Лекция 19: Большая картина. Полный жизненный цикл приложения: от git push до наблюдаемого сервиса

Введение: Собираем пазл

Здравствуйте! Сегодня у нас финальная лекция. На протяжении трех семестров мы с вами, как инженеры, изучали и собирали отдельные узлы сложнейшего механизма — современного backend-приложения. Мы научились:

- **Проектировать** его (Agile/Scrum, Jira).
- **Писать код** (Node.js, Express).
- **Работать с данными** (SQL, Prisma, 1C).
- **Обеспечивать качество** (Тесты, Документация, Безопасность).
- **"Упаковывать"** его (Docker).
- **Автоматизировать сборку** (CI/CD).
- **Развертывать и эксплуатировать** (Kubernetes, Облака).
- **Наблюдать** за ним (Логи, Метрики).

Сегодня наша цель — собрать из этих разрозненных "кубиков" единую, целостную картину. Мы проследим за "жизнью" одной маленькой фичи — от момента, когда разработчик пишет git push, до того, как миллионы пользователей начинают ей пользоваться, а мы видим ее "пульс" на наших дашбордах.

Эта лекция — взгляд с высоты птичьего полета на весь современный цикл разработки и эксплуатации ПО (DevOps).

1. Сценарий: Наша задача

Представим, что мы работаем в компании, которая разрабатывает наш To-Do API.

- **Команда:** Работает по Scrum, двухнедельные спринты.
- **Инфраструктура:** Приложение развернуто в управляемом Kubernetes-кластере в Yandex.Cloud.
- **CI/CD:** Настроен на GitHub Actions.
- **Наблюдаемость:** Используется стек Prometheus/Grafana для метрик и ELK для логов.

Задача на спринт: *«Как пользователь, я хочу видеть, когда была создана задача, чтобы лучше планировать свои дела»*.
Технически это означает, что нам нужно:

1. Добавить поле createdAt в ответ API для эндпоинта GET /tasks.
2. Добавить соответствующий тест.

2. Этап 1: Разработка (Локальная машина)

- **Действие:** Разработчик Анна берет задачу PROJ-123 из Jira.

- **Git:** Она не работает в main! Она создает новую ветку: git checkout -b feature/PROJ-123-add-created-at.
- **Код:** Анна вносит изменения в код. В Prisma select для GET /tasks она добавляет поле createdAt: true.
- **Тесты:** Она запускает тесты **локально** (npm test) и пишет новый тест, который проверяет наличие и формат поля createdAt в ответе. Убеждается, что все тесты "зеленые".
- **Docker Compose:** Она запускает docker-compose up и проверяет работу нового эндпоинта вручную через Postman.
- **Git:** Анна делает коммит с осмысленным сообщением: git commit -m "feat(tasks): PROJ-123 Add createdAt field to task response" и отправляет ветку в репозиторий: git push origin feature/PROJ-123....

3. Этап 2: Код-ревью и CI (GitHub)

- **Pull Request:** Анна создает Pull Request (PR) из своей ветки в ветку develop.
- **CI-пайплайн (GitHub Actions): Автоматически** запускается наш CI-пайплайн для этого PR. Он:
    1. Скачивает код на "чистый" раннер.
    2. Устанавливает зависимости (npm ci).
    3. Прогоняет все тесты (npm test).
    4. *Не собирает и не пушит Docker-образ* (т.к. это не push в main).
- **Код-ревью:** Другой разработчик, Петр, видит "зеленую галочку" от CI (тесты прошли), просматривает код, оставляет пару комментариев. Анна вносит правки, пушит их, CI снова проходит.
- **Merge:** Петр одобряет PR. Анна нажимает кнопку "Merge".

4. Этап 3: Сборка и Доставка на Staging (CI/CD)

- **Триггер:** Слияние кода в ветку develop является триггером для **второго пайплайна** (или второй части нашего основного).
- **CI-часть:**
    1. Снова запускаются тесты (на всякий случай, чтобы проверить, что после слияния ничего не сломалось).
    2. Собирается **Docker-образ** приложения.
    3. Образу присваивается уникальный тег, например, ghcr.io/my-company/todo-api:develop-a1b2c3d.
    4. Образ **загружается (push)** в наш приватный реестр контейнеров.
- **CD-часть (Continuous Delivery):**

1. Пайплайн **автоматически** запускает задание для развертывания на **тестовый сервер (Staging)**.
2. Он подключается к нашему Staging Kubernetes-кластеру и обновляет Deployment, указывая новый тег образа.

5. Этап 4: Тестирование и приемка (Staging)

- **Kubernetes в действии:** Kubernetes на Staging-кластере выполняет **Rolling Update**, плавно заменяя старые поды на новые.
- **QA:** Тестировщик Мария получает уведомление, что новая версия выкачена на Staging.
- **Тестирование:** Она открывает Postman (или автотесты), проверяет, что в ответе GET /tasks действительно появилось поле createdAt и что старый функционал не сломался.
- **Приемка:** QA и Product Owner подтверждают, что фича реализована корректно и готова к выпуску.

6. Этап 5: Релиз в Production (CD)

- **Процесс:** В конце спринта (или по готовности) тимлид создает Pull Request из ветки develop в ветку main.
- **CI:** Снова прогоняются все тесты.
- **Merge:** После одобрения PR вливается в main.
- **Триггер для Production:** push в ветку main запускает **финальный пайплайн**.
    1. Снова собирается и тегируется Docker-образ (теперь уже с тегом main-f0e9d8c или v1.2.0).
    2. Образ пушится в реестр.
    3. **РУЧНОЕ ПОДТВЕРЖДЕНИЕ:** Пайплайн доходит до этапа "Deploy to Production" и **останавливается**, ожидая ручного одобрения от ответственного лица в интерфейсе GitHub Actions.
- **Deploy:** Тимлид нажимает кнопку "Approve". Пайплайн продолжает работу и обновляет Deployment уже на **боевом (Production)** Kubernetes-кластере.

7. Этап 6: Эксплуатация и Наблюдаемость (Production)

- **Kubernetes:** Начинает Rolling Update в продакшене. Трафик плавно перетекает на новую версию.
- **Мониторинг (Grafana):** DevOps-инженер и команда наблюдают за дашбордами:
    - Не выросла ли доля ошибок (Error Rate)?
    - Не увеличилось ли время ответа (Latency)?
    - Справляется ли система с нагрузкой (Traffic, Saturation)?
- **Логирование (Kibana):** Если на дашборде виден всплеск ошибок, инженер идет в Kibana, ищет логи ошибок за последние 5 минут, чтобы понять причину.

- **Алертинг (Alertmanager):** Если бы что-то пошло не так (например, доля 5xx ошибок превысила 5% на 1 минуту), Alertmanager уже отправил бы уведомление в Slack, и команда начала бы реагировать еще до того, как пользователи что-то заметят.
- **Откат (Rollback):** Если проблема серьезная, DevOps-инженер одной командой kubectl rollout undo или нажатием кнопки в CI/CD может мгновенно откатить приложение к предыдущей, стабильной версии.

Заключение: Вы — часть большого конвейера

Сегодня мы увидели, что современная разработка — это не хаотичный процесс, а хорошо отлаженный, автоматизированный конвейер, где каждая технология и каждый специалист играют свою важную роль.

Вы, как разработчики, находитесь в самом начале этого конвейера. Ваша задача — писать качественный, протестированный код и поставлять его в виде небольших, независимых изменений. А дальше в дело вступает магия DevOps-инструментов, которые мы изучили в этом семестре: CI/CD доставляет ваш код, Docker гарантирует его консистентность, Kubernetes обеспечивает его жизнь и стабильность, а системы наблюдаемости дают нам уверенность, что все работает как надо.

Этот курс дал вам карту современного IT-мира. Вы получили фундаментальные знания на каждом уровне: от написания одной строчки JavaScript до понимания сложных архитектурных паттернов. Ваш путь в профессиональную разработку только начинается, но теперь у вас есть прочный фундамент, на котором можно строить блестящую карьеру. Удачи