Отчёт DV по практикам P07 (контейнеризация) и P08 (CI/CD)

Контейнеризация и харднинг

Критерий	Что требовалось	Как выполнено	Где в репо
C1 Dockerfile (multi-stage, размер)	Multi-stage, убрать временные deps, оптимизация размера	Два стейджа: build (wheel + pytest) и runtime на `python:3.11.9-slim`, установка зависимостей через кэш pip и wheel-dir, в runtime копируются только артефакты. Пример: `docker build -t wishlist-api .` не тянет dev-зависимости в финальный слой	`Dockerfile`
C2 Безопасность контейнера	Non-root, HEALTHCHECK, тома/FS	Создаётся `app` user/group, `USER app`. `HEALTHCHECK` через python-запрос к `/health`. Каталог uploads получает chown и монтируется в volume	`Dockerfile`
C3 Compose/ локальный запуск	docker compose up поднимает сервис	`compose.yaml` описывает `api` с `env_file: .env.example`, healthcheck, volume `uploads-data`, restart policy `unless-stopped`, порт 8000. Пример: `make docker-run` → сервис отвечает на `http://localhost:8000/health`	`compose.yaml`
C4 Сканирование образа	Включить linters/scan	Hadolint конфиг для Dockerfile, `make docker-scan` вызывает `trivy image` по образу, `make lint` запускает hadolint	`.hadolint.yaml`, `Makefile`
C5 Контейнеризация своего приложения	Запуск своего сервиса через compose	Собран реальный Wishlist API: копируются `app/` и `src/`, uvicorn entrypoint, volume для вложений, монтирование uploads в контейнер. Пример: после `docker build` → `docker compose up` выдаёт доступный API	`Dockerfile`, `compose.yaml`, `Makefile`

Зачем это сделано

- Цель: воспроизводимый и безопасный образ Wishlist API для dev/stage/prod, минимальный слой зависимостей и контроль запуска (healthcheck, non-root).

- Альтернативы и +/-: база образа `distroless` или `alpine` (меньше размер, сложнее дебаг и SSL), healthcheck через `curl` (проще, но тянет лишний пакет), монтирование uploads как read-only с отдельным writer-сервисом (больше изоляция, но требует изменений в коде).
- Rollout plan: 1) локально `make lint` и `make docker-scan`, 2) `make docker-run` для smoke-теста, 3) пуш образа в реестр и подключение того же compose/helm в окружениях, 4) при необходимости усилить hardening (cap-drop, seccomp) отдельным патчем.

Минимальный стабильный CI

Критерий	Что требовалось	Как выполнено	Где в репо
C1 Сборка и тесты	Build + unit-тесты в CI	Workflow `CI` выполняет checkout, устанавливает зависимости, запускает `ruff`, `black --check`, `isort --check-only`, `pytest -q`. При ошибке любого шага job падает	`.github/workflows/ci.yml`
C2 Кэширование/ конкурренси	Cache зависимостей, concurrency	`actions/setup-python@v5` с `cache: pip`, `concurrency ${{ github.workflow }}-${{ github.ref }}`, `cancel-in-progress: true`, timeout 12 минут	`.github/workflows/ci.yml`
C3 Секреты и конфиги	Секреты не должны утекать	Workflow не использует секреты, вывод не содержит чувствительных данных. Для локали переменные вынесены в `.env.example`, подхватываются compose	`.github/workflows/ci.yml`, `compose.yaml`, `.env.example`
C4 Артефакты/ репорты	Сохранять отчёты/ артефакты	`pytest` пишет `reports/junit.xml`, шаг `actions/upload-artifact@v4` сохраняет артефакт `test-reports`	`.github/workflows/ci.yml`
C5 CD/ промоушн (эмуляция)	Базовый конвейер без деплоя	Pipeline ограничен линтами и тестами. Возможное расширение — публикация образа в GHCR или staging deploy как отдельный шаг	`.github/workflows/ci.yml`

Зачем сделано

- Цель: быстрый и предсказуемый feedback loop на push/PR, чтобы регрессии ловились до мержа и артефакты тестов были доступны в UI.
- Альтернативы и +/-: матрица Python/OS (шире охват, но дольше ран), разделение на независимые job линтов и тестов (больше параллелизм, сложнее конфигурация), cache по `requirements.lock` для более точного ключа (лучше кэш-хит, нужно поддерживать lock).
- Rollout plan: 1) запуск workflow в ветке `p08-cicd-minimal`, 2) открыть PR и проверить зелёный ран с артефактом junit, 3) при необходимости добавить SCA (`pip-audit`) и публикацию coverage/образа, 4) закрепить бейдж CI в README (уже добавлен).

Итог

- P07: multi-stage контейнеризация Wishlist API с non-root, healthcheck, volume для вложений, базовый харденинг, локальные цели для линтов и скана
- P08: минимальный стабильный CI на push/pull_request с кэшем pip, concurrency, линтами и тестами, публикацией junit-отчёта и бейджем статуса в README