production-среду и используется для финального тестирования перед выпуском. Здесь проверяется совместимость компонентов, нагрузочная устойчивость и корректность работы системы в условиях, приближенных к реальным. В stage-окружении разворачиваются стабильные версии кода, прошедшие CI/CD-pipeline, и проводится тестирование с участием QA-инженеров. Prod-окружение (Production Environment) – это реальная среда, в которой приложение

доступно конечным пользователям. Здесь отключены отладочные инструменты, включено логирование ошибок, и система настроена на максимальную производительность и безопасность. Изменения в prod-окружение вносятся только после полного тестирования в dev и stage, а деплой выполняется через автоматизированные CI/CD-процессы.

Continuous Integration (CI) фокусируется на непрерывной интеграции изменений в общий репозиторий системы контроля версий. Разработчики многократно вносят правки в код, который автоматически собирается и проверяется на соответствие стандартам качества. На этом этапе выполняется планирование по реализации функций, написание программного кода, далее осуществляется автоматическая сборка артефактов с использованием инструментов, таких как Maven для Java или npm для React. После сборки проводится тестирование, включая модульные тесты, интеграционные проверки и статический анализ кода, что позволяет выявлять ошибки на ранних стадиях. Каждая сборка ассоциируется с конкретным коммитом, что обеспечивает отслеживаемость изменений и возможность отката к стабильным версиям.

Continuous Deployment (CD) обеспечивает автоматическую доставку проверенного кода в production-окружение, исключая ручные этапы развертывания. Ключевым элементом здесь является контейнеризация: микросервисы Spring Boot и React-клиент упаковываются в Docker-контейнеры, что гарантирует идентичность окружений (dev, stage, prod) и устраняет проблемы с зависимостями. Для управления развёртыванием контейнеров используется Kubernetes, который автоматически масштабирует сервисы, балансирует нагрузку и обеспечивает откат ошибочных версий. Важным аспектом CD являются канареечные релизы, при которых обновления постепенно внедряются для части пользователей, что снижает риски критических сбоев.

Для использования CI/CD-процессов при разработке сервиса такси предлагается использовать платформу GitLab [1] как основу для pipeline. Pipeline – ядро CI/CD-процессов, представляет собой последовательность скриптов, управляющих сборкой, тестированием и развертыванием. Для разрабатываемого программного средства он включает следующие этапы: серверная часть собирается с помощью фреймворка Maven с последующей упаковкой в Docker-образы, которые содержат зависимости, такие как JRE и библиотеки Spring. Тестирование API и проверка совместимости с клиентской частью также входят в pipeline. Клиентская часть требует установки npm-пакетов, сборки оптимизированной версии приложения, проверки типов и E2E-тестирования [2]. Для управления контейнерами подходит Kubernetes, автоматизирующий развёртывание, масштабирование и балансировку нагрузки. Обратная совместимость обеспечивается проверкой изменений в API сервера через Swagger или OpenAPI перед развертыванием, благодаря этому можно избежать конфликтов в запросах между клиентом и севером.

Специфика клиент-серверных приложений связана с неоднородностью компонентов. Серверные микросервисы зависят от Java-библиотек и требуют настройки Spring Cloud Config для управления конфигурациями, при этом клиентская часть интегрируется с Node.js и требует оптимизации статических файлов для CDN [3]. Управление окружениями для сервиса такси предлагается реализовать через разделение переменных для dev, stage и prod с использованием инструментов, таких как Vault или Kubernetes Secrets.

Важным этапом в процессах поставки и интеграции является мониторинг многокомпонентной системы. Он позволяет отслеживать метрики и автоматически откатывать неудачные обновления. Такой подход снижает нагрузку на команды, ускоряет цикл разработки и повышает надёжность сервиса. В качестве инструмента мониторинга для сервиса такси можно использовать Prometheus.

Для минимизации рисков при обновлениях сервиса такси предлагается применение канареечных релизов. Данные релизы осуществляются через развертывание двух версий программного обеспечения: более старой, но стабильной, которая доступна большей части пользователей и новой, с применением последних изменений, но имеющей потенциальные проблемы и доступной лишь небольшой группе пользователей. Для каждой из версий контролируется трафик и отслеживается состояние. В случае, когда обновленная версия работает без возникновения критических ошибок и с приемлемым уровнем производительности, осуществляется постепенный перевод всех пользователей на новую версию программного обеспечения.

CI/CD-процессы являются необходимым элементом современной разработки, особенно для распределённых систем. Предложенные в статье методы и инструменты автоматизации сборки, тестирования и развёртывания сервиса такси позволяют адаптировать клиент-серверное приложение к production-среде, минимизируя риски и ускоряя выход обновлений. Данные решения для сервиса такси могут быть по шаблону применены на различные проекты с клиент-серверной архитектурой.

Список использованных источников:

1. Gitlab CI: Features Overview, Tutorials and Best Practices [Электронный ресурс]. – Электронные данные. – Режим доступа: https://codefresh.io/learn/gitlab-ci/ Дата доступа: 01.04.2025.

2. E2E Testing [Электронный ресурс]. – Электронные данные. – Режим доступа: https://microsoft.github.io/code-with-engineering-playbook/automated-testing/e2e-testing/ Дата доступа: 03.04.2025.

3. Gilbert Held. A Practical Guide to Content Delivery Networks (2nd edition) / Gilbert Held – CRC Press, 2018. – 304 с.