

Быстрые тесты на реальных SQL бд (postgres и не только)

Булавский Сергей
Рейнджер, Бюро разработки

Контур



О себе

- 10+ лет в разработке
- Люблю улучшать DevEx
- Люблю тесты
- Currently at: Контур — Рейнджеры



Кадры из шоу «細かすぎて伝わらないモノマネ選手権», Fuji TV

Кликбэйт

Переход на ... дал:

~ *30 min* → ...

Кликбэйт

Переход на ... дал:

~ 30 min → ~3 min

10 лет назад



С чего мы начинали?

С чего мы начинали?

- Медицинский продукт
- Распределенная команда
- Докер только набирает популярность
- TDD/тесты в каждом пулл реквесты
- .NET Core 1.0-2.0
- EF Core 1.0-2.0
- Реальные постгрес инстансы в ограниченном количестве

Какие тесты мы писали?

- **Юнит** – для тестирования отдельных сервисов
- **Интеграционные на уровне сервиса** для тестирования в связке с реальной БД
- **Интеграционные на несколько сервисов** и несколько БД
- **E2E** – для всей системы

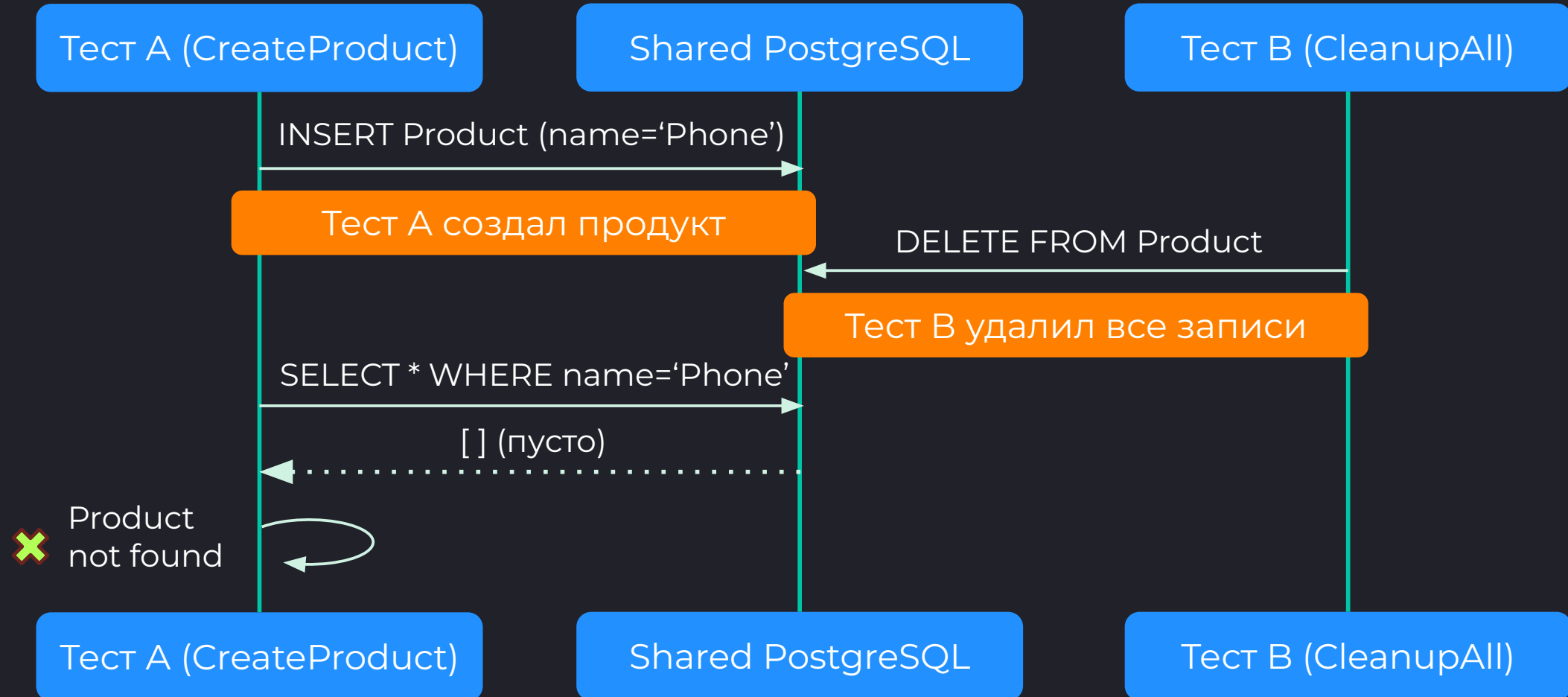
Почему тесты с реальной БД?

- Основная часть бизнес правил часто живет в схеме БД
- Слой работы с данными обычно самый критичный в ошибках и производительности
- SQL сам по себе обычно не тестируется
- До EF Core 3.0 Query Client Evaluation был по умолчанию и транспайлинг запросов был еще не так прокачан

Проблемы с тестами на реальной БД

- Необходимость поддерживать БД в «чистом» состоянии – мануальные действия = время
- Параллельное выполнение необходимо для экономии времени
- Нарушение изоляции между тестами на всех уровнях в случае параллельных запусков

Нарушение изоляции



Проблемы с тестами на реальной БД

- Необходимость поддерживать бд в «чистом» состоянии – мануальные действия = время
- Параллельное выполнение необходимо для экономии времени
- Нарушение изоляции между тестами на всех уровнях в случае параллельных запусков
- Написание логики с расчетом на грязное состояние БД – не работает долгосрочно

Как писать тесты.md

- Из `users` ничего не удалять – на первые 50 записей завязаны тесты авторизации.

Как писать тесты.md

- Из `users` ничего не удалять — на первые 50 записей завязаны тесты авторизации.
- Проверять наличие по `id`, не по `Count()` — соседние тесты добавляют строки.

Как писать тесты.md

- Из `users` ничего не удалять — на первые 50 записей завязаны тесты авторизации.
- Проверять наличие по `id`, не по `Count()` — соседние тесты добавляют строки.
- Email в тестах — только `@example.com`, потому что `cleanup`-скрипт удаляет по маске.

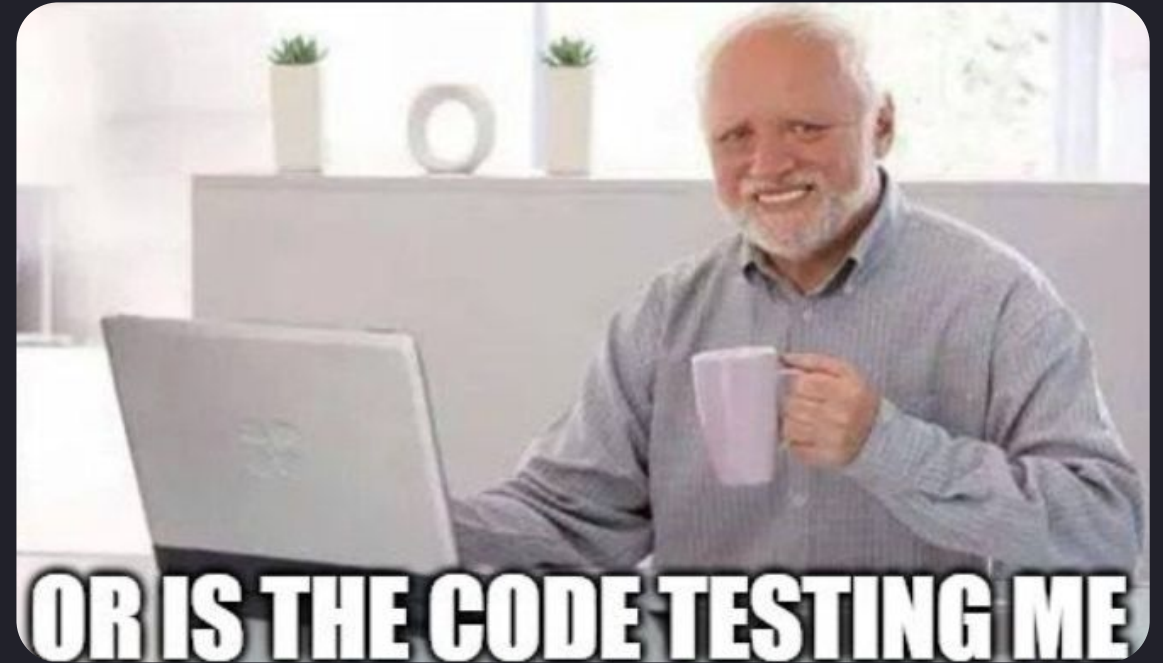
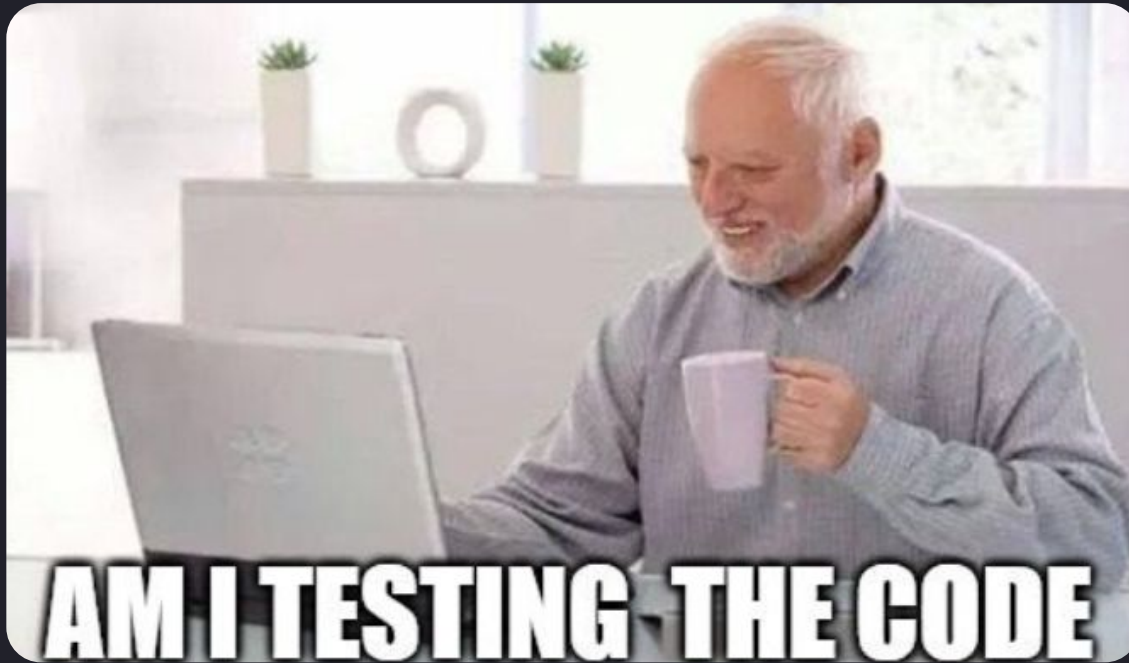
Как писать тесты.md

- Из `users` ничего не удалять — на первые 50 записей завязаны тесты авторизации.
- Проверять наличие по `id`, не по `Count()` — соседние тесты добавляют строки.
- Email в тестах — только `@example.com`, потому что `cleanup`-скрипт удаляет по маске.
- Категорию «Разное» нельзя переименовывать — на неё захардкожен `id=7` в трёх местах.

Как писать тесты.md

- Из **users** ничего не удалять — на первые 50 записей завязаны тесты авторизации.
- Проверять наличие по id, не по Count() — соседние тесты добавляют строки.
- Email в тестах — только **@example.com**, потому что cleanup-скрипт удаляет по маске.
- Категорию «Разное» нельзя переименовывать — на неё захардкожен **id=7** в трёх местах.
- **// НЕ УДАЛЯТЬ**. Ломает тесты отчётов.

Эксперименты



Источник: Dreamstime (стоковое фото)

Moq/Substitute Repository/DbSet

- Нет нормального состояния
- Нет выполнения запросов в SQL

EF In-memory

- Провайдер базы данных, предназначенный исключительно для тестирования приложений
- Позволяет разработчикам имитировать базу данных в оперативной памяти системы вместо использования физического хранилища данных

EF In-memory

- Не проверяет целостность данных (UNIQUE, FK, CASCADE, CHECK/NOT NULL)
- Транзакции не поддерживаются
- SQL и трансляция (RAW SQL, JSONB и т.д. не работают)
- Миграции и raw-SQL объекты не применяются

SQLite

- Реальная БД

SQLite

- Свой диалект
- Миграции для SQLite и PostgreSQL расходятся
- Некоторые операции вообще не поддерживаются (ALTER/DROP COLUMN в старых версиях)
- Блокируется вся база данных на запись (SQLITE_BUSY)

Фейки делают **только хуже!**

PostgreSQL с роллбэками

- Очень быстрый подход
- Рабочий подход до определенной степени сложности логики
- Если в коде есть создание и коммит транзакций – возникнут конфликты с внешней транзакцией
- Всё должно работать через одно соединение с внешней открытой транзакцией



"Hapyto" (Naruto), © Studio Pierrot / Masashi Kishimoto

Как бы я делал это **сейчас**

Требования к тестам

- **Скорость выполнения** – медленные тесты гоняют реже или вообще не гоняют

Стоимость скорости

Команда из 10 разработчиков, 5 PR в день на команду, по 3 прогона CI на PR.

250 рабочих дней в году, средний прогон тестов 20 минут, средневзвешенная стоимость часа разработчика 4500 руб (для бизнеса), стоимость часа CI раннера ~100 руб.

Потери разработчика на ожидание и переключение контекста 50%

Рыночные штатные ставки: junior 1 300–2 700, middle 3 100–5 600, senior от 5 700 Р/час. (Клерк, обзор рынка, октябрь 2025)

CI: GitHub Actions Linux 4-core, \$0,016/мин.

Стоимость скорости

5 PR x 3 попыток x 250 рабочих дней = 3750
прогонов в год

625 часов машинного времени CI ~60 тыс. рублей
Время разработчиков (50%) ~310 часов - ~1.4 млн Р.

Требования к тестам

- **Скорость выполнения** – медленные тесты гоняют реже или вообще не гоняют
- **Изоляция** – для предсказуемости, простоты тестов и возможности параллелизма

Требования к тестам

- **Скорость выполнения** – медленные тесты гоняют реже или вообще не гоняют
- **Изоляция** – для предсказуемости, простоты тестов и возможности параллелизма
- **Возможность локального запуска** – дебаг, более быстрый feedback loop для агентов и для меня



Требования к тестам

- **Скорость выполнения** – медленные тесты гоняют реже или вообще не гоняют
- **Изоляция** – для предсказуемости, простоты тестов и возможности параллелизма
- **Возможность локального запуска** – дебаг, более быстрый feedback loop для агентов и для меня
- **Простота** в написании

Как создавать **реальные** бд?

Создание реальных БД под тесты

- Одна БД на весь запуск – быстро создается, один раз мигрируется – худший вариант изоляции
- Одна БД на тест класс – создание и миграция на тест класс – изоляция внутри тест класса нарушена
- Одна БД на тест – полная изоляция – создание и миграции БД на тест

Testcontainers

 **testcontainers-dotnet** Public

 Sponsor

 Watch **32**

 Fork **348**

 Star **4.3k**

 **develop**

 **6 Branches**

 **44 Tags**

 Add file

 Code

 HofmeisterAn	chore: Migrate to LoggerMessageAttribute (#1697) ✓	32acbf4 · 2 days ago	 947 Commits
 .devcontainer	feat: Add module connection string provider (#1632)	3 months ago	
 .github	chore(deps): Bump the actions group with 5 updates (#1687)	last month	
 build	feat: Add Docker Engine v29 support (#1609)	5 months ago	
 docs	docs: Add note about unsupported BuildKit Dockerfile featur...	2 days ago	
 examples	chore(examples): Enable NuGet restore lock mode (#1659)	2 months ago	
 src	chore: Migrate to LoggerMessageAttribute (#1697)	2 days ago	
 tests	feat: Add Floci module (#1690)	2 weeks ago	
 .editorconfig	chore: Remove StyleCop.Analyzers (#848)	3 years ago	
 .gitattributes	chore: Remove Git LFS tracking for .snk (#1655)	2 months ago	

About

A library to support tests with throwaway instances of Docker containers for all compatible .NET Standard versions.

dotnet.testcontainers.org

testing docker automation
dotnet testcontainers hacktoberfest
testcontainers-dotnet

 Readme
 MIT license
 Code of conduct
 Contributing
 Security policy
 Activity

Testcontainers – container per test

.NET Test Host

TestContainers

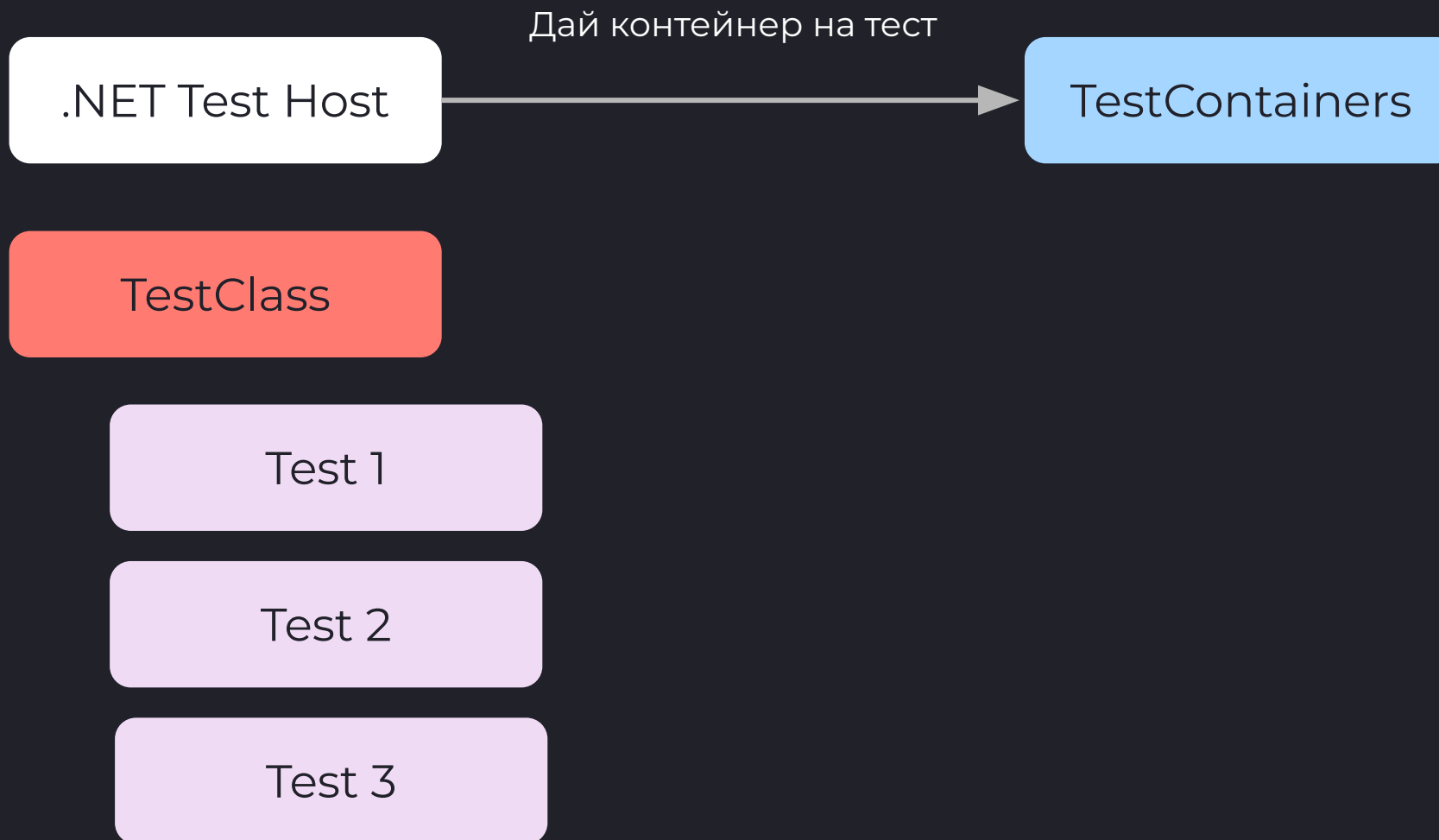
TestClass

Test 1

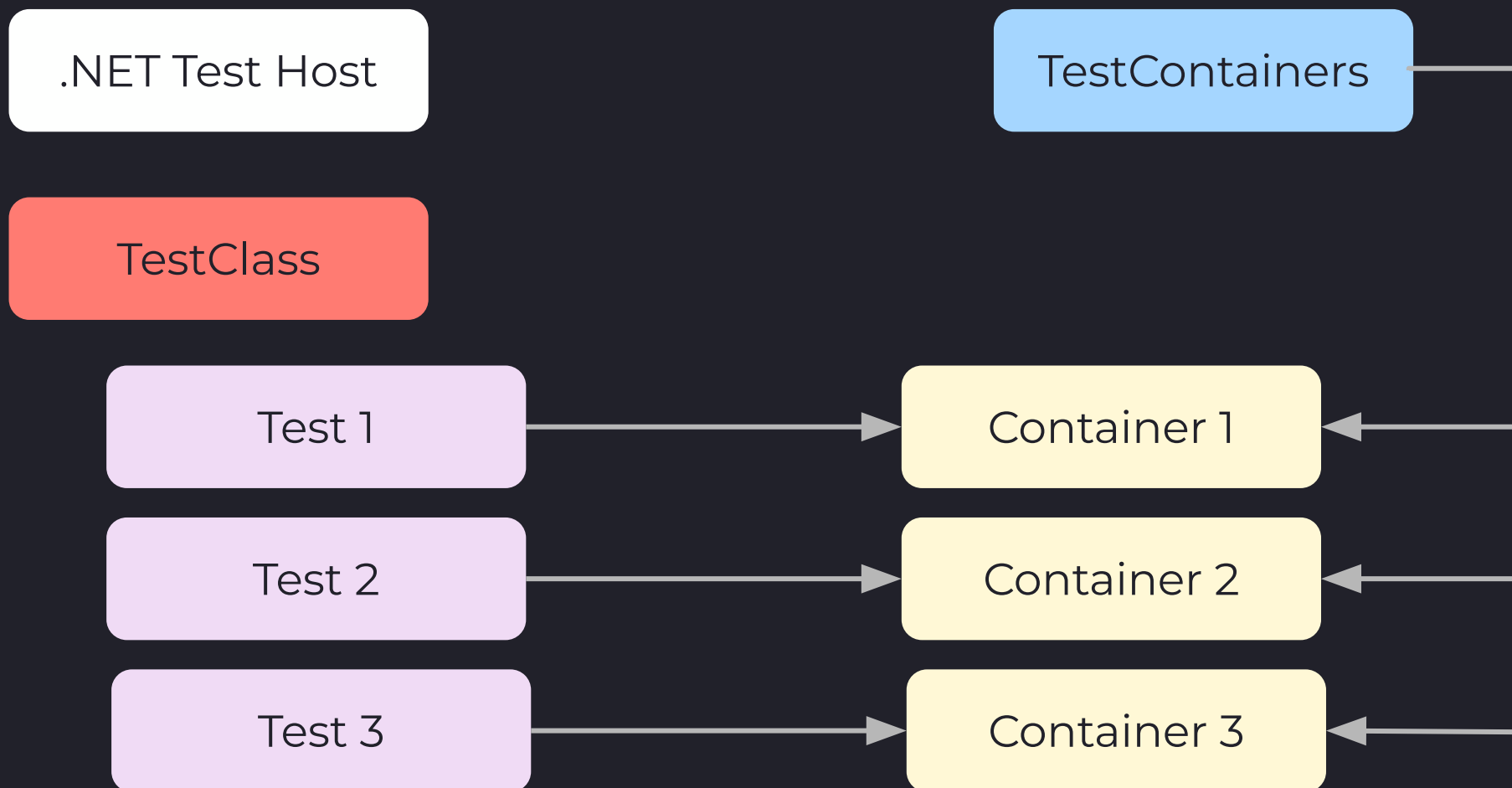
Test 2

Test 3

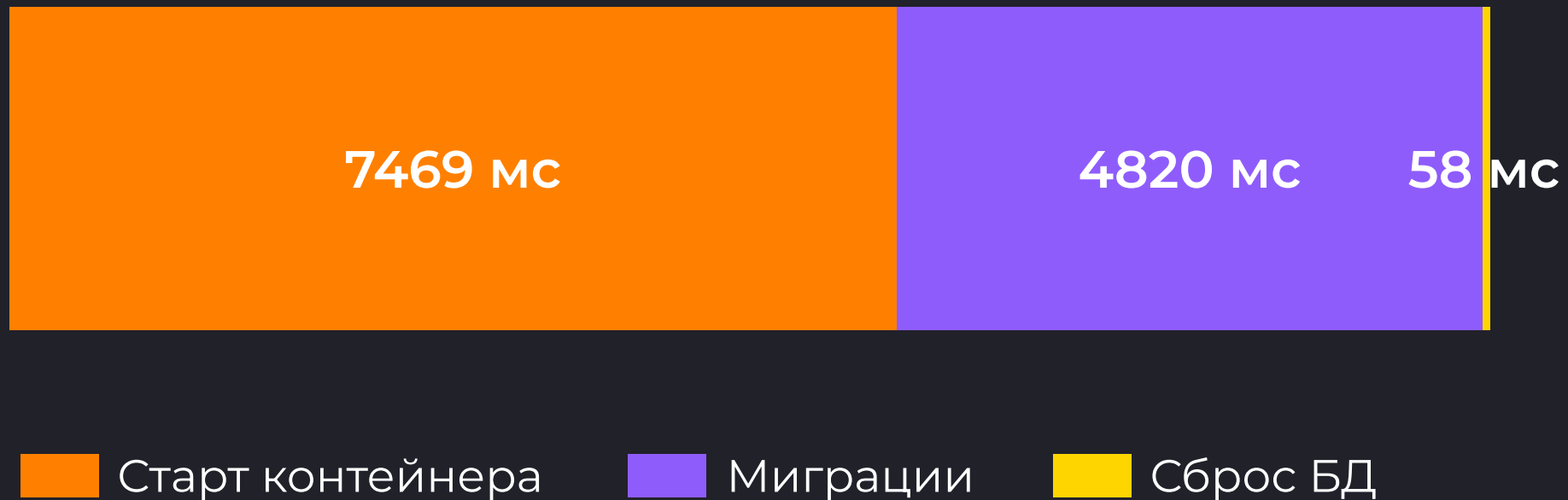
Testcontainers – container per test



Testcontainers – container per test



Testcontainers – container per test



Testcontainers – container per test

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~12.3 сек

инфраструктуры
(в среднем на тест)

время на тест:

~7469 мс

старт контейнера
(в среднем на тест)

~4820 мс

117 миграций:

Testcontainers – container per test

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~12.3 сек

инфраструктуры
(в среднем на тест)

время на тест:

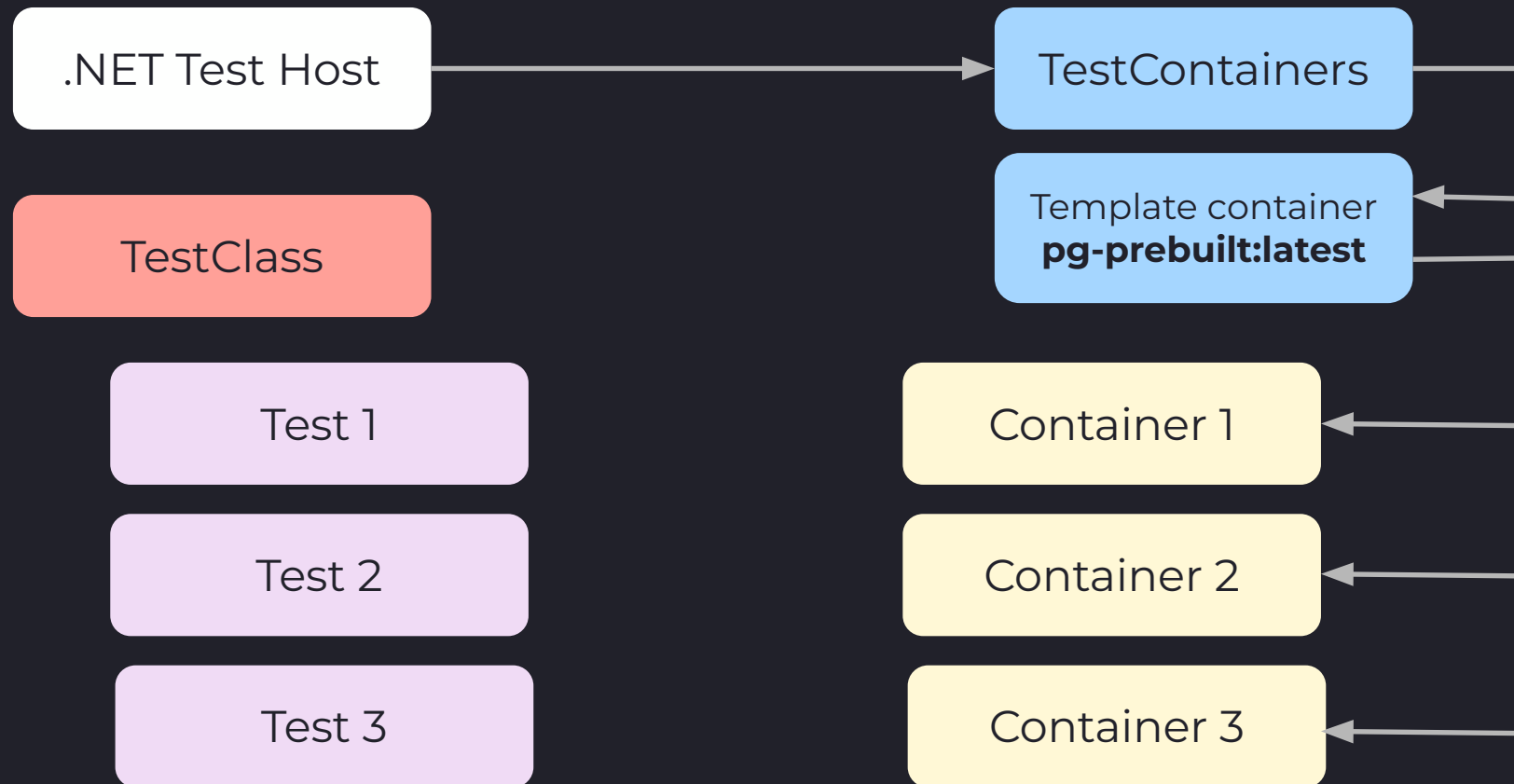
~7469 мс ~4820 мс

старт контейнера
(в среднем на тест)

117 миграций:

на 2340 тестов ~60 минут
инфраструктуры

Testcontainers – container per test – Docker commit



Docker commit

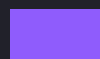
Docker commit сохраняет writable-слой запущенного контейнера в новый образ — «снимок» файловой системы.

Поднимаем Postgres, накатываем миграции, останавливаем (docker stop, для checkpoint) и коммитим — новые контейнеры из этого образа стартуют уже с готовой схемой.

Testcontainers – container per test – Docker commit



Старт контейнера



Миграции



Сброс БД

Testcontainers – container per test – Docker commit

время на тест:

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~7183 мс

старт контейнера
(в среднем на тест)

~5-10 мс

117 миграций:

~7.2 сек

инфраструктуры
(в среднем на тест)

Testcontainers – container per test – Docker commit

время на тест:

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~7183 мс

старт контейнера
(в среднем на тест)

~5-10 мс

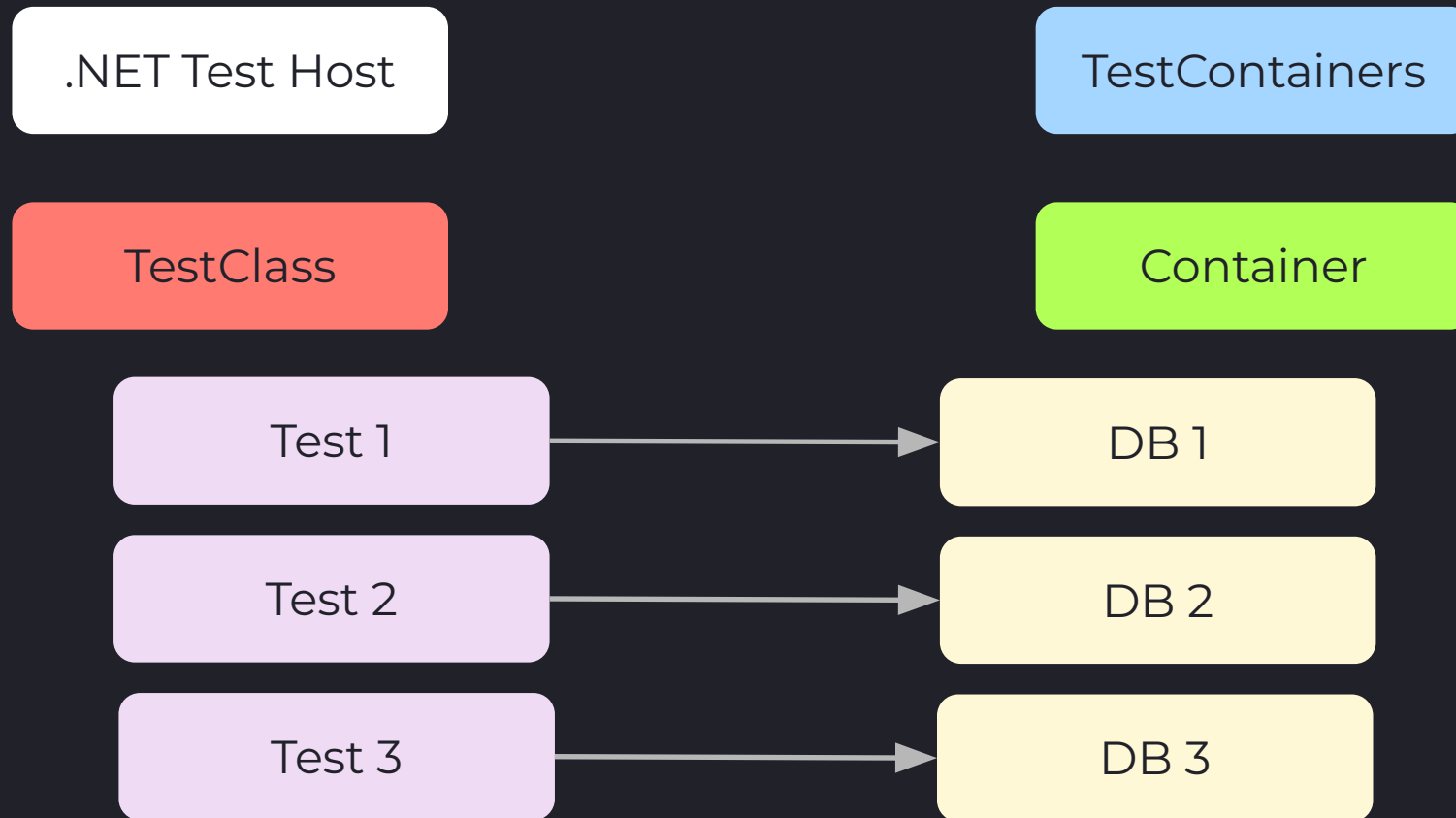
117 миграций:

~7.2 сек

инфраструктуры
(в среднем на тест)

на 2340 тестов ~35 минут
инфраструктуры

Testcontainers – container per class



Testcontainers – container per class



■ Старт контейнера ■ Миграции ■ Сброс БД

Testcontainers – container per class

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~5.3 сек

инфраструктуры
(в среднем на тест)

время на тест:

~536 мс

старт контейнера
(в среднем на тест)

~4733 мс

117 миграций:

Testcontainers – container per class

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~5.3 сек

инфраструктуры
(в среднем на тест)

время на тест:

~536 мс

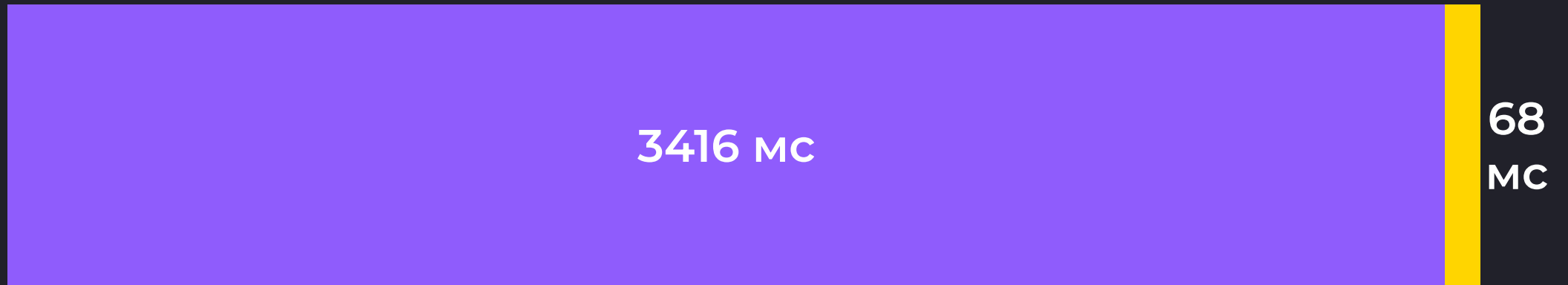
старт контейнера
(в среднем на тест)

~4733 мс

117 миграций:

на 2340 тестов ~ 26 минут
инфраструктуры

Testcontainers – container per run



■ Старт контейнера ■ Миграции ■ Сброс БД

Testcontainers – container per run

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~3.5 сек

в среднем на тест

время на тест:

~3 мс

старт контейнера
(на тест)

~3416 мс

117 миграций:

Testcontainers – container per run

2340

всего тестов
выполняется (12 тест
классов, 8 потоков)

~3.5 сек

в среднем на тест

время на тест:

~3 мс

старт контейнера
(на тест)

~3416 мс

117 миграций:

на 2340 тестов ~ 17 минут

Быстрее уже не будет...



Linkin Park — "In the End" (2001) © Warner Bros. Records / Machine Shop Recordings Пер. Joseph Kahn

Или бюджет?



"Шрек" (2001), © DreamWorks Animation, источник: giphy.com



IntegreSQL

  allaboutapps / integresql

<> Code

Issues 7

Pull requests 2

Agents

Actions

Projects

Security and quality

Insights

On April 24 we'll start using GitHub Copilot interaction data for AI model training unless you opt out. [Review this update](#) and manage your preferences in your [GitHub account settings](#).

 integresql Public

Watch 12

Fork 16

Star 799

master

6 Branches

3 Tags

Go to file

Add file

Code

 majodev Merge pull request #16 from allaboutapps/mr/v1.1.0 9e395cd · 2 years ago 201 Commits

.devcontainer	fix toolchain, reenale pipefails, fix linting, cleanup	3 years ago
.github/workflows	add github actions multiarch build and publish	2 years ago
.vscode	remove unused utils	3 years ago
cmd/server	intro zerolog and add proper log statements in pool and ma...	3 years ago
docs	rm no longer needed extra md docs	2 years ago
internal	WIP docs and changelog, generic timeout handling	2 years ago
pkg	WIP docs and changelog, generic timeout handling	2 years ago
tests	bump deps	3 years ago
tmp	add tmp directory	3 years ago
.dockerignore	WIP docs and changelog, generic timeout handling	2 years ago
.drone.yml	fix toolchain, reenale pipefails, fix linting, cleanup	3 years ago
.gitignore	add tmp directory	3 years ago
.golangci.yml	fix toolchain, reenale pipefails, fix linting, cleanup	3 years ago

About

IntegreSQL manages isolated PostgreSQL databases for your integration tests.

go

golang

postgres

server

integration-testing

postgresql

id-allaboutapps-backend

Readme

MIT license

Activity

Custom properties

799 stars

12 watching

16 forks

Report repository

Releases 1

v1.1.0

Latest

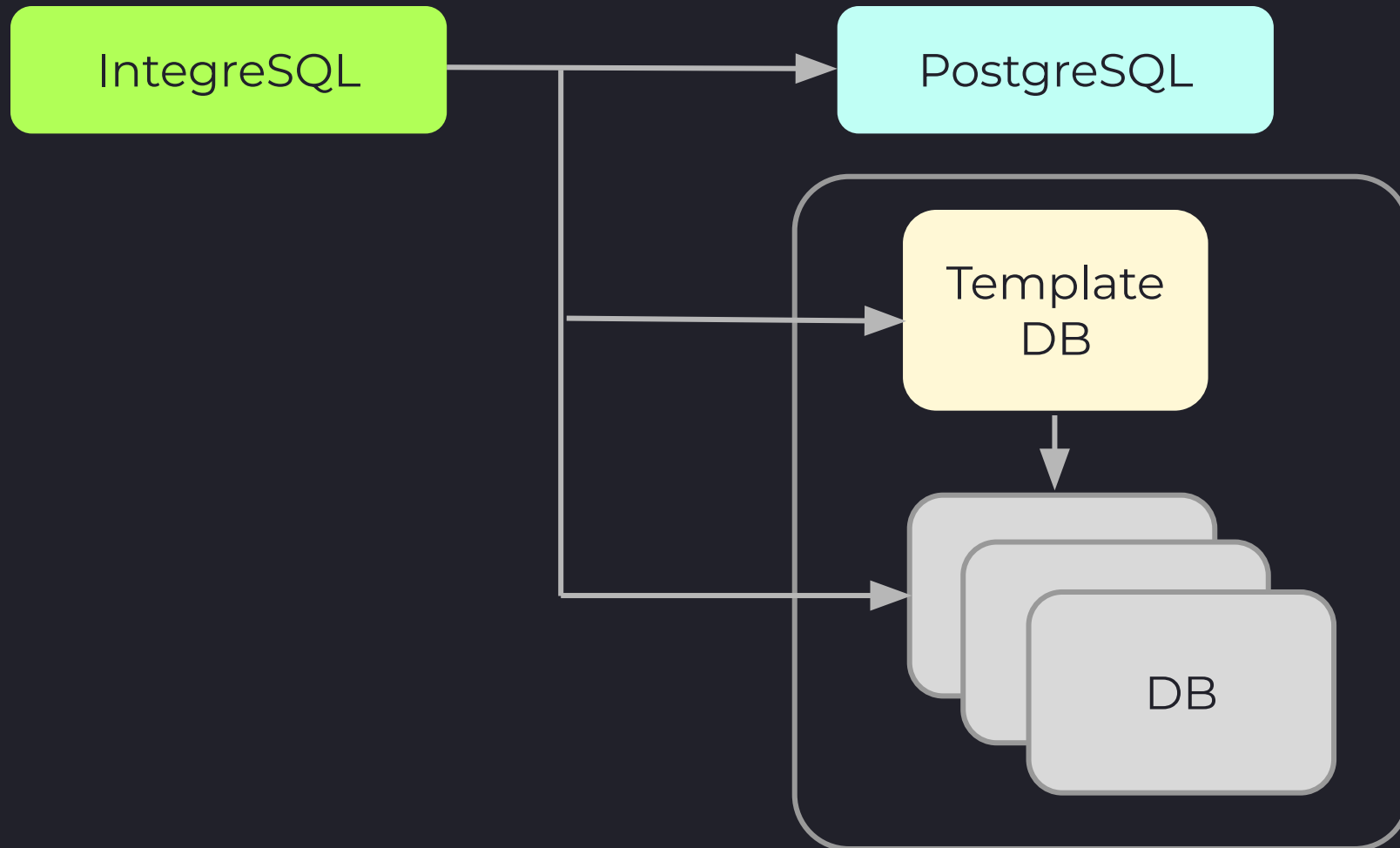
on Jan 31, 2024

Packages 1

IntegreSQL

- Server with REST API
- Docker container
- PostgreSQL
- Clients:
 - Go: [integresql-client-go](#) by [Nick Müller - @MorpheusXAUT](#)
 - Python: [integresql-client-python](#) by [Marcin Sztolcman - @msztolcman](#)
 - .NET: [IntegreSQL.EF](#) by [Artur Drobinskiy - @Shaddix](#)
 - JavaScript/TypeScript: [@devoxa/integresql-client](#) by [Devoxa - @devoxa](#)

IntegreSQL



Postgres TEMPLATE

-- 1. Создать базу

```
CREATE DATABASE tmp_db;
```

-- ... применить миграции к tmp_db ...

-- 2. Закрыть все соединения и запретить новые

```
UPDATE pg_database SET dataallowconn = false WHERE datname = 'tmp_db';
```

-- 3. Пометить как шаблон

```
UPDATE pg_database SET datistemplate = true WHERE datname = 'tmp_db';
```

-- 4. Создание из шаблона

```
CREATE DATABASE testdb_001 TEMPLATE tmp_db;
```

Что происходит при создании?

```
psql test_db
```

Что происходит при создании?

```
psql test_db
```

```
migration_001.sql
```

```
migration_002.sql
```

```
...
```

```
migration_200.sql
```

Что происходит при создании?

7

Парсинг, анализ
и планирование каждого
SQL стејтмента

Что происходит при создании?

1

Парсинг, анализ
и планирование каждого
SQL стейтмента

2

Вставки в каталоги:
pg_class, pg_attribute,
pg_type, pg_constraint,
pg_index, ...

Что происходит при создании?

1

Парсинг, анализ
и планирование каждого
SQL стейтмента

2

Вставки в каталоги:
pg_class, pg_attribute,
pg_type, pg_constraint,
pg_index, ...

3

Построение индексов:
скан+сортировка+
запись Btree страниц

Что происходит при создании?

1

Парсинг, анализ
и планирование каждого
SQL стейтмента

2

Вставки в каталоги:
pg_class, pg_attribute,
pg_type, pg_constraint,
pg_index, ...

3

Построение индексов:
скан+сортировка+
запись Btree страниц

4

Проверка значений PK,
Unique, FK, CHECK

Что происходит при создании?

1

Парсинг, анализ
и планирование каждого
SQL стейтмента

2

Вставки в каталоги:
pg_class, pg_attribute,
pg_type, pg_constraint,
pg_index, ...

3

Построение индексов:
скан+сортировка+
запись Btree страниц

4

Проверка значений PK,
Unique, FK, CHECK

5

Срабатывание триггеров
на вставках

Что происходит при создании?

1

Парсинг, анализ
и планирование каждого
SQL стейтмента

2

Вставки в каталоги:
pg_class, pg_attribute,
pg_type, pg_constraint,
pg_index, ...

3

Построение индексов:
скан+сортировка+
запись Btree страниц

4

Проверка значений РК,
Unique, FK, CHECK

5

Срабатывание триггеров
на вставках

6

Блокировки...

Итог



Сотни/тысячи изменений
системных каталогов БД



Кэши, индексы, проверки
на целостность



Каждый раз на каждую
БД для каждого теста

Что делает CREATE TEMPLATE?

- base/
- — 16384
- — 16385
- — 16386

Что делает CREATE TEMPLATE?

- base/
- — 16384 - файлы БД 1
- — 16385 - файлы БД 2
- — 16386

Что делает CREATE TEMPLATE?

1

Создает новый OID

2

Создает новый
каталог базы

3

Копирует файлы
шаблонной БД

4

Регистрирует
новую БД в pg_database

Template strategy – FILE_COPY

Копируются каталоги целиком на уровне файловой системы

Требуется checkpoint до и после операции

Template strategy – WAL_LOG

Страницы копируются и пишутся в WAL

Template strategy – WAL_LOG

template_db



page 1

page 2

page 3

...

Template strategy – WAL_LOG

template_db -----> db_copy

↓

↓

page 1 -----> page1

page 2 -----> page2

page 3 -----> page3

...

Где какая лучше?

На больших базах данных FILE_COPY лучше себя показывает

WAL лучше на небольших

Ограничения

Ограничения



Нельзя иметь активные
подключения к template

Ограничения



Нельзя иметь активные подключения к `template`



Базы данных не связаны – после копирования, добавленные в `template` данные не перенесутся в копию

Ограничения



Нельзя иметь активные подключения к template



Базы данных не связаны – после копирования, добавленные в template данные не перенесутся в копию



Нельзя выполнить копирование внутри транзакции

Ограничения



Нельзя иметь активные подключения к template



Базы данных не связаны – после копирования, добавленные в template данные не перенесутся в копию



Нельзя выполнить копирование внутри транзакции



Некоторые расширения или внешние компоненты могут хранить состояние вне базы

Ограничения



Нельзя иметь активные подключения к template



Базы данных не связаны – после копирования, добавленные в template данные не перенесутся в копию



Нельзя выполнить копирование внутри транзакции



Некоторые расширения или внешние компоненты могут хранить состояние вне базы



Нет инкрементального клонирования (обновления после первого копирования нельзя «довезти»)

Ограничения



Нельзя иметь активные подключения к template



Базы данных не связаны – после копирования, добавленные в template данные не перенесутся в копию



Нельзя выполнить копирование внутри транзакции



Некоторые расширения или внешние компоненты могут хранить состояние вне базы



Нет инкрементального клонирования (обновления после первого копирования нельзя «довезти»)



Не копируются GRANT права оригинальной БД, права будут дефолтные

IntegreSQL

Name	Container ID	Image
● testcontainers-ryuk-3979fceb-670a-4350-9312-9933b8761881	7240d7140390	testcontainers/ryuk:0.9.0
● tender_euler	84acfc730f42	postgres:16-alpine
● friendly_bhabha	72779fe8a3ba	allaboutapps/integresql:latest

IntegreSQL

Приложение

TestRunner

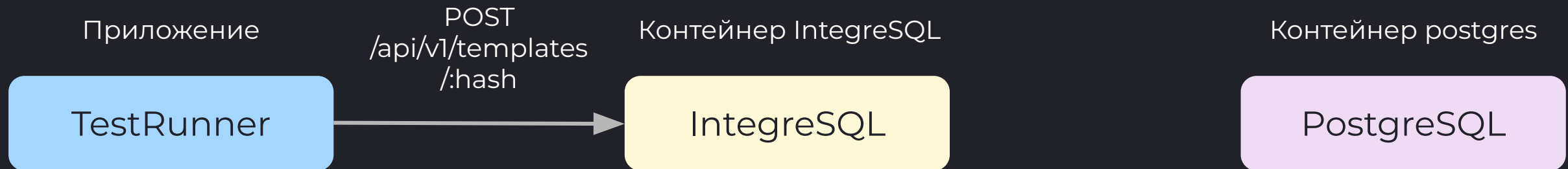
Контейнер IntegreSQL

IntegreSQL

Контейнер postgres

PostgreSQL

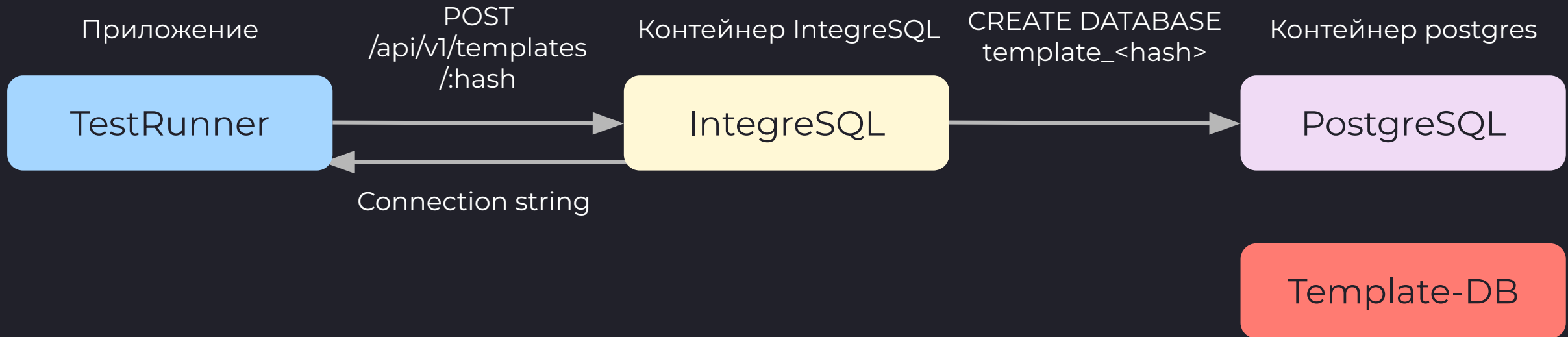
IntegreSQL



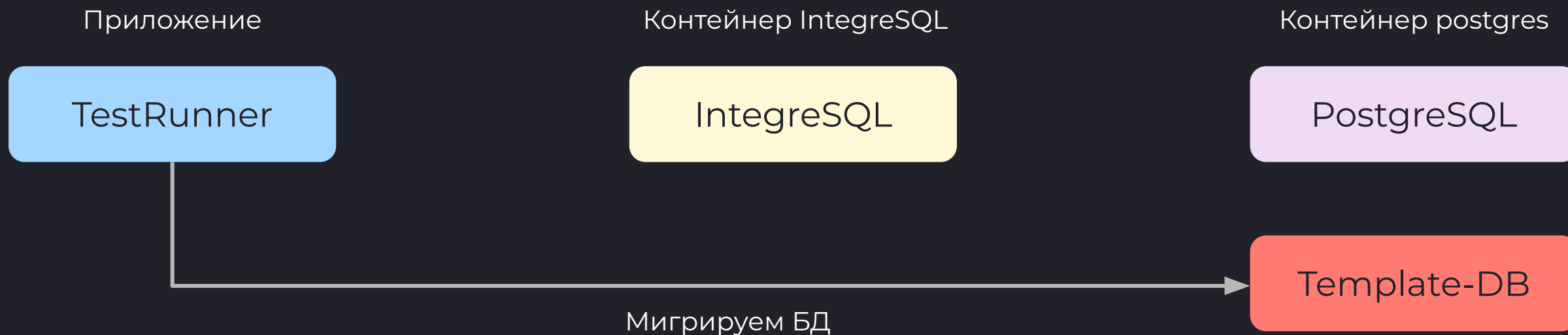
IntegreSQL



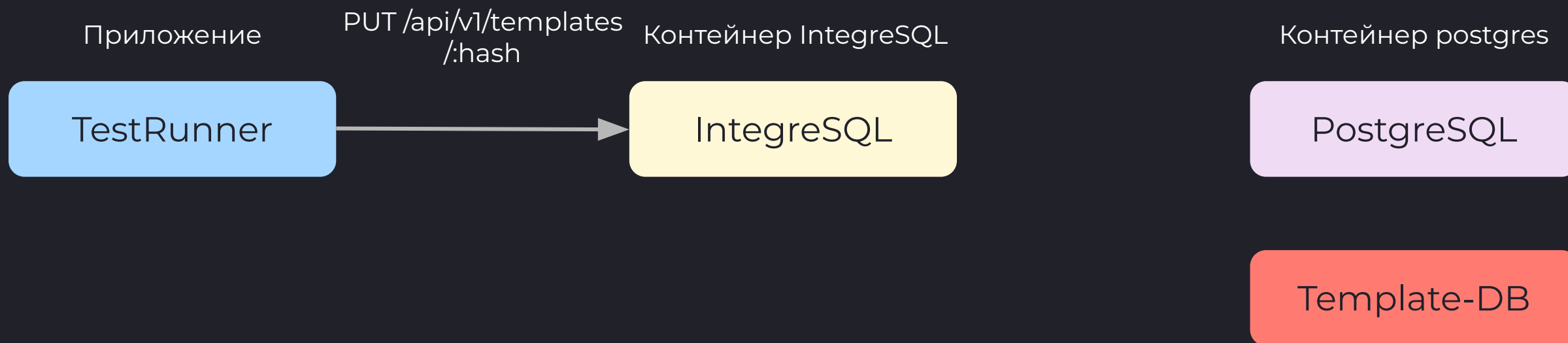
IntegreSQL



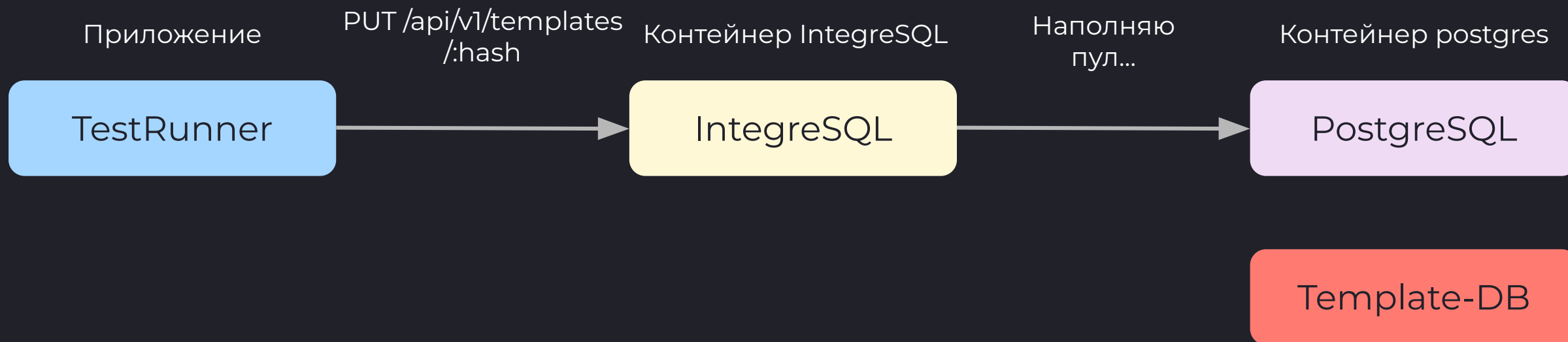
IntegreSQL



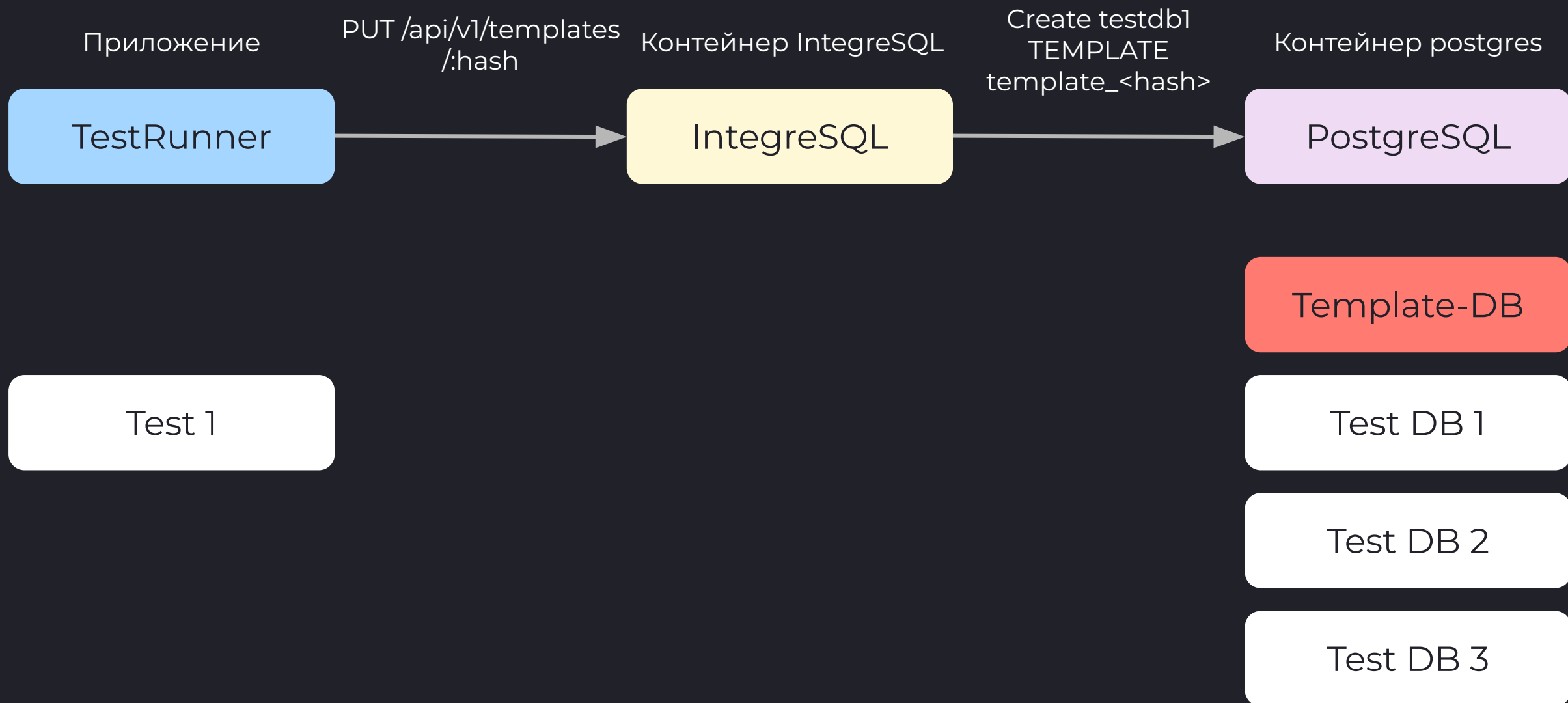
IntegreSQL



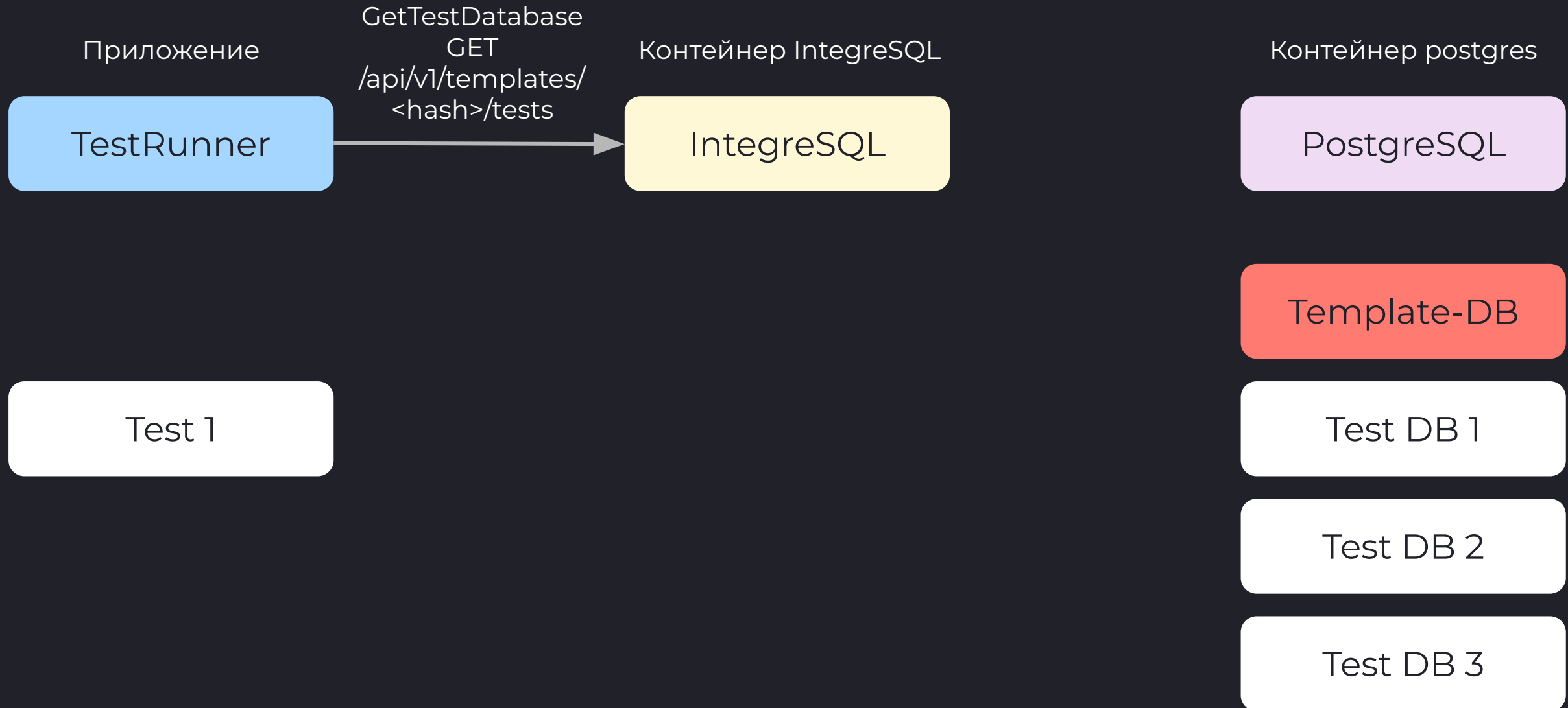
IntegreSQL



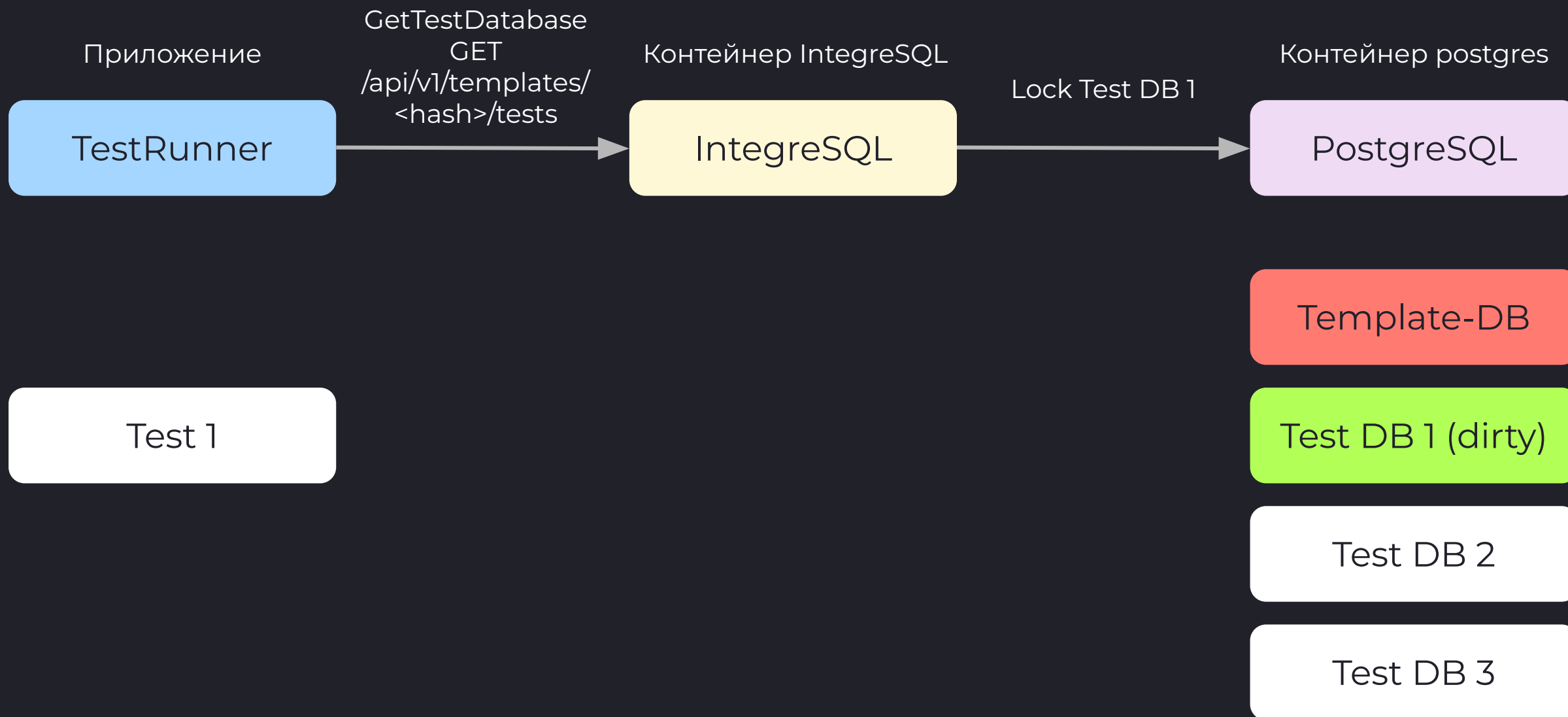
IntegreSQL



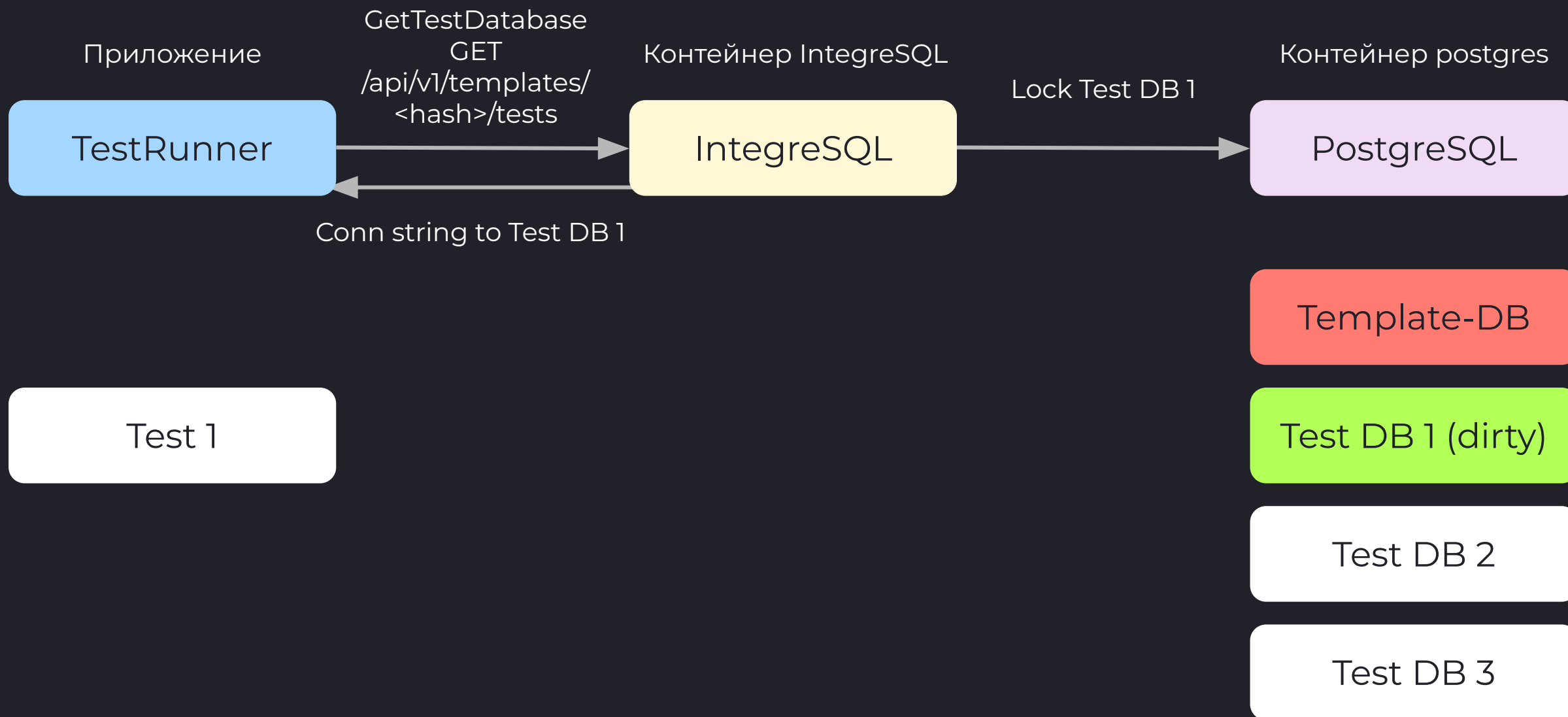
IntegreSQL



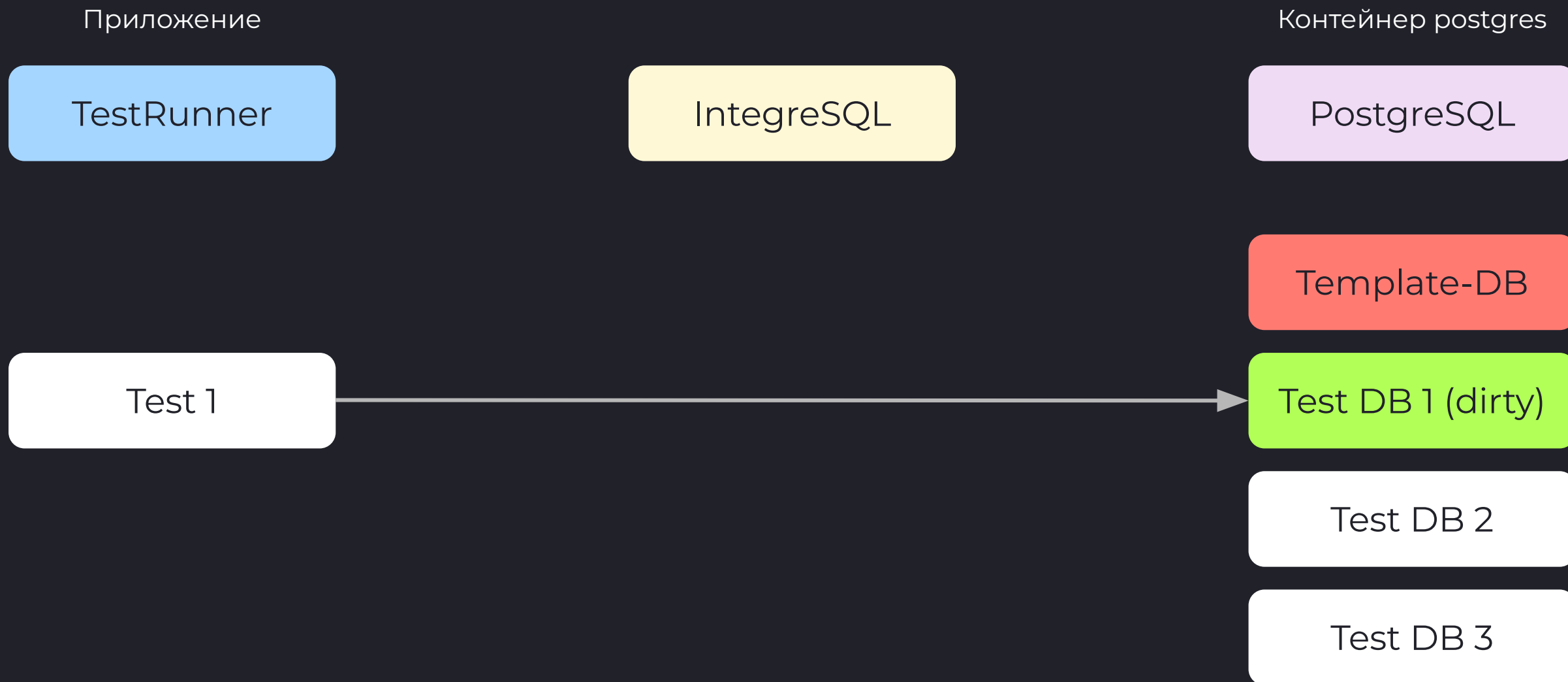
IntegreSQL



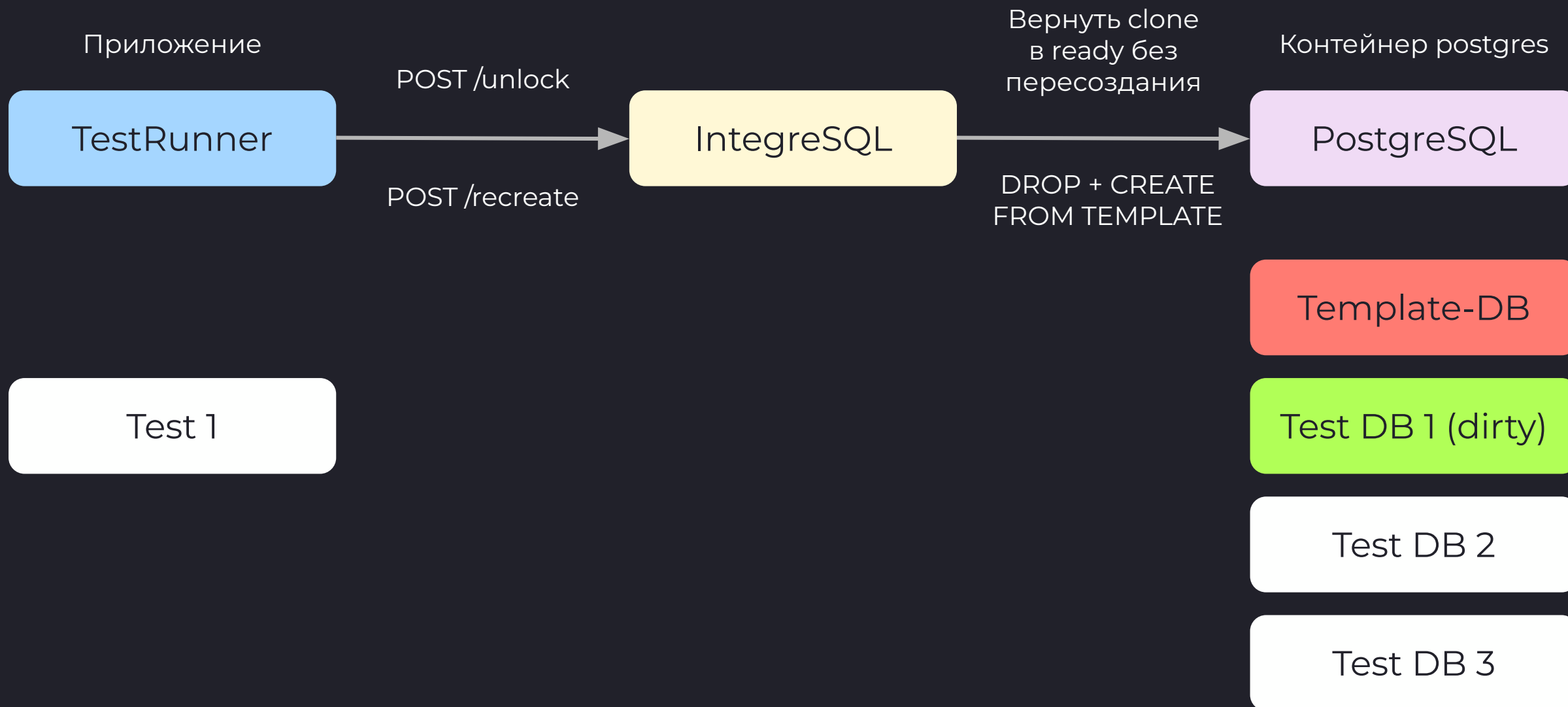
IntegreSQL



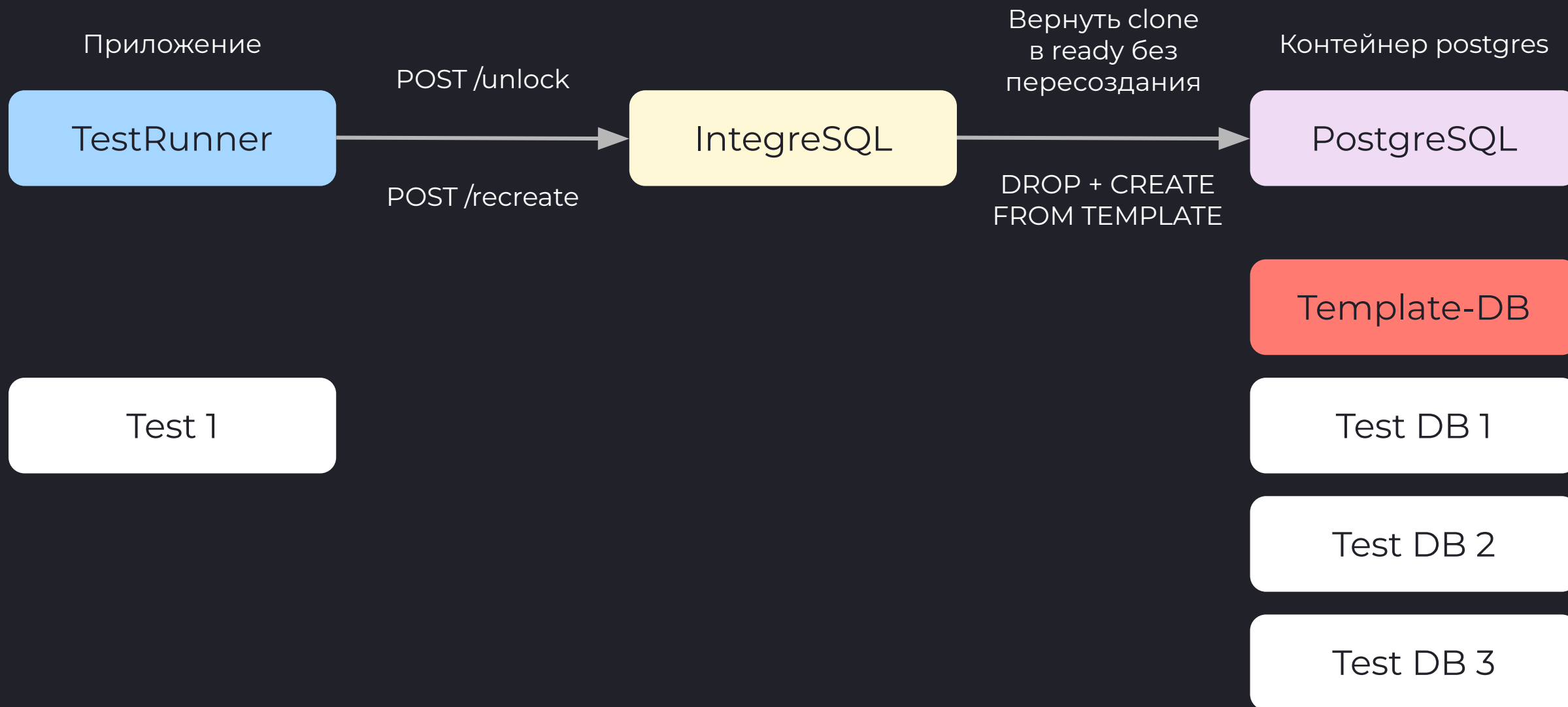
IntegreSQL



IntegreSQL



IntegreSQL



IntegreSQL – код

```
public abstract class AppServiceTestBase : IAsyncLifetime
{
    private NpgsqlDatabaseInitializer _initializer = null!;
    private string _connectionString = null!;
    protected ShopDbContext Context { get; private set; } = null!;

    public async Task InitializeAsync()
    {
        _initializer = await IntegresSqlManager.GetAsync();

        // GET /api/v1/templates/:hash/tests – клон из пула (~50 мс)
        _connectionString = await _initializer
            .CreateDatabaseGetConnectionString<ShopDbContext>(SeedingOptions);

        var options = new DbContextOptionsBuilder<ShopDbContext>()
            .UseNpgsql(_connectionString).Options;
        Context = new ShopDbContext(options);
    }

    public async Task DisposeAsync()
    {
        await Context.DisposeAsync();

        // POST /api/v1/templates/:hash/tests/:id/recreate (DropDatabaseOnRemove=true)
        await _initializer.RemoveDatabase(_connectionString);
    }
}
```

IntegrerSQL - App level

Test
Infrastructure

InitializeAsync

Test

DisposeAsync

IntegreSQL – код

```
public abstract class AppServiceTestBase : IAsyncLifetime
{
    private NpgsqlDatabaseInitializer _initializer = null!;
    private string _connectionString = null!;
    protected ShopDbContext Context { get; private set; } = null!;

    public async Task InitializeAsync()
    {
        _initializer = await IntegresSqlManager.GetAsync();

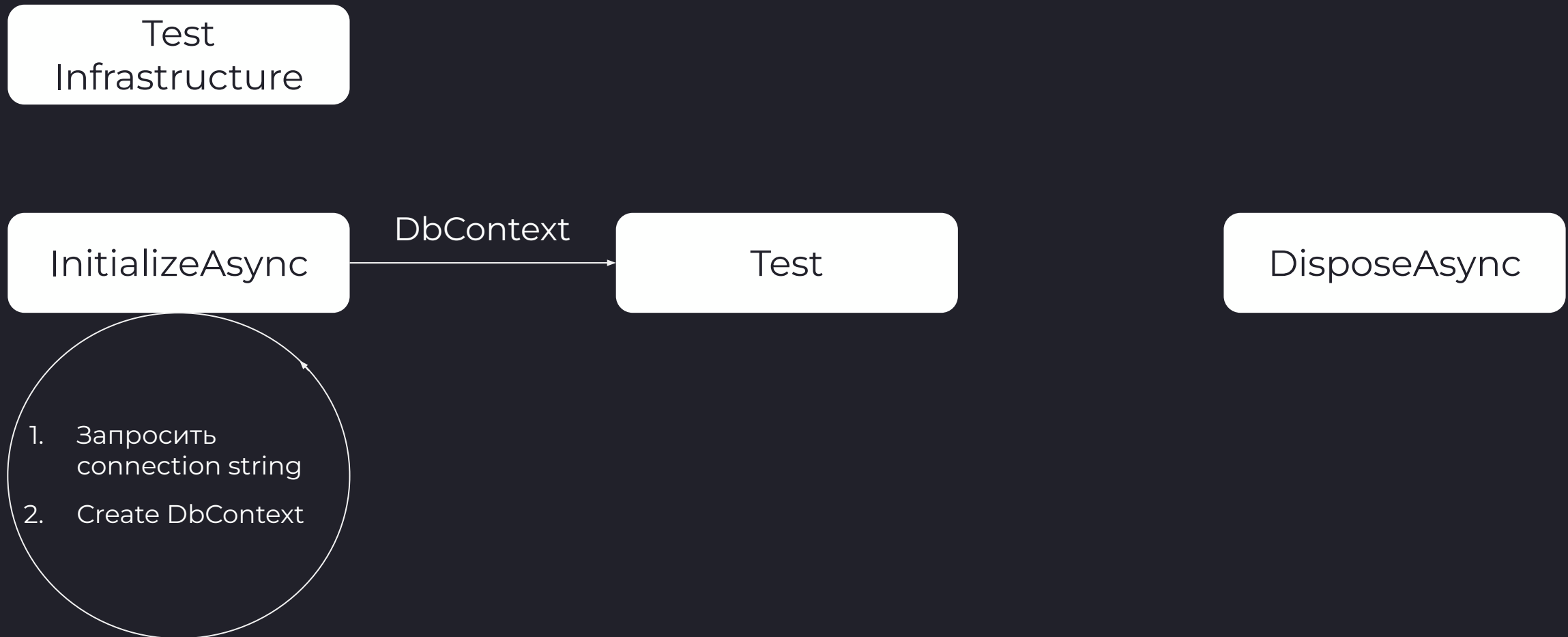
        // GET /api/v1/templates/:hash/tests – клон из пула (~50 мс)
        _connectionString = await _initializer
            .CreateDatabaseGetConnectionString<ShopDbContext>(SeedingOptions);

        var options = new DbContextOptionsBuilder<ShopDbContext>()
            .UseNpgsql(_connectionString).Options;
        Context = new ShopDbContext(options);
    }

    public async Task DisposeAsync()
    {
        await Context.DisposeAsync();

        // POST /api/v1/templates/:hash/tests/:id/recreate (DropDatabaseOnRemove=true)
        await _initializer.RemoveDatabase(_connectionString);
    }
}
```

IntegrerSQL - App level



IntegreSQL – код

```
public abstract class AppServiceTestBase : IAsyncLifetime
{
    private NpgsqlDatabaseInitializer _initializer = null!;
    private string _connectionString = null!;
    protected ShopDbContext Context { get; private set; } = null!;

    public async Task InitializeAsync()
    {
        _initializer = await IntegresSqlManager.GetAsync();

        // GET /api/v1/templates/:hash/tests – клон из пула (~50 мс)
        _connectionString = await _initializer
            .CreateDatabaseGetConnectionString<ShopDbContext>(SeedingOptions);

        var options = new DbContextOptionsBuilder<ShopDbContext>()
            .UseNpgsql(_connectionString).Options;
        Context = new ShopDbContext(options);
    }

    public async Task DisposeAsync()
    {
        await Context.DisposeAsync();

        // POST /api/v1/templates/:hash/tests/:id/recreate (DropDatabaseOnRemove=true)
        await _initializer.RemoveDatabase(_connectionString);
    }
}
```

IntegreSQL - App level



IntegrerSQL – код

```
public class ProductServiceTests : AppServiceTestBase
{
    [Fact]
    public async Task Add_NewProduct_PersistsAndReturnsId()
    {
        var service = new ProductService(new ProductRepository(Context));

        var id = await service.AddAsync(new ProductDto { Name = "Phone", Price = 100 });

        var saved = await Context.Products.FindAsync(id);
        Assert.Equal("Phone", saved!.Name);
    }
}
```

Бенчмарк

Бенчмарк – сценарии измерений

- **Кол-во тестов** – как растёт время с числом тестов

Бенчмарк – сценарии измерений

- **Кол-во тестов** – как растёт время с числом тестов
- **Кол-во миграций** – как дорожает изоляция с ростом схемы и времени миграции

Бенчмарк – сценарии измерений

- **Кол-во тестов** – как растёт время с числом тестов
- **Кол-во миграций** – как дорожает изоляция с ростом схемы и времени миграции
- **Параллелизм** – какой прирост производительности дает параллельное выполнение

Бенчмарк

- .NET Core 8 Web Api
- 7 сущностей в БД - Categories, Customers, Discounts, Orders, Products, Reviews, Suppliers
- Тесты с DbContext напрямую в тестируемом сервисе и интеграционные тесты с полным хостом
- 195 базовых тестов на сервис
- 17 реальных и 100 сгенерированных миграций

Бенчмарк – сценарии тестов

```
[Fact]
public async Task GetAll_WhenOrdersExist_Returns200WithOrders()
{
    await CreateOrderWithProductAsync();
    await CreateOrderWithProductAsync();

    var response = await Client.GetAsync("/api/orders");
    var orders = await response.Content.ReadFromJsonAsync<List<OrderDto>>();

    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
    Assert.Equal(2, orders!.Count);
}
```

Бенчмарк – сценарии тестов

```
/// <summary>
/// Создаёт категорию, обновляет через PUT, проверяет GET, удаляет – полный HTTP-цикл.
/// </summary>
[Fact]
public async Task CreateUpdateDelete_VerifyEachStep_AllPersist()
{
    var created = await CreateCategoryAsync("Спорт");

    var putResp = await Client.PutAsJsonAsync($"/api/categories/{created.Id}",
        new UpdateCategoryRequest { Name = "Спорт и фитнес", Description = "Обновлено" });
    Assert.Equal(HttpStatusCode.OK, putResp.StatusCode);
    var updated = await putResp.Content.ReadFromJsonAsync<CategoryDto>();
    Assert.Equal("Спорт и фитнес", updated!.Name);

    var getResp = await Client.GetAsync($"/api/categories/{created.Id}");
    var fetched = await getResp.Content.ReadFromJsonAsync<CategoryDto>();
    Assert.Equal("Спорт и фитнес", fetched!.Name);

    var delResp = await Client.DeleteAsync($"/api/categories/{created.Id}");
    Assert.Equal(HttpStatusCode.NoContent, delResp.StatusCode);
    Assert.Equal(HttpStatusCode.NotFound, (await Client.GetAsync($"/api/categories/{created.Id}")).StatusCode);

    // benchmark: искусственное увеличение продолжительности теста и объёма работы с БД
    for (var i = 0; i < 4; i++)
    {
        var extra = await CreateCategoryAsync($"Доп кат {i}");
        await Client.PutAsJsonAsync($"/api/categories/{extra.Id}",
            new UpdateCategoryRequest { Name = $"Доп кат {i} v2" });
        await Client.GetAsync($"/api/categories/{extra.Id}");
    }
    await Client.GetAsync("/api/categories");
}
```

Бенчмарк – множитель тестов

Бенчмарк – множитель тестов

- ScaleManager в бенчмарке находит пути до необходимых тестов
- По заданному **scaleFactor** копирует исходные тесты необходимое количество раз меняя имя и наследуя оригинальные тесты

```
public class ProductServiceTests_2 : ProductServiceTests { ... }  
public class ProductServiceTests_3 : ProductServiceTests { ... }  
public class ProductServiceTests_4 : ProductServiceTests { ... }  
public class ProductServiceTests_5 : ProductServiceTests { ... }
```

Бенчмарк – миграции

17 обычных миграций, которые добавляют вышеупомянутые сущности (в некоторых правда происходит вайбкод)

Migrations	
20260415121117_InitialCreate.cs	
20260415121117_InitialCreate.Designer.cs	
20260416000001_Add_IsAvailable_To_Products.cs	
20260416000002_Add_Sku_To_Products.cs	
20260416000003_Add_Notes_And_Weight.cs	
20260416000004_Add_Performance_Indexes.cs	
20260416000005_Add_ExternalRefs_To_Orders.cs	
20260416000006_Create_Categories.cs	
20260416000007_Create_Tags_System.cs	
20260416000008_Create_Customers_And_Addresses.cs	
20260416000009_Create_Coupons.cs	
20260416000010_Create_ProductReviews_And_Helpfulness.cs	
20260416000011_Create_PriceHistory_With_Trigger.cs	
20260416000012_Create_AuditLog_With_Triggers.cs	
20260416000013_Create_Inventory_System.cs	
20260416000014_Add_FullTextSearch_To_Products.cs	
20260416000015_Create_OrderStatistics_Materialized_View.cs	
20260421152746_AddCategoryCustomerSupplierReviewDisc...	
20260421152746_AddCategoryCustomerSupplierReviewDisc...	

Бенчмарк – миграции для нагрузки

100 сгенерированных парных
миграций для симуляции линейного
роста времени выполнения
миграций

- 20990101000001_AddShippingRates.cs
- 20990101000001_AddShippingRates.Designer.cs
- 20990101000002_DropShippingRates.cs
- 20990101000002_DropShippingRates.Designer.cs
- 20990101000003_CreateProductVariants.cs
- 20990101000003_CreateProductVariants.Designer.cs
- 20990101000004_DropProductVariants.cs
- 20990101000004_DropProductVariants.Designer.cs
- 20990101000005_AddLoyaltyPoints.cs
- 20990101000005_AddLoyaltyPoints.Designer.cs
- 20990101000006_DropLoyaltyPoints.cs
- 20990101000006_DropLoyaltyPoints.Designer.cs
- 20990101000007_CreateWarehouseSlots.cs
- 20990101000007_CreateWarehouseSlots.Designer.cs
- 20990101000008_DropWarehouseSlots.cs
- 20990101000008_DropWarehouseSlots.Designer.cs
- 20990101000009_AddReturnRequests.cs
- 20990101000009_AddReturnRequests.Designer.cs
- 20990101000010_DropReturnRequests.cs
- 20990101000010_DropReturnRequests.Designer.cs

Бенчмарк – миграции для нагрузки

```
namespace FastIntegrationTests.Infrastructure.Migrations
{
    /// <inheritdoc />
    public partial class AddShippingRates : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"CREATE TABLE load_tmp_001 (
id          SERIAL          PRIMARY KEY,
code        VARCHAR(20)     NOT NULL,
name        VARCHAR(100)    NOT NULL,
created_at  TIMESTAMP       NOT NULL DEFAULT NOW()
);
INSERT INTO load_tmp_001 (code, name)
SELECT 'CODE_' || gs, 'Value ' || gs
FROM generate_series(1, 300) gs;");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"DROP TABLE IF EXISTS load_tmp_001;");
        }
    }
}
```

Бенчмарк – миграции для нагрузки

```
namespace FastIntegrationTests.Infrastructure.Migrations
{
    /// <inheritdoc />
    public partial class AddShippingRates : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"CREATE TABLE load_tmp_001 (
id            SERIAL          PRIMARY KEY,
code          VARCHAR(20)     NOT NULL,
name          VARCHAR(100)    NOT NULL,
created_at    TIMESTAMP       NOT NULL DEFAULT NOW()
);
INSERT INTO load_tmp_001 (code, name)
SELECT 'CODE_' || gs, 'Value ' || gs
FROM generate_series(1, 300) gs;");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"DROP TABLE IF EXISTS load_tmp_001;");
        }
    }
}
```

Бенчмарк – миграции для нагрузки

```
namespace FastIntegrationTests.Infrastructure.Migrations
{
    /// <inheritdoc />
    public partial class AddShippingRates : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"CREATE TABLE load_tmp_001 (
id            SERIAL            PRIMARY KEY,
code          VARCHAR(20)      NOT NULL,
name          VARCHAR(100)     NOT NULL,
created_at    TIMESTAMP        NOT NULL DEFAULT NOW()
);
INSERT INTO load_tmp_001 (code, name)
SELECT 'CODE_' || gs, 'Value ' || gs
FROM generate_series(1, 300) gs;");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"DROP TABLE IF EXISTS load_tmp_001;");
        }
    }
}
```

Бенчмарк – миграции для нагрузки

```
namespace FastIntegrationTests.Infrastructure.Migrations
{
    /// <inheritdoc />
    public partial class DropShippingRates : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql(
                @"DROP TABLE IF EXISTS load_tmp_001;");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
        }
    }
}
```

Бенчмарк – миграции для нагрузки

Целевое N	Прячем	Остаётся
17	100	17 реальных
42	75	17 + 25
67	50	17 + 50
92	25	17 + 75
117	0	всё

Бенчмарк – параллелизм

Параллелизм реализуется просто

```
-- xUnit.MaxParallelThreads={scenario.MaxParallelThreads}"
```

Бенчмарк – прогрев

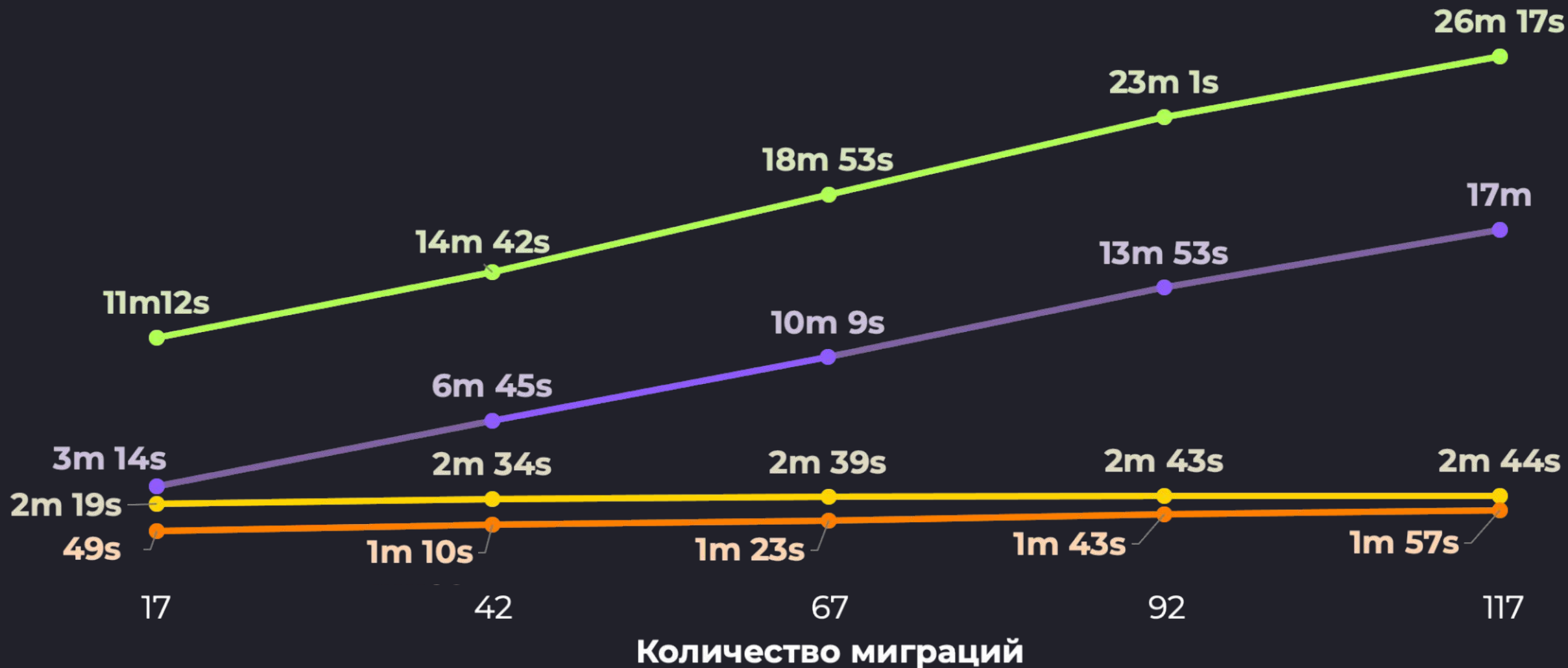
Прогреваем Docker простым dotnet test запуском, тянем все необходимые образы и один раз все поднимаем

Бенчмарк – cooldown

Между прогонами нужен обязательно `Task.Delay`, чтобы докер успел подчистить все созданные сети и освободить порты, на большом количестве контейнеров это становится проблемой

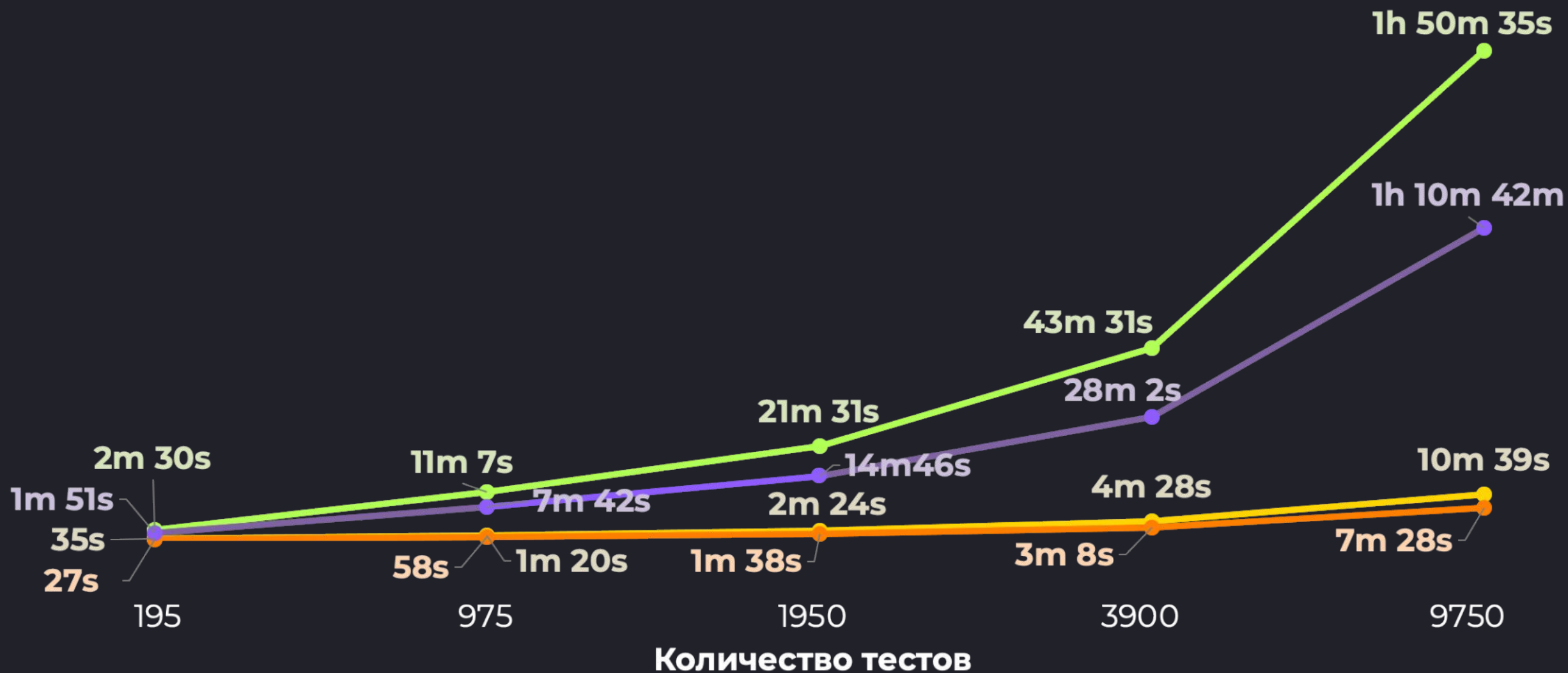
Бенчмарк – кол-во миграций

IntegreSQL Respawn Тест контейнер на класс Один тест контейнер на весь запуск



Бенчмарк – кол-во тестов

● IntegreSQL ● Respawn ● Тест контейнер на класс ● Один тест контейнер на весь запуск



Бенчмарк – параллелизм

IntegreSQL Respawn Тест контейнер на класс Один тест контейнер на весь запуск



Respawn

☰

 jbogard / Respawn

 ▾

 ▾







<> Code

🕒 Issues 11

🔗 Pull requests 3

🛠 Agents

🔄 Actions

📁 Projects

📖 Wiki

🛡 Security and quality

📊 Insights

🔔

On April 24 we'll start using GitHub Copilot interaction data for AI model training unless you opt out. [Review this update](#) and manage your preferences in your [GitHub account settings](#).

 Respawn

Public

 Watch 42 ▾

 Fork 153 ▾

 Star 3k ▾

main ▾

🔗 4 Branches

🏷 19 Tags

T

Add file ▾

<> Code ▾

 jbogard

Merge pull request #163 from 0xcd/Testcontainers-Fixture

🗨

✓

f550203 · 3 months ago

🕒 253 Commits

📁 .github/workflows	Add test reports to GitHub actions	3 months ago
📁 .nuget	0.2	10 years ago
📁 Respawn.DatabaseTests	Run MySQLTests on CI	3 months ago
📁 Respawn.UnitTests	Run Respawn.UnitTests on .NET 8 instead of .NET 6	last year
📁 Respawn	Checkpoint	5 months ago
📁 informix-server	Adding a message when the database is empty	3 years ago
📄 .gitattributes	Respawn with SQL Server and Postgres support	12 years ago
📄 .gitignore	Add IBM DB2 DbAdapter,	2 years ago
📄 Build.ps1	Add test reports to GitHub actions	3 months ago
📄 Directory.Build.props	C# 12	5 months ago
📄 Directory.Build.targets	Add test reports to GitHub actions	3 months ago
📄 LICENSE	Initial commit	12 years ago

About

Intelligent database cleaner for integration tests

📖 Readme

📄 Apache-2.0 license

📈 Activity

☆ 3k stars

👁 42 watching

🔗 153 forks

Report repository

Releases 10

🔗 v7.0.0

Latest

on Dec 1, 2025

[+ 9 releases](#)

Packages

No packages published

Contributors 31

Respawn

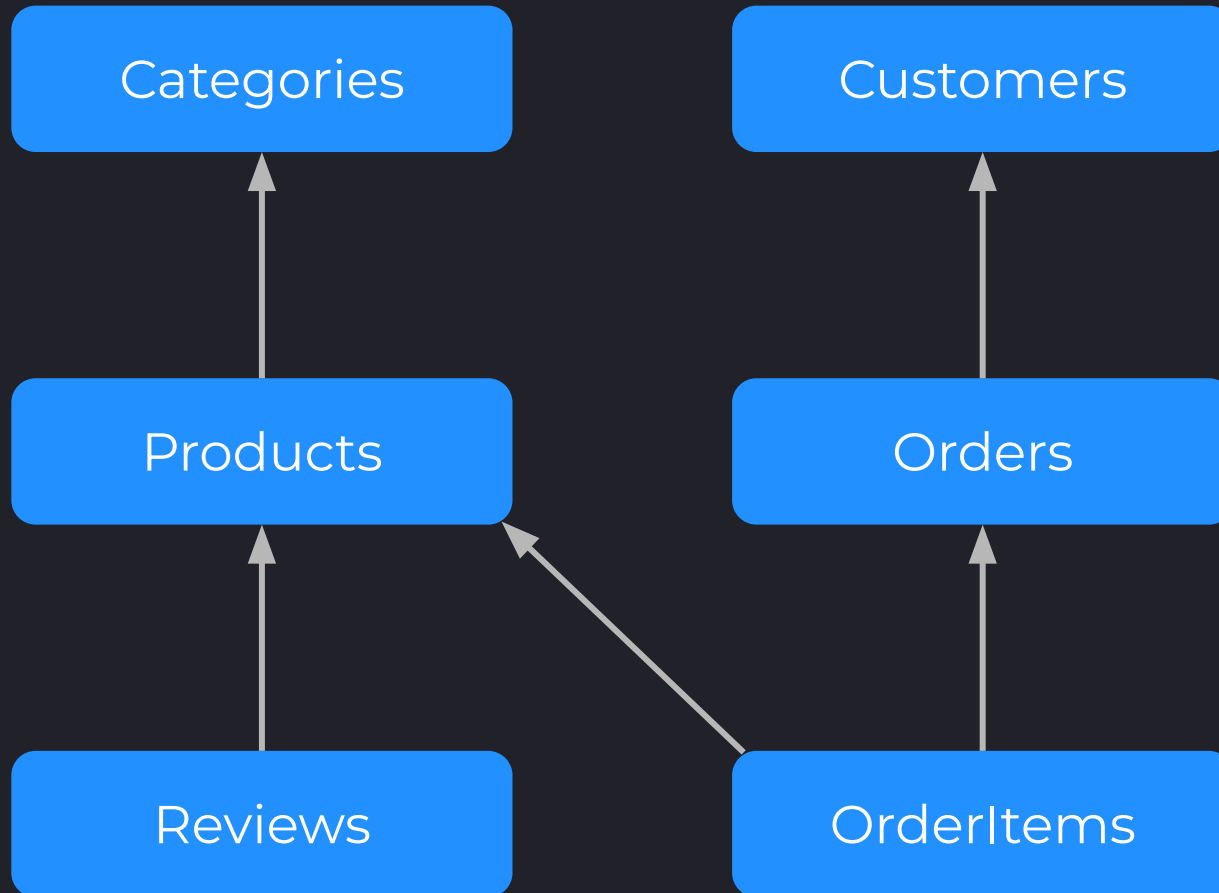
- .NET
- NuGet
- 3K stars
- Last release: 1 dec, 2025
- PostgreSQL, mssql, oracle, mysql, informix, db2...

Respawn

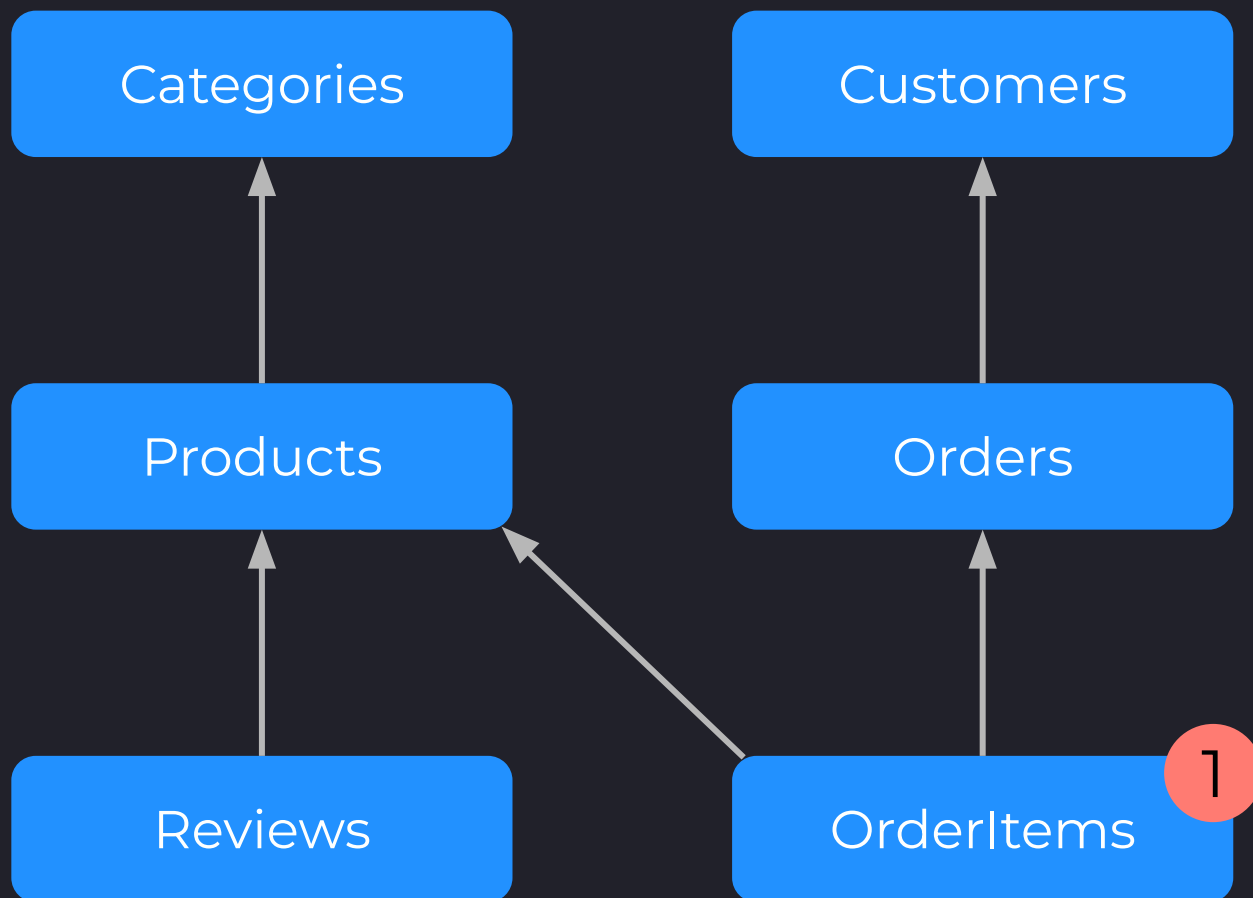
- Быстрая очистка
- При инициализации строится топологически отсортированный граф FK зависимостей – сначала дочерние таблицы, потом родительские
- Кэширует итоговый SQL для сброса:

```
DELETE FROM "public"."OrderItems";  
DELETE FROM "public"."Orders";  
DELETE FROM "public"."Products";  
DELETE FROM "public"."Categories";
```

Respawn

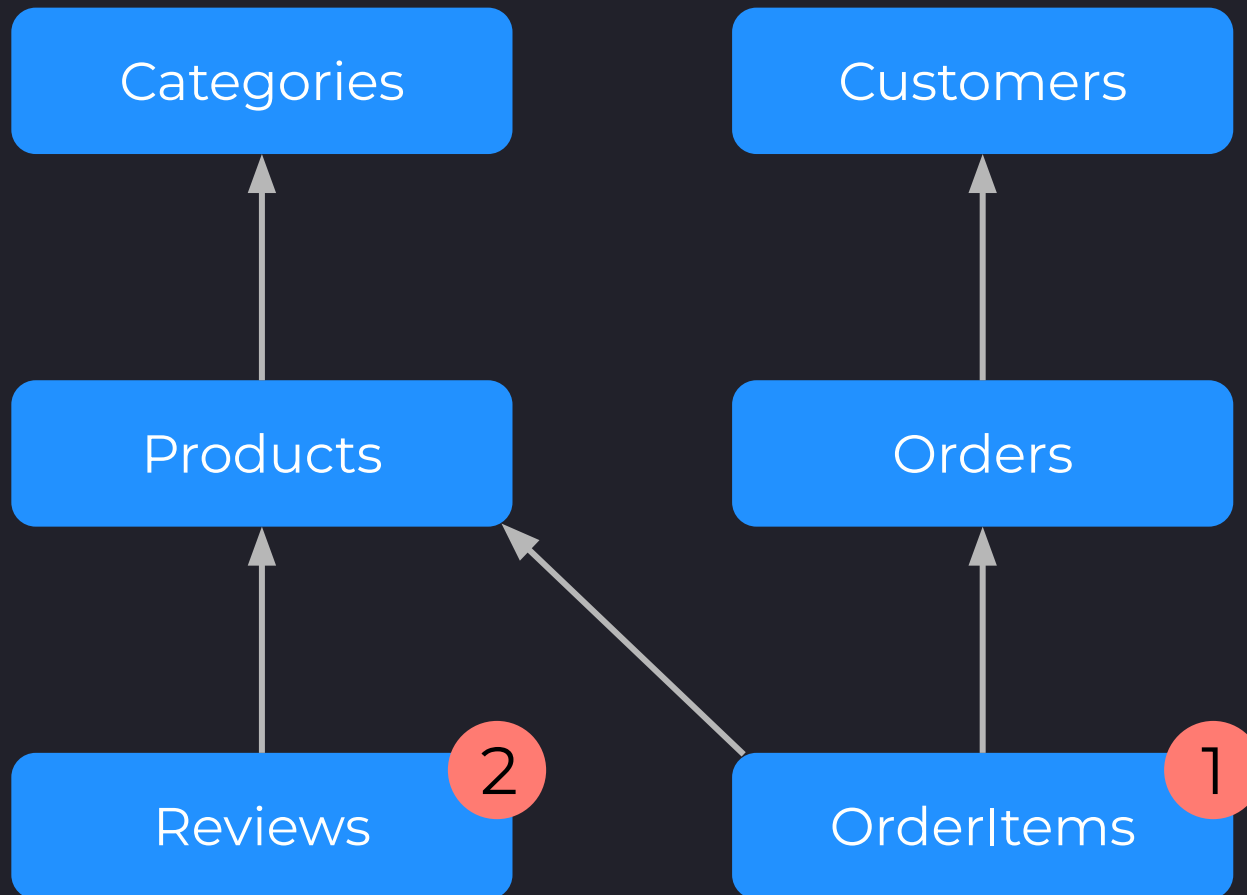


Respawn



DELETE FROM "OrderItems"

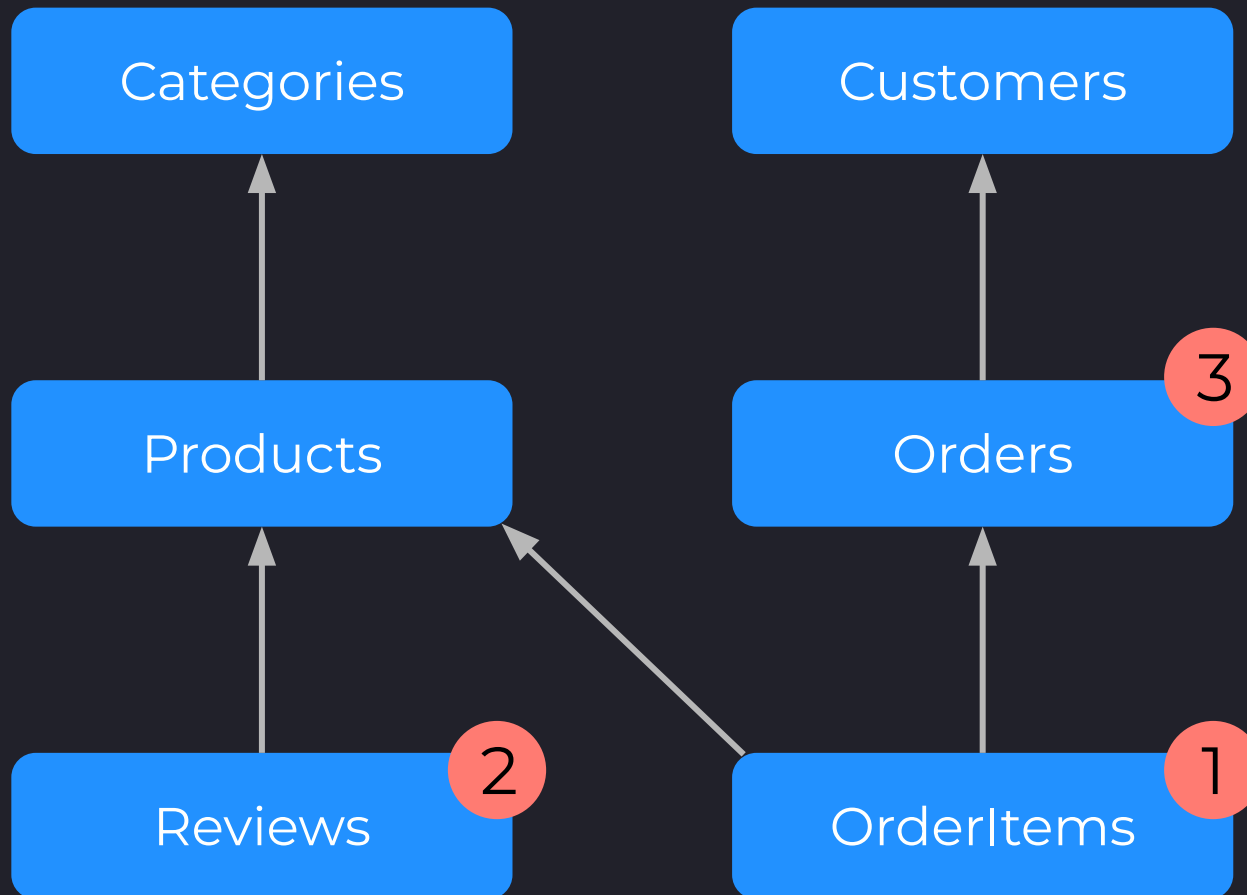
Respawn



`DELETE FROM "OrderItems"`

`DELETE FROM "Reviews"`

Respawn

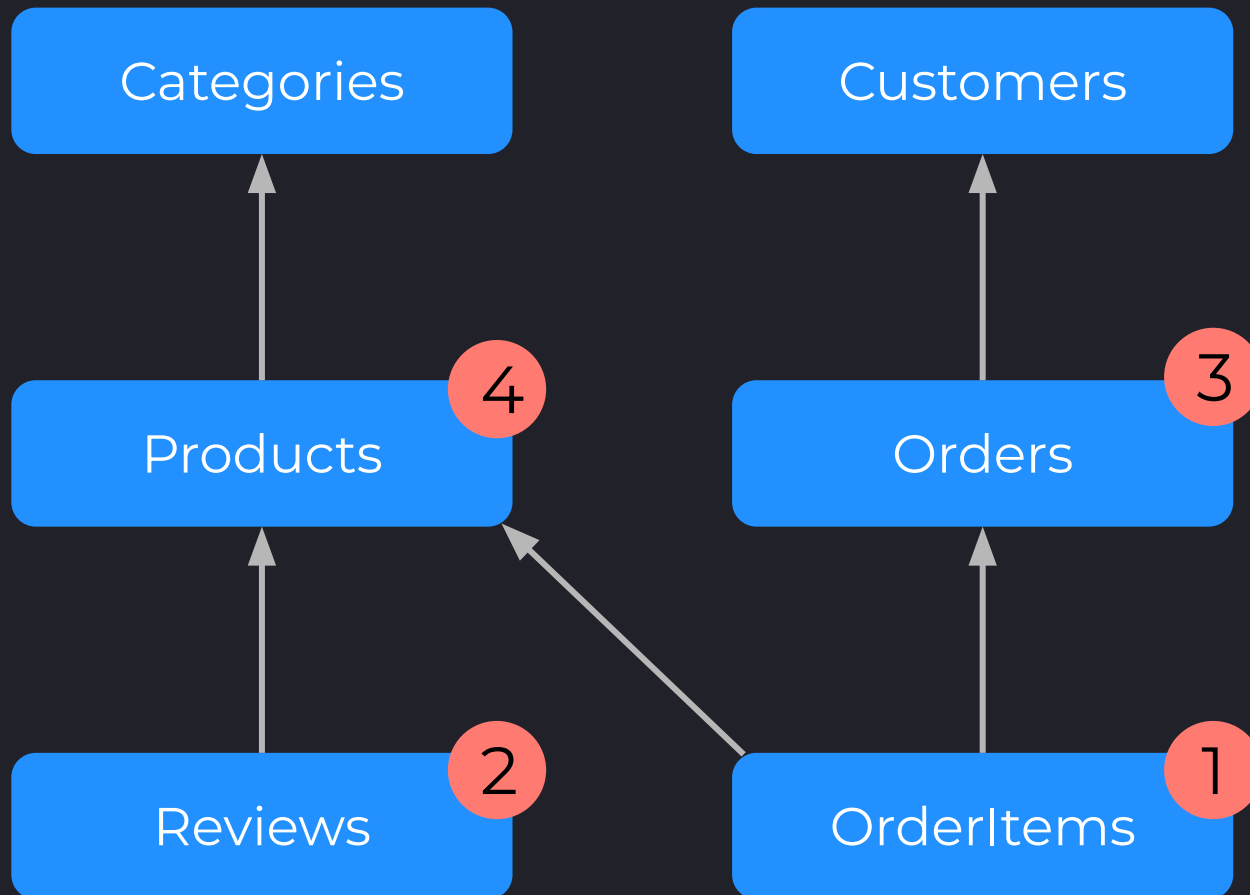


```
DELETE FROM "OrderItems"
```

```
DELETE FROM "Reviews"
```

```
DELETE FROM "Orders"
```

Respawn



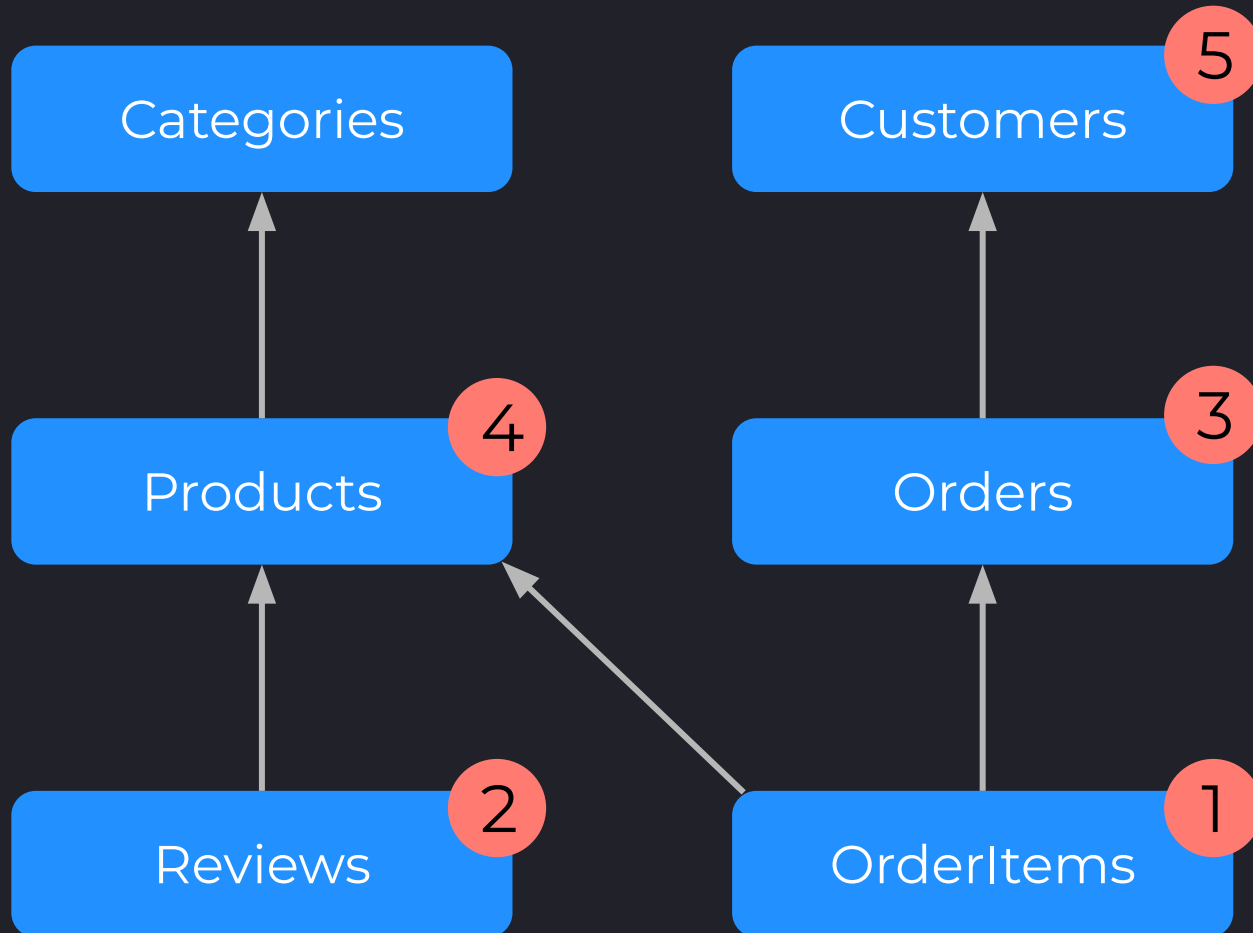
DELETE FROM "OrderItems"

DELETE FROM "Reviews"

DELETE FROM "Orders"

DELETE FROM "Products"

Respawn



```
DELETE FROM "OrderItems"
```

```
DELETE FROM "Reviews"
```

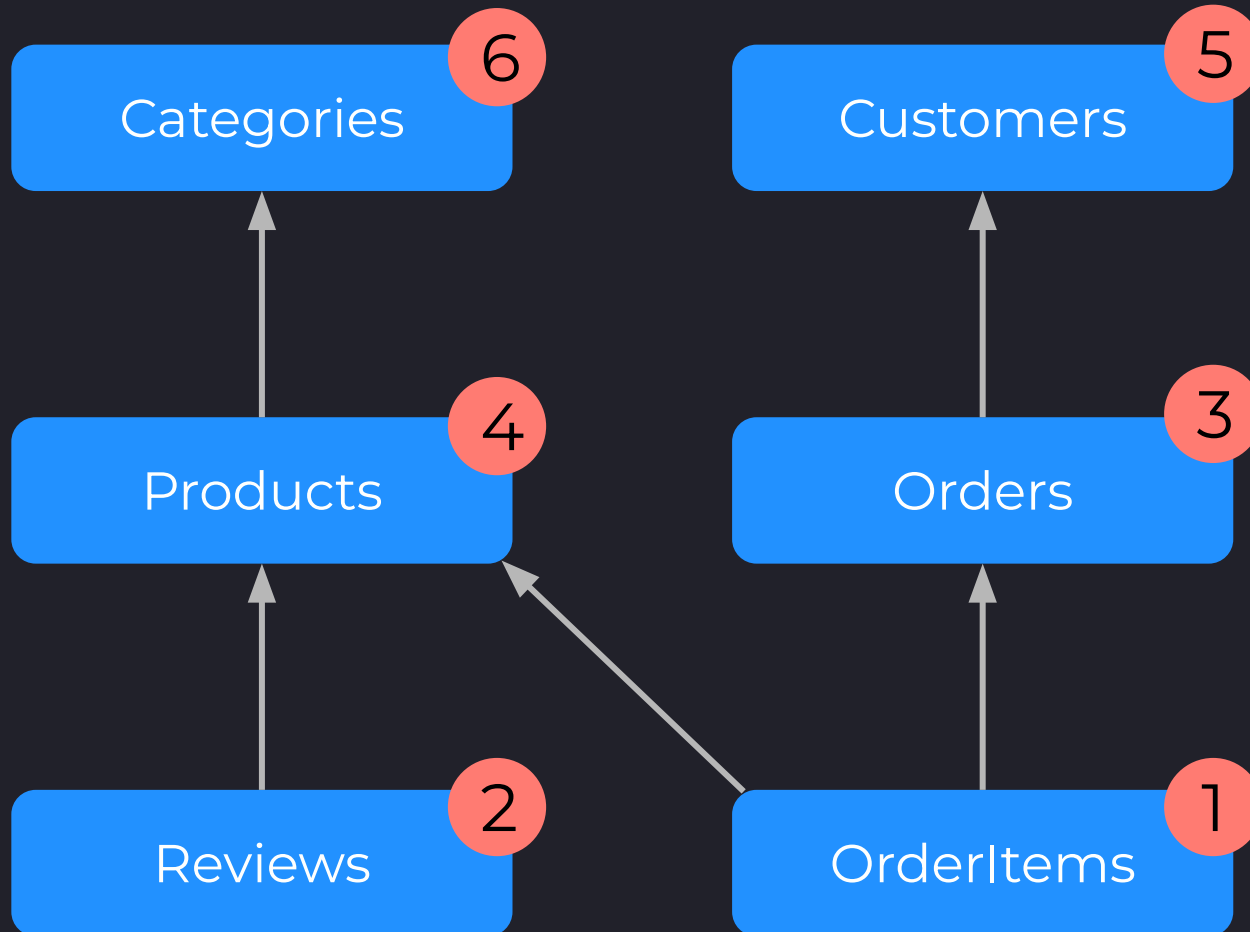
```
DELETE FROM "Orders"
```

```
DELETE FROM "Products"
```

```
DELETE FROM "Customers"
```

```
DELETE FROM "Categories"
```

Respawn



```
DELETE FROM "OrderItems"
```

```
DELETE FROM "Reviews"
```

```
DELETE FROM "Orders"
```

```
DELETE FROM "Products"
```

```
DELETE FROM "Customers"
```

```
DELETE FROM "Categories"
```

Respawn - настройки

```
new RespawnerOptions
{
    DbAdapter          = DbAdapter.Postgres,
    SchemasToInclude  = ["public"],           // сбрасывать только эти схемы
    SchemasToExclude  = ["audit", "lookup"],  // или исключить конкретные
    TablesToIgnore     = [                    // не трогать эти таблицы
        new Table("Countries"),
        new Table("Currencies"),
    ],
    TablesToInclude    = [...],               // сбрасывать только явно указанные
    WithReseed         = true,                // сбросить IDENTITY-счётчики
}
```

Respawn

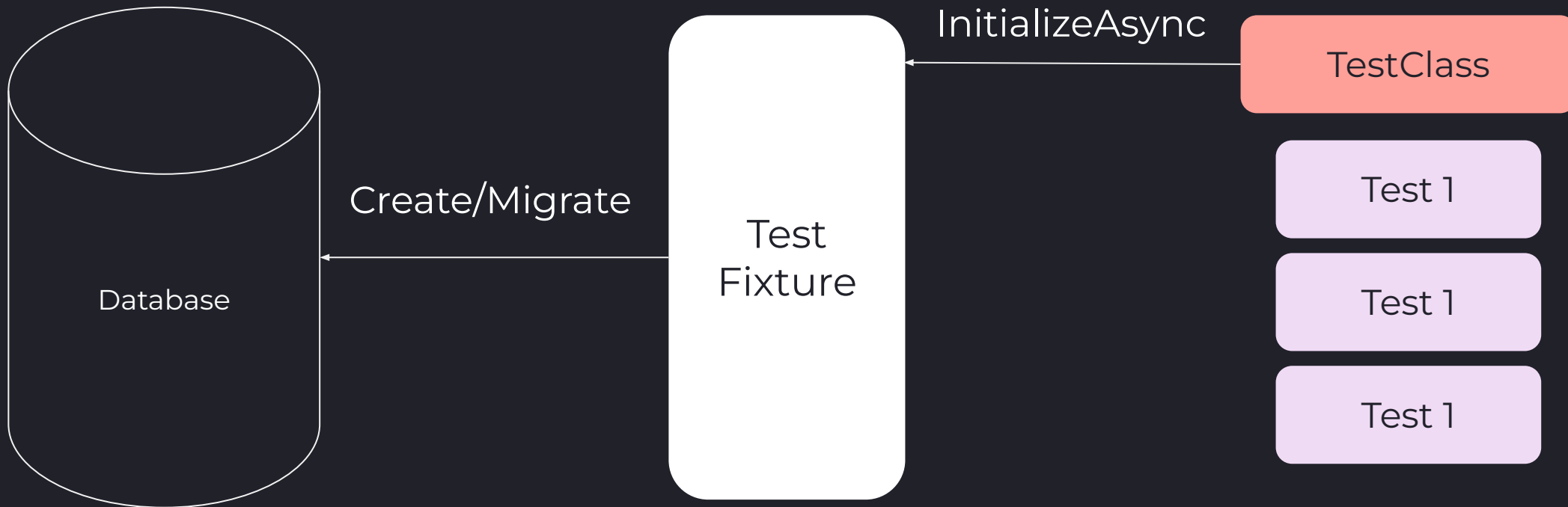
```
public class MyTestFixture : IAsyncLifetime
{
    private Respawner _respawner = null!;

    public async Task InitializeAsync()
    {
        // запустить контейнер, применить миграции...
        _respawner = await Respawner.CreateAsync(conn, new RespawnerOptions
        {
            DbAdapter = DbAdapter.Postgres,
            SchemasToInclude = ["public"],
        });
    }

    public async Task ResetAsync()
    {
        await _respawner.ResetAsync();
    }

    public Task DisposeAsync() { /* остановить контейнер */ }
}
```

Respawn



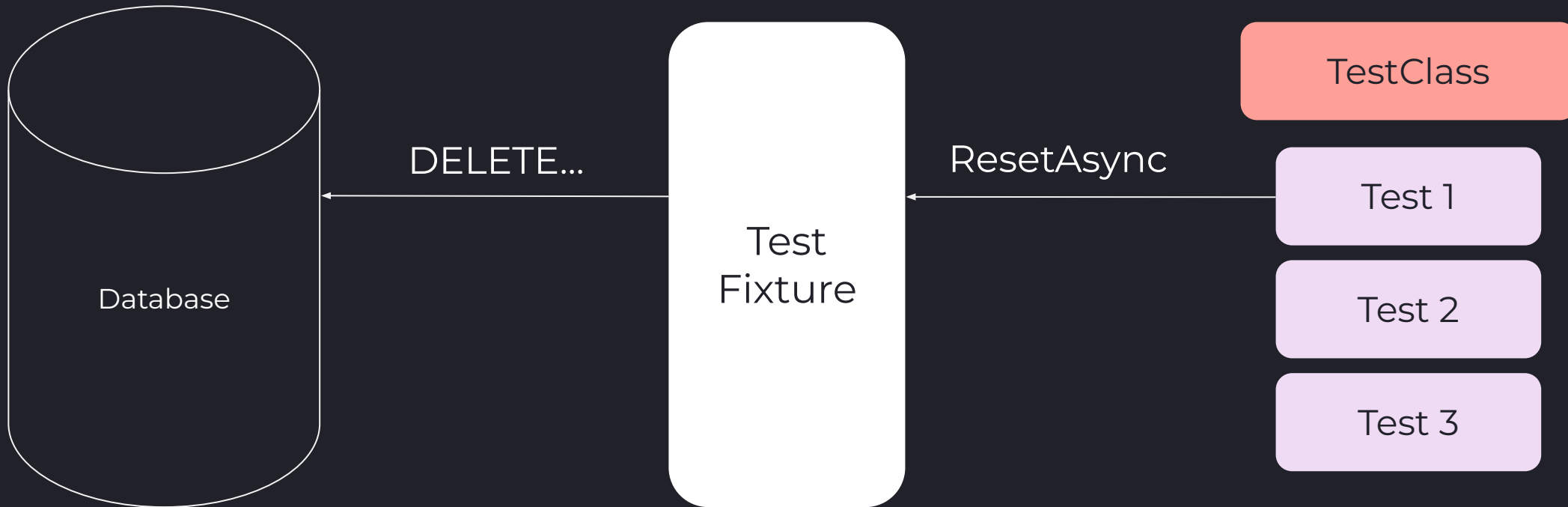
Respawn

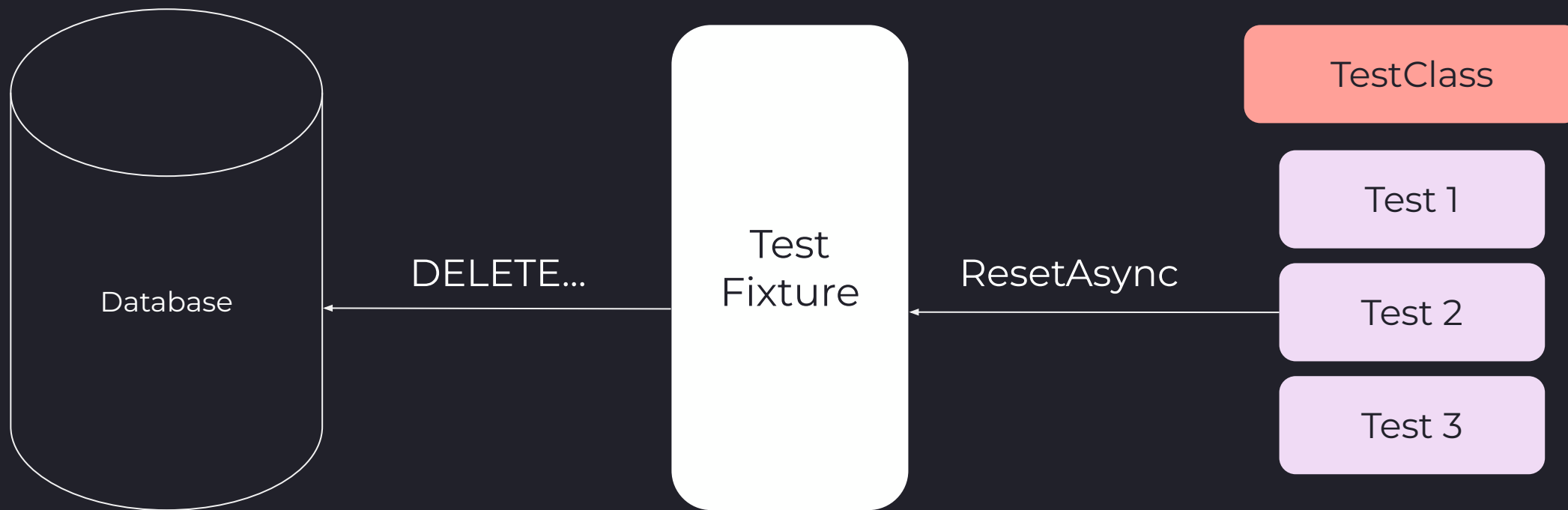
```
public class MyTests : IAsyncLifetime, IClassFixture<MyTestFixture>
{
    . . .

    public async Task InitializeAsync() => await _fixture.ResetAsync();
    public Task DisposeAsync() => Task.CompletedTask;

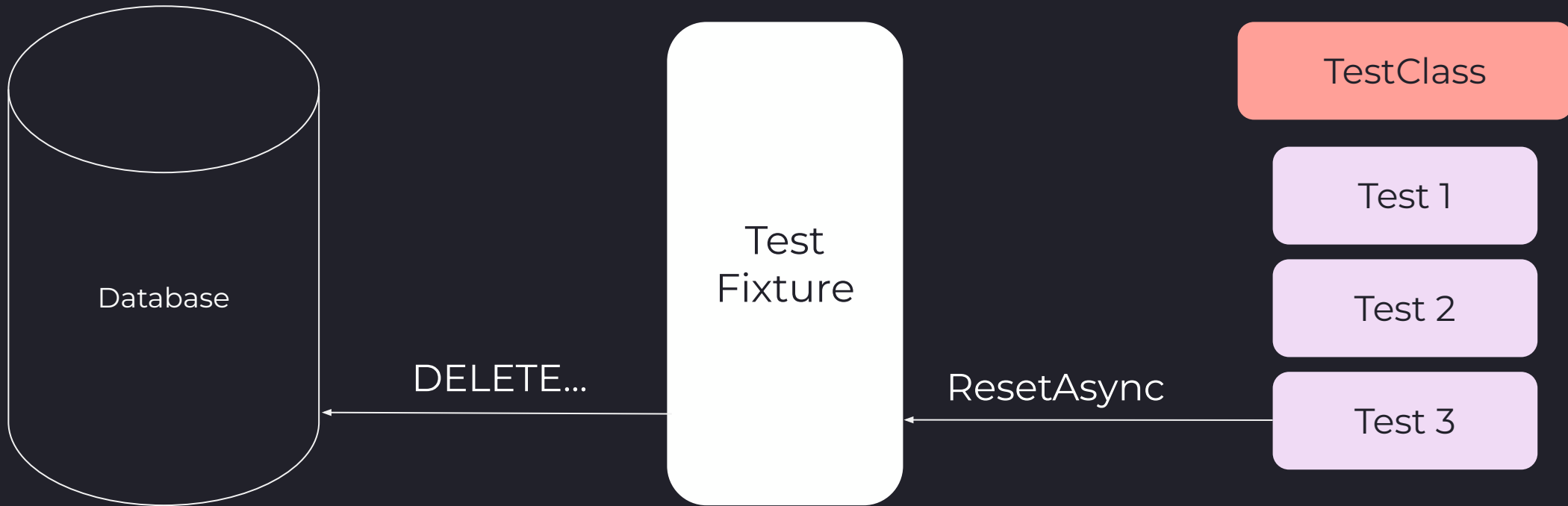
    [Fact]
    public async Task Test() { /* ... */ }
}
```

Respawn





Respawn



Respawn – проблемы?

Respawn – проблемы?

- Изоляция на уровне тест класса
- Параллелизм только между тест классами – 1-2 длинных теста внутри тест класса тормозят остальные
- Проекты с большим количеством тест-классов
- Могут не подтираться seq идентификаторов

IntgreSQL – проблемы?

- Postgres only

IntgreSQL – проблемы?

- Postgres only
- На маленьком количестве тестов может быть оверкилом

IntgreSQL – проблемы?

- Postgres only
- На маленьком количестве тестов может быть оверкилом
- ?

Что и когда выбирать?

- PostgreSQL – IntegreSQL

Что и когда выбирать?

- PostgreSQL – IntegreSQL
- !Postgre but .NET and SQL – Respawn

Что и когда выбирать?

- PostgreSQL – IntegreSQL
- !Postgre but .NET and SQL – Respawn
- Testcontainers

Заключение

Изоляция не бесплатна — важно определить подходящий уровень изоляции окружения в тестах конкретно для ваших задач

Быстрые и стабильные CI/CD и тесты — экономия времени разработчиков (денег), вычислительных ресурсов (денег), более быстрое выявление регрессов и TTM.

Спасибо за внимание



Сергей Булавский
t.me/sergey_bulavskiy



ТГ-канал
Технологии в Контуре



Пример
инфраструктуры
для тестов

Контур