

Методы и средства сборки и развертывания ПО, 25

ФИО преподавателя: Смирнов Михаил Вячеславович

e-mail: smirnovmgupi@gmail.com

Лекция 9

Тестирование в конвейерах непрерывного развертывания.

Место тестирования в конвейере CI/CD



Пирамида тестирования



Пирамида тестирования - это концепция, которая рекомендует структурировать автоматизированные тесты по трем основным уровням:

Основание: Модульные тесты (Unit). Проверяют изолированные функции и методы, не требуют внешних зависимостей, выполняются за миллисекунды и дают самую быструю обратную связь в CI/CD.

Середина: Интеграционные тесты (Integration). Проверяют взаимодействие между модулями, базами данных, API. Требуют поднятия тестового окружения, выполняются за секунды, минуты.

Вершина: Сквозные тесты (End-to-End, E2E). Имитируют поведение реального пользователя в полном стеке приложения, выполняются минуты, часы и сложны в поддержке.

Модульное тестирование (Unit testing)



Модульное тестирование - это практика проверки наименьших неделимых частей программы (юнитов), обычно функций, методов или классов, в полной изоляции от внешнего мира (баз данных, файловой системы, сети).

Цель в CI/CD: Обеспечить мгновенную обратную связь разработчику о корректности логики его кода.

Ключевые характеристики:

Скорость: Выполняются за миллисекунды.

Изоляция: Не зависят от внешних систем.

Детерминированность: Результат (прошел/не прошел) всегда однозначен при одних и тех же входных данных.

Пример: Написать тест для функции `calculateDiscount(price, percent)`, который проверит, что при цене 1000 и проценте 10 возвращается значение 900.

Интеграционное тестирование (Integration Testing)



Интеграционное тестирование проверяет, как несколько модулей или сервисов взаимодействуют друг с другом. Оно отвечает на вопрос: «Правильно ли мои компоненты обмениваются данными и работают вместе?».

Цель в CI/CD: Обнаружить дефекты на стыках компонентов: несовместимые интерфейсы, неправильные форматы данных, ошибки в работе с реальной БД, проблемы с сетью.

Ключевые характеристики:

Зависимости: Требуют развертывания реальных или тестовых версий внешних сервисов (БД, кэш, брокеры сообщений).

Скорость: Выполняются медленнее unit-тестов (секунды, минуты).

Сложность отладки: При падении такого теста не всегда сразу ясно, в каком именно компоненте ошибка.

Пример: Протестировать сервис регистрации пользователя: проверить, что при отправке HTTP-запроса на `/api/register` данные корректно валидируются, хэшируются и сохраняются в тестовой базе данных, а затем возвращается корректный JSON-ответ.

Сквозное тестирование (End-to-End, E2E)

Сквозное тестирование — это имитация поведения реального пользователя в работающем приложении, которое развернуто в среде, максимально приближенной к продакшену. Оно проверяет весь продукт целиком, от фронтенда до бэкенда.

Цель в CI/CD: Убедиться, что ключевые бизнес-сценарии работают от начала до конца, и что все компоненты системы, работая вместе, дают ожидаемый результат.

Ключевые характеристики:

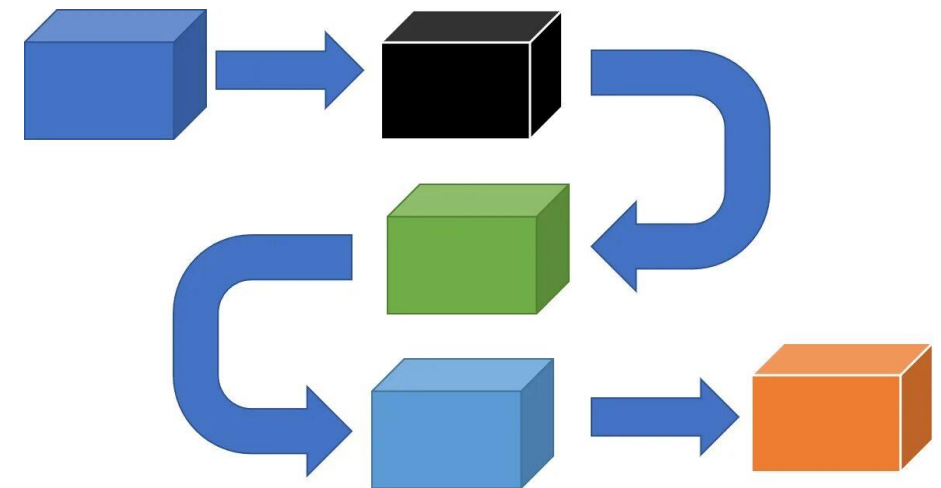
Реализм: Максимально близко к действиям живого пользователя.

Медлительность: Самые медленные тесты.

Хрупкость: Часто ломаются при малейших изменениях в UI.

Сложность поддержки: Требуют значительных усилий для написания и обновления.

Пример: Автоматизированный сценарий: браузер открывает сайт -> пользователь вводит логин/пароль -> переходит в каталог -> добавляет товар в корзину -> переходит к оформлению заказа -> вводит данные карты -> подтверждает заказ -> видит сообщение «Заказ успешно создан».



End to End Testing

Тестирование производительности (Performance Testing)

Тестирование производительности не связано с функциональностью. Тесты проверяют насколько хорошо система работает под нагрузкой (стабильность, скорость, масштабируемость и эффективность использования ресурсов).

Основные типы в контексте CI/CD:

Нагрузочное (Load Testing): Проверка поведения под ожидаемой пиковой нагрузкой.

Стресс-тестирование (Stress Testing): Постепенное увеличение нагрузки за пределы нормы, чтобы найти точку разрыва и проверить, как система восстанавливается после сбоя.

Тестирование на стабильность/надежность (Soak Testing): Длительный прогон под средней нагрузкой для поиска утечек памяти или постепенной деградации производительности.

Цель в CI/CD: Обнаружить регрессию производительности на ранних этапах. Если после нового коммита время отклика API выросло на 50%, конвейер должен остановиться.



Тестирование безопасности (Security Testing / DevSecOps)

Тестирование безопасности в CI/CD — это практика «сдвига безопасности влево» (shift-left), то есть интеграция проверок на уязвимости на самых ранних этапах жизненного цикла разработки.

Основные практики, встраиваемые в конвейер:

Статический анализ кода (SAST): Сканирование исходного кода на наличие шаблонов, связанных с уязвимостями (SQL-инъекции, XSS, небезопасная работа с памятью). Запускается на этапе сборки.

Динамический анализ (DAST): Сканирование работающего продукта (например, на staging-среде) для поиска уязвимостей, которые видны только во время работы.

Анализ зависимостей (SCA): Сканирование сторонних библиотек (npm, Maven, PyPI) на наличие известных уязвимостей из публичных баз данных (CVE).

Цель в CI/CD: Не допустить попадания известных уязвимостей в релиз, сделав безопасность автоматизированной и непрерывной.



Дымовое тестирование (Smoke Testing)



Дымовое тестирование — это минимальный набор тестов, который проверяет базовую работоспособность приложения после сборки или развертывания на новом окружении. Его название происходит от занятого принципа электроники: «Если при включении устройства пошел дым, дальнейшее тестирование бессмысленно».

Цель в CI/CD: Быстро принять решение: «Готова ли очередная сборка к дальнейшему, более глубокому тестированию?».

Ключевые характеристики:

Скорость: Выполняется быстро (несколько минут).

Ограниченность: покрывает "пути счастья" (happy paths) ключевых функций.

Применение: запускается сразу после развертывания в тестовой среде.

Примеры сценариев: Продукт запускается? Пользователь может залогиниться? Главная страница загружается? API возвращает ответ на базовый запрос?

Программное обеспечение: Инструменты для Unit/Integration-тестов

JUnit 5

Java: JUnit 5 (стандарт) и **TestNG** (более мощный для сложных сценариев).

Python: pytest - инструмент для тестирования с простым синтаксисом и мощными возможностями. **unittest** входит в стандартную библиотеку Python.

JavaScript/Node.js: Jest – наиболее популярный лидер, «из коробки» предлагает все необходимое: runner, assertions, mock'и. **Mocha** + **Chai** - модульные компоненты.

Как интегрируются в CI/CD: CI-сервер (например, Jenkins) просто выполняет команду для запуска тестов (mvn test, pytest, npm test). Код возврата 0 означает успех, любой другой код — неудачу, что и приводит к "покраснению" конвейера.



Программное обеспечение: Инструменты для E2E-тестирования



Selenium WebDriver: API, которое позволяет управлять браузером программно на разных языках (Java, Python, C#, JavaScript). Требуется отдельного драйвера для каждого браузера. Мощный, но может быть сложным в настройке и медленным.



Playwright

Cypress: Современный инструмент, который работает непосредственно внутри браузера вместе с продуктом. Это дает ему преимущество в скорости, простоте отладки (встроенный DevTools) и стабильности. Работает только на Chromium-движках.

Playwright: Поддерживает Chromium, Firefox и WebKit (Safari). Очень быстрый, надежный и обладает мощным API для автоматизации сложных сценариев (перехват сетевых запросов, эмуляция мобильных устройств).

Как интегрируется в CI/CD: Для запуска этих тестов нужен браузер. Обычно это решается с помощью Docker-образов, которые содержат браузеры «без головы» (headless).



Программное обеспечение: Инструменты тестирования производительности



Apache JMeter: Имеет графический интерфейс для создания тестовых планов в XML. Изначально создан для тестирования веб-приложений, но теперь поддерживает множество протоколов.

k6: Тесты пишутся на JavaScript, что делает код понятным и удобным для версионирования. Создан специально для CI/CD: легковесный, запускается из командной строки, легко интегрируется с системами мониторинга.

Gatling: Мощный инструмент на Scala. Также удобный для CI/CD конвейера. Хорош своей эффективностью и низким потреблением ресурсов при генерации высокой нагрузки.

CI/CD контейнер с тестированием (пример)

1.Начало: Разработчик пушит код в ветку `main` в GitLab.

2.Стадия «Build»:

1. **Сборка:** `docker build` создает образ продукта.
2. **Статический анализ:** Запускается `sonar-scanner` (утилита SonarQube) для проверки кода на уязвимости. Если нужно – «дымим».

3.Стадия «Test»:

1. **Модульные тесты:** Запускается `npm test` (Jest).
2. **Интеграционные тесты:** Запускается `npm run integration-tests` (pytest для API-тестов). Поднимаем тестовую базу данных в Docker.

5. Стадия «Deploy to Staging»:

1. Проверенный образ пушится в Docker Registry.
2. Образ разворачивается на staging-кластер Kubernetes с помощью `kubectl`.

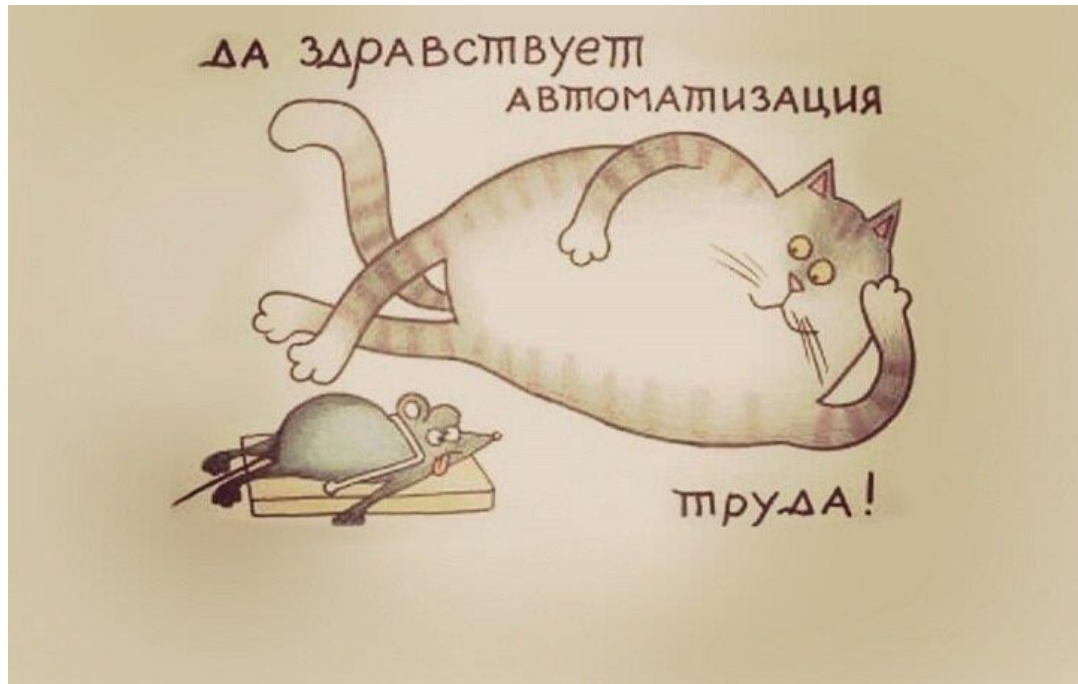
6. Стадия «E2E & Security»:

1. **Сквозные тесты:** Запускается `npm run playwright test` в URL staging-среды.
2. **Сканирование безопасности:** Запускается OWASP ZAP в режиме DAST в той же среде.

7. Стадия «Deploy to Production» (Автоматическое или ручное):

1. После прохождения всех тестов и мануального подтверждения, образ разворачивается в релизной среде Kubernetes. online.mirea.ru

Автоматизация тестирования в CI/CD контейнере



Скорость и Частота: Автоматизация позволяет проводить тысячи тестов за минуты. Это дает командам уверенность при выпуске небольших, но частых изменений (десятки раз в день) и позволяет отказываться от «больших взрывов».

Качество и Надежность: Дефекты обнаруживаются намного быстрее, пока разработчик еще помнит, что он менял.

Экономия ресурсов: Время на написание тестов и автоматизацию многократно окупается за счет сокращения времени на ручное тестирование, отладку и «тушение пожаров» в продакшене.

«Живая документация»: Набор тестов служит точным, всегда актуальным описанием того, как система должна себя вести.

Чтение на дом

- Д. Хамбл, Д. Фарли, Непрерывное развертывание ПО, стр. 103-118.

Спасибо за внимание!