



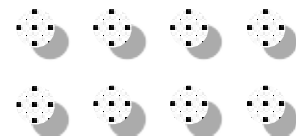
Simple project

Simple is better than complex (Tim Peters)

Санкт-Петербург, ИТМО
05.07.25

ПЛАН

- Цели
- Задачи
- Теория
- Реализованные задачи
- Архитектура
- Паттерны
- Демонстрация



Цели

- Получить навыки разработки и развертывания микросервисных приложений с использованием Docker и Docker compose.
- Использовать в разработке полученные знания по языкам программирования Golang, Java Script, Python.

Задачи

- Разработать отдельные приложения:
 - ✓ Frontend с использованием FastAPI и JS
 - ✓ RestAPI с использованием GO Gorilla
 - ✓ Сервис аутентификации на JVT token
 - ✓ Кеширование данных в Redis
 - ✓ Подключение базы данных Postresql (SQLite)
 - ✓ Подключить сервис Adminer для контроля Postresql
- Осуществить деплой всех контейнеров с помощью Docker compose
- Использовать в разработке полученные знания по языкам программирования Golang, Java Script, Python.

Особенности

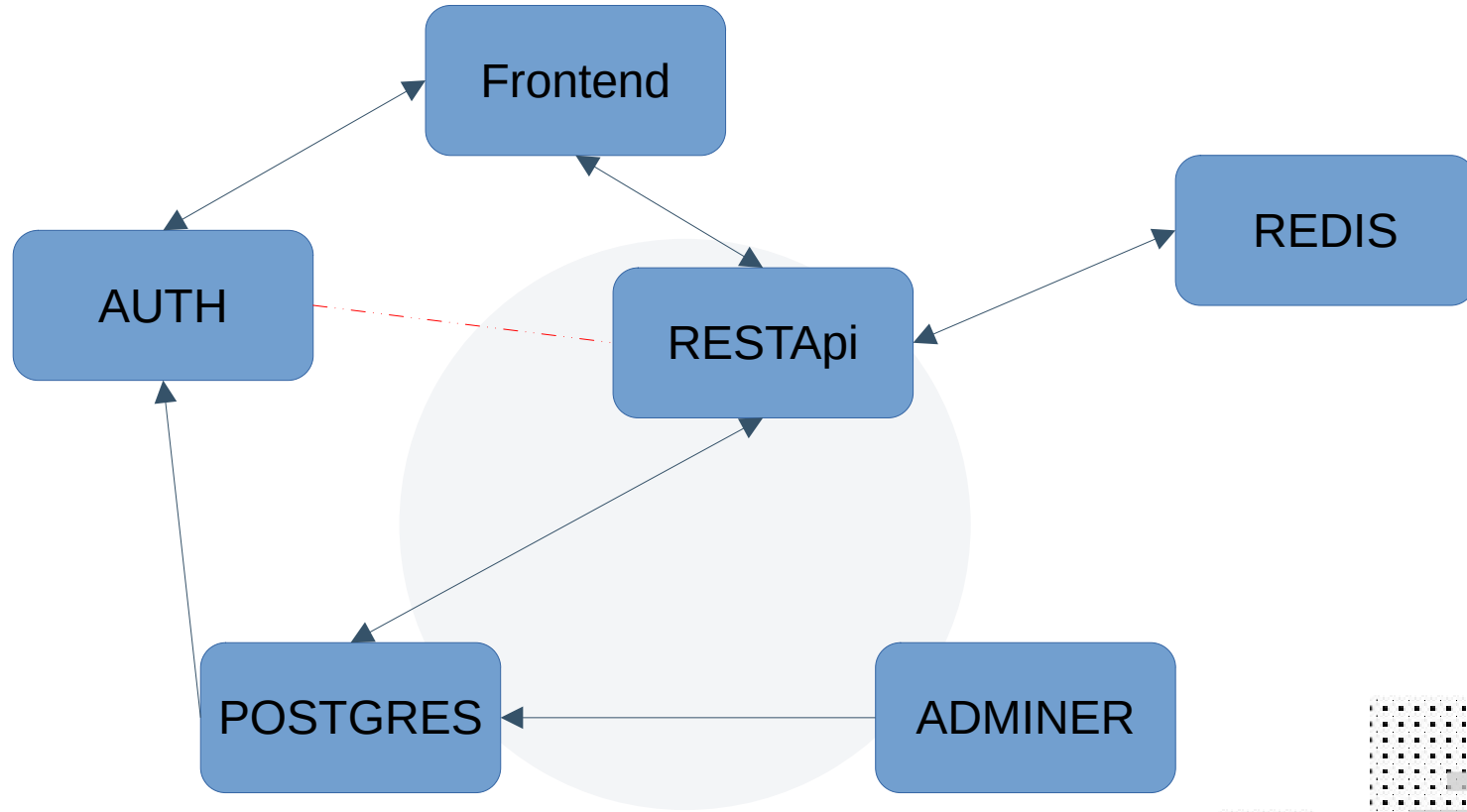
- Приложения максимально простые и примитивные, фокус работы на осуществление взаимодействия между приложениями.
- Проект может быть использован как образец для взаимодействия между отдельными микросервисами.
- При частичном использовании (часть контейнеров через Docker compose, часть отдельный запуск) необходимо правильно использовать `host: localhost` или названия сервисов в `compose.yaml`.

Функционал

Общий сервис состоит из 6 docker контейнеров, запускающихся через docker compose. Доступ к приложению на localhost:8001 либо через демонстрационный сайт <https://strrv.ru> (проксирование с помощью nginx).

- 1) **Frontend** (FastAPI + HTML/CSS + JS) Раздает статику, html-страницы, отправляет запросы к REST-API и принимает ответы, передает запросы к серверу аутентификации, устанавливает полученный JWT токен в cookies.
- 2) **REST-API** (Golang) Принимает GET, POST, PUT и DELETE запросы, обрабатывает их, записывает информацию в базу данных Postgres и Redis.
- 3) **Auth** (Golang) Выполняет аутентификацию пользователя проверяя логин и пароль из базы данных, генерирует JWT токен, отправляет его на фронтенд.
- 4) **Postgres** Основная база данных
- 5) **Redis** Кеширующая база данных.
- 6) **Adminer** Сервис управления базами данных.

Архитектура




Основные тонкости при реализации

- ✓ При сборе контейнеров через Docker compose взаимодействие происходит через внутреннюю сеть Docker (даже если она не объявлена), что требует правильной настройки хостов и портов.
- ✓ Запросы и ответы на фронтенд требуют настройки CORS.
- ✓ При доступе через NGINX настройки могут отличаться. Необходимо согласованно применять настройки NGINX и CORS в приложениях.



Дальнейшее развитие

- ✓ Связь RESTApi и Auth
 - ✓ Логирование работы сервиса
 - ✓ CI/CD
- 



Спасибо за внимание!