Методичка к защите

Лабораторная 3 — балансировка и отказ API-реплики

Автор: Щербацевич Владислав Сергеевич
Группа: БПИ2302
Целевая оценка: 5
Трек: 1 — цена масштабирования (Compose + HAProxy, не Swarm, не трек 2)
Вход: http://127.0.0.1:8080 и http://localhost:8080 (оба должны отвечать)
Этот файл — шпаргалка. Не читать с листа целиком: открывать нужный раздел.

Как пользоваться на защите. Сначала 90 секунд «что это за система». Потом живой стенд: два X-Instance-Id, docker ps, stop-one. Код показывают только когда спрашивают «где». Цифры трека 1 — из REPORT.md и evidence/, не из головы. Если не помните число — откройте JSON.

0. Карта выступления (10–15 минут)

- 1 мин — задача лабы и схема Клиент → HAProxy → api×2 → Postgres + Redis.
- 1 мин — живой стенд: curl/браузер, два разных X-Instance-Id, одна сессия.
- 2 мин — docker compose ps: API без host-порта, имена api-N, лимиты.
- 2 мин — идемпотентность: ключ в заголовке, таблица в Postgres, гонка 12 потоков.
- 2 мин — отказ: stop-one, recovery ~2.5 с, вторая реплика.
- 2 мин — уровень 4: ready 1 с fall 2, SIGTERM, rolling, migrate one-shot.
- 3 мин — трек 1: гипотеза, 128/107/69 и 49/105/221, честно про saturation, IPv6.
- Оставшееся — вопросы.

1. Что говорить в первые 90 секунд

ГОВОРИТЬ: Это мессенджер с lab2: пользователи, чаты, сообщения, сессии. В lab3 я не размножаю данные, а размножаю процессы API. Снаружи одна дверь — HAProxy на порту 8080. За ней минимум два одинаковых контейнера api. Сообщения пишет PostgreSQL, сессии лежат в Redis, не в памяти процесса. Поэтому можно убить один api — логин не слетает, дублей сообщения нет, если клиент повторил тот же Idempotency-Key. Sticky-сессий нет: один пользователь разными запросами попадает на разные реплики — это видно по заголовку X-Instance-Id. Для пятёрки выбран трек 1: не «ещё узлов», а цена лишних реплик при фиксированном CPU.

НЕ ВРАТЬ: Не говорить «у нас высокая доступность системы». Выдерживается отказ ОДНОГО процесса API. Хост, HAProxy, Postgres, Redis — точки отказа. Трек 2 не делался.

2. Схема и роли файлов

Клиент (браузер, smoke, level3) бьёт в localhost:8080. Контейнер lb — HAProxy. Он round-robin раздаёт на контейнеры сервиса api. Имя api в Docker DNS имеет несколько IP. Postgres — источник истины (users, chats, messages, idempotency_records, schema_migrations). Redis — сессии, AOF appendfsync always. Контейнер migrate один раз накатывает SQL и выходит. API сами DDL не гоняют.

Куда ткнуть курсор, если просят «покажи файл»

- compose.yaml — весь стенд: сервисы, реплики, лимиты, секреты, нет ports у api, нет container_name.
- infra/haproxy.cfg — roundrobin, httpchk /health/ready, inter 1s fall 2, live отвечает сам LB.
- src/app/middleware.py — X-Request-Id и X-Instance-Id, 503 при shutdown.
- src/app/config.py — instance_id из HOSTNAME.
- src/app/api/health.py — live vs ready.
- src/app/main.py — accepting, lifespan SIGTERM.

- src/app/__main__.py — uvicorn timeout_graceful_shutdown=5.
- src/app/sessions.py — Redis PXAT, не память процесса.
- src/app/api/messages.py и src/app/crud/messages.py — Idempotency-Key.
- migrations/002_idempotency.sql — таблица ключей.
- src/app/migrate.py и сервис migrate в compose — one-shot.
- scripts/stop-one.sh (.cmd) — стоп по X-Instance-Id.
- scripts/rolling-update.sh — новый контейнер до удаления старого.
- student-tests/level3.py — две реплики + гонка ключа.
- student-tests/track1.py и evidence/track1.json — пятёрка.
- REPORT.md — цифры, которые можно зачитать.

3. Команды: стенд, тесты, доказательства

Защита на Windows. В PowerShell из корня репозитория. PYTHONUTF8=1 — чтобы make/python не падали на кириллице. Секреты не печатать в чат и не открывать на проекторе содержимое secrets/.

3.1. Поднять с нуля (если стенд ещё нет)

```
cd C:\Users\Admin\PycharmProjects\mtusi2026lab3
powershell -ExecutionPolicy Bypass -File scripts\bootstrap.ps1
docker compose --env-file .env config --quiet
docker compose --env-file .env up -d --build --wait --scale api=2
```

bootstrap создаёт .env и файлы секретов, не коммитятся. config --quiet проверяет compose. --wait ждёт healthcheck. scale api=2 обязательно: дефолт тоже 2, но на защите произнесите число.

3.2. Живой пульс

```
curl.exe -i http://127.0.0.1:8080/health/live
curl.exe -i http://127.0.0.1:8080/health/ready
docker compose --env-file .env ps
```

live с улицы отдаёт HAProxy: X-Instance-Id: load-balancer — это касса, не реплика. ready идёт на API. В ps у api колонка PORTS должна быть 8080/tcp БЕЗ 0.0.0.0:xxxx — порты на хост не проброшены. Доказательство уже снято: evidence/compose-ps.txt.

НЕ ВРАТЬ: Не доказывать разные реплики через /health/live. Только /api/v1/users/me или POST сообщения.

3.3. Два instance id одной сессией (сценарий README пункт 1)

Проще всего готовый тест — он логинится сам и 40 раз бьёт /users/me:

```
$env:PYTHONUTF8='1'
python student-tests/level3.py
```

PASS и evidence/level3.json: unique_instance_ids длина ≥2. Если хотят руками: зарегистрировать пользователя (examples/requests.http), взять Bearer, повторять GET /api/v1/users/me — заголовок X-Instance-Id должен чередоваться.

3.4. Публичные проверки курса

```
python tests/smoke.py --base-url http://127.0.0.1:8080
python student-tests/persistence_hook.py
python student-tests/failover.py
scripts\restore-api.cmd
python student-tests/failover.py
python student-tests/rolling.py
```

smoke — контракт lab3 (ключ, replay, 409). persistence — restart стенда, тот же токен, volumes на месте

(scripts/restart.cmd внутри хука). failover — stop-one.cmd по TARGET_INSTANCE_ID. После первого failover реплик станет 1 — restore-api.cmd возвращает scale 2, второй прогон бьёт уже другую. На POSIX в README: make test-failover ACTION_SCRIPT=scripts/stop-one.sh. У вас Windows, поэтому .cmd.

ПОКАЗАТЬ: Готовые логи: evidence/smoke.txt, persistence.txt, failover.txt, sequential_failover.json, rolling.json, level3.json.

3.5. Трек 1 (долго, на защите обычно НЕ гонять заново)

```
python student-tests/track1.py
python tests/load.py --duration 20 --concurrency 8 --output evidence/load_ipv4.json
```

Открыть evidence/track1.json и SVG. Повтор прогона — много минут и сменит цифры; для защиты достаточно снятого замера 2026-10-30.

3.6. Остановить / поднять

```
docker compose --env-file .env down          # данные в volumes остаются
docker compose --env-file .env down -v       # учебное обнуление БД — на защите не делать
```

4. Оценка 3 — что говорить и какой код

4.1. Стенд в одном compose, две реплики, одна дверь

ГОВОРИТЬ: Весь стенд в compose.yaml, не Swarm. Имя проекта mtusi-lab3. Сервисы: postgres, redis, migrate, api, lb. Реплик api две. С хоста торчит только lb 127.0.0.1:8080 и [::1]:8080. База и redis на loopback для отладки, это не API. Образы с digest, не latest.

ПОКАЗАТЬ: compose.yaml — services api и lb; блок deploy.replicas; отсутствие ports у api.

```
  api:
    deploy:
      replicas: ${API_REPLICAS:-2}
    # НЕТ ports: и НЕТ container_name
  lb:
    ports:
      - "127.0.0.1:8080:8080"
      - "[::1]:8080:8080"
```

Зачем нет container_name: имя контейнера в Docker уникально. Если написать messenger-api, вторую реплику scale не создаст. Compose сам даёт mtusi-lab3-api-1, api-2.

4.2. HAProxy: очередь без прилипания

ГОВОРИТЬ: HAProxy — касса с одной дверью. Клиент не знает про два официанта. Алгоритм roundrobin: запросы по кругу. Cookie нет, IP не клеим — sticky запрещён заданием, иначе скрыли бы, что сессия не общая. Список бэкендов не руками: server-template спрашивает Docker DNS имя api, до 4 слотов. Проверка живости — GET /health/ready каждые 1 с, два провала — вон из круга.

ПОКАЗАТЬ: infra/haproxy.cfg целиком, особенно backend api.

```
frontend public
    bind *:8080
    http-request return status 200 ... hdr X-Instance-Id "load-balancer" ... if { path /health/live }
    default_backend api
backend api
    balance roundrobin
    option httpchk
    http-check send meth GET uri /health/ready ...
    http-check expect status 200
    default-server inter 1s fall 2 rise 1 resolvers docker ...
    server-template api 4 api:8080 check
```

Почему не Nginx: свободный выбор, взяли HAProxy 3.2 с docker resolvers. Почему live на LB: контракт хочет, чтобы публичный live проверял процесс ВХОДА. Ready — можно ли давать работу этой реплике (БД+Redis+accepting).

4.3. X-Instance-Id и X-Request-Id

ГОВОРИТЬ: Request-Id — имя письма, каждый запрос новый UUID, его клеит middleware. Instance-Id — имя почтальона: пока жив процесс, строка одна. Берётся из переменной INSTANCE_ID, иначе HOSTNAME контейнера. В базу не пишется. Нужен, чтобы доказать две реплики и чтобы stop-one нашёл контейнер. Request-Id для docker stop бесполезен.

ПОКАЗАТЬ: src/app/config.py функция _instance_id; src/app/middleware.py; src/app/main.py app.state.instance_id

```
# config.py
configured = os.environ.get("INSTANCE_ID", "").strip()
if configured: return configured
host = os.environ.get("HOSTNAME", "").strip() or socket.gethostname()
return host

# middleware.py
response.headers["X-Request-Id"] = request_id
response.headers["X-Instance-Id"] = _instance_id(request)
```

4.4. Общие сессии — почему не память API

ГОВОРИТЬ: Если сессию хранить в словаре Python, второй контейнер не узнает токен — после round-robin будет 401, придётся логиниться снова. Redis общий. Ключ sess:sha256(token), TTL абсолютный PXAT, чтобы после рестарта Redis из AOF срок не продлился. AOF appendfsync always — учебная сохранность при restart.

ПОКАЗАТЬ: src/app/sessions.py create_session / resolve_session; compose redis command --appendonly yes

```
token = secrets.token_urlsafe(32)
await redis.set(session_key(token), payload, pxat=int(expires_at.timestamp() * 1000))
```

4.5. Идемпотентность POST сообщения

ГОВОРИТЬ: Ключ придумывает клиент, не человек в UI и не сервер. Сервер только читает заголовок Idempotency-Key. Без него 400. Область: пользователь + POST + путь чата + строка ключа. Хранится в Postgres таблица idempotency_records. Повтор того же текста — тот же message id. Другой текст — 409. Другой пользователь с тем же ключом — другое сообщение. Параллель: pg_advisory_xact_lock, чтобы два API не вставили два сообщения. Пишем Message, flush, потом запись ключа — в одной транзакции. 5xx не сохраняем: если упали до INSERT ключа, клиент может повторить тот же ключ.

ПОКАЗАТЬ: src/app/api/messages.py require_idempotency_key; crud/messages.py create_message; migrations/002_idempotency.sql

```
# api: без заголовка — 400
value = request.headers.get("idempotency-key")

# crud: замок по user+path+key, потом SELECT ключа
SELECT pg_advisory_xact_lock(:a, :b)
# тот же hash текста → вернуть старое сообщение
# другой hash → 409
# нет записи → INSERT messages, INSERT idempotency_records status 201
```

Генерация ключа в тестах: tests/client.py — uuid.uuid4().hex если вызывающий не передал key. Гонка: student-tests/level3.py функция test_concurrent_key — Barrier на 12 потоков, один ключ, ожидаем 12×201 и один id, в истории одна копия. Запускается внутри python student-tests/level3.py.

4.6. Failover ≤ 10 секунд

ГОВОРИТЬ: Тест читает X-Instance-Id из ответа, кладёт в TARGET_INSTANCE_ID, вызывает scripts/stop-one.cmd. Скрипт сверяет hostname/имя/id контейнера и docker stop -t 5. HAProxy за ~2 с (1 с × fall 2) выкидывает труп из круга. Живая реплика продолжает. Сессия в Redis, сообщения в Postgres — повторный логин не нужен. Измеренный recovery 2.46 с, ошибок 2 из 93. Курс просит не больше 10 с. Краткие ошибки в момент обнаружения допустимы.

ПОКАЗАТЬ: scripts/stop-one.sh и stop-one.cmd; student-tests/failover.py; evidence/failover.txt и sequential_failover.json

```
# stop-one.sh
hostname=$(docker inspect -f '{{.Config.Hostname}}' "$id")
if [ "$TARGET_INSTANCE_ID" = "$hostname" ] || ...; then found=$id; fi
docker stop -t 5 "$found"
```

После первого стопа restore-api.cmd поднимает вторую реплику. Повтор failover.py бьёт уже её. Отдельно в отчёте docker stop второй: пишет оставшийся api, recovery почти 0, потому что касса уже на живом. Показать docker compose ps — один Exited, второй healthy.

5. Оценка 4

5.1. Health, таймауты, SIGTERM, период проверки

ГОВОРИТЬ: Live — процесс существует. Ready — принимаем работу: accepting=True, SELECT 1 в Postgres, ping Redis. HAProxy проверяет ready, не live: иначе труп без базы остался бы в круге. Период проверок: inter 1s. Максимум обнаружения отказа: fall 2 → около 2 секунд; с DNS hold valid 2s верхняя оценка около 4 с. SIGTERM: compose stop_signal SIGTERM, stop_grace_period 8s, uvicorn timeout_graceful_shutdown 5. В finally lifespan accepting=False — ready и бизнес отдают 503, касса выкидывает бэкенд до убийства процесса.

ПОКАЗАТЬ: health.py; main.py lifespan; __main__.py uvicorn; compose stop_grace_period; haproxy inter/fall

```
# health.py ready
if not request.app.state.accepting: 503
SELECT 1; redis.ping()

# main.py finally
app.state.accepting = False

# middleware: если not accepting и путь не /health/live → 503
```

5.2. По очереди обе реплики, RPS, идемпотентность параллельно

Цифры зачитать из REPORT / sequential_failover.json, не путать с 128 RPS трека 1 — другой клиент.

- stop той, что ответила: recovery 2.46 с, 91 успех / 2 ошибки, 6.07 RPS, окно 15 с.
- restore + ещё раз: 2.74 с, 90/2, 6.00 RPS.
- stop второй: ~0.03 с, 46/1, 3.83 RPS.
- параллельный ключ: level3, 12 потоков, один id.

5.3. Лимиты, секреты, нет порта API

ГОВОРИТЬ: У каждого сервиса deploy.resources.limits. api 0.50 CPU / 256M каждая. Секреты Docker: postgres_password только postgres, database_url — migrate и api, cache_url только api. HAProxy паролей не видит. Volume только postgres_data и redis_data — у api своего диска с чатами нет.

ПОКАЗАТЬ: compose.yaml secrets, volumes, deploy.resources, отсутствие ports у api, отсутствие container_name.

5.4. Rolling и миграции один раз

ГОВОРИТЬ: Rolling не равен failover. Failover — убили живого. Rolling — сначала подняли нового, потом стопнули старого, по одной штуке, под нагрузкой. Скрипт: scale N+1, wait, docker stop -t 5 старого, rm, вернуть scale N. Замер: rolling.json, 1980 ok / 24 err (1.2%), оба старых instance id исчезли. Ненулевые ошибки на стыке DNS нормальны.

ПОКАЗАТЬ: scripts/rolling-update.sh; student-tests/rolling.py

ГОВОРИТЬ: Миграции: сервис migrate, restart no, python -m app.migrate, API depends_on service_completed_successfully, RUN_MIGRATIONS=0. API ждут schema_migrations. Совместимость: rolling одной версии API; схему заранее накатил migrate. Не обещаем lab2 и lab3 одновременно. Ограничение честно: откат схемы не делали.

ПОКАЗАТЬ: compose.yaml service migrate; src/app/migrate.py pg_advisory_xact_lock; main.py wait_for_schema

6. Оценка 5 — трек 1

6.1. Речь 2–3 минуты

ГОВОРИТЬ: Трек 1 — цена масштабирования. Гипотеза: при фиксированном суммарном CPU добавление реплик не ускоряет POST, потому что режется CPU каждого процесса, а не потому что кончилась база. Методика: один профиль, 8 closed-loop клиентов через HAProxy, прогрев 5 с, замер 20 с. Серия A: всегда 1.0 CPU и 512 MiB на все API, реплик 1, 2, 4. Серия B: всегда 2 реплики, CPU растёт 0.5 → 1 → 2 суммарно. A: 128 → 107 → 69 RPS. Четыре реплики на том же ватте почти вдвое медленнее одной. Ошибок 0. B: 49 → 105 → 221. Дали CPU — почти линейно. Значит запас по Postgres был. Улучшение: bind [::1]:8080, иначе Windows localhost давал p50 две секунды.

6.2. Где цифры

ПОКАЗАТЬ: REPORT.md таблица; evidence/track1.json; графики track1_fixed_budget.svg и track1_growing_cpu.svg

- fixed-1: 1×1.00 CPU, 128.1 RPS, p50 52 мс
- fixed-2: 2×0.50, 107.5 RPS, p50 72 мс
- fixed-4: 4×0.25, 69.3 RPS, p50 95 мс
- grow-2×0.25: сумма 0.50 CPU, 49.1 RPS
- grow-2×0.50: сумма 1.00, 105.5 RPS
- grow-2×1.00: сумма 2.00, 221.0 RPS

Скрипт student-tests/track1.py: scale_api, docker update --cpus/--memory, run_load 8 потоков POST /messages. Closed-loop: следующий запрос только после ответа, RPS ≈ 8 / среднее время.

6.3. Saturation point — как не попасть

НЕ ВРАТЬ: В README написано «найдите saturation point БД/LB/пула». Классического плато этих трёх НЕТ. Серия B ещё растёт до 221, postgres в docker stats доли процента, пул 10+10 на реплику, при 4 репликах до 80 соединений при max_connections 100, ошибок 0. Мы НЕ дожали базу до потолка. Это дырка по букве задания. Закрываем её честным текстом, не выдуманным «Postgres на 100%».

ГОВОРИТЬ: Искали насыщение Postgres, HAProxy и пула. На этом профиле они не насытились: stats пустые, при росте CPU RPS ещё линейный. То, что видно: при фиксированном бюджете рост ЧИСЛА реплик перестаёт помогать и вредит — это нарезка CPU API. Чтобы поймать плато базы, нужен более жёсткий профиль, его нет в evidence. Это ограничение опыта, записано в конце REPORT.md.

Где «показано» в работе: абзац под таблицей в REPORT.md и массивы docker_stats в track1.json (имена

postgres-1, lb-1, api-*). Отрицательный результат: они не bottleneck. Не ищите слайд «база 100%» — его нет.

6.4. Подтверждённое улучшение

Задание не требует конкретно IPv6. Требует одно измеренное до/после. В отчёте взяли вход. Было только 127.0.0.1:8080. Windows localhost идёт в ::1, p50 ≈ 2 с, ≈1.9 RPS. Добавили [::1]:8080. После: localhost 87.6 RPS p50 75 мс, 127.0.0.1 94.7 RPS. Файлы: compose.yaml ports lb; evidence/load_localhost.json. Это не «серия B». Серия B — обязательный второй опыт, не фикс.

6.5. Ограничения трека, которые сами назвать первым

- Один Docker-host. Отказ VM/ноутбука не переживается.
- Один HAProxy, одна Postgres, один Redis — SPOF.
- Closed-loop занижает абсолютный RPS; сравнивать отношения точек, не 128 с одногруппником.
- docker stats --no-stream — кадр в конце, не среднее; не объяснять 24% vs 0.7% как «вторая не работала».
- Две реплики для failover, не для скорости при 1 CPU. Это разные оси.
- Гипотеза в исходнике track1.py чуть старая про Postgres; в JSON и REPORT исправлена — читать REPORT.

7. Живой сценарий защиты (чеклист)

- Стенд уже up --scale api=2, ready 200.
- docker compose ps — два api healthy без host-порта, lb 8080.
- python student-tests/level3.py — две реплики + идемпотентность. Открыть level3.json.
- По желанию: два curl GET /api/v1/users/me с одним Bearer — разные X-Instance-Id.
- python student-tests/failover.py — не паниковать из-за 1–2 ошибок. Потом docker ps: один Exited.
- scripts\restore-api.cmd, второй failover или руками docker stop второго имени из ps.
- Открыть REPORT таблицу трека 1 и два SVG. Произнести 128/107/69 и 49/105/221.
- Если просят код HAProxy / идемпотентности / stop-one — разделы 4–5 этой методички.
- Не гонять track1.py с нуля, если нет 15+ минут.

8. Примерные вопросы и ответы

В: Чем отказ процесса отличается от отказа узла?

О: Процесс — один контейнер api, docker stop. Второй api на том же host + общие Redis/Postgres живут. Узел — вся ВМ/ноутбук/Docker Desktop: вместе падают LB, обе реплики, база. Это трек 2, мы его не заявляем.

В: Почему depends_on и healthcheck — не балансировщик?

О: depends_on решает порядок СТАРТА (сначала postgres healthy, потом api). Healthcheck Compose красит контейнер healthy для --wait. Трафик клиентов они не делят. Делит HAProxy: у него свой httpchk и roundrobin. Без LB клиент должен был бы сам знать два IP.

В: Какие POST можно повторять после timeout и что защищает Idempotency-Key?

О: Клиент не знает, записалось ли сообщение. Имеет право повторить ТОТ ЖЕ POST с ТЕМ ЖЕ ключом и тем же телом. Сервер вернёт старый id. Другое тело с тем же ключом — 409, это уже другое сообщение, новый ключ. Ключ обязателен только у POST сообщения. Регистрацию/чаты так не защищали.

В: Что сломается, если сессии в локальной памяти, а API станет больше?

О: Round-robin отправит следующий запрос на соседний процесс — токена там нет — 401. Sticky это запрещённый костыль: убили «свою» реплику — сессия всё равно умерла. Поэтому Redis.

В: Зачем X-Instance-Id, если есть X-Request-Id?

О: Request-Id уникален на каждый хит даже на одном процессе. Им нельзя доказать две реплики и нельзя docker stop. Instance-Id стабилен на жизнь процесса и равен hostname.

В: Как HAProxy выбирает первое или второе API?

O: Счётчик round-robin + кто проходит ready. «Второе вместо первого» — следующий слот, не баг. Порядок слотов из DNS, не «api-1 важнее».

B: Период проверок и максимальное время обнаружения отказа?

O: inter 1s, fall 2 → типично ~2 с после 503/смерти. timeout check 1s. DNS hold valid 2s → верх ~4 с. Измерили recovery 2.46 с. Бюджет курса 10 с.

B: Где генерируется Idempotency-Key?

O: На клиенте. В сервере генерации нет, иначе сервер не отличит повтор от нового. Тесты: uuid4 в tests/client.py и student-tests/level3.py. Человек в чате ключ не печатает.

B: Почему 12 параллельных POST не делают 12 сообщений?

O: Оба API идут в одну таблицу. Замок advisory на время транзакции: второй ждёт и видит ключ первого. UNIQUE на (user, method, path, key) — страховка. Тест test_concurrent_key.

B: Почему публичный live отвечает HAProxy?

O: Live входа = жива ли касса. Если live отдавать API, при мёртвом LB клиент не достучится, а при живом LB и мёртвых API live всё равно можно отдать с кассы. Ready специально идёт на API.

B: Зачем два одинаковых контейнера, если трек 1 показал, что один быстрее?

O: Скорость при фиксированном 1 CPU и выживаемость — разные цели. Один толстый процесс быстрее на этом профиле. Два нужны, чтобы docker stop одного не обнулил сервис. На защите развести оси.

B: Где saturation point БД/LB/пула?

O: Не найден как плато. Stats postgres/lb ≈0%, серия grow ещё линейна до 221 RPS, пул не упёрся. Нашли другое: при фиксированном CPU рост реплик вреден. Классическое насыщение склада не показали — ограничение профиля 8×20c. Так и написано в REPORT.

B: Почему не нулевые ошибки на failover/rolling?

O: Между SIGTERM и исключением из LB есть запросы в умирающий процесс. Курс это разрешает, если измеряли. У нас единицы процентов, recovery секунды.

B: Миграции на каждой реплике?

O: Нет. Отдельный one-shot migrate, API RUN_MIGRATIONS=0. Иначе гонка CREATE TABLE.

B: Почему нет sticky?

O: Sticky прячет раздельное состояние. Задание прямо запрещает чинить им локальные сессии. Состояние общее — прилипать незачем.

B: Что будет, если убить Postgres?

O: Все api станут unready (ping БД падает), HAProxy перестанет слать бизнес-трафик, сервис ляжет. Две реплики API базу не спасают. Не заявлять обратное.

B: Что будет, если убить Redis?

O: Новые логины и проверка Bearer не работают (503). Старые сообщения в Postgres живы, но API без сессии их не отдаст. Redis SPOF для аутентификации.

B: Closed-loop / coordinated omission?

O: Клиент ждёт ответа, медленные запросы уменьшают число попыток. Для сравнения 1/2/4 одним клиентом ок. Это не абсолютный потолок стенда как open-loop.

B: Почему в stats одна реплика 24%, сосед 0.7%?

O: Один кадр docker stats в конце прогона, не интеграл за 20 с. Не строить на этом теорию. Смотреть RPS и лимиты.

B: Зачем [::1], это же не про реплики?

O: На Windows localhost ≠ 127.0.0.1. Без IPv6-bind измерение «через localhost» врало бы ~2 с на запрос. Это подтверждённое улучшение входа, без него трек 1 на localhost не про API.

B: Можно ли сравнить ваш 128 RPS с чужим ноутбуком?

O: Нет. Курс: универсального проходного RPS нет. Сравнивают отношения на одной машине: 128:107:69.

В: workers uvicorn больше 1?
О: workers=1. Несколько воркеров в одном контейнере — уже не наш X-Instance-Id=hostname на всех. Горизонталь — отдельные контейнеры.

В: Как stop-one понимает, кого убивать?
О: Тест передаёт TARGET_INSTANCE_ID = значение заголовка. Скрипт сравнивает с Hostname контейнера (мы как раз кладём hostname в заголовок).

В: Rolling под нагрузкой зачем, если есть failover?
О: Обновление образа без «выключить всех». Сначала лишний здоровый, потом убрать старого. Иначе дыра на время сборки.

В: Совместимы ли миграции с работающей версией?
О: Additive: новая таблица ключей. Rolling гоняет одну версию API после migrate. Не заявляем blue-green двух схем.

В: Почему Postgres слушает 5433 на хосте, это же дырка?
О: Учебный отладочный bind на 127.0.0.1, не публикация API. Снаружи интернет не торчит. Можно признать упрощением.

В: Аргон2 и lab2?
О: Пароли как в lab2, Basic только на создание сессии, дальше Bearer. Это не тема lab3, но контракт сохранён.

В: Что в PR / отчёте ещё TODO?
О: В REPORT строка Commit/PR: TODO после публикации. Если PR не открыт — сказать как есть, не выдумывать номер.

9. Мини-шпаргалка чисел

- Реплики: 2. Вход: 8080. Ready LB: 1 с, fall 2, обнаружение ~2 с (верх ~4 с). Failover recovery ~2.5 с.
- Grace: stop -t 5, uvicorn 5, compose 8 с.
- TTL сессии 3600, ключа идемпотентности 86400.
- Лимит api: 0.50 CPU / 256M. Трек1 fixed сумма всегда 1 CPU / 512M.
- Трек1 RPS: 128 / 107 / 69 и 49 / 105 / 221. Ошибки 0.
- IPv6 фикс: ~1.9 RPS → ~88 RPS на localhost.
- Гонка: 12 POST, 1 message id. Rolling: ~1.2% ошибок.

10. Если что-то пойдёт не так на защите

- Стенд не ready: docker compose ps, логи docker compose logs api --tail 80, часто migrate не прошёл или Docker Desktop спал.
- Один instance id: реплика одна — compose up --scale api=2; или смотрите live вместо /users/me.
- localhost тупит, 127.0.0.1 быстрый: проверяйте, что в compose есть [::1]:8080 и контейнер lb пересоздан.
- failover не находит TARGET: instance id должен совпасть с hostname (inspect). Не сравнивать с load-balancer.
- Не лезьте чинить публичные tests/ и contracts/ на глазах — это запрещено курсом.

Конец методички. Источник цифр: REPORT.md и evidence/ на дату 2026-10-30. Если перегоните тесты — замените числа, не зубрите старые. Код открывать из репозитория, не из памяти.