Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б15-мм

Автоматизация сборки компонентов и развертывания инфраструктуры приложения Desbordante

ШАЛЬНЕВ Владислав Александрович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ассистент кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2024

Оглавление

Введение

В последние годы автоматизация процессов разработки и развертывания стала одной из ключевых тенденций в индустрии программного обеспечения. Это обусловлено необходимостью быстрого внедрения изменений, и обеспечения надежной работы сложных распределенных систем.

Для решения задач автоматизации широко применяются практики DevOps, направленные на ускорение разработки и надежное развертывание программного обеспечения. DevOps базируется на таких подходах, как представление инфраструктуры как кода (IaC), непрерывная интеграция и доставка (CI/CD), управление конфигурациями, а также мониторинг и логирование. Эти подходы, в совокупности с использованием современных инструментов и методологий, способствуют минимизации ручных операций, стандартизации процессов разработки и обеспечивают стабильную работу приложений даже в условиях частых изменений.

Активно развивающиеся приложения, такие как открытый профилировщик данных Desbordante, требуют использования систем развертывания инфраструктуры, соответствующих высоким требованиям гибкости, масштабируемости и безопасности. После реимплементации серверной части [22] на языке Python и перепроектирования архитектуры [13], старое решение на основе Docker Compose перестало удовлетворять потребностям приложения. Оно стало ограничением, препятствующим дальнейшему росту и развитию системы.

В рамках данной работы предстояло разработать систему автоматической сборки, доставки и развертывания компонентов инфраструктуры Desbordante, способную устранить недостатки существующего решения и обеспечить поддержку развертывания в production-среде.

1. Постановка задачи

1.1. Цель работы

Целью работы является проектирование и разработка системы, обеспечивающей автоматическую сборку, доставку и развертывание компонентов инфраструктуры приложения Desbordante. Для достижения этой цели были поставлены следующие задачи:

1. сделать обзор:
    - архитектуры приложения с последующим анализом взаимодействия между его внутренними сервисами;
    - существующего решения с анализом его преимуществ и выявлением недостатков;
2. реализовать автоматизированную сборку компонентов, необходимых для работы приложения;
3. организовать доставку полученных в результате сборки артефактов к системе развертывания;
4. разработать систему для развертывания инфраструктуры приложения в соответствии с заданными требованиями;
5. проверить корректность работы разработанной системы и развернутых на удаленном сервере сервисов приложения. Тестирование сервисов должно включать в себя:
    - загрузку датасета;
    - создание задачи обработки;
    - получение текущего статуса задачи и ее результата.

1.2. Требования к системе

К системе развертывания инфраструктуры приложения были предъявлены следующие требования. Система должна:

1. обеспечивать возможность взаимодействия компонентов внутри системы;

2. предоставлять разработчикам приложения доступ к административной панели, при этом уровень доступа должен зависеть от роли разработчика в проекте;

3. обеспечивать доступность приложения для внешних пользователей;

4. поддерживать автоматическое обновление ранее собранных компонентов приложения;

5. обеспечивать сохранность персистентных данных, включая данные баз данных и пользовательские файлы, при выполнении перезагрузки или выключении;

6. обеспечивать защиту конфиденциальных данных, таких как пароли баз данных и токены доступа. Эти данные не должны быть доступны в исходном коде системы;

7. поддерживать две независимые среды развертывания: среда разработки и тестирования и production. Обновление компонентов в production-окружении должно происходить в полуавтоматическом[1] режиме.

В целях обеспечения безопасности и предотвращения случайных ошибок, возникающих в процессе работы над проектом, реализация данных требований должна исключать предоставление сторонним приложениям возможности прямого управления удаленным сервером [8]. Такой подход минимизирует риск несанкционированного вмешательства и способствует поддержанию стабильности инфраструктуры.

[1]Под полуавтоматическим режимом здесь понимается процесс обновления компонентов, при котором требования и инструкции задаются вручную, но само обновление выполняется с минимальным вмешательством, например, путем нажатия кнопки или выпуска новой релизной версии.

2. Обзор

2.1. Обзор архитектуры приложения

Архитектура веб-приложения Desbordante построена с применением клиент-серверной модели, что обеспечивает четкое разделение функций между клиентской и серверной частями. Эта модель позволяет предоставлять пользователю интерфейс для взаимодействия с системой, а также эффективно обрабатывать запросы, управлять данными и выполнять вычислительные задачи за счет серверных ресурсов [18].

Серверная часть приложения состоит из нескольких независимых сервисов, которые взаимодействуют между собой для выполнения различных функций, таких как обработка пользовательских запросов, управление данными и асинхронное выполнение задач. Разделение между сервисами организовано так, чтобы обеспечивать гибкость и масштабируемость системы, позволяя каждому компоненту развиваться независимо [13]. Архитектура приложения на момент начала работы представлена на Рис. 1.

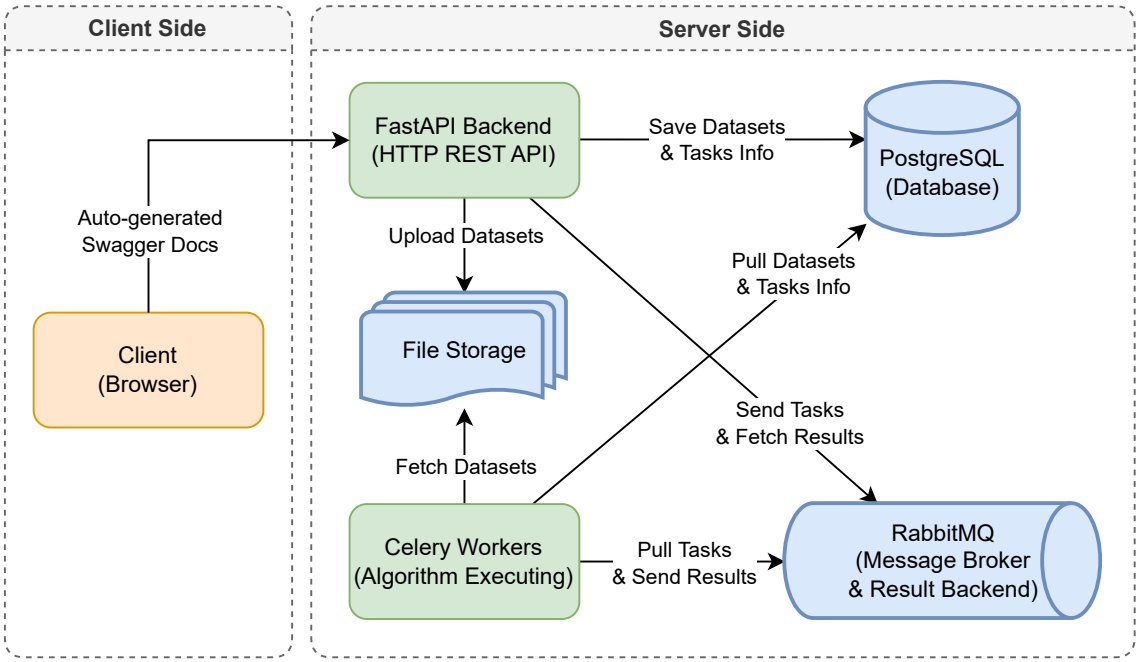

Рис. 1: Архитектура приложения к моменту начала работы

Сервисы конфигурируются с помощью переменных окружения, что позволяет изменять параметры системы без необходимости вносить изменения в исходный код. Такой подход обеспечивает гибкость в управлении конфигурациями, например, дает возможность использовать различные настройки для разных сред (разработки, тестирования и production). Это также повышает безопасность, так как конфиденциальные данные, такие как ключи API[2] или параметры подключения к базе данных, могут быть сохранены в защищенном окружении и не включены в исходный код системы.

2.2. Клиентская часть

На момент выполнения данной работы основная клиентская часть приложения находится на стадии переработки. В связи с этим в качестве клиентской части рассматривается веб-интерфейс, который автоматически генерируется с помощью Swagger [19] и поставляется веб-сервером FastAPI [16]. Данный интерфейс предоставляет пользователю возможность отправлять запросы к серверу, загружать данные и тестировать функциональность API без необходимости разрабатывать полноценный пользовательский интерфейс.

2.3. Основное приложение

Веб-сервер на основе FastAPI выполняет роль ключевого звена, которое обрабатывает пользовательские запросы, взаимодействует с другими сервисами архитектуры и управляет основными процессами приложения.

Он обеспечивает возможность взаимодействия между клиентской частью и серверной инфраструктурой: предоставляет пользователю документацию Swagger для облегчения тестирования и принимает входящие запросы через HTTP REST [14] API.

[2]API (Application Programming Interface) — это программный интерфейс, с помощью которого приложения, веб-сервисы и программы обмениваются информацией.

2.3.1. Взаимодействие с сервисами

Для выполнения своих функций Backend взаимодействует с несколькими серверными компонентами, каждый из которых должен быть доступен и корректно настроен после развертывания системы. Эти взаимодействия затрагивают следущие сервисы:

- **Файловое хранилище.** Используется для сохранения пользовательских датасетов. Backend должен иметь доступ к папке файловой системы, путь к которой указан в переменной `UPLOADED_FILES_DIR_PATH` для загрузки и извлечения данных.
- **База данных PostgreSQL [20].** Хранит метаданные датасетов, информацию о задачах и результаты их обработки. Доступ к базе данных должен быть настроен с указанием учетных данных `POSTGRES_USER` и `POSTGRES_PASSWORD`, адреса сервиса `POSTGRES_HOST` и порта `POSTGRES_PORT`, названия `POSTGRES_DB` и диалекта драйвера `POSTGRES_DIALECT_DRIVER`.
- **Очередь сообщений RabbitMQ [15].** Служит для передачи задач на обработку и получения результатов. Backend взаимодействует с RabbitMQ для создания, отправки и получения сообщений, что требует настройки подключения к брокеру сообщений с помощью переменных `RABBITMQ_DEFAULT_USER`, `RABBITMQ_DEFAULT_PASSWORD`, `RABBITMQ_HOST` и `RABBITMQ_PORT`.

2.4. Обработчики фоновых задач

Для выполнения вычислительных задач, связанных с обработкой датасетов, используется система асинхронной обработки задач — Celery [3]. Celery Workers[3] обрабатывают задачи, которые отправляются в очередь сообщений, с использованием примитивов Desbordante.

[3] https://docs.celeryq.dev/en/stable/userguide/workers.html (дата обращения: 29 ноября 2024 г.).

2.4.1. Взаимодействие с сервисами

Из-за специфики библиотеки Celery, код обработчиков интегрирован непосредственно с основным приложением, что подразумевает использование идентичной конфигурации и доступ к тем же сервисам, что и у Backend-сервиса. Рабочий процесс Celery Workers включает взаимодействия со следующими компонентами:

- **Очередь сообщений.** Обработчики получают задачи из очереди сообщений, которые включают конфигурации алгоритмов для обработки датасета. После завершения обработки, результаты записываются обратно в очередь.
- **База данных.** Celery Workers извлекают из базы данных необходимые метаданные для обработки датасетов, такие как разделитель колонок и индекс заголовка.
- **Файловое хранилище.** Обработчики загружают пользовательские датасеты из файлового хранилища для их последующей обработки.

Все сервисы конфигурируются с использованием тех же переменных окружения, что и для Backend-части. Однако, в отличие от веб-сервера, обработчики Celery недоступны извне, и их взаимодействие с другими компонентами системы возможно только через очередь сообщений.

2.5. Обзор существующего решения

Существующее решение используется командой разработчиков во время работы над проектом для упрощения процесса развертывания и настройки необходимых компонентов приложения.

2.5.1. Контейнеризация

Для обеспечения изолированной работы компонентов приложения используется контейнеризация с помощью Docker [4]. Данный подход

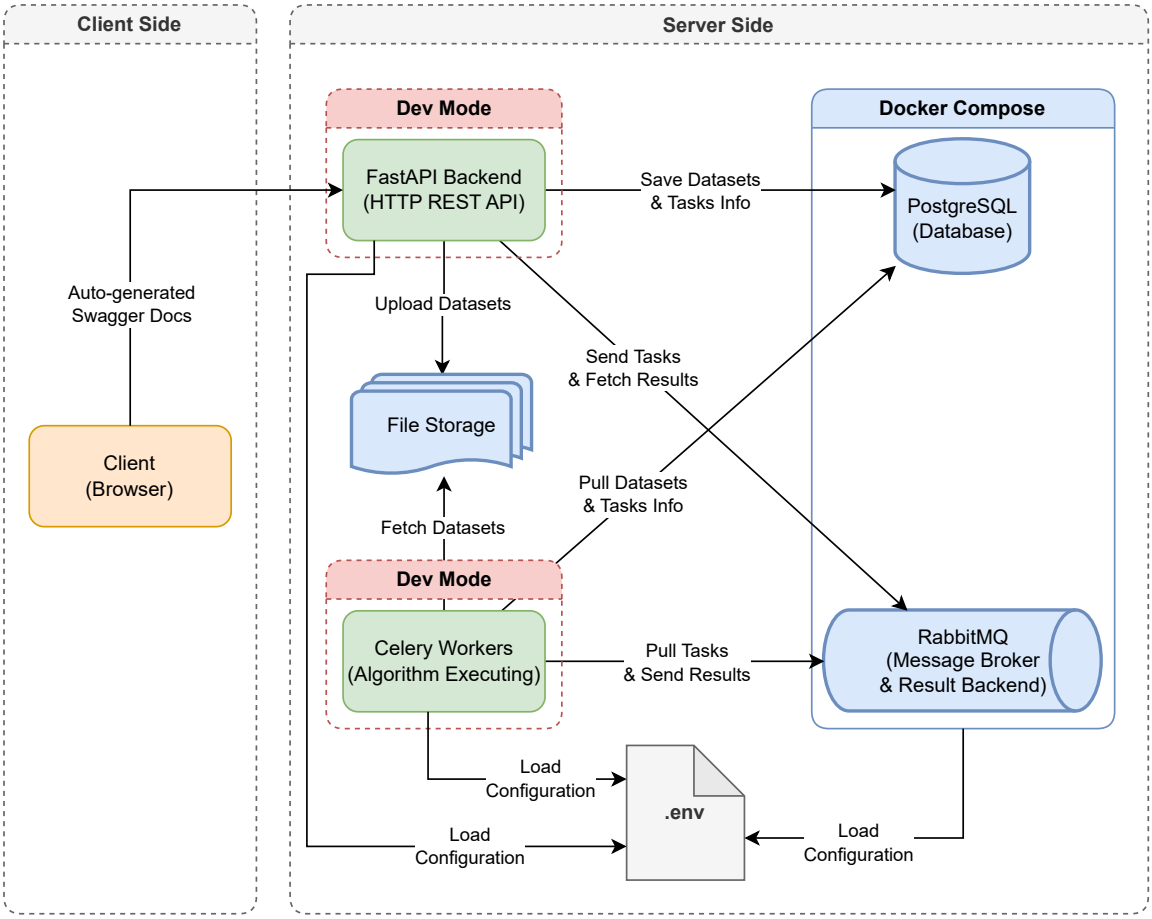

Рис. 2: Архитектура существующего решения

позволяет создавать изолированные среды для каждого компонента системы, что обеспечивает их независимость и упрощает управление зависимостями. Контейнеры Docker позволяют упаковать приложение с необходимыми библиотеками и конфигурациями в единую, легко переносимую единицу, которая может быть развернута на любой машине, поддерживающей Docker. Это минимизирует проблемы, связанные с различиями операционных систем, обеспечивая стабильность работы приложения независимо от платформы.

2.5.2. Упаковка основного приложения

Для Backend-сервиса был написан `Dockerfile`[4], который обеспечивает установку зависимостей, настройку окружения, и запуск веб-сервера

[4]Файл с инструкциями, используемый Docker для сборки образа.

в изолированной виртуальной среде. Однако полученный образ не используется при развертывании в существующем решении.

2.5.3. База данных и очередь сообщений

Для запуска контейнеров PostgreSQL и RabbitMQ используются готовые образы Docker `postgres:16.0-alpine` и `rabbitmq:3.12-management-alpine`, доступные на платформе Docker Hub[5]. Эти образы основаны на легковесном дистрибутиве Linux — Alpine, что позволяет снизить размер контейнеров и ускорить их загрузку. Использование готовых образов упрощает процесс развертывания и настройки сервисов с помощью переменных окружения.

2.5.4. Развертывание

Для развертывания и управления частью контейнеров используется утилита Docker Compose[6]. Она позволяет описать конфигурацию всех сервисов приложения в едином YAML-файле, что упрощает процесс развертывания и обеспечения взаимодействия между компонентами.

В текущем решении Docker Compose используется только для запуска базы данных PostgreSQL и очереди сообщений RabbitMQ. В файле конфигурации указываются используемые версии образов, порты для связи с внешними приложениями, а также переменные окружения, хранящиеся в файле `.env`, конфиденциальные данные в котором изначально заменены на заглушки и требуют ручной настройки. Благодаря этому подходу разработчики могут развернуть инфраструктуру, необходимую для разработки с помощью пары команд.

Backend и Celery Workers запускаются отдельно с использованием скриптов из файла `Makefile`[7] с поддержкой автоматической перезагрузки кода, что удобно для разработки, так как изменения сразу применяются без перезапуска сервисов вручную. Однако, этот подход требует значительных ресурсов, поскольку постоянно отслеживает изменения в

[5]Облачный сервис, который служит для хранения и распространения Docker-образов.

[6]https://docs.docker.com/compose/ (дата обращения: 29 ноября 2024 г.).

[7]Файл с инструкциями для утилиты Make, которая используется для управления проектом.

файлах и перезапускает сервисы, следовательно, увеличивает нагрузку на систему. Данный метод не подходит для production-окружения, где важны производительность и стабильность.

2.5.5. Преимущества и недостатки

В результате анализа существующего решения, были выявлены следующие преимущества:

- **Изоляция и независимость компонентов.** Docker обеспечивает изоляцию сервисов, что упрощает управление зависимостями и гарантирует стабильную работу приложения на различных платформах.
- **Упрощенное развертывание.** Docker Compose позволяет легко настраивать сервисы и взаимодействие между ними, а также масштабировать инфраструктуру.
- **Сохранность конфиденциальных данных.** Конфиденциальные данные, такие как пароли и токены, хранятся в файле `.env` и изначально заменены на заглушки. Это улучшает безопасность и упрощает настройку окружения.
- **Использование готовых образов.** Использование официальных Docker-образов упрощает развертывание и минимизирует время на настройку сервисов.

И недостатки:

- **Отсутствие применения миграций[8] к базе данных.** В текущем решении нет автоматического применения миграций, хотя соответствующая команда есть в `Makefile`. Это может привести к проблемам при изменении структуры базы данных.

[8]Миграция базы данных — это процесс изменения её структуры и содержимого с целью обновления версии, переноса на другую платформу или слияния с другой базой данных.

- **Отсутствие автоматического запуска обработчиков.** Процесс запуска Celery Workers не автоматизирован, что требует дополнительной настройки и управления.
- **Решение не готово к production-окружению.** Backend и Celery Workers запускаются в режиме разработки, что требует дополнительных ресурсов и не позволяет сервисам работать в оптимизированном режиме.
- **Отсутствие интерфейса для просмотра задач.** В решении нет пользовательского интерфейса для мониторинга и управления задачами обработки, что затрудняет отладку приложения.
- **Отсутствие разделения на стадии разработки и production.** Нет разделения конфигураций для разных окружений, что может привести к ошибкам в настройке при переходе на более высокие стадии разработки.
- **Отсутствие обратного прокси-сервера.** Решение не включает настройку обратного прокси-сервера[9] для маршрутизации запросов. Это ограничивает возможности масштабирования и защиты приложения в production-окружении.

2.6. Выводы

В рамках обзора были рассмотрены архитектура приложения, существующее решение для его развертывания и используемые при этом технологии, а также проведен анализ взаимодействия между внутренними сервисами и выявлены преимущества и недостатки текущего решения.

Архитектура приложения состоит из веб-сервера на FastAPI, Celery обработчиков, очереди сообщений RabbitMQ, базы данных PostgreSQL и файлового хранилища. Веб-сервер поставляет пользователю автоматически сгенерированную клиентскую часть, а также имеет доступ к

[9]Тип прокси-сервера, который ретранслирует запросы из внешней сети на сервера, логически расположенные во внутренней сети. Он выполняет роль барьера между клиентами и внутренними сервисами, в некоторых случаях может балансировать нагрузку, выполнять кеширование и фильтрацию входящего трафика.

базе данных, очереди сообщений и файловому хранилищу. Обработчики Celery получают задачи из RabbitMQ, используют данные из PostgreSQL и файлового хранилища, а результаты отправляют обратно в очередь. Все параметры для взаимодействий сервисов настраиваются через переменные окружения.

Полученные результаты показывают, что существующее решение может использоваться для разработки и тестирования, но требует значительных улучшений для подготовки к использованию в production-среде на удаленном сервере. В частности, требуется внедрение дополнительных компонентов, таких как обратный прокси-сервер для маршрутизации запросов, реализация автоматического запуска обработчиков и применения миграций к базе данных, а также улучшение управления конфигурациями для различных сред. Также необходимо предоставить интерфейс для просмотра и отладки задач обработки, что упростит мониторинг выполнения операций и ускорит процесс устранения ошибок.

3. Реализация

В соответствии с целью работы необходимо было разработать систему автоматической сборки, доставки и развертывания, соответствующую сформулированным требованиям. При этом особое внимание было уделено сохранению преимуществ существующего решения, а также устранению выявленных в результате анализа недостатков. Новый подход направлен на создание удобной, масштабируемой и безопасной инфраструктуры сервера, адаптированной для работы в production-режиме.

3.1. Используемые технологии

Для реализации нового решения были частично использованы наработки из существующего решения. В частности, сохранились такие элементы, как контейнеризация с помощью Docker, конфигурация через переменные окружения, применение готовых официальных образов, а также использование `Makefile` для объединения команд в скрипты и упрощения выполнения рутинных операций.

Кроме того, было принято решение использовать новые технологии и методологии, широко применяемые в современной практике DevOps, что позволяет упростить автоматизацию процессов и повысить эффективность системы.

3.1.1. GitHub организации, команды и роли

В рамках нового решения использовались возможности организации Desbordante[10] на GitHub [6]. GitHub-организация представляет собой централизованное пространство для хранения и управления репозиториями проекта.

Организации позволяют управлять правами доступа своих участников и обеспечивать централизованное администрирование ресурсов команды. В частности, возможность создания внутренних команд и на-

[10] https://github.com/Desbordante (дата обращения: 29 ноября 2024 г.).

значения ролей внутри организации помогает контролировать доступ к различным частям кода и инфраструктуры.

Это и последующее использование сервисов и возможностей GitHub обусловлено тем, что команда разработчиков проекта длительное время использует эту платформу для хранения и версионирования кода, взаимодействия между участниками и совместной работы. Это означает хорошую осведомленность участников команды о процессах работы с платформой, что существенно упрощает процесс взаимодействия с новой системой развертывания, а также ускоряет ее освоение и интеграцию в рабочий процесс.

Кроме того, GitHub и его сервисы в полной мере предоставляют возможности для реализации части требований к новой системе, такие как управление доступом, автоматизация CI/CD[11] процессов и обеспечивает централизованное хранение кода, что делает его хорошим выбором для управления инфраструктурой.

3.1.2. GitHub Actions и GitHub Container Registry

GitHub Actions — это инструмент автоматизации рабочих процессов, встроенный в платформу GitHub. Он позволяет создавать и управлять CI/CD пайплайнами — наборами инструкций для автоматической сборки, тестирования и развертывания приложений. GitHub Actions интегрируется с репозиториями GitHub, что упрощает запуск пайплайнов на основе событий, таких как коммиты, pull requests или выпуск релизов[12].

GitHub Container Registry — это реестр контейнеров, предоставляемый GitHub для хранения и управления Docker-образами. GitHub Container Registry интегрируется с организациями GitHub, что позволяет легко работать с образами в рамках одного экосистемного решения. Он предоставляет возможности для безопасного хранения контейнеров,

[11] CI/CD (Continuous Integration, Continuous Delivery) — это технология автоматизации сборки, тестирования и доставки модулей разрабатываемого проекта заинтересованным сторонам (разработчикам, аналитикам, инженерам качества, конечным пользователям и др.).

[12] Возможность GitHub создавать версии кода, которые помимо набора изменений часто сопровождаются бинарными файлами, документацией и пометкой о новых возможностях или исправлениях.

их версионирования и использования.

3.1.3. Kubernetes

Kubernetes [11] — это система оркестрации контейнеров, предназначенная для автоматического развертывания, масштабирования и управления контейнеризированными приложениями. Он признан стандартом в индустрии [21], обеспечивая автоматизацию процессов, высокую доступность и надежность работы приложений.

Для работы с Kubernetes необходимо понимание как минимум следующего набора терминов:

- **Кластер** — совокупность серверов, которые работают совместно для обеспечения отказоустойчивости и масштабируемости.
- **Нода (узел)** — отдельный сервер в кластере.
- **Под** — наименьшая единица развертывания, содержащая один или несколько контейнеров.
- **Ресурсы Kubernetes** — абстракции, такие как Deployment, Service, Volume и VolumeClaim, которые управляют развертыванием приложений, сетевыми соединениями и хранением данных соответственно.

Для взаимодействия с Kubernetes используется CLI-утилита `kubectl`, которая позволяет управлять кластерами и ресурсами, просматривать логи и диагностировать проблемы. Для настройки и изменения Kubernetes-манифестов без их непосредственной модификации используется инструмент Kustomize [17], встроенный в `kubectl`.

Преимущества Kubernetes:

- **Масштабируемость и высокая доступность.** Kubernetes может автоматически масштабировать количество экземпляров сервисов в зависимости от нагрузки, что обеспечивает стабильность работы даже в условиях пикового трафика.

- **Автоматическое восстановление.** В случае сбоя Kubernetes перезапускает контейнеры, заменяет неработающие узлы и поддерживает требуемое состояние приложения, сводя время простоя к минимуму.

- **Бесшовное обновление сервисов.** Kubernetes позволяет обновлять приложения постепенно, без прерывания работы для пользователей, направляя запросы только на работающие экземпляры.

- **Балансировка нагрузки.** Встроенные механизмы распределения запросов между доступными экземплярами сервисов обеспечивают равномерную загрузку ресурсов и более эффективное использование инфраструктуры.

- **Экосистема инструментов.** Для Kubernetes разработаны такие инструменты, как ArgoCD и Flux, упрощающие процессы доставки и развертывания приложений, облегчая управление их состоянием и ускоряя цикл разработки.

С учетом вышеперечисленных преимуществ, было решено заменить им Docker Compose в новом решении для повышения масштабируемости, отказоустойчивости и облегчения выполнения поставленных требований.

Для реализации был выбран облегченный дистрибутив Kubernetes — K3s [10], оптимизированный для использования в средах с ограниченными ресурсами. Он сохраняет основные функции Kubernetes, но снижает требования к памяти и процессору, что делает его идеальным выбором для малых и средних приложений.

3.1.4. Nginx Ingress Controller

Ingress Controller в Kubernetes управляет маршрутизацией запросов к сервисам кластера путем настройки ресурсов Ingress.

Выбор Ingress Controller на основе Nginx [12] обусловлен его доступностью, стабильностью, активной поддержкой сообщества и широкими

возможностями настройки, что делает его оптимальным для управления внешним трафиком [5].

3.1.5. ArgoCD

ArgoCD [1] — это GitOps [7] инструмент для автоматизации процессов развертывания приложений в Kubernetes. Он обеспечивает синхронизацию состояния приложения, описанного в Git-репозиториях, с текущим состоянием кластера Kubernetes, автоматически применяя необходимые изменения.

Выбор в пользу ArgoCD был сделан по следующему ряду причин:

- **Интеграция с Git и GitHub.** ArgoCD построен на принципах GitOps, обеспечивая прямую синхронизацию с Git-репозиториями. Поддерживаются такие платформы, как GitHub, GitLab, Bitbucket, а также другие Git-совместимые системы. Решение позволяет работать с защищенными репозиториями с использованием SSH-ключей, токенов доступа и других средств аутентификации.

- **Веб-интерфейс.** ArgoCD предлагает графический интерфейс, который позволяет в реальном времени управлять синхронизациями и развертываниями, в отличие от CLI-ориентированных решений, таких как Flux.

- **Поддержка сложных сценариев.** ArgoCD поддерживает конфигурацию приложений через Helm, Kustomize, YAML-манифесты и предоставляет дополнительные решения, такие как ApplicationSet, что упрощает управление сложными проектами.

- **Широкая экосистема и поддержка.** ArgoCD активно развивается, о чем свидетельствуют его статус зрелого проекта в CNCF[13] и регулярные обновления[14]. Инструмент также поддерживает и

[13]Cloud Native Computing Foundation (CNCF) — это фонд, основанный Linux Foundation, который распространяет open source-компоненты Cloud Native приложений, https://www.cncf.io/projects/argo/ (дата обращения: 29 ноября 2024 г.).

[14]https://github.com/argoproj/argo-cd/releases (дата обращения: 29 ноября 2024 г.).

интегрируется с другими решениями, например SSO-системами[15], такими как GitHub OAuth.

- **Управление доступом.** ArgoCD реализует собственную систему RBAC[16]. Эта система позволяет настраивать доступ к приложениям, синхронизациям и настройкам в зависимости от ролей, что значительно упрощает управление безопасностью.

3.1.6. ArgoCD ApplicationSet

ApplicationSet — это Kubernetes-контроллер, добавляемый ArgoCD, который упрощает автоматическое создание и управление множеством приложений. Он предоставляет декларативный способ описания приложений через шаблоны и динамически управляет их состоянием. Инструмент позволяет генерировать приложения на основе данных из различных источников, таких как Git, списки и другие[17].

Для автоматизации создания приложений в ArgoCD существуют два основных подхода:

- **App of Apps** — структура, где одно приложение создает управляет другими.
- **ApplicationSet** — более современный подход, основанный на генерации приложений из единого шаблона.

Выбор был сделан в пользу ApplicationSet, так как он значительно улучшает переиспользование кода, уменьшает дублирование и упрощает поддержку конфигураций.

3.1.7. Helm Charts

Helm [9] — это инструмент для управления Kubernetes-приложениями, который функционирует как пакетный менеджер. Он использу-

[15]Технология единого входа (Single Sign-On, SSO) — это возможность использовать один идентификатор для доступа ко всем разрешенным ИТ-ресурсам и системам.

[16]Управление доступом на основе ролей (RBAC) — это метод управления доступом к ресурсам, который основывается на назначении ролей пользователям.

[17]Генераторы, доступные для ApplicationSet https://argo-cd.readthedocs.io/en/stable/operator-manual/applicationset/Generators/ (дата обращения: 29 ноября 2024 г.).

ет концепцию Helm Charts — пакетов, содержащих ресурсы Kubernetes, необходимые для развертывания приложения. Helm упрощает управление конфигурациями, зависимостями и версиями, позволяя автоматически генерировать и обновлять ресурсы Kubernetes.

Преимущества Helm, используемые в решении:

- **Шаблонизация.** Helm использует шаблоны для генерации манифестов, что позволяет централизованно управлять конфигурациями и упрощать развертывание различных сервисов.
- **Параметризация.** Через файл `values.yaml` Helm позволяет изменять параметры приложений для разных окружений (например, staging и production), что упрощает управление конфигурациями и повышает гибкость.
- **Переиспользование кода.** Helm позволяет вынести повторяющиеся фрагменты конфигураций в файл `_helpers.tpl`, улучшая читаемость и упрощая поддержку приложений.

Другие поддерживаемые ArgoCD инструменты, такие как Kustomize, предлагают альтернативные способы управления конфигурациями и больше подходят для настройки уже существующих манифестов, что не подходит для развертывания приложений текущего решения.

3.1.8. ArgoCD Image Updater

ArgoCD Image Updater [2] — это расширение для ArgoCD, предназначенное для автоматизации обновлений контейнерных образов в Kubernetes-приложениях.

Расширение поддерживает два способа обновления: напрямую через ArgoCD и через Git с записью изменений в файлы `.argocd-source-<appName>.yaml`[18]. В решении используется второй

[18]Файлы, используемые для переопределения параметров https://argo-cd.readthedocs.io/en/stable/user-guide/parameters/#store-overrides-in-git (дата обращения: 29 ноября 2024 г.).

вариант, так как он обеспечивает лучшую синхронизацию состояния приложений с репозиторием и позволяет отслеживать изменения образов через Git.

Принцип работы:

- ArgoCD Image Updater отслеживает изменения в репозиториях образов, например GitHub Container Registry;
- Записывает обновления в Git-репозиторий в файл `.argocd-source-<appName>.yaml`.
- Состояние кластера синхронизируется с репозиторием через ArgoCD.

Достоинства:

- **Автоматизация обновлений.** Исключает необходимость вручную обновлять манифесты и позволяет ускорить процесс развертывания.
- **Настройка стратегии обновлений.** Возможность настраивать стратегию (semver[19], latest и т.д.) обновления путем аннотаций.
- **Разрабатывается командой ArgoCD.** Решение активно поддерживается и развивается командой ArgoCD.
- **Гибкость.** Работает с различными репозиториями контейнерных образов, включая GitHub Container Registry.

3.1.9. Celery Flower

Celery Flower — это веб-интерфейс для мониторинга и управления задачами Celery. Flower работает как отдельный сервер, подключаемый к кластеру Celery и обеспечивает визуализацию информации о задачах в реальном времени.

[19]Обновление до последней версии образа с учетом семантического версионирования.

В предлагаемом решении он используется для устранения недостатка предыдущей системы, где не был предоставлен подобный интерфейс, что затрудняло отладку приложения и отслеживание состояния задач в реальном времени.

3.2. Автоматизация сборки и доставки компонентов

3.2.1. Контейнеризация

Для развертывания сервисов Backend, Celery Workers и Flower было решено использовать единый Docker-образ, поскольку файлы сервисов Backend и Celery Workers уже были упакованы подобным образом с помощью `Dockerfile` в предыдущем решении.

В зависимости от переданной при запуске контейнера команды должен запускаться запрашиваемый сервис. Для реализации этого подхода были написаны три bash-скрипта: `start.sh`, `celery.sh` и `flower.sh`.

- **`start.sh`**. Этот скрипт выполняет миграции базы данных, приводя ее состояние к актуальному, и запускает веб-сервер основного приложения;
- **`celery.sh`**. Запускает обработчики Celery с использованием конфигурации, сформированной на основе файлов основного приложения. Она включает в себя данные для подключения к брокеру сообщений и согласование формата передачи сообщений;
- **`flower.sh`**. Ожидает доступности Celery Workers, после чего запускает веб-интерфейс Flower. В процессе устанавливаются данные для авторизации пользователей при доступе к интерфейсу.

Также был доработан сам `Dockerfile`, чтобы включить написанные скрипты в итоговый образ и настроить запуск `start.sh` в качестве команды по умолчанию. Это позволило обеспечить корректный запуск основного приложения при развертывании контейнера без необходимости указывать дополнительные команды.

```
#!/bin/bash

set -o errexit
set -o pipefail
set -o nounset

alembic -c internal/infrastructure/data_storage/relational/postgres/migrations/alembic.ini upgrade head
uvicorn --workers 1 --host 0.0.0.0 --port $APPLICATION_SERVER_PORT internal:app
```

Листинг 1: Скрипт `start.sh`, выполняющий миграции базы данных и запускающий веб-сервер приложения.

3.2.2. Автоматическая сборка и доставка к регистру образов

Для автоматизации сборки образов и их доставки в GitHub Container Registry был создан CI/CD-пайплайн для GitHub Actions. Этот пайплайн запускается при следующих событиях:

- коммиты в ветки `main`, `feat/ci-cd` и `dev`;
- создание релизов на GitHub. При создании релиза автоматически создается новый тег в формате `semver`, например, `v1.2.3`, что инициирует сборку образа с соответствующей версией;
- ручное добавление тега в `semver` формате.

В процессе работы пайплайн выполняет следующие действия:

1. Загружает код из репозитория.
2. Авторизуется в GitHub Container Registry с использованием токена доступа, поставляемого в переменных GitHub Actions.
3. Извлекает метаданные для формирования тегов образа. Теги могут включать хэш коммита (`sha`) и версии, соответствующие `semver`.
4. Собирает Docker-образ на основе `Dockerfile` проекта.
5. Отправляет образ в GitHub Container Registry с применением сформированных тегов.

Название версии образа формируется автоматически на основе одного из следующих источников:

- текущий хэш коммита (`sha`);
- семантическая версия, извлекаемая из тега (`v*.*.*`), если сборка инициирована созданием релиза.

Таким образом, поддерживается актуальность образов в контейнерном регистре и облегчается их использование при развертывании приложений.

3.3. Система автоматического развертывания

3.3.1. Архитектура

С точки зрения архитектуры, каждый сервис приложения Desbordante рассматривается как отдельное ArgoCD-приложение. Приложения создаются на основе Helm-шаблонов, которые используют параметры, определенные в соответствующих файлах конфигурации для каждого окружения. Эта структура позволяет управлять каждым компонентом инфраструктуры как независимым модулем, что упрощает обновления и масштабирование.

Структурно система разделена на три основных раздела:

- **Core.** Этот раздел включает конфигурационные файлы, отвечающие за настройку ядра системы:
  - `argocd-cm.yaml` — содержит основные настройки ArgoCD;
  - `argocd-image-updater-config.yaml` — задает параметры для ArgoCD Image Updater;
  - `argocd-rbac-cm.yaml` — отвечает за управление ролями и правами доступа;
  - `argocd-server-ingress.yaml` — определяет правила для входящего трафика через Ingress;

  - `root.yaml` — играет ключевую роль в автоматизации развертывания всей инфраструктуры.

- **Charts.** В папке `charts` находятся шаблоны Helm, которые используются для создания приложений ArgoCD. Шаблоны внутри папки `templates` позволяют генерировать ресурсы Kubernetes, предоставляя гибкость в параметризации через использование переменных.

- **Apps.** В разделе `apps` содержатся файлы конфигурации приложений для двух окружений:

  - **stage** — тестовое окружение для проверки изменений;
  - **prod** — рабочее окружение для эксплуатации.

Для первоначальной настройки системы, например, для задания доменных имен для окружений `stage` и `prod`, используются переменные окружения, заданные в файле `.env`.

Папка `volumes` служит хранилищем для персистентных данных, таких как базы данных или пользовательские датасеты. Данные из разных окружений хранятся в соответствующих подпапках, что позволяет разделить данные для тестового и рабочего окружений.

3.3.2. Административная панель

После создания состоящего из одной ноды кластера Kubernetes и установки всех необходимых зависимостей, для доступа к панели администратора ArgoCD был настроен Ingress `argocd-server-ingress.yaml`.

Для ArgoCD особенность заключается в том, что API сервера прослушивают единый порт, поддерживая одновременно несколько протоколов (gRPC и HTTPS). Это требует использования аннотации `nginx.ingress.kubernetes.io/ssl-passthrough` как показано на листинге 2, которая позволяет пробрасывать соединения напрямую к API серверу.

Данная настройка обеспечивает возможность доступа разработчиков к панели ArgoCD через доменное имя.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: argocd-server-ingress
  namespace: argocd
  annotations:
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  ingressClassName: nginx
  rules:
    - host: argo.$STAGE_DOMAIN
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: argocd-server
                port:
                  name: https
```

Листинг 2: Ingress, обеспечивающий доступ к панели администрирования ArgoCD.

3.3.3. App Helm Chart

Для унифицированного процесса развертывания сервисов был написан Helm Chart, включающий все необходимые ресурсы Kubernetes для удовлетворения потребностей любого из сервисов Desbordante.

Этот Helm Chart состоит из следующих элементов:

- **Chart.yaml** — метаинформация о шаблоне, включая имя, версию и описание. Обязательный элемент Helm Charts.
- **templates/** — каталог, содержащий шаблоны ресурсов Kubernetes, которые создаются при развертывании.

Каждый из этих ресурсов имеет свое назначение:

- **Deployment** — управляет созданием и обновлением подов. В этот шаблон передаются образ контейнера, порт, команда для запуска и переменные окружения, если они необходимы. Это основной ресурс для обеспечения масштабируемости и обновлений приложения.

- **Service** — создает сервис, обеспечивающий доступ к приложению внутри кластера. С помощью сервисов Kubernetes маршрутизирует трафик к правильным подам, основываясь на метках.

- **Ingress** — используется для настройки маршрутизации трафика к Service, включая маршруты для различных хостов и портов. Он помогает направить внешний трафик на нужные приложения внутри Kubernetes.

- **PersistentVolume** — управляет хранилищем данных. Этот ресурс создает постоянные тома, которые используются для хранения данных в Kubernetes, например, для баз данных. Он включает настройки объема, режима доступа и привязки к конкретным разделам файловой системы.

- **PersistentVolumeClaim** — позволяет приложению запросить хранилище с нужными параметрами, такими как объем и режим доступа.

Результат[20] развертывания такого Helm Chart изображен на Рис. 3. Благодаря этим ресурсам, развернутые приложения могут взаимодействовать друг с другом внутри инфраструктуры, быть доступными извне, а также сохранять персистентные данные.

[20]Ресурсы типа ReplicaSet и Pod создаются автоматически при развёртывании Deployment. Pod является «оболочкой» для контейнеров, а ReplicaSet отвечает за горизонтальное масштабирование сервиса.

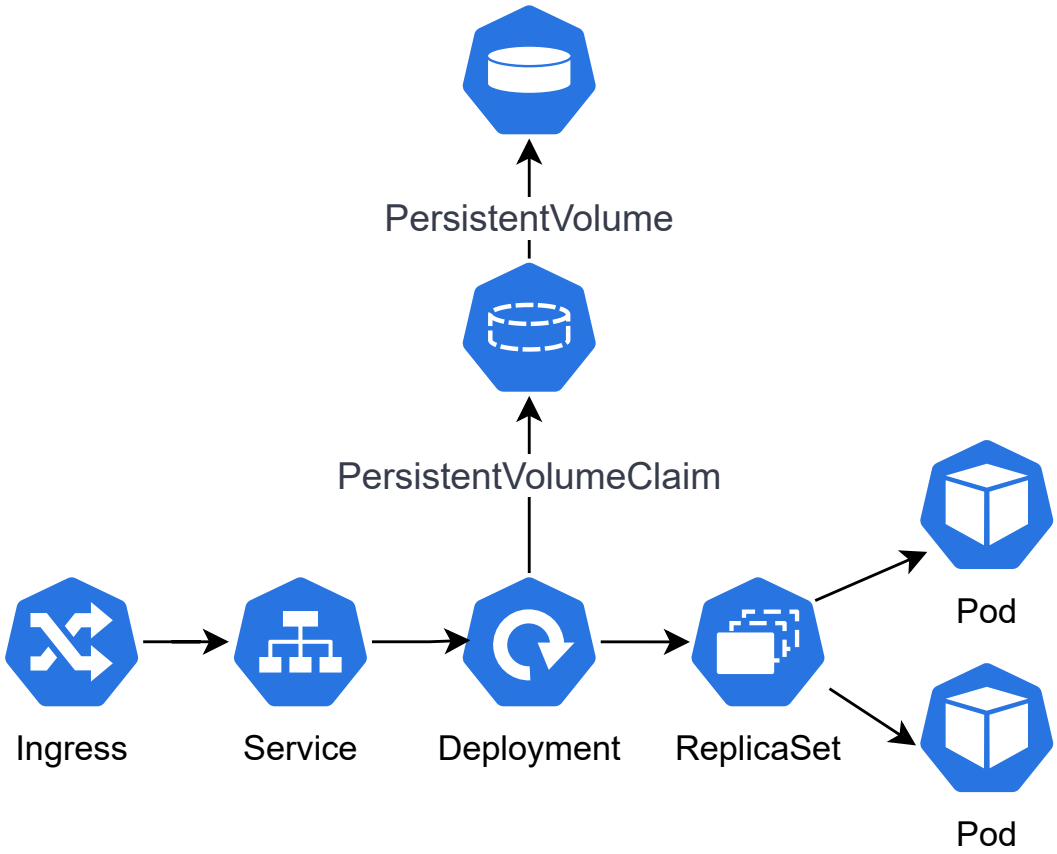

Рис. 3: Kubernetes-ресурсы, полученные в результате развертывания App Helm Chart.

3.3.4. Хранение конфиденциальных данных

Для безопасного хранения конфиденциальных данных, таких как пароли и API-ключи, системой используется Kubernetes-ресурс Secret. Этот ресурс позволяет хранить чувствительную информацию в зашифрованном виде, обеспечивая ее доступность только для тех подов и контейнеров, которым она предназначена.

3.3.5. Root ApplicationSet

Root ApplicationSet является важнейшим компонентом новой системы в автоматизации развертывания и обновления сервисов инфраструктуры. Он управляет процессом динамического создания и синхрониза-

ции приложений в различных окружениях на основе App Helm Chart.

Для создания ArgoCD Application объектов Root использует генератор на основе файлов Git-репозитория. Этот подход позволяет гибко конфигурировать приложения на основе файлов настроек путем изменения их через Git-коммиты, что упрощает управление и обновление сервисов.

Генератор выбирает файлы конфигурации, которые находятся в указанном репозитории, и на основе их содержимого создает параметры для каждого приложения, такие как имя, окружение в котором приложение должно быть развернуто, порты и другие настройки.

Root ApplicationSet развертывает приложения в двух окружениях: `stage` и `prod`. Для каждого из окружений используются отдельные настройки из соответствующих файлов конфигурации, что позволяет адаптировать параметры приложений в зависимости от среды.

Данный ApplicationSet также управляет стратегиями обновления образов для приложений. Для окружения `prod` используется стратегия на основе семантического версионирования (`semver`), что гарантирует стабильные и проверенные обновления [21]. В окружении `stage` применяется стратегия, обновляющая приложение на основе новейшего собранного образа, что позволяет быстрее тестировать новые версии и собранные изменения. Эти настройки управления версиями образов реализуются через аннотации, как показано на листинге 3.

```
1  {{- if (dig "autoUpdate" false .) }}
2      metadata:
3        annotations:
4          argocd-image-updater.argoproj.io/image-list: image={{ .image.name }}
5          {{- if eq .values.env "stage" }}
6          argocd-image-updater.argoproj.io/image.update-strategy: newest-build
7          argocd-image-updater.argoproj.io/image.allow-tags: regexp:^[0-9a-f]{7}
8          {{- end }}
9          {{- if eq .values.env "prod" }}
10         argocd-image-updater.argoproj.io/image.update-strategy: semver
```

[21] Поскольку образы с семантическим версионированием генерируются только при выпуске релиза на GitHub или при ручном добавлении тега, для production-окружения реализована полуавтоматическая система обновления версий.

```
11          {{- end }}
12          argocd-image-updater.argoproj.io/write-back-method: git:repocreds
13          argocd-image-updater.argoproj.io/git-branch: {{ .values.branch }}
14      {{- end }}
```

Листинг 3: Часть Root ApplicationSet, обеспечивающая выбор стратегии обновления образов на основе окружения.

3.3.6. Apps

В разделе Apps находятся файлы настроек сервисов для развертывания в окружениях `prod` и `stage`. Эти файлы содержат параметры, которые определяют конфигурацию сервисов и запрашиваемые ими возможности системы.

Каждый файл может включать следующие ключевые параметры:

- **autoUpdate** — флаг, указывающий, следует ли автоматически обновлять образы при появлении новых версий в регистре;
- **image** — параметры образа контейнера, включая имя образа и тег;
- **command** — команда, с помощью которой должен быть запущен контейнер;
- **subdomain** — поддомен, через который будет доступен сервис;
- **port** — порт, используемый сервисом для входящих соединений;
- **env** — переменные окружения, которые могут включать параметры для подключения к базе данных, очереди сообщений или других сервисов;
- **secrets** — чувствительные данные, такие как пароли и ключи доступа, предоставляемые сервисам;
- **volume** — параметры для подключения хранилища, включая путь монтирования, объем хранилища и режим доступа.

3.3.7. Ограничение доступа

Для ограничения доступа к административной панели ArgoCD в организации GitHub было создано OAuth-приложение, а также определена роль “devops”. В ArgoCD была настроена авторизация через GitHub OAuth с использованием системы разграничения прав RBAC.

В результате настройки все участники организации Desbordante имеют доступ к панели ArgoCD в режиме “readonly”, что позволяет, например, просматривать логи или следить за состоянием приложений. Полный доступ с возможностью управления панелью предоставлен только участникам с ролью “devops”, что обеспечивает защиту от несанкционированных изменений в системе.

3.3.8. Дополнительные возможности

При реализации системы также были учтены специфические требования к инфраструктуре и предоставлены дополнительные возможности для повышения гибкости и функциональности:

- **Совместный доступ к хранилищу данных.** Реализована возможность множественного доступа сервисов к одному Volume, что позволяет обработчикам Celery использовать сохраненные в файловом хранилище пользовательские датасеты.
- **Динамическое управление ресурсами.** В зависимости от заданных параметров определенные ресурсы Kubernetes могут быть включены или отключены. Например, если сервис не должен быть доступен извне, можно отключить Ingress, исключив его из процесса развертывания.
- **Автоматизация настройки.** Был написан `Makefile`, который выполняет ключевые этапы настройки системы, включая:
  - установку необходимых для работы зависимостей;
  - создание кластера Kubernetes;

– генерацию паролей для сервисов;

– настройку ключа доступа к репозиторию;

– запуск системы.

3.4. Результаты

В результате была разработана система автоматического развертывания приложений в Kubernetes, основанная на ArgoCD. Новая система позволяет автоматизировать весь процесс обновления приложений, обеспечивая их стабильную и бесшовную работу.

Все сформулированные требования к новой системе были успешно выполнены:

- Взаимодействие компонентов внутри системы, доступность для внешних пользователей и сохранность персистентных данных обеспечены использованием Kubernetes и его ресурсов Service, Ingress, PersistentVolume и PersistentVolumeClaim.

- Доступ разработчиков к административной панели настроен через интеграцию ArgoCD с GitHub OAuth, с разграничением прав доступа с помощью ролей RBAC.

- Автоматическое обновление компонентов реализовано с помощью ArgoCD Image Updater.

- Конфиденциальные данные хранятся в защищенных ресурсах Kubernetes Secret и генерируются при настройке системы.

- Поддержка двух независимых сред развертывания — staging и production — реализована с использованием ApplicationSet и переменных окружения, а полуавтоматическое обновление компонентов в production обеспечено через процесс управления релизами с помощью GitHub.

Процесс работы системы, изображенный на Рис. 4 можно описать следующим образом:

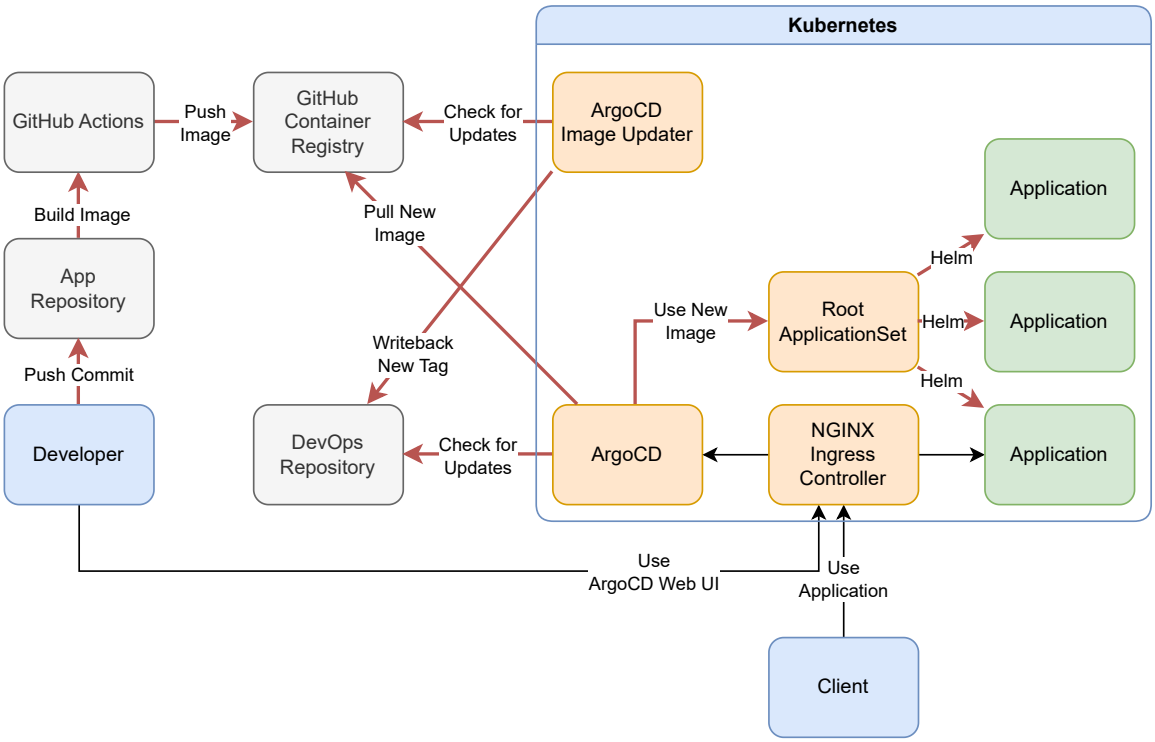

Рис. 4: Схема работы новой системы развертывания.

1. Разработчик выполняет коммит в репозиторий.
2. GitHub Actions автоматически собирает образ на основе `Dockerfile`.
3. Собранный образ отправляется в GitHub Container Registry.
4. ArgoCD Image Updater отслеживает появление новой версии образа и обновляет соответствующий файл в DevOps-репозитории.
5. ArgoCD замечает изменения и обновляет приложение в кластере Kubernetes.

Такая архитектура обеспечивает минимальные временные затраты на обновление приложений, снижает вероятность ошибок при ручном вмешательстве и позволяет быстро адаптироваться к изменениям.

4. Тестирование и сравнение

4.1. Тестирование

Проведено ручное тестирование реализованной системы и развернутой с ее помощью инфраструктуры. Тестирование включало следующие этапы:

- установка системы на удаленный сервер и проверка ее работоспособности;
- проверка автоматического развертывания инфраструктуры при запуске системы;
- тестирование доступа к административной панели через три типа GitHub аккаунтов:
  - пользователь, не являющийся участником организации Desbordante, не имел доступа;
  - участник организации без роли `devops` имел доступ только для просмотра;
  - разработчик с ролью `devops` имел полный доступ к управлению сервисами.
- проверка автоматической сборки образов и их загрузки в GitHub Container Registry после изменений в ветке `main`;
- проверка обновления сервисов Backend, Celery Workers и Flower в `staging` окружении до последней версии образа;
- проверка обновления указанных сервисов в production окружении до версии образа, связанных с релизом на GitHub;
- тестирование загрузки датасета, создания задачи обработки, получения ее статуса и результата через веб-интерфейс Backend;
- проверка доступности веб-интерфейса Flower и отображения списка задач обработки;

- проверка сохранения данных баз данных и пользовательских датасетов после перезагрузки системы и сервера.

Для тестирования использовался удаленный сервер от компании Beget с 2 ядрами CPU, 2 ГБ оперативной памяти и 10 ГБ NVMe хранилища.

Данные тесты подтвердили соответствие системы заявленным требованиям и ее готовность к использованию.

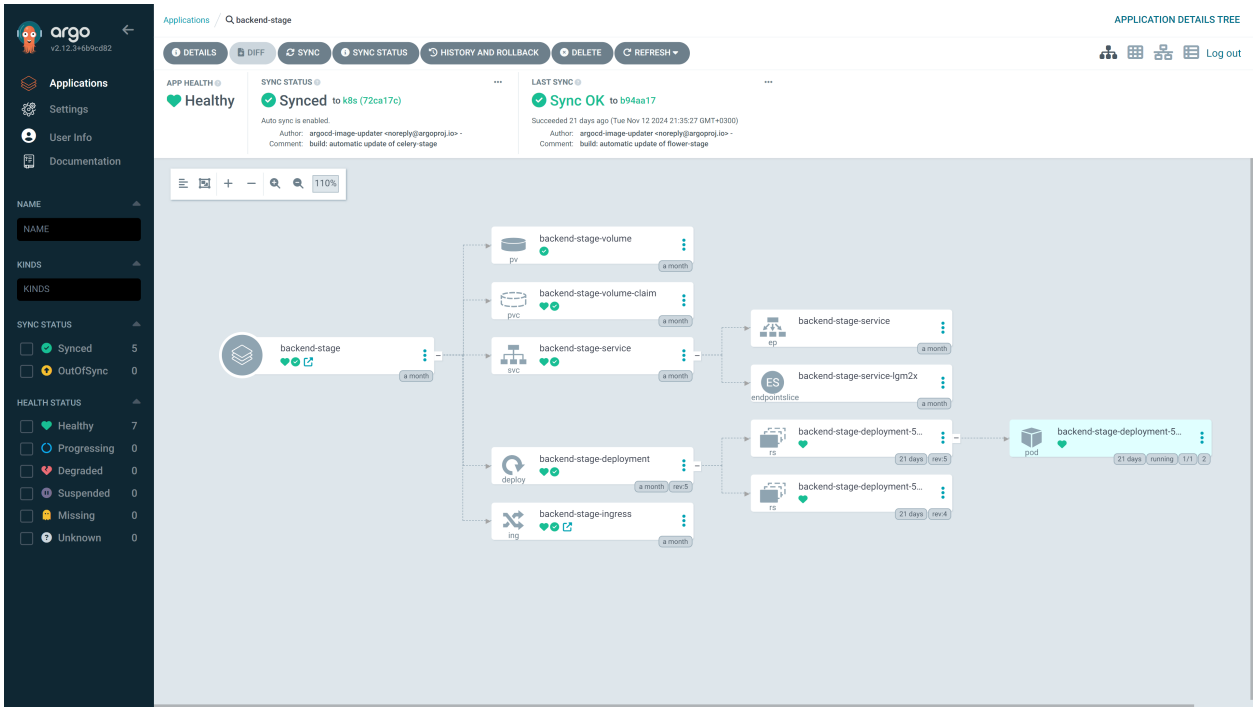

Рис. 5: Административная панель ArgoCD.

4.2. Сравнение с предыдущей системой

Ниже представлена Таблица 1, которая проводит сравнение нового решения с предыдущей системой.

Для подсчета строк кода использовалась утилита `cloc`, учитывались файлы `Makefile`, `Dockerfile`, CI/CD пайплайны, YAML-манифесты и скрипты запуска.

Исходя из таблицы, можно сделать вывод, что новая система сохранила ключевые преимущества старого решения. При этом удалось

Таблица 1: Сравнение старой и новой системы

Параметр	**Старая система**	**Новая система**
Контейнеризации	Docker	Docker
Развертывание	Docker Compose	Kubernetes и ArgoCD
Хранение паролей	`.env`	Kubernetes Secrets
Использование готовых образов	Да	Да
Применение миграций БД	Вручную	Автоматически
Запуск Celery Workers	Вручную	Автоматически
Интерфейс просмотра задач обработки	Нет	Flower
Поддержка нескольких окружений	Нет	staging и production
Обратный прокси-сервер	Нет	Nginx
Количество строк кода	96	657

устранить недостатки, такие как отсутствие автоматического применения миграций базы данных, автоматического запуска Celery Workers, и отсутствие интерфейса для мониторинга задач. Вдобавок, новая система обеспечивает поддержку нескольких окружений, таких как staging и production, и интегрирует обратный прокси-сервер Nginx.

Однако, несмотря на улучшения, новая система стала значительно сложнее в настройке и управлении. Внедрение Kubernetes, ArgoCD и других современных технологий требует большего уровня знаний и опыта в области DevOps, что делает систему более подходящей для опытных разработчиков, но и более мощной и гибкой в долгосрочной перспективе.

5. Заключение

В результате семестровой практики была спроектирована и реализована система автоматической сборки, доставки и развертывания на удаленном сервере инфраструктуры приложения Desbordante. А именно, были выполнены следующие задачи:

1. проведен обзор:
    - архитектуры приложения и проанализированы взаимодействия между его сервисами;
    - существующего решения, а также анализ его преимуществ и выявление недостатков;
2. с помощью GitHub Actions реализована автоматическая сборка необходимых для работы приложения компонентов в Docker-образ;
3. организована доставка полученного образа в GitHub Container Registry и обеспечен доступ системы развертывания к нему;
4. разработана система развертывания инфраструктуры в Kubernetes на основе ArgoCD, удовлетворяющая заданным требованиям;
5. проведена проверка работоспособности реализованной системы и развернутого с ее помощью приложения, включая загрузку датасета, создание задачи обработки, получение ее статуса и результата.

Ссылка на GitHub-репозитории системы развертывания: https://github.com/Desbordante/desbordante-devops/tree/k8s

И серверной части с системой сборки и доставки: https://github.com/Desbordante/desbordante-server (имя пользователя — vladyoslav).

Список литературы

[1] Argo Project. — Argo CD Documentation, 2024. — URL: https://argo-cd.readthedocs.io/ (дата обращения: 29 ноября 2024 г.).

[2] Argo Project. Argo CD Image Updater. — 2024. — URL: https://argocd-image-updater.readthedocs.io/ (дата обращения: 29 ноября 2024 г.).

[3] Celery Project. Celery - Distributed Task Queue. — 2024. — URL: https://docs.celeryproject.org (дата обращения: 29 ноября 2024 г.).

[4] Docker Inc. — Docker Documentation, 2024. — URL: https://docs.docker.com (дата обращения: 29 ноября 2024 г.).

[5] Flant. Обзор и сравнение контроллеров Ingress для Kubernetes. — 2019. — URL: https://habr.com/ru/companies/flant/articles/447180/ (дата обращения: 29 ноября 2024 г.).

[6] GitHub Inc. — GitHub Documentation, 2024. — URL: https://docs.github.com/ (дата обращения: 29 ноября 2024 г.).

[7] GitOps Community. GitOps is Continuous Deployment for cloud native applications. — 2024. — URL: https://www.gitops.tech/ (дата обращения: 29 ноября 2024 г.).

[8] Guide to General Server Security : Rep. : Special Publication 800-123 / National Institute of Standards and Technology (NIST) ; Executor: Karen Scarfone, Wayne Jansen, Murugiah Souppaya, Amanda Cody : 2008. — URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf (дата обращения: 29 ноября 2024 г.).

[9] Helm Authors. — Helm Documentation, 2024. — URL: https://helm.sh/docs/ (дата обращения: 29 ноября 2024 г.).

[10] K3s Project Authors. — K3s Documentation, 2024. — URL: https://docs.k3s.io/ (дата обращения: 29 ноября 2024 г.).

[11] Kubernetes Authors. — Kubernetes Documentation, 2024. — URL: https://kubernetes.io/docs/ (дата обращения: 29 ноября 2024 г.).

[12] Kubernetes Community. — Ingress NGINX Controller Documentation, 2024. — URL: https://kubernetes.github.io/ingress-nginx/ (дата обращения: 29 ноября 2024 г.).

[13] Nurmuhametov Rafik. — Backend architecture refactoring. — Desbordante, 2024. — spring. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/Backend%20architecture%20refactoring%20-%20Rafik%20Nurmuhametov%20-%202024%20spring.pdf (дата обращения: 29 ноября 2024 г.).

[14] Pautasso Cesare, Wilde Erik. Introduction // REST: From Research to Practice / Ed. by Erik Wilde, Cesare Pautasso. — New York, NY : Springer New York, 2011. — P. 1–18. — ISBN: 978-1-4419-8303-9. — URL: https://doi.org/10.1007/978-1-4419-8303-9_0 (дата обращения: 29 ноября 2024 г.).

[15] Pivotal Software. — RabbitMQ Documentation, 2024. — URL: https://www.rabbitmq.com/documentation.html (дата обращения: 29 ноября 2024 г.).

[16] Ramírez Sebastián. — FastAPI Documentation, 2024. — URL: https://fastapi.tiangolo.com (дата обращения: 29 ноября 2024 г.).

[17] SIG-CLI. — Kustomize Documentation, 2024. — URL: https://kustomize.io/ (дата обращения: 29 ноября 2024 г.).

[18] Sulyman Shakirat. Client-Server Model. — 2015. — URL: https://www.researchgate.net/profile/Shakirat-Sulyman/publication/271295146_Client-Server_Model/links/

5864e11308ae8fce490c1b01/Client-Server-Model.pdf (дата обращения: 29 ноября 2024 г.).

[19] Swagger RESTful API Documentation Specification, 2024. — URL: https://docs.swagger.io/spec.html (дата обращения: 29 ноября 2024 г.).

[20] The PostgreSQL Global Development Group. — PostgreSQL Documentation, 2024. — URL: https://www.postgresql.org/docs/ (дата обращения: 29 ноября 2024 г.).

[21] VK Team. Как жили до Kubernetes: сравниваем самый популярный оркестратор с другими решениями. — 2021. — URL: https://habr.com/ru/companies/vk/articles/543232/ (дата обращения: 29 ноября 2024 г.).

[22] Vadim Yakshigulov. — Backend Reimplementation. — Desbordante, 2023. — spring. — URL: https://github.com/Desbordante/desbordante-core/blob/main/docs/papers/Backend%20reimplementation%20-%20Yakshigulov%20Vadim%20-%202023%20spring.pdf (дата обращения: 29 ноября 2024 г.).