ARCTB.RU

Инструкция по развёртыванию микросервисной системы

8 микросервисов • Go + Python + React • Docker Compose

Версия 1.0 | 2025

1. Обзор архитектуры

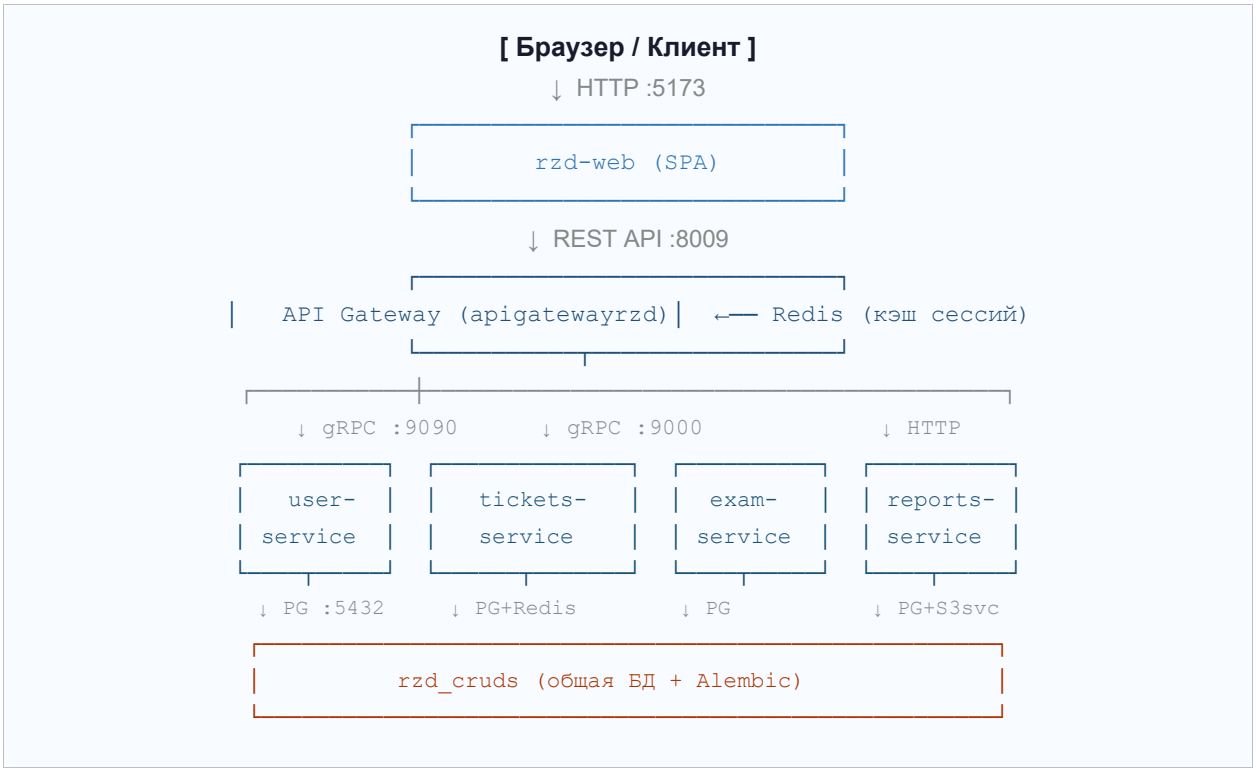
Система представляет собой микросервисную платформу для управления экзаменами и аттестацией сотрудников ARCTB.RU. Все сервисы взаимодействуют через общую Docker-сеть и используют внешнюю базу данных PostgreSQL.

1.1 Состав системы

Сервис	Технология	HTTP порт	gRPC порт	БД / Кэш	Назначение
`rzd-web`	Node 20 / Vite / React	5173	—	—	Фронтенд SPA
`apigatewayrzd`	Python 3.13 / Sanic	8009	—	Redis	API-шлюз, JWT, маршрутизация
`user-service`	Go 1.23 / Gin	8080	9090	PostgreSQL	Пользователи, JWT, роли
`tickets-service`	Go 1.23 / Gin	8000	9000	PostgreSQL + Redis	Билеты, вопросы, кэш
`exam-service`	Python 3.12 / gRPC	—	по .env	PostgreSQL	Логика экзаменов
`reports-service`	Python 3.13 / Sanic	по .env	—	PostgreSQL + Redis	Отчёты (Dramatiq workers)
`s3servicerzd`	Python 3.13 / Sanic	API+UI	—	PostgreSQL + S3	Файловое хранилище
`rzd-cruds`	Python / Alembic	—	—	PostgreSQL	Библиотека ORM + миграции

1.2 Схема взаимодействия сервисов

Ниже приведена логическая схема взаимодействия компонентов системы:



```
                                        ↕ S3 API
                                  [ s3servicerzd ]
```

1.3 Стек технологий

- Фронтенд: React + Vite + TypeScript (Node 20)
- Бэкенд (Go): user-service, tickets-service — Go 1.23, Gin, gRPC, PostgreSQL
- Бэкенд (Python): apigatewayrzd, exam-service, reports-service, s3servicerzd — Python 3.12–3.13, Sanic/FastAPI, Poetry
- Общая библиотека: rzd_cruds — Python ORM (SQLAlchemy + Alembic), публикуется в GitLab PyPI
- Коммуникация: REST HTTP + gRPC (Protocol Buffers)
- Кэш и очереди: Redis 7
- БД: PostgreSQL 15–16
- Контейнеризация: Docker + Docker Compose, общая внешняя Docker-сеть

2. Требования и предварительная подготовка

2.1 Системные требования

Компонент	Требование
ОС	Linux (Ubuntu 20.04+) / macOS / Windows 10+ с WSL2
Docker	версия 24.0+
Docker Compose	версия 2.20+ (plugin или standalone)
RAM	минимум 4 ГБ, рекомендуется 8 ГБ
CPU	2+ ядра
Диск	10 ГБ свободного места
Сеть	Доступ к Docker Hub и GitLab (gitlab.picpall.ru)
GitLab Token	glpat-*** — для установки rzd_cruds из приватного PyPI

2.2 Создание общей Docker-сети

Все сервисы используют единую внешнюю Docker-сеть. Перед запуском любого сервиса необходимо создать её:

Команда	`docker network create rzd_network`

Имя сети (rzd_network) должно совпадать с переменной CONTAINER_NETWORK в .env-файлах каждого сервиса.

2.3 GitLab Token для приватного PyPI

Сервисы exam-service, reports-service, s3servicerzd и apigatewayrzd используют пакет rzd_cruds, опубликованный в приватном GitLab PyPI (gitlab.picpall.ru).

Убедитесь, что токен доступа прописан в Dockerfile или передаётся через переменную окружения:

```
poetry config http-basic.gitlab __token__ <ваш_gitlab_token>
```

Токен встроен в Dockerfile s3servicerzd (glpat-Ue5CTjPwmwxwAyyXndxS). Для остальных Python-сервисов при необходимости передайте токен через build args.

3. Порядок развёртывания

Сервисы необходимо запускать в строгом порядке из-за зависимостей (gRPC-вызовы, общая БД). Ниже приведена рекомендуемая последовательность:

№	Сервис	Что делает	Зависит от
1	`rzd_cruds`	Запускает PostgreSQL, применяет миграции Alembic	—
2	`user-service`	PostgreSQL + приложение, gRPC :9090	rzd_cruds DB
3	`tickets-service`	PostgreSQL + Redis + приложение, gRPC :9000	rzd_cruds DB
4	`exam-service`	gRPC-сервер экзаменов	user-service, tickets-service
5	`s3servicerzd`	Файловый сервис + Streamlit UI	rzd_cruds DB
6	`reports-service`	Sanic API + Dramatiq worker + Redis	s3servicerzd, API Gateway
7	`apigatewayrzd`	Точка входа REST API :8009	user-service, exam-service, tickets-service
8	`rzd-web`	SPA фронтенд :5173	apigatewayrzd

4. Пошаговая инструкция развёртывания

Шаг 1 — rzd_cruds (база данных + миграции)

rzd_cruds — центральная библиотека ORM, содержащая схему БД и миграции Alembic. Сначала нужно поднять PostgreSQL и применить миграции.

1.1 Создание .env файла

```
cd rzd_cruds-main
cp .env.example .env  # или создайте вручную
```

Переменная	Значение по умолчанию / Пример	Описание
DB_HOST	rzd_postgres	Имя контейнера PostgreSQL
DB_PORT	5432	Внутренний порт PostgreSQL
DB_NAME	russian_railways	Имя базы данных
DB_USER	postgres	Пользователь БД
DB_PASS	postgres	Пароль БД
DOCKER_PORT	5434	Внешний порт для подключения
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

1.2 Запуск PostgreSQL

```
docker compose up -d
```

1.3 Применение миграций

Если установлен Poetry локально:

```
cd rzd_cruds-main
poetry install
export DB_HOST=localhost DB_PORT=5434 DB_NAME=russian_railways DB_USER=postgres
DB_PASS=postgres
poetry run alembic upgrade head
```

Или через Docker (если alembic доступен внутри контейнера):

```
docker exec -it <container_name> alembic upgrade head
```

1.4 (Опционально) Заполнение тестовыми данными

```
poetry run python cli/cli.py auto-fill-full    # случайные данные
poetry run python cli/cli.py fill-tst-q       # вопросы от методистов
```

Шаг 2 — user-service (сервис пользователей)

Go-микросервис: HTTP :8080 (health/metrics) и gRPC :9090 (основной API).

2.1 Создание .env файла

```
cd user-service-main
```

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME_DB	user_service_db	Имя контейнера БД
CONTAINER_NAME_APP	user_service_app	Имя контейнера приложения
DOCKER_DB_PORT	5432	Внешний порт PostgreSQL
DOCKER_APP_PORT	8080	Внешний HTTP порт
DOCKER_APP_GRPC_PORT	9090	Внешний gRPC порт
JWT_SECRET	your_secret_key	Секрет для JWT токенов
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

2.2 Запуск

```
docker compose up -d --build
```

2.3 Проверка

```
curl http://localhost:8080/health
```

Шаг 3 — tickets-service (сервис билетов)

Go-микросервис: REST :8000 и gRPC :9000. Использует PostgreSQL и Redis.

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME_DB	railway-postgres	Имя контейнера БД
CONTAINER_NAME_RE	railway-redis	Имя контейнера Redis
CONTAINER_NAME_APP	railway-app	Имя контейнера приложения
DOCKER_DB_PORT	5433	Внешний порт PostgreSQL (не конфликтовать с user-service!)
DOCKER_REDIS_PORT	6379	Внешний порт Redis
DOCKER_APP_PORT	8000	Внешний HTTP порт
DOCKER_APP_GRPC_PORT	9000	Внешний gRPC порт
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

```
docker compose up -d --build
```

3.1 Проверка

```
curl http://localhost:8000/health
```

Шаг 4 — exam-service (сервис экзаменов)

Python gRPC-сервис. Подключается к БД rzd_cruds и к tickets-service по gRPC.

4.1 Создание .env файла

```
cd exam-service-main
```

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME	exam_service	Имя контейнера
DOCKER_PORT	50051	Внешний gRPC порт
DB_NAME	russian_railways	БД (та же, что rzd_cruds)
DB_USER	postgres	Пользователь БД
DB_PASS	postgres	Пароль БД
DB_HOST	rzd_postgres	Хост БД (имя контейнера rzd_cruds)
TICKET_SERVICE_URL	tickets_app:9000	gRPC URL tickets-service
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

```
docker compose up -d --build
```

Шаг 5 — s3servicerzd (файловый сервис)

Python-сервис: Sanic REST API + Streamlit UI. Использует S3-совместимое хранилище и PostgreSQL.

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME	s3_service	Имя контейнера
DOCKER_PORT_API	8010	Внешний порт Sanic API
DOCKER_PORT_UI	8501	Внешний порт Streamlit UI
PORT	8010	Внутренний порт API
STREAMLIT_SERVER_PORT	8501	Внутренний порт Streamlit
DB_HOST	rzd_postgres	Хост БД
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

```
docker compose up -d --build
```

Шаг 6 — reports-service (сервис отчётов)

Python Sanic-сервис + Dramatiq worker для фоновой генерации PDF-отчётов. Включает собственный Redis.

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME	reports_service	Имя контейнера API
DRAMATIQ_CONTAINER_NAME	reports_worker	Имя контейнера worker
REDIS_CONTAINER_NAME	reports_redis	Имя контейнера Redis
DOCKER_PORT	8011	Внешний порт сервиса
SERVICE_PORT	8011	Внутренний порт
REDIS_PORT	6380	Порт Redis (не конфликтовать с :6379!)
DB_HOST	rzd_postgres	Хост основной БД
API_GATEWAY_HOST	api_gateway	Хост API Gateway
API_GATEWAY_PORT	8009	Порт API Gateway
S3_SERVICE_HOST	s3_service	Хост S3-сервиса

Переменная	Значение по умолчанию / Пример	Описание
S3_SERVICE_PORT	8010	Порт S3-сервиса
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

```
docker compose up -d --build
```

Шаг 7 — apigatewayrzd (API-шлюз)

Главная точка входа для всех клиентов. Маршрутизирует запросы к user-service, tickets-service, exam-service через gRPC и HTTP.

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME	api_gateway	Имя контейнера
DOCKER_PORT	8009	Внешний порт
API_PORT	8009	Внутренний порт
SECRET	your_jwt_secret	JWT секрет (должен совпадать с user-service)
ALGORITHM	HS256	Алгоритм JWT
ACCESS_TOKEN_EXPIRE	3600	TTL токена (секунды)
EXAMS_CLIENT_HOST	exam_service	Хост exam-service
EXAMS_CLIENT_PORT	50051	gRPC порт exam-service
TICKET_CLIENT_HOST	tickets_app	Хост tickets-service
TICKET_CLIENT_PORT	9000	gRPC порт tickets-service
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

```
docker compose up -d --build
```

7.1 Проверка API Gateway

```
curl http://localhost:8009/api/v1/users/login \
  -X POST -H 'Content-Type: application/json' \
  -d '{"uin": "admin_uin", "password": "admin123!"}'
```

Шаг 8 — rzd-web (фронтенд)

React SPA, собирается во время docker build. URL API Gateway задаётся через build argument.

Переменная	Значение по умолчанию / Пример	Описание
CONTAINER_NAME	rzd_web	Имя контейнера
DOCKER_PORT	5173	Внешний порт
VITE_API_URL	http://<IP>:8009/api/v1	URL API Gateway (пример: http://192.168.13.68:8009/api/v1)
CONTAINER_NETWORK	rzd_network	Имя Docker-сети

Важно: VITE_API_URL должен содержать реальный IP или hostname сервера (не localhost), так как он встраивается в клиентский JS-код во время сборки.

```
docker compose up -d --build
```

5. Сводная таблица портов

Порт	Протокол	Сервис	Описание
5173	HTTP	rzd-web	Фронтенд (Vite preview)
8009	HTTP	apigatewayrzd	API Gateway (основной вход)
8000	HTTP	tickets-service	REST API билетов
9000	gRPC	tickets-service	gRPC сервер билетов
8080	HTTP	user-service	HTTP (health, metrics)
9090	gRPC	user-service	gRPC (основной API пользователей)
5432	TCP	PostgreSQL (user)	БД user-service
5433	TCP	PostgreSQL (tickets)	БД tickets-service
5434	TCP	PostgreSQL (cruds)	БД rzd-cruds (основная)
6379	TCP	Redis (tickets)	Кэш tickets-service
6380	TCP	Redis (reports)	Очередь reports-service

Примечание: порты PostgreSQL для разных сервисов следует разнести (5432, 5433, 5434…), чтобы избежать конфликтов при запуске на одном хосте.

6. Проверка работоспособности

6.1 Проверка запущенных контейнеров

```
docker ps --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
```

Все контейнеры должны иметь статус Up (healthy).

6.2 Проверка сетевого взаимодействия

```
docker network inspect rzd_network
```

В выводе должны быть перечислены все запущенные контейнеры.

6.3 Health-checks

Сервис	Команда проверки	Ожидаемый ответ
rzd-web	curl http://localhost:5173/	HTML страница
apigatewayrzd	curl http://localhost:8009/api/v1/	JSON или 404
user-service	curl http://localhost:8080/health	{"status":"ok"}
tickets-service	curl http://localhost:8000/health	{"status":"healthy"}

7. Устранение типичных проблем

7.1 Сервис не может подключиться к БД

- Убедитесь, что rzd_cruds PostgreSQL запущен и принимает соединения.
- Проверьте, что контейнер БД находится в той же Docker-сети (rzd_network).
- Убедитесь, что DB_HOST в .env — это имя контейнера БД, а не localhost.

```
docker exec -it <app_container> ping <db_container_name>
```

7.2 Ошибка gRPC Connection refused

- Убедитесь, что целевой сервис запущен и здоров.
- Проверьте, что EXAMS_CLIENT_HOST / TICKET_CLIENT_HOST — имена контейнеров, а не localhost.
- gRPC порты: exam-service → tickets-service :9000, apigatewayrzd → exam-service :50051, apigatewayrzd → tickets-service :9000.

7.3 Фронтенд не видит API

- VITE_API_URL должен быть IP/hostname сервера, доступный из браузера клиента.
- После изменения VITE_API_URL необходимо пересобрать контейнер: docker compose up -d --build.

7.4 Ошибка установки rzd_cruds (GitLab PyPI 401)

- Убедитесь, что GitLab token актуален и имеет права read_package_registry.
- Передайте токен при сборке или обновите Dockerfile.

```
docker compose build --build-arg GITLAB_TOKEN=glpat-xxxxxxxx
```

7.5 Конфликт портов

- Если несколько сервисов используют один и тот же внешний порт — измените DOCKER_DB_PORT, DOCKER_APP_PORT и т.д. в соответствующих .env файлах.
- Особенно следите за PostgreSQL (по умолчанию у нескольких сервисов стоит 5432) и Redis (6379).

7.6 Просмотр логов

```
docker logs -f <container_name>
docker compose logs -f --tail=100
```

8. Полезные команды

8.1 Управление сервисами

Остановить все	`docker compose down`
Пересборка и запуск	`docker compose up -d --build`
Перезапустить сервис	`docker compose restart <service>`

8.2 Просмотр состояния

```
# Все контейнеры проекта
docker ps -a

# Использование ресурсов
docker stats

# Список образов
docker images | grep rzd
```

8.3 Очистка

```
# Остановить и удалить контейнеры, тома, сети
docker compose down -v

# Удалить неиспользуемые образы
docker image prune -f
```

8.4 Миграции Alembic

```
# Применить все миграции
alembic upgrade head

# Откатить последнюю миграцию
alembic downgrade -1

# Статус миграций
alembic current
```

9. Рекомендации по безопасности

- Замените все пароли по умолчанию (user_service_password_2024, railway_password_2024, postgres) перед развёртыванием в production.
- Используйте уникальный JWT_SECRET для каждой среды. Секрет API Gateway и user-service должны совпадать.
- Не храните GitLab-токены в Dockerfile; используйте Docker BuildKit secrets или CI/CD переменные.
- Закройте порты PostgreSQL и Redis файрволом; они должны быть доступны только внутри Docker-сети.

- Включите SSL для PostgreSQL и Redis в production-среде.
- Периодически обновляйте базовые образы Docker (python:3.13, golang:1.23-alpine, postgres:15-alpine).

Документ подготовлен автоматически на основе исходного кода проекта