📦 Контейнеризация требования

Замечания и контекст

Контейнеризация должна обеспечивать воспроизводимое развертывание приложения на любой машине с Docker. Все компоненты проекта (кроме сторонних сервисов типа Redis/PostgreSQL, где можно использовать официальные образы) должны быть упакованы в контейнеры.

📋 МИНИМАЛЬНЫЕ ТРЕБОВАНИЯ (на 5 баллов)

1. Docker Images

Обязательные требования к сборке:

- Отдельный Dockerfile для каждого сервиса
- Правильное разделение на слои (неизменяемое в начале, часто меняемое в конце)
- Использование `.dockerignore` для исключения ненужных файлов
- ENV переменные для всех настроек (никаких хардкодов)
- Volumes для персистентных данных
- Правильная настройка портов (EXPOSE)

Оптимизация размера:

- Размер образа должен быть разумным (<1GB backend, <500MB frontend)
- Удаление временных файлов, кэша пакетных менеджеров после установки
- Не включать ненужные утилиты (компиляторы, debug-tools, IDE)
- Multi-stage build где применимо

Безопасность:

- Никакой sensitive информации в образах (пароли, токены, ключи)
- Запуск от non-root пользователя (USER directive)
- Версионирование образов (не использовать latest в production)

2. Docker Compose

Обязательные элементы:

- `docker-compose.yml` для локального развертывания
- Все сервисы запускаются командой `docker-compose up`

- Правильные зависимости между сервисами (depends_on)
- Изолированные сети для компонентов
- Volumes для БД и загруженных файлов
- Переменные окружения через `.env` файл
- `.env.example` с примерами значений

3. Документация

Обязательно задокументировать:

- README с инструкцией запуска
- Описание каждого образа и его назначения
- Список всех ENV переменных с описанием
- Схема/диаграмма архитектуры контейнеров
- Требования к ресурсам (минимальные RAM/CPU)

🚀 МАКСИМАЛЬНЫЕ ТРЕБОВАНИЯ (на 10 баллов)

Все минимальные требования ПЛЮС:

1. Продвинутая оптимизация

- **Собственный base image** с преднастроенными зависимостями
- **Условное кэширование** без пересборки при неизменном коде
- **Build pipeline** с возможностью очистки старых images
- **Distroless или Alpine** базовые образы где возможно
- **Layer caching** стратегия документирована

2. Production-ready features

- **Healthcheck** для каждого контейнера
- **Resource limits** (memory, CPU) для контейнеров
- **Graceful shutdown** с обработкой сигналов
- **Автоматический restart** при падении (restart policies)
- **Логирование** в структурированном формате (JSON)

3. Расширенная конфигурация

- Отдельные compose файлы для dev/test/prod окружений
- **Override файлы** для различных конфигураций
- **Secrets management** (Docker secrets или внешний vault)
- **Network policies** для дополнительной изоляции

4. Мониторинг и отладка

- Интеграция с системой мониторинга (Prometheus metrics)
- Централизованное логирование
- Debug режим с возможностью подключения debugger
- Performance метрики контейнеров

⚠️ ОГРАНИЧЕНИЯ - ЧТО НЕЛЬЗЯ ДЕЛАТЬ

Критические ошибки безопасности:

- ❌ Хранить секреты в Dockerfile, образах или коммитах
- ❌ Использовать `--privileged` флаг без крайней необходимости
- ❌ Оставлять дефолтные пароли в контейнерах
- ❌ Публиковать приватные образы в публичные registry

Ошибки в архитектуре:

- ❌ Запускать контейнеры от root без обоснования
- ❌ Использовать `latest` тег для production
- ❌ Хардкодить пути, URL и конфигурации
- ❌ Делать контейнеры stateful без volumes
- ❌ Использовать один контейнер для всего (анти-паттерн "fat container")

Ошибки оптимизации:

- ❌ Копировать `node_modules`, `venv`, `.git` в образ
- ❌ Игнорировать порядок слоев (часто меняемое в начале)
- ❌ Устанавливать пакеты в runtime вместо build time
- ❌ Не очищать кэш пакетных менеджеров после установки
- ❌ Использовать образы с известными CVE уязвимостями

Ошибки в реализации:

- ❌ Отсутствие `.dockerignore`
- ❌ Игнорирование exit codes и сигналов ОС
- ❌ Логирование в файлы вместо stdout/stderr
- ❌ Использование `COPY . .` без фильтрации
- ❌ Запуск нескольких процессов в одном контейнере без supervisor

---

📝 ФОРМАТ ПРЕЗЕНТАЦИИ РАБОТЫ

1. Docker Compose файл

- Полный `docker-compose.yml` с комментариями
- Диаграмма/схема pipeline сборки и взаимодействия

2. Описание образов

Для каждого образа указать:

- Назначение и функциональность
- Base image и его обоснование
- Размер финального образа
- Основные оптимизации

3. Описание контейнеров

Для каждого контейнера указать:

- Роль в системе
- Внешние и внутренние порты
- Volumes и их назначение
- Переменные окружения
- Healthcheck стратегия
- Resource limits (если установлены)

4. Метрики

- Время сборки образов
- Размеры образов до/после оптимизации
- Время холодного старта всей системы
- Потребление ресурсов в runtime