Учебный проект: БД зоомагазина "Лап-Ландия" и её интересное окружение

Александр Гоппе

26 февраля 2025 г.

Содержание

1 Описание проекта

Учебный проект **Лап-Ландия** посвящён построению базы данных для зоомагазина с развёрнутой инфраструктурой, включающей реплицируемую БД, балансировщик нагрузки, инструменты мониторинга и анализатора данных. Демонстрационный стенд разворачивается “по клику“ через docker compose.

1.1 Общая структура

- **База данных**: кластер Patroni из трёх узлов.
- **Балансировка**: HAProxy (1 демонстрационный экземпляр).
- **Распределённая Система Контроля (DCS)**: Zookeeper 3 экземпляра.
- **Инициализация БД**: мини-контейнер db-script-runner (создаёт базу и юзеров).
- **Миграции**: Flyway.
- **Мониторинг**: PostgreSQL Exporter + Prometheus + Grafana.
- **Бизнес-аналитика**: Apache Superset.
- **Демонстрационное приложение**: Spring Boot (Java).
- **Демонстрационное тестирование API**: контейнер curl_runner (отправляет запросы в приложение).
- **Фронт (бонус)**: TypeScript (в разработке).
- **Логирование в ClickHouse (бонус)**: через Logstash.

Один экземпляр балансировщика HAProxy был развёрнут с пониманием того, что в реальном производстве их должно быть минимум два. Но, т.к. всё окружение разворачивается в тестовой среде на персональном компьютере ученика и точка отказа, так или иначе, одна - работает один экземпляр. Теоретически можно было бы обойтись двумя узлами Zookeeper и Patroni, но, как и многое в этом проекте, эти настройки были взяты из открытых источников и, ввиду и без того не самой тривиальной архитектуры стенда, было решено не тратить время на переделку готовых наработок коллег-ремесленников.

Аналогично не реплицировались Superset, ClickHouse и другие системы, т.к. предпочтительной задачей в контексте курса была выбрана настройка некоторого одного “целевого“ кластера СУБД.

1.2 Диаграмма инфраструктуры

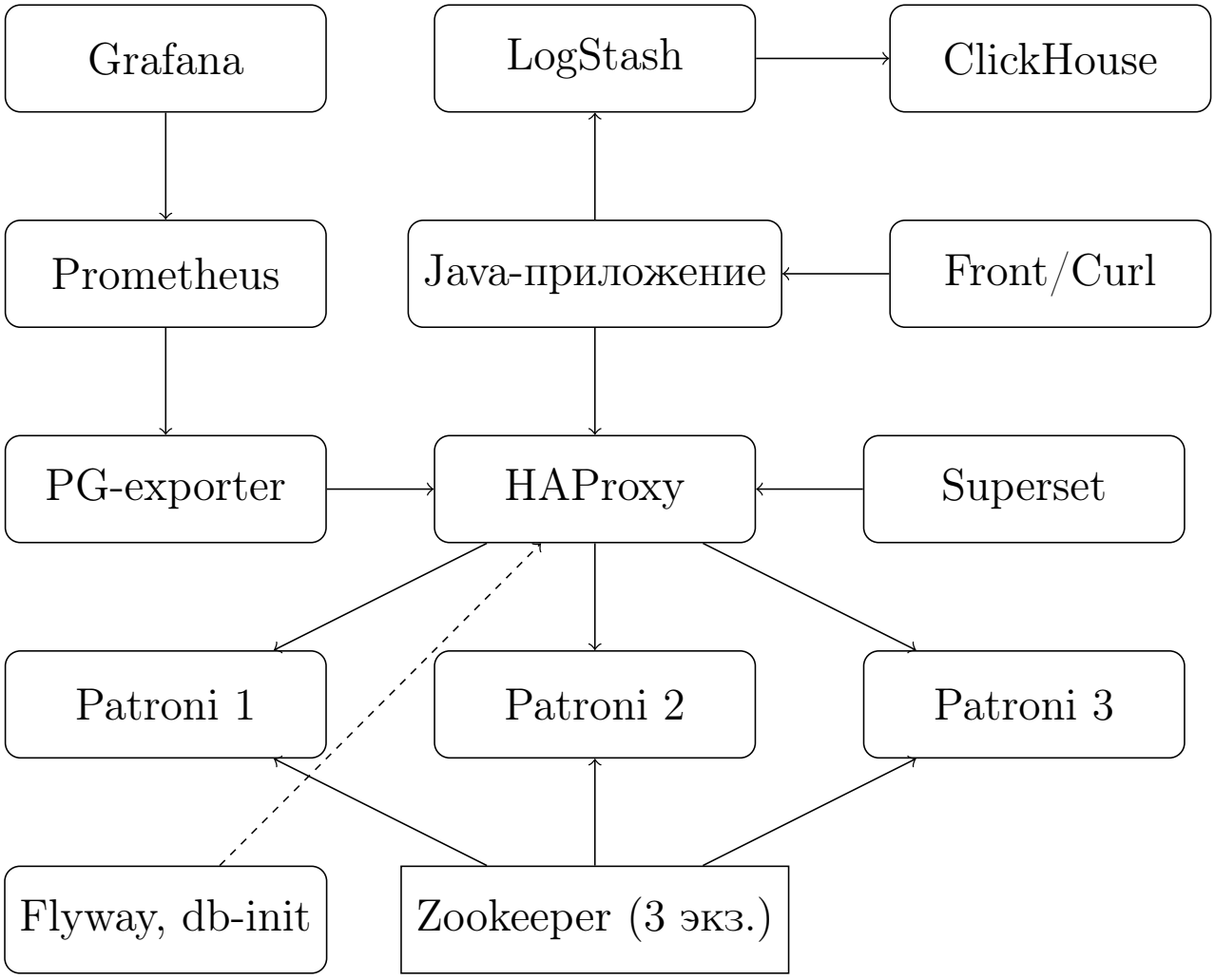

Рис. 1: Схема инфраструктуры проекта

2 Структура и элементы хранилища данных

2.1 Схемы

В базе созданы следующие схемы:

- **archive**: архивированные данные, для разгрузки оперативной схемы.
- **business**: оперативная схема с данными о покупках в сети зоомагазинов.
- **cron**: техническая схема для элементов расширения cron.
- **migrations**: техническая схема инструмента миграции flyway.
- **public**: общая начальная схема PostgreSQL.

2.2 Схема archive: описание

Архивируются только данные о покупках, как наиболее тяжёлые и интенсивные. Структура и индексы полностью дублируют прототипы из бизнес-схемы.

2.3 Схема business: описание

Оперативные бизнес-данные о покупках, магазинах, покупателях.

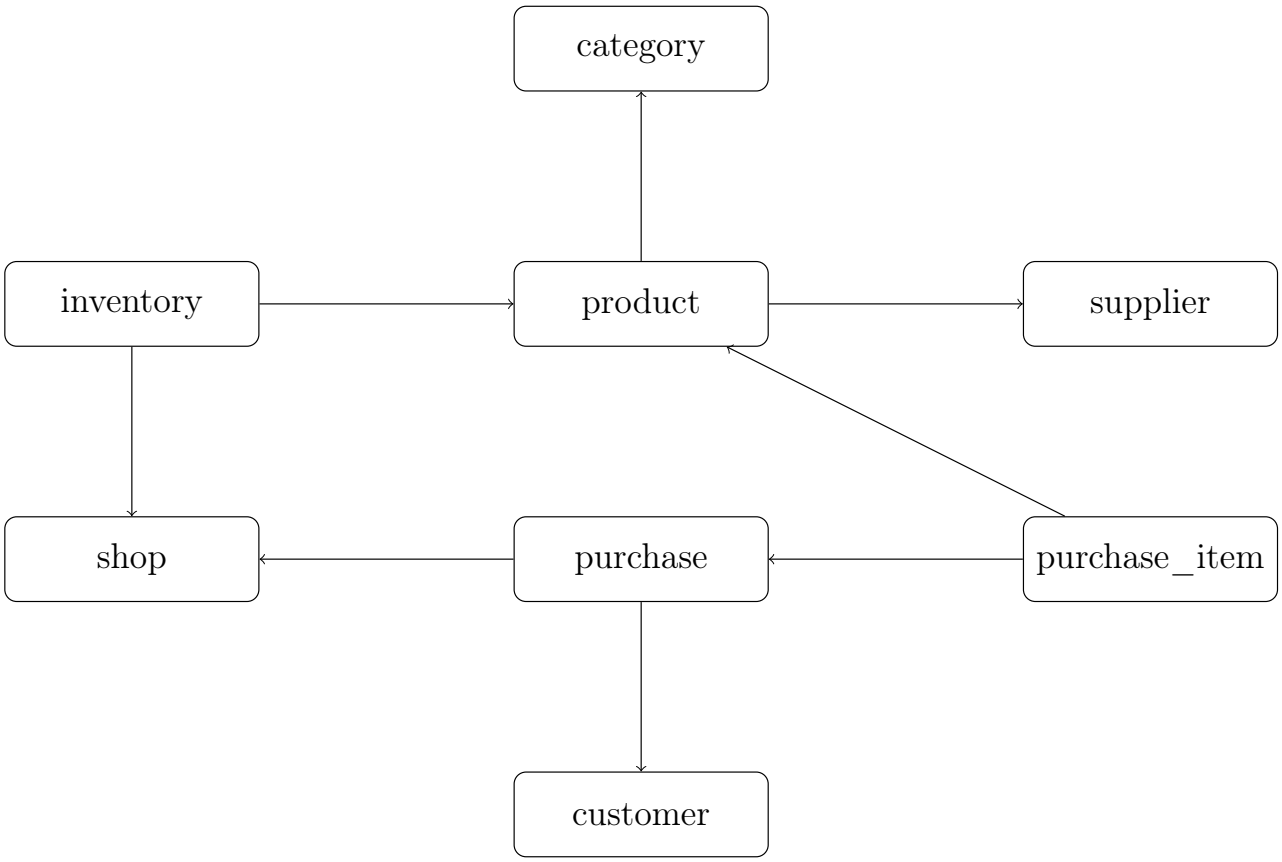

Рис. 2: Схема связей таблиц

2.3.1 Таблица category

Имя столбца	**Тип**	**Описание**
id	SERIAL	Уникальный идентификатор категории
name	TEXT	Название категории

2.3.2 Таблица product

Имя столбца	**Тип**	**Описание**
id	SERIAL	Уникальный идентификатор продукта
name	TEXT	Название продукта
description	TEXT	Описание продукта
category_id	INT	Ссылка на категорию (category.id)
price	NUMERIC(10,2)	Цена продукта
characteristics	JSONB	Характеристики продукта в формате JSON

2.3.3 Таблица shop

Имя столбца	Тип	Описание
id	SERIAL	Уникальный идентификатор магазина
name	TEXT	Название магазина
location	TEXT	Местоположение магазина

2.3.4 Таблица inventory

Имя столбца	Тип	Описание
shop_id	INT	Ссылка на магазин (shop.id)
product_id	INT	Ссылка на продукт (product.id)
quantity	INT	Количество товара в магазине

2.3.5 Таблица supplier

Имя столбца	Тип	Описание
id	SERIAL	Уникальный идентификатор поставщика
name	TEXT	Название поставщика
contact_info	TEXT	Контактная информация поставщика

2.3.6 Таблица customer

Имя столбца	Тип	Описание
id	SERIAL	Уникальный идентификатор клиента
phone	phone_domain	Телефон клиента
email	email_domain	Электронная почта клиента
name	TEXT	Имя клиента
loyalty_status	loyalty_status	Статус лояльности клиента
bonus_points	NUMERIC(10,2)	Количество бонусных баллов клиента

Несмотря на предполагаемую валидацию на бэкенде, правильные базовые типы в целом в базе не помешают. Перечисление поможет избежать случайных описок и логически ограничит значения.

2.3.7 Таблица purchase

Имя столбца	Тип	Описание
id	BIGSERIAL	Уникальный идентификатор покупки
customer_id	INT	Ссылка на клиента (customer.id)
shop_id	INT	Ссылка на магазин (shop.id)
purchase_date	TIMESTAMP	Дата покупки
total_amount	NUMERIC(10,2)	Общая сумма покупки

Используем BIGSERIAL для подстраховки от переполнения номеров покупок.

2.3.8 Таблица purchase_item

Имя столбца	Тип	Описание
purchase_id	BIGINT	Ссылка на покупку (purchase.id)
product_id	INT	Ссылка на продукт (product.id)
quantity	INT	Количество товара в покупке

2.3.9 Индексы

Имя индекса	Таблица	Описание
idx_product_search	product	Индекс для поиска по описанию и характеристикам продукта (используется GIN)
idx_product_category	product	Индекс по категории продукта (category_id)
idx_inventory_product_shop	inventory	Индекс по продуктам и магазинам в инвентаре
idx_purchase_customer_date	purchase	Индекс по покупателю и дате покупки
idx_purchase_brin	purchase	Индекс с использованием BRIN для диапазона дат покупок
idx_customer_phone	customer	Уникальный индекс по телефону клиента

2.4 Пользовательские типы данных

В данной секции приведены пользовательские типы и домены, используемые в базе данных.

- **email_domain** – текстовый тип, содержащий email-адрес. Соответствует регулярному выражению:

  ```
  ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$
  ```

- **phone_domain** – текстовый тип для хранения телефонных номеров. Допустимые значения:

  ```
  ^\+?\d{10,15}$
  ```

- **loyalty_status** – перечислимый тип, определяющий уровень лояльности клиента. Возможные значения:

Значение	Описание
BRONZE	Базовый уровень
SILVER	Средний уровень
GOLD	Высший уровень

2.5 Процедуры, функции и задания по расписанию

В данной секции приведены хранимые процедуры, функции и задания, выполняемые в базе данных.

2.5.1 Процедуры

- **archive_old_purchases** – процедура для архивации устаревших данных о покупках.
  - Выполняет перенос устаревших записей в архивную таблицу.
  - Освобождает основную таблицу от старых данных.

2.5.2 Функции

- **transliterate** – функция для транслитерации текста.
  - Принимает строку на входе.
  - Возвращает строку, в которой символы заменены на их латинские аналоги.

Функция может пригодиться при миграциях и расширении БД. В данном проекте она нашла применение для эстетичности генерируемых данных :)

2.5.3 Задание по расписанию

В базе используется планировщик задач для автоматического выполнения архивации старых покупок.

- **Задание архивации:** выполняется ежедневно в 04:00.

```
SELECT cron.schedule('0 4 * * *',
    $$CALL business.archive_old_purchases();$$);
```

2.6 Схема cron: описание

Данная схема создана подключенным в ходе инициализации БД расширением pg_cron и содержит технические таблицы job и job_run_details с информацией о заданиях.

2.7 Схема migrations: описание

Данная схема создана используемым для управления миграциями инструментом flyway и содержит единственную техническую таблицу flyway_schema_history с информацией о миграциях.

3 Кластер Patroni и настройки узлов БД

3.1 Кластер Patroni

Кластер Patroni разворачивается благодаря подготовленному образу, включающему в себя установку различных python-библиотек и расширений и настроечный файл patroni.yml, содержащий различные настройки для Распределённой Системы Контроля (DCS), в качестве которой был выбран Zookeeper. В целом, настройки любой DCS практически идентичны и подкрепляются соответствующими библиотеками python.

```
bootstrap:
  dcs:
    ttl: 30
...
zookeeper:
  hosts:
    - zoo1:2181
    - zoo2:2181
    - zoo3:2181
  scope: patroni
  namespace: /service/patroni
```

3.2 Распределённый контроль с помощью Zookeeper

DCS была выбрана исходя из наличия минимального знакомства с ней благодаря работе с Apache Kafka. Zookeeper поддерживает иерархичное хранение ключей и является устоявшейся системой, и хорошо подходит для хранения не часто меняющихся данных. При выборе продуктового варианта, однако, можно рассмотреть выбор и ETCD, популярность которого растёт на фоне, как принято считать, отличается более простым настраиванием и интеграцией с kubernetes. В самом Zookeeper можно поинтересоваться данными patroni, такими как список узлов, конфигурацией, переданной для DCS, или, например, кто сейчас является лидером:

```
$ root@zoo1:/zookeeper-3.4.14/bin# zkCli.sh
Connecting to localhost:2181
...
get /service/patroni/leader
{"leader": "patroni1", "epoch": 0}
...
```

3.3 Балансировщик с высокой доступностью HAProxy

HAProxy настроен на API нашего Patroni так, что healthcheck проверяет HTTP-статус 200. Такой статус возвращает только мастер-узел, а реплики возвращают 503.

Однако, при желании или необходимости, можно разгрузить мастер-узел от чтений, добавив в HAProxy дополнительный backend. Это потребует от нас развить работающие с БД приложения на использование двух источников данных с балансировкой, например, в контексте инструментария Spring Boot JPA. В данной работе этот вариант только обозначим, а настроим на прямую работу только с мастером.

4 Мониторинг

4.1 Общее описание мониторинга

Мониторинг организован с помощью:

- **postgres_exporter** - инструмента, который выполняет запросы к базе, подключаясь под специально созданным юзером с ограниченными правами.
- **Prometheus** - известного сборщика метрик, обращающегося по http к postgres_exporter.
- **Grafana** - инструмента визуализации метрик.

В качестве изюминки проекта было реализовано предварительное встраивание панелей (дашбордов) при разворачивании образа grafana, делая проект готовым к демонстрации уже “с порога“. Готовая красивая панель для postgresql была скачана с сайта grafana.

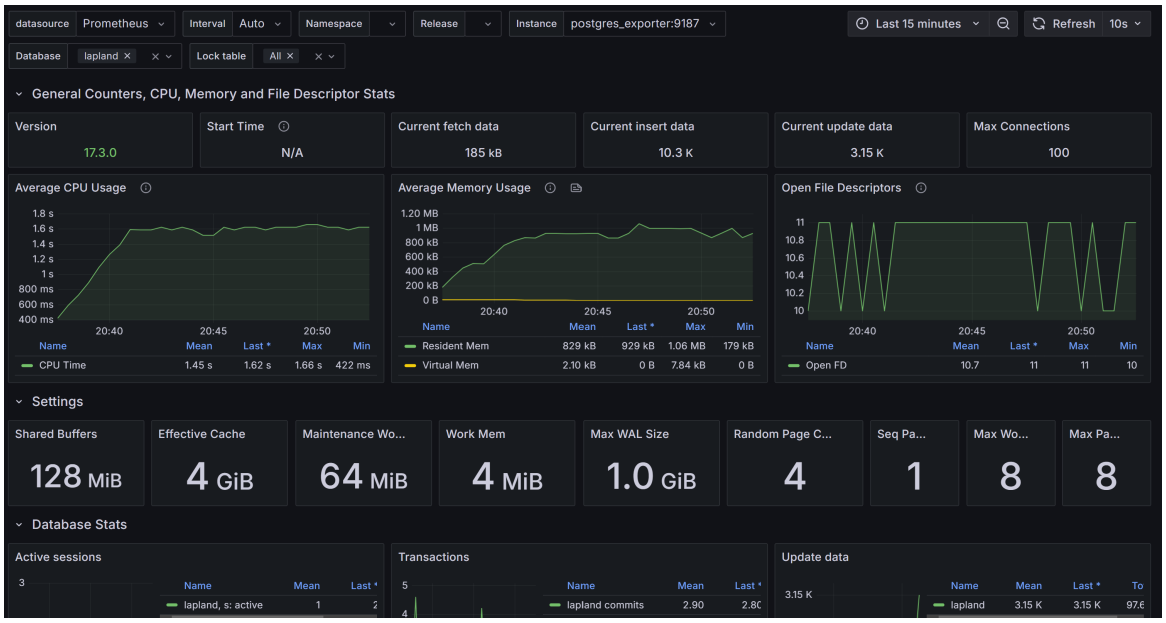

Рис. 3: Панель мониторинга в Grafana

5 Бизнес-аналитика

5.1 Настройка Apache Superset

Это известное средство анализа и визуализации данных с открытым исходным кодом для настройки “по кнопке“ требует ощутимого упорства. Для автоматизации запуска Superset с предустановленными панелями потребовалась усиленная команда запуска ПО в виде:

- Установки библиотек для подключения к целевой БД (PostgreSQL).
- Инициализации внутренней базы (которой по умолчанию является встроенная sqlite3, это решение хорошо подходит для демонстрации, но в продуктовой среде к использованию не рекомендуется).
- Инициализации самого Superset.

- Создания администратора с паролем.
- Ожидания запуска Superset по определённому порту.
- Запуска скрипта импорта с использованием API Superset, включающего:
  - Получение токена для обращений по API.
  - Импорту подготовленного архива с панелями и чартами.

Существенной сложностью в отладке и построении данного процесса было диагностирование и обход требования использования CSRF-токена для работы с flask-сервером, реализующим API Superset. Изучались заголовки запросов в отладчике браузера, документация Superset, форумы и GPT-объяснения. В результате в изначальные настройки программы, расположенные в python-скрипте были добавлены переменные, отключающие механизм CSRF-токена и ослабляющие настройки безопасности на демонстрационном стенде.

5.2 Панели

Ниже приведена автоматически импортируемая панель. График был собран из данных таблицы покупок, агрегированных по дате. Чарт - из запроса со связями.

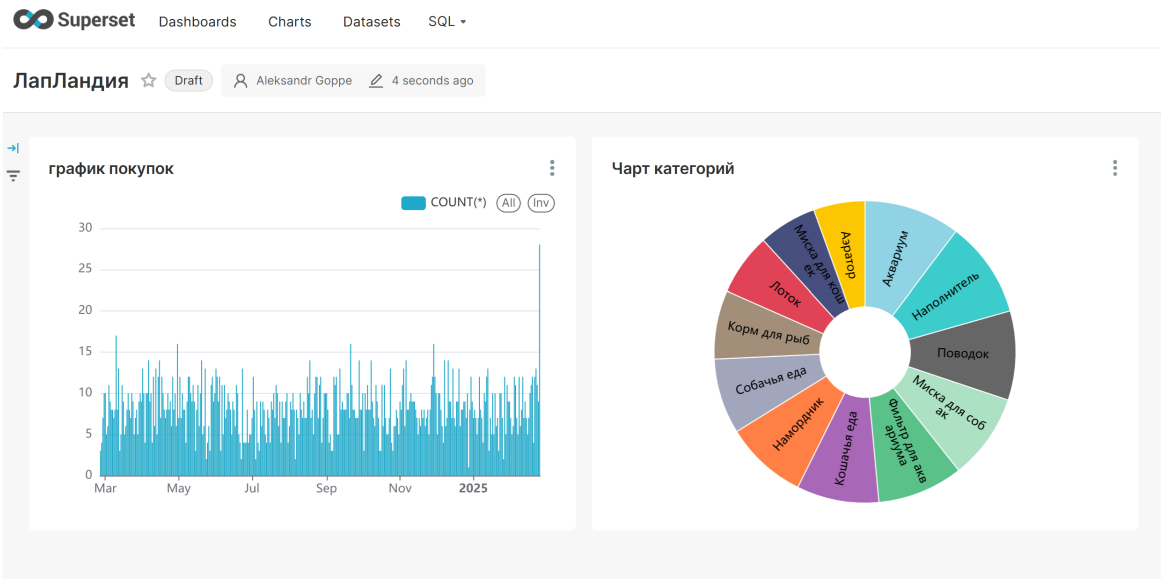

Рис. 4: Панель в Superset

6 Демонстрационное серверное Java-приложение

В демонстрационных целях было разработано веб-приложение, работающее с базами данных через актуальный подход Spring-Boot и Hibernate

7 Демонстрационное клиентское React.js-приложение

Ну тут что-то ещё.