ПРИМЕНЕНИЕ ТЕХНОЛОГИЙ КОНТЕЙНЕРИЗАЦИИ ДЛЯ СБОРКИ И ДОСТАВКИ МУЛЬТИСЕРВИСНОГО ВЕБ-ПРИЛОЖЕНИЯ

Маркер В. А.[1], Кочегурова Е. А.[2]
[1]Томский политехнический университет, ОИТ ИШИТР, гр. 8ВМ31, e-mail: vam52@tpu.ru
[2]Томский политехнический университет, ОИТ ИШИТР, доцент, e-mail: kocheg@tpu.ru

Аннотация

В работе рассматривается процесс сборки и поставки мультисервисного веб-приложения, реализующего функционал автоматизированного рабочего места маркетолога. Описаны особенности контейнеризации компонентов системы с использованием *Docker*, а также управление набором контейнеров с помощью утилиты *Compose*.

Ключевые слова: контейнеризация, Docker, микросервисная архитектура.

Введение

Современные веб-приложения все чаще состоят из нескольких взаимодействующих компонентов. Архитектуру приложений, организованных таких образом, называют микросервисной. Развитие микросервисной архитектуры является ответом на потребность бизнеса в создании гибких и масштабируемых систем, чье функционирование не зависит от одного монолитного блока, а предполагает комбинирование различных сервисов. Этот подход находит всё более широкое применение в современных информационных технологиях.

Было разработано мультисервисное веб-приложение – автоматизированное рабочее место маркетолога-поставщика товаров. Это решение отвечает на потребность современного рынка в инструментах, способствующих повышению эффективности маркетинговых исследований, оптимизации как процесса работы с клиентами, так и расходов на проведение маркетинговых кампаний.

Целью настоящей работы является организация сборки и поставки разработанного веб-приложения в виде, максимально приближенном к коробочному – готовому к передаче заказчику. Развитие технологий контейнеризации, особенно платформы Docker, предоставляет эффективные инструменты для достижения этой цели, позволяя автоматизировать процессы сборки и развертывания приложений, что непосредственно отражается на ускорении и упрощении поставки программных продуктов.

Описание веб-приложения

Веб-приложение автоматизированного рабочего места (АРМ) маркетолога включает серверную и клиентскую части. Пользовательский интерфейс состоит из следующих компонентов:

– информационная панель аналитики (статистика продаж, портрет потребителя);

– панель управления целевыми аудиториями (формирование по личностным и поведенческим критериям);

– панель управления маркетинговыми кампаниями (формирование персональных предложений, оценка их эффективности).

Серверная часть приложения была разработана с использованием микросервисной архитектуры. На рисунке 1 представлена структурная схема веб-приложения.

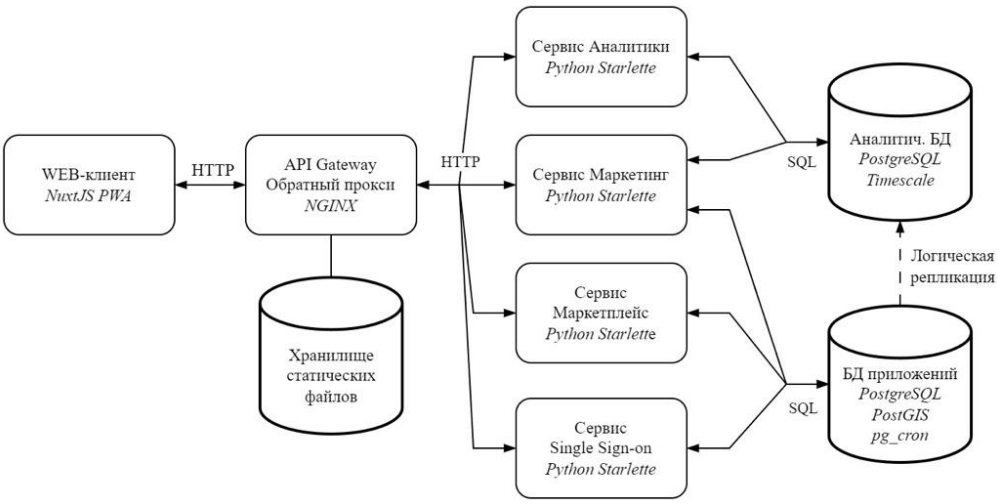
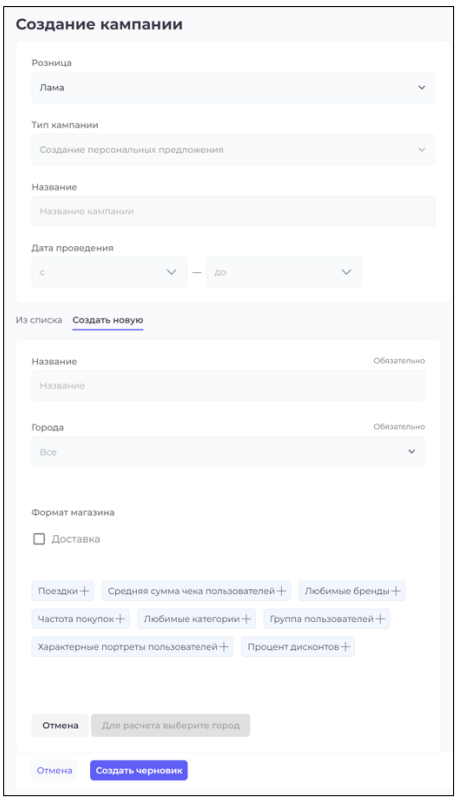
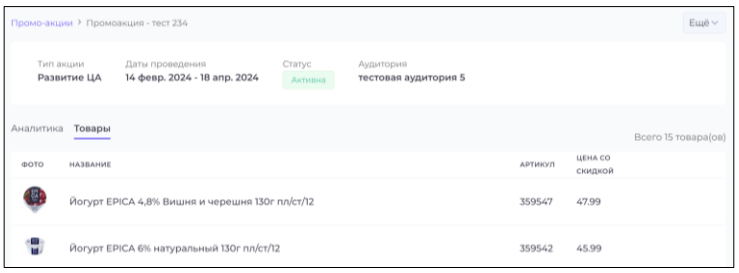


Рис. 1. Структурная схема веб-приложения

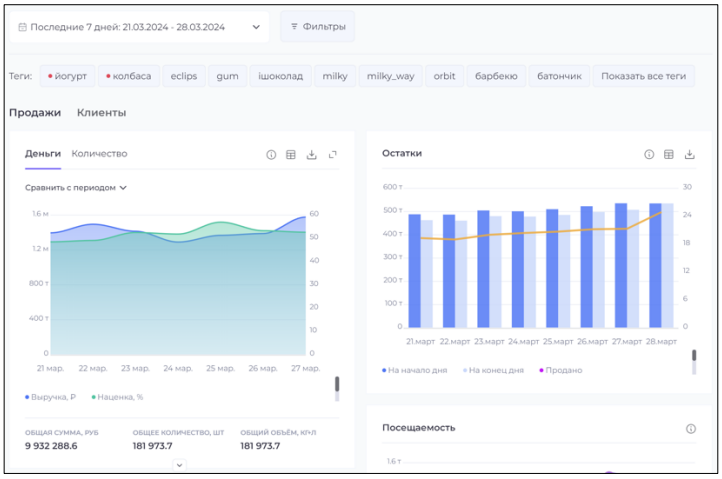
Веб-клиент – пользовательский интерфейс, реализованный в виде *HTML5 SPA* (Single Page Application) с использованием фреймворка *NuxtJS* [1]. На рисунке 2 представлены некоторые фрагменты пользовательского интерфейса АРМ маркетолога.



а



б



в

Рис. 2. Интерфейс АРМ маркетолога:
а – форма создания черновика маркетинговой кампании,
б – панель управления маркетинговой кампанией,
в – информационная панель «Аналитика»

API Gateway. Обратный прокси и сервер статических файлов Nginx [2], организующий единую точку доступа к сервисам Backend.

Backend-сервис Single Sign-On (SSO) – HTTP-сервер, реализующий API OAuth2 [3] для аутентификации и авторизации, а также предоставляющий WEB-клиенту общую информацию о пользователе и его правах.

Backend-сервис Аналитика – HTTP-сервер, предоставляющий API для получения агрегированных данных о продажах в разрезе конкретных товаров, категорий товаров, конкретных магазинов и городов; данных о потребителях в разрезе их демографических характеристик, поведенческих паттернов и местоположения. Также в рамках сервиса выполняются периодические задачи расчёта таблицы факто о пользователях, необходимой сервису Маркетинг.

Backend-сервис Маркетинг – HTTP-сервер, предоставляющий API для управления целевыми аудиториями и маркетинговыми кампаниями и реализующий соответствующую бизнес-логину: формирование персональных предложений на основе различных стратегий, доставка информации о предложениях и акциях клиентам.

Backend-сервис Маркетплейс – HTTP-сервер, предоставляющий API для управления ассортиментом, карточками товаров: ценами, описанием, постоянными скидками. Также в рамках сервиса выполняются периодические выгрузки номенклатуры торговой сети и данных о финансовых транзакциях (покупках).

Для хранения исторических данных и выполнения аналитических запросов используется СУБД PostgreSQL [4] с установленным расширением Timescale DB [5]. Расширение предоставляет функции, упрощающие обработку данных временных рядов, автоматического секционирования и сжатия реляционных таблиц.

Прикладная логика backend-сервисов также требует хранилище данных, в качестве которого используется отдельный экземпляр PostgreSQL с установленными расширениями PostGIS для работы с географическими данными и pg_cron для запуска периодических задач внутри СУБД.

Контейнеризация компонентов

Контейнеризация – это набор технологий, позволяющих упаковывать приложения и их зависимости в изолированные модули – контейнеры. Эти модули являются переносными и могут быть запущены в рамках любой среды, обеспечивая максимально возможную совместимость и независимость от окружения. Кроме того, такие платформы контейнеризации как *Docker* [6] предоставляют механизмы для автоматизации развертывания и управления запущенными контейнерами.

Далее приведено описание процесса контейнеризации компонентов системы.

1. API Gateway и WEB-клиент.

Веб-клиент – это HTML5-приложение. Чтобы сделать его доступным для конечных пользователей достаточно веб-сервера статических файлов. Также необходимо организовать единую точку входа для всех запросов к бэкенд-сервисам. Для этого используется сервер обратного прокси *Nginx*. Разработчики *Nginx* предоставляют готовый образ контейнера [7], который можно использовать в качестве базового для API Gateway.

Так как веб-клиент реализован с использованием фреймворка NuxtJS его код нужно предварительно скомпилировать. Эту процедуру можно выполнить во время сборки образа контейнера. Негативным фактором является то, что в этом случае система сборки NuxtJS и другие зависимости также попадут в конечный образ контейнера, что увеличит его размер. Платформа Docker предоставляет возможность многоэтапной сборки образов контейнеров, что позволяет выполнить сборку приложения в одном контейнере и использовать другой контейнер для его запуска.

Разработчики *NodeJS* (среда выполнения JavaScript), необходимого для сборки веб-клиента, предоставляют готовый образ контейнера, который можно использовать в качестве базового для контейнера-сборщика [8].

Сборка образа контейнера API Gateway состоит из следующих шагов:

1. Копирование файлов манифеста JavaScript приложения в файловую систему контейнера-сборщика, основанного на образе NodeJS.
2. Установка зависимостей, необходимых для сборки проекта.
3. Копирование исходного кода приложения в файловую систему контейнера-сборщика.
4. Сборка WEB-приложения.
5. Копирование статических файлов собранного приложения из файловой системы контейнера-сборщика в файловую систему контейнера-сервера, основанного на образе Nginx Alpine Slim.

6. Копирование шаблона конфигурации Nginx в файловую систему итогового контейнера. Данный файл определяет правила маршрутизации HTTP-трафика, как для статических файлов, так и для конечных точек Backen-сервисов.

Значения шаблонных параметров определяются переменными среды, с которыми запущен контейнер. Логика выполнения шаблонизации конфигурации и запуска веб-сервера реализована в базовом образе Nginx.

Рисунок 3 иллюстрирует описанные шаги, а также содержит команды Dockerfile, необходимые для выполнения многоэтапной сборки.

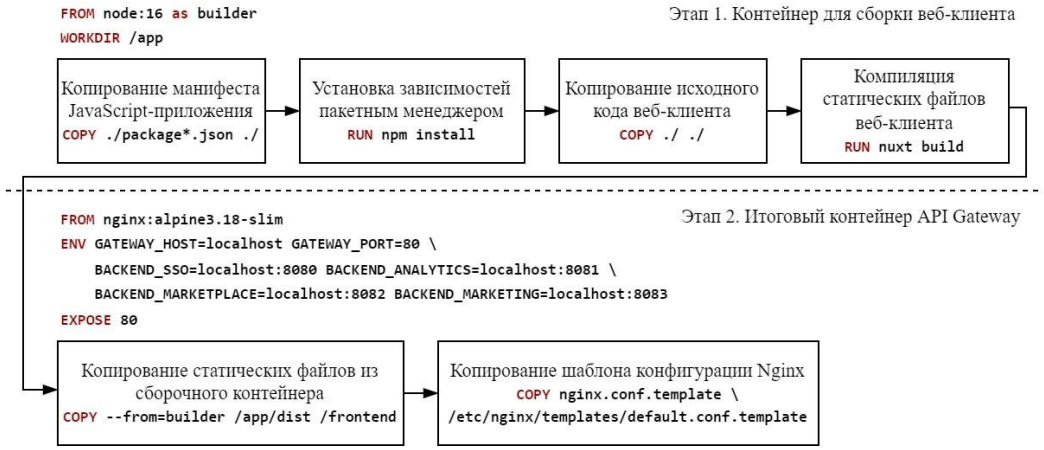


Рис. 3. Этапы сборки образа контейнера API Gateway с указанием команд Dockerfile

Применение многоэтапной сборки позволило уменьшить размер итогового образа контейнера на 95 % с 1,46 ГБ при использовании NodeJS в качестве веб-сервера, до 65,1 МБ при использовании Nginx.

2. Backend-сервисы.

Все backend-сервисы АРМ маркетолога реализованы на интерпретируемом языке программирования Python. Он не требует компиляции, поэтому во время сборки образов контейнеров необходимо только установить соответствующие зависимости, скопировать исходный код и файлы конфигурации, а также определить команду запуска веб-сервера. Разработчики Python предоставляют в общее пользование образы контейнеров с предустановленным интерпретатором языка и пакетным менеджером. В качестве базового был выбран образ, основанный на облегчённом Debian 10 [10] с предустановленным Python 3.10.

В отличии от WEB-клиента, для сборки которого были необходимы только публично доступные зависимости, backend-сервисы требуют установки библиотек из частного git-репозитория [11]. В эти библиотеки были вынесены некоторые утилиты, общие для всех сервисов. Доступ к частному репозиторию можно получить через SSH-подключение. При этом нельзя допустить, чтобы ключи доступа или иные данные аутентификации SSH остались в образе контейнера после сборки. Специально для таких случаев платформа Docker предоставляет директиву, позволяющую выполнить любую команду процесса сборки, пробросив сессию клиента SSH с хостовой операционной системы.

Сборка контейнера каждого backend-сервиса состоит из следующих шагов:

1. Внутрь контейнера с помощью встроенного пакетного менеджера устанавливаются пакеты *openssh-client* и *git*. Также производится подготовка служебных директорий SSH-клиента, чтобы он был способен выполнить подключение к удалённому серверу с частным git-репозиторием.

2. В файловую систему контейнера копируется файл *requirements.txt*, содержащий список зависимостей Python-приложения.

3. Выполняется установка зависимостей через пакетный менеджер *pip*. Сессия SSH-клиента пробрасывается с хостовой системы при помощи параметра *–mount=type=ssh*.

4. В файловую систему контейнера копируется исходный код backend-сервиса.

Рисунок 4 иллюстрирует описанные шаги, а также содержит команды Dockerfile, выполняющие сборку, а также другую метаинформацию о контейнере: базовый образ; рабочую директорию; точку подключения раздела, содержащего файлы конфигурации приложения; команду, с которой будет запущен сервис.

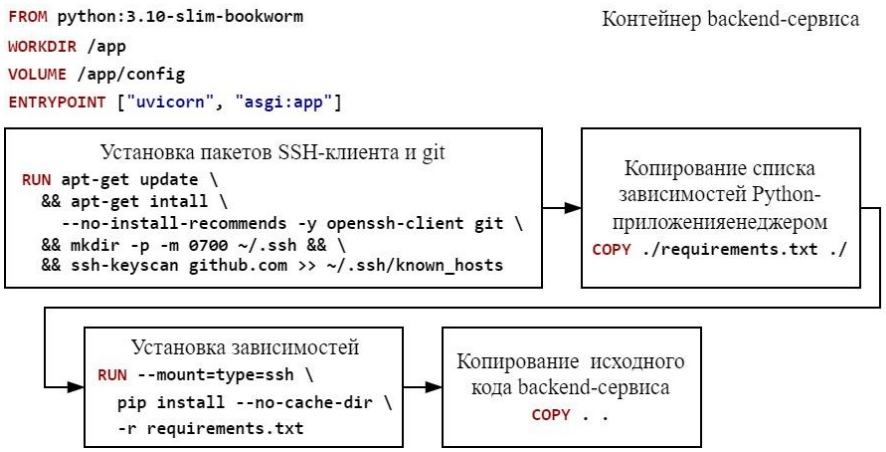


Рис. 4. Этапы сборки образа контейнера backend-сервиса указанием команд Dockerfile

Стоит обратить внимание, что размер полученного отдельного образа контейнера получится не минимально возможным, т. к. пакеты, установленные для загрузки зависимостей, остались в файловой системе контейнера.

Чтобы избежать этого, необходимо выполнить установку пакетов openssh-client и git, затем установку зависимостей Python-приложения и последующее удаление вышеуказанных пакетов в рамках одной команды *RUN*. В этом случае системные зависимости не будут зафиксированы в файловой системе ни одного слоя контейнера, за счёт чего удастся уменьшить размер образа с 575 МБ до 486 МБ.

Негативным фактором описанного подхода является увеличения времени сборки образа при изменении списка зависимостей. Это связано с тем, что слой, созданный командой *RUN*, установившей зависимости, не будет кеширован отдельно. При этом, в первоначально описанной конфигурации данный слой может быть переиспользован между образами контейнеров всех backend-сервисов.

3. СУБД PostgreSQL.

Экземпляры используемых СУБД также могут быть контейнеризованы. Разработчики PostgreSQL предоставляют официальные образы контейнеров с предустановленной СУБД. Один из этих образов может быть базовым при сборке с установкой требуемых расширений и последующим применением миграции реляционной схемы. Также возможно использование готовых образов контейнеров, предоставляемыми разработчиками самого расширения в случае с TimescaleDB [12] и сообществом в случае PostGIS и pg_cron [13].

Управление набором контейнеров

Для работы АРМ маркетолога требуется одновременное выполнение нескольких компонентов-контейнеров. Для облегчения процесса управления мультиконтейнерными приложениями платформа Docker предоставляет утилиту Compose [14]. Для её использования необходимо описать конфигурацию всех контейнеров в декларативном стиле в формате *YAML*. После этого появится возможность управлять их запуском, остановкой и сборкой минимальным набором команд.

Описанные раннее файлы Dockerfile, манифест Docker Compose, определяющий конфигурацию контейнеров веб-приложения АРМ маркетолога, а также набор команд, выполняющий сборку и запуск веб-приложения представлены в публичном репозитории [15].

Заключение

В рамках данной работы был рассмотрен процесс контейнеризации компонентов мультисервисного веб-приложения, реализующего функционал автоматизированного рабочего места маркетолога.

Были описаны особенности сборки и запуска контейнеров для различных компонентов системы. В частности, при сборке образа контейнера сервиса API Gateway была применена техника сборки в

несколько этапов, что позволило значительно уменьшить размер конечного образа (на 95 %). Также было рассмотрено управление набором контейнеров с помощью утилиты Docker Compose.

Применение технологий контейнеризации позволило упростить процесс развертывания и управления приложением, а также уменьшить накладные расходы на его поддержку.

Список использованных источников

1. Введение – Начало работы с Nuxt // Nuxt: интуитивно понятный фреймворк Vue: сайт. – 2024. – URL: https://nuxt.com/docs/getting-started/introduction.
2. Nginx: документация // Nginx: сайт. – 2024. – URL: https://nginx.org/ru/docs/.
3. OAuth 2.0 Authorization Framework // Auth0 by Okta: сайт. – 2024. – URL: https://auth0.com/docs/authenticate/protocols/oauth.
4. PostgreSQL: Documentation: 16: 1. What Is PostgreSQL? // PostgreSQL: сайт. – 2024. – URL: https://www.postgresql.org/docs/16/intro-whatis.html.
5. Get started with Timescale // Timescale documentation: сайт. – 2024. – URL: https://docs.timescale.com/getting-started/latest/.
6. Docker overview // Docker docs: сайт. – 2024. – URL: https://docs.docker.com/get-started/overview/.
7. Nginx – official image // Docker Hub: сайт. – 2024. – URL: https://hub.docker.com/_/nginx.
8. Node – official image // Docker Hub: сайт. – 2024. – URL: https://hub.docker.com/_/node.
9. Dockerfile reference // Docker Docs: сайт. – 2024. – URL: https://docs.docker.com/reference/dockerfile/.
10. Python – official image // Docker Hub: сайт. – 2024. – URL: https://hub.docker.com/_/python.
11. Сведения о GIT // Документация по GitHub: сайт. – 2024. – URL: https://docs.github.com/ru/get-started/using-git/about-git.
12. Install TimescaleDB from a pre-built container // Timescale documentation: сайт. – 2024. – URL: https://docs.timescale.com/self-hosted/latest/install/installation-docker/.
13. Docker image for PostgreSQL and Patrony // GitHub: сайт. – 2024. – URL: https://github.com/mmartinello/patroni-docker.
14. Docker Compose overview // Docker Docs: сайт. – 2024. – URL: https://docs.docker.com/compose/.
15. Контейнеризация. Дополнительные материалы // GitHub: сайт. – 2024. URL: https://github.com/victormarker/containerization-article-2024.