Проектирование архитектуры программных систем

Д6

Проект: Платформа для управления персональными финансами "FinTrack"

Выполнили:

Сидоров Артемий БПИ225

Шмараев Артём БПИ225

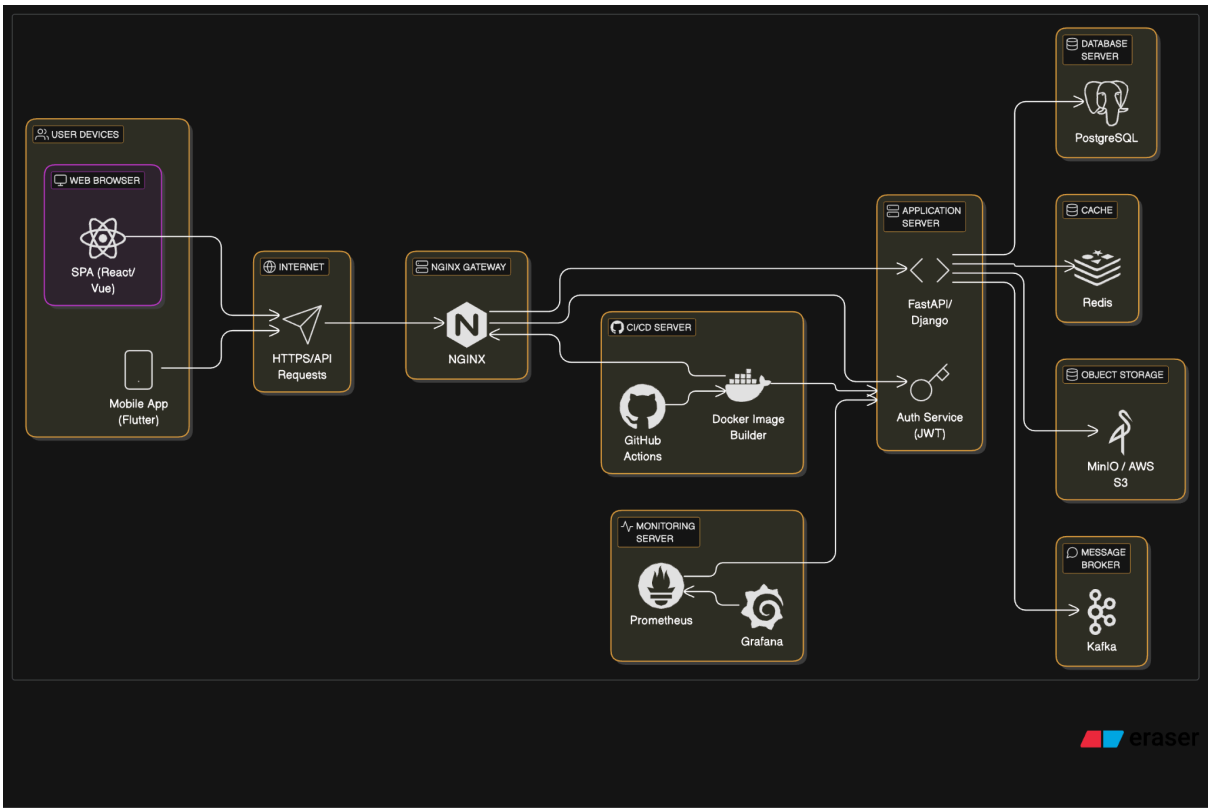
(БПИ228)

ПИ ФКН НИУ ВШЭ · 2024/2025 учебный год

Содержание

1. Идентификация внешних систем для "FinTrack"

1.1. UML Deployment Diagram (PlantUML)



Пояснения:

- User Device — конечное устройство пользователя (браузер или мобильное приложение).
- NGINX — входная точка, проксирует запросы к backend.
- Application Server — FastAPI/Django backend с auth-сервисом.
- Database, Redis, S3, Kafka — типичная структура серверной части.
- CI/CD Server — автоматическая сборка, доставка образов.
- Monitoring — система мониторинга (Prometheus + Grafana).

ЗАР: Доставка компонентов системы до среды исполнения

Компоненты, подлежащие доставке:

- frontend — SPA (React)
- backend — API-сервер на FastAPI
- auth-service — микросервис авторизации
- migrations — утилита для Alembic/SQLAlchemy
- worker — асинхронный обработчик задач (Celery/Kafka consumer)
- nginx — статичный конфиг для обратного прокси
- infra — файлы описания деплоя (docker-compose, Helm чарты, secrets)

Хранение артефактов:

- **GitHub Container Registry** (ghcr.io)
    - Контейнеры публикуются с тегами :latest, :staging, :prod, а также :commit-hash
    - Доступ к registry осуществляется через GITHUB_TOKEN или PAT (Personal Access Token)
- **GitHub Releases** используются для хранения:
    - Миграционных скриптов (если не входят в образ)
    - Статических билдов фронта (если не в контейнере)

CI/CD Конвейер (GitHub Actions):

Шаг 1. Secrets Scan GitLeaks

- Проверка по GitLeaks.
- Фейлит сборку, если найден незафиксированный ключ/токен.

Шаг 2. Static Code Analysis Black, Flake8, Mypy

- Применяются к backend + worker.

Шаг 3. Unit Tests Pytest, Coverage

- Обязателен pytest-cov с порогом покрытия (например, --cov-fail-under=85).

Шаг 4. Docker Build + Push

- Используется multi-stage build (отделение runtime от build-time слоёв)

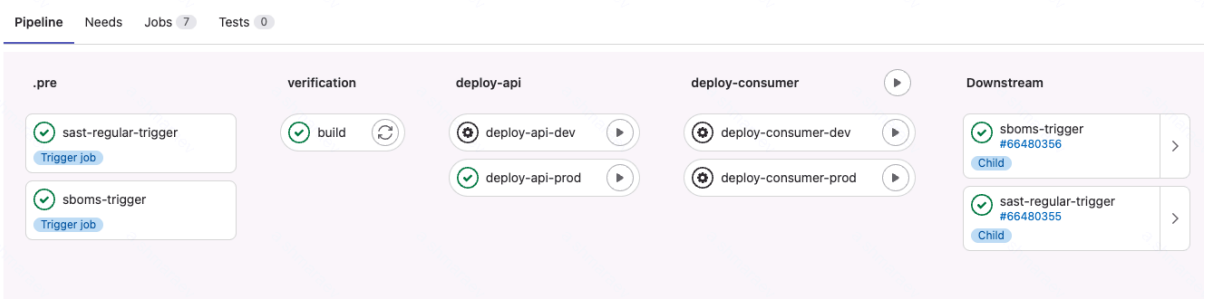
- docker-compose.yml использует образы по commit hash

Шаг 5. Автоматическое развертывание

- **Development**:
  - Используется docker-compose + ssh-deploy на тестовый сервер.
  - Конфигурация .env хранится отдельно и передаётся как секрет.

- **Production** (в случае Kubernetes):
  - Используется Helm GitOps-подход.
  - Helm-чарты хранятся в отдельном репозитории infra/helm-fintrack.
  - Обновление производится путём изменения версии образа в values.yaml.

Пример пайплайна:



ЗАР: Стратегия обновления FinTrack без остановки (Zero Downtime Deployment)

Цель

Обеспечить безопасное и непрерывное обновление компонентов системы без прерывания пользовательского трафика и снижения доступности. Включает обновление backend-сервисов, фронтенда, воркеров и БД.

Стратегия развертывания

1. Rolling Update (Kubernetes)

Применяется как основная стратегия:

- Используется объект Deployment с типом стратегии RollingUpdate.
- Гарантирует, что трафик не будет направлен на под, пока он не готов (через readiness probe).
- Обеспечивает откат: kubectl rollout undo.

2. Blue-Green Deployment (опционально)

Для высокорисковых обновлений:

- Создаются две параллельные среды (blue — текущая, green — новая).
- Новая версия разворачивается отдельно, получает нагрузочное тестирование.
- После прогрева осуществляется переключение трафика (через Ingress Controller или Service).
- Старая версия остаётся доступной на случай отката.

Health Checks: Liveness и Readiness Probes

Используются для контроля состояния подов:

- readinessProbe предотвращает подачу трафика на неготовый под.
- livenessProbe перезапускает под при зависании или утечках ресурсов.

Миграции базы данных

Инструмент: Alembic (если SQLAlchemy) или Django Migrations (если Django)

Принципы:

- Миграции разделяются на **совместимые** и **разрушающие**.
- Первым этапом создаются и применяются только **обратимые** миграции (например, добавление колонок).
- Изменения, разрушающие старую логику, откладываются до завершения переключения версии.

Шаги:

Генерация миграций:

Применение миграций до выката:
alembic upgrade head

1. Применение rolling update сервиса.
2. Удаление устаревших сущностей — в следующем релизе.

Возможные сбои и меры по предотвращению

Сценарий	Причина	Решение
Простой при обновлении	Отсутствие readinessProbe	Включить readinessProbe, установить maxUnavailable: 0
Разрушение текущей версии	Несовместимые миграции	Применять только обратимые миграции, использовать feature toggles
Перезапуск всех подов	Одновременное обновление всех реплик	Использовать rollingUpdate, maxSurge: 1
Ошибки при миграциях	Миграции не протестированы	Проверять миграции в staging, использовать dry-run
Потеря данных при деплое	Удаление или переименование колонок	Следовать стратегии "add, switch, remove"

Обработка обновлений воркеров и фоновых задач

- При наличии фоновых задач в Kafka, Redis или Celery — необходимо реализовывать backward-compatible логику.

- Для структурированных payload'ов желательно использовать versioned schema (например, через protobuf или JSON Schema с версией).
- Использовать флаги и контроллеры в коде для обработки разных версий событий (например, if event.version == 2:).

Дополнительные меры безопасности

- Все миграции проходят проверку на staging-среде.
- Все деплои фиксируются в git с отдельным changelog.
- Используется логгирование ключевых этапов деплоя и проверок через CI/CD pipeline.

ЗАР: Вычислительные ресурсы и инфраструктура для FinTrack

Цель

Определить требования к ресурсам продакшен-среды системы FinTrack на основе предполагаемой нагрузки, числа пользователей, архитектурных решений и профиля потребления ресурсов компонентами.

Архитектура решения

Система включает следующие основные компоненты:

- **Backend (FastAPI)**
- **PostgreSQL (основная БД)**
- **Redis (кэш и брокер фоновых задач)**
- **Kafka (очереди задач)**
- **Worker-процессы (асинхронные задачи: парсинг, уведомления)**
- **Frontend (SPA, отдаётся NGINX)**
- **Object Storage (S3-совместимый)**

Развёртывание осуществляется в **Kubernetes-кластере (k8s)** или **VPS-инфраструктуре**, в зависимости от масштаба.

Планируемая нагрузка

- **Одновременных пользователей**: до 500 в пике
- **Операций API в пике**: до 50 RPS
- **Фоновые задачи**: ~10 тыс. задач/день
- **Хранение файлов (чеков, скриншотов)**: ~100 МБ/день
- **Исторические данные**: до 5 млн записей транзакций

Рекомендованные ресурсы (на компонент)

1. Backend (FastAPI, Python)

- **CPU**: 2 vCPU (autoscaling до 4)
- **RAM**: 4 GB
- **Storage**: 5 GB (persistent volume не обязателен)
- **Обоснование**: CPU-bound при высокой RPS (JSON-сериализация, auth, валидация); поддержка нескольких Gunicorn workers; активное взаимодействие с Redis и PostgreSQL.

2. PostgreSQL

- **CPU**: 2 vCPU (желательно с поддержкой AVX2)
- **RAM**: 8 GB
- **SSD**: 50 GB (расширяется с ростом пользователей)
- **Обоснование**: Работа с большим количеством read/write транзакций, индексы, JOIN’ы. Использование pg_stat_statements и планов запросов требует памяти. SSD критичен для быстрого доступа к индексам и WAL.

3. Redis

- **CPU**: 1 vCPU
- **RAM**: 2 GB
- **Storage**: Ephemeral (RDB snapshot в S3)
- **Обоснование**: Используется как брокер (для фоновых задач) и кэш. Высокая скорость доступа, умеренная нагрузка, но необходимо резервирование snapshot'ов.

4. Kafka (в кластере или через облачный провайдер)

- **CPU**: 2 vCPU
- **RAM**: 4 GB
- **Storage**: 20 GB SSD (сохранение сообщений на 7 дней)
- **Обоснование**: Асинхронная доставка задач, устойчивость к пиковым всплескам нагрузки. Используется для очередей между микросервисами.

5. Worker (Python, Celery/FastStream)

- **CPU**: 1–2 vCPU (в зависимости от интенсивности задач)
- **RAM**: 2 GB
- **Количество реплик**: 2+ (autoscaling по очередям Kafka)
- **Обоснование**: Асинхронные операции: парсинг чеков, ML-задачи, рассылка уведомлений. Возможны нагрузки на CPU при парсинге/обработке изображений.

6. Frontend (NGINX + SPA)

- **CPU**: 0.5 vCPU
- **RAM**: 512 MB
- **Storage**: 500 MB (build + assets)

- **Обоснование**: Отдача статики, SPA-приложение загружается и кэшируется клиентом. Почти не требует ресурсов.

7. Object Storage (S3-совместимый: MinIO или Яндекс Object Storage)

- **Storage**: 100 GB (расширяемо)
- **Обоснование**: Хранение чеков, изображений, архивов отчётов. Доступ к файлам идёт через URL и не нагружает backend.

8. CI/CD Runner (self-hosted или GitHub Actions)

- **CPU**: 2 vCPU
- **RAM**: 4 GB
- **Storage**: 20 GB (build cache, Docker)
- **Обоснование**: Сборка Docker-образов, тесты, lint, деплой. Желательно использовать удалённый Docker Registry.

Конфигурация для продакшена (на старте)

Вариант 1: Kubernetes (K8s)
Развёртывание в облаке (например, Yandex Managed Kubernetes или AWS EKS):

Cluster Nodes: 3× (4 vCPU, 8 GB RAM, 100 GB SSD)
+ S3 Bucket (100 GB)
+ Yandex Managed PostgreSQL (2 vCPU, 8 GB RAM, SSD)
+ External Redis & Kafka (облачные сервисы)

Вариант 2: VPS (для MVP или пилота)

VPS-1:
- 4 vCPU, 8 GB RAM, 100 GB SSD (PostgreSQL + Redis + Backend + Workers)
VPS-2:
- 2 vCPU, 4 GB RAM, 50 GB SSD (CI/CD + Frontend + Kafka)
S3: Облачный бакет (Object Storage)

Запас ресурсов и масштабирование

- Все Stateful-компоненты (PostgreSQL, Redis) размещаются на StatefulSet (K8s) или отдельных VPS с резервным копированием.
- Horizontal Pod Autoscaler (HPA) применяется к backend и worker'ам.
- Storage расширяется автоматически (S3, PVC в Kubernetes).

- Метрики и алерты через Prometheus + Grafana.

Риски и меры

Риск	Мера
Нехватка RAM в PostgreSQL	Мониторинг через pg_stat_activity, tuning shared_buffers, work_mem
Просадка производительности Redis	Резервирование RDB-снапшотов, eviction policy
Заполнение диска S3	Жизненные политики хранения, мониторинг объёма
Падение worker'ов	Лимиты по CPU, retry-логика в Kafka