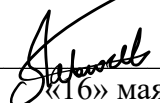


**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Образовательная программа «Программная инженерия»

УТВЕРЖДАЮ  
Академический руководитель  
образовательной программы  
«Программная инженерия»,  
старший преподаватель департамента  
программной инженерии

 Н.А. Павлов  
«16» мая 2025 г.

**Выпускная квалификационная работа**

на тему «Управляемые базы данных в Kubernetes как сервис. Модуль  
автоматизации взаимодействия с базами данных.»

по направлению подготовки 09.03.04 «Программная инженерия»

Научный руководитель  
Профессор, НИУ ВШЭ

Должность, место работы


Профессор ДПИ, к.т.н

ученая степень, ученое звание

**В.В. Шилов**

И.О. Фамилия

Подпись, Дата


 16.05.2025

Выполнил студент группы БПИ217  
4 курса бакалавриата  
образовательной программы  
«Программная инженерия»

**С. А. Мохов**

И.О. Фамилия

Подпись, Дата

 16.05.2025


Научный руководитель  
Приглашенный преподаватель,  
НИУ ВШЭ

Должность, место работы

**Л.В. Немировский**

И.О. Фамилия

Подпись, Дата

 16.05.2025

**Москва 2025**

## Реферат

На данный момент облачная инфраструктура является наиболее распространенным решением для развертывания крупных платформ. Одной из ключевых проблем облачной инфраструктуры является нехватка автоматизации в обслуживании распределенных баз данных. Каждая компания стремится реализовать собственное решение для развертывания и дальнейшего обслуживания баз данных в рамках своей инфраструктуры. Данная работа призвана предоставить решение с открытым исходным кодом для автоматизации управления распределенными базами данных в облачной инфраструктуре Kubernetes.

Общий объем отчета: 74 страницы, 3 главы, 34 иллюстрации, 2 таблицы, 102 источника и 17 листингов.

Ключевые слова: облачные технологии, облачная инфраструктура, Kubernetes, облачный оператор, Kubernetes оператор, CloudOps, управляемые базы данных, управляемые базы данных как сервис, обслуживание баз данных, распределенные базы данных, автоматизация мониторинга, автоматизация эксплуатации баз данных, MongoDB, автоматизация MongoDB, управляемая база данных MongoDB.

## Abstract

At the moment, cloud infrastructure is the most common solution for deploying platforms. One of the key problems of cloud infrastructure is the lack of automation in the maintenance of distributed databases. Each company strives to implement its own solution for the deployment and further maintenance of databases within its infrastructure. This work aims to provide an open source solution for automating the management of distributed databases in the Kubernetes cloud infrastructure.

The work contains 74 pages, 3 chapters, 34 figures, 2 tables, 102 sources and 17 listings.

Keywords: cloud technologies, cloud infrastructure, Kubernetes, cloud operator, Kybernetes operator, CloudOps, managed databases, managed databases as a service, database maintenance, distributed databases, monitoring automation, database operation automation, MongoDB, MongoDB automation, managed MongoDB.

# Содержание

Глоссарий . . . . .	5
Введение . . . . .	6
<b>1 Глава 1. Предметная область и обзор существующих решений . . . . .</b>	<b>7</b>
1.1 Описание предметной области и актуальность . . . . .	7
1.2 Обзор рынка . . . . .	8
1.3 Сравнительный анализ аналогов . . . . .	9
1.4 Выводы по главе . . . . .	10
<b>2 Глава 2. Проектирование системы и технологии разработки . . . . .</b>	<b>11</b>
2.1 Проектирование системы и технологии разработки . . . . .	11
2.2 Выбор технологий реализации . . . . .	15
2.3 Вывод по главе . . . . .	23
<b>3 Глава 3. Программная реализация системы . . . . .</b>	<b>24</b>
3.1 Описание схемы базы данных и ассоциируемых сущностей . . . . .	24
3.2 Реализация модуля MDB . . . . .	31
3.3 Описание API . . . . .	52
3.4 Объем выполненной работы . . . . .	64
3.5 Вывод по главе . . . . .	67
<b>Заключение . . . . .</b>	<b>68</b>
<b>Список используемой литературы . . . . .</b>	<b>69</b>

# Глоссарий

В настоящем отчете применяются следующие термины с соответствующими определениями:

- 1) CloudOps - практики автоматизации инфраструктурных процессов в облачных средах.
- 2) DBaaS (Databases as a service) - платформа управляемых баз данных.
- 3) DBaaS агент (MDB агент) - агент, используемый DBaaS платформой для автоматизации баз данных.
- 4) MDB (Managed databases) — модуль DBaaS платформы, обеспечивающий управление базами данных в среде Kubernetes.
- 5) OnyxDB - внутреннее название DBaaS платформы, используемое в разработке.
- 6) On-premise развертывание - развертывание сервиса внутри собственной инфраструктуры.
- 7) Биллинг ресурсов - процесс сбора данных о потреблении ресурсов.
- 8) Кластер - объединение нескольких реплик базы данных.
- 9) Облачная инфраструктура - среда для развертывания контейнеризированных приложений.
- 10) Облачный оператор - средство автоматизации управления ресурсами, размещенных в облачной среде.
- 11) Продукт - абстракция в рамках DBaaS платформы, которая объединяет группу людей в подразделение и предоставляет квоту на ресурсы для управления базами данных.
- 12) Проект - абстракция в рамках DBaaS платформы, которая объединяет в себе группу кластеров баз данных и привязана к продукту, который предоставляет квоту на ресурсы.

# Введение

Одним из наиболее быстро развивающихся направлений являются облачные технологии, которые предоставляют удобный функционал для развертывания и дальнейшего обслуживания больших распределенных систем. В ходе развития современных технологий и паттернов проектирования микросервисная архитектура стала наиболее распространенной при организации взаимодействия бизнес-логики крупных платформ. С переходом на микросервисную архитектуру и облачные технологии многие компании столкнулись с проблемой автоматизации инфраструктурных процессов. Это побудило крупные компании начать разработку собственных решений для различных облачных платформ. К одной из таких проблем можно отнести нехватку автоматизации при эксплуатации распределенных баз данных. Исследуя доступные предложения, можно столкнуться с рядом ограничений, которые могут быть неприемлемы для потенциального пользователя.

Цель данной работы заключается в автоматизации управления распределенными базами данных в облачной инфраструктуре Kubernetes[11] посредством разработки модуля управления базами данных. В качестве базы данных для автоматизации выбрана база данных MongoDB[12].

Реализация поставленной задачи включает в себя следующий список задач:

- 1) Анализ существующих решений для Kubernetes и других проприетарных облаков.
- 2) Изучение паттернов проектирования для автоматизации облачной инфраструктуры.
- 3) Изучение специфики базы данных MongoDB.
- 4) Проектирование отказоустойчивой и масштабируемой архитектуры платформы.
- 5) Разработка модуля MDB.
- 6) Автоматизация процессов мониторинга.
- 7) Интеграция модуля MDB с модулем IDM.
- 8) Интеграция модуля MDB с модулем Web.

Первая глава содержит описание предметной области и сравнительный анализ с обоснованием актуальности выполняемой работы.

Вторая глава описывает архитектурные решения, а также выбор средств реализации программного решения.

В третьей главе рассмотрена реализация поставленных задач с описанием ключевых элементов системы.

В заключении описаны выводы по итогам выполненной работы с планом дальнейшего развития проекта.

# 1. Глава 1. Предметная область и обзор существующих решений

## 1.1. Описание предметной области и актуальность

На данный момент облачная инфраструктура является распространенным окружением для развертывания масштабируемых информационных систем. Облачная инфраструктура предоставляет обширные возможности для развития платформ, состоящих из множества сервисов, однако одной из ключевых проблем является необходимость автоматизации процессов эксплуатации распределенных систем с учетом потенциального масштабирования. Одним из таких процессов является автоматизация управления базами данных.

Анализируя возможные решения для автоматизации эксплуатации баз данных, можно встретить различные технологии, однако при подробном изучении функционала и ключевых особенностей потенциальный пользователь может столкнуться с рядом ограничений, которые не удовлетворяют его потребностям. Среди таких ограничений:

- Недоступность сервиса на территории РФ.
- Использование проприетарной облачной технологии, что требует передачи персональных данных третьим лицам.
- Недостаточность функционала в существующих открытых решениях.

Таким образом, актуальность данной работы обусловлена рядом ограничений в существующих решениях. Разрабатываемый программный продукт призван предоставить платформу для управления базами данных в облачной инфраструктуре Kubernetes[11] с поддержкой базы данных MongoDB[12], которая позволит автоматизировать рутинные процессы, предоставить расширенный функционал для конечного пользователя и централизовать инфраструктуру, связанную с эксплуатацией баз данных. Ключевой особенностью платформы является предоставление решения с открытым исходным кодом.

## 1.2. Обзор рынка

При изучении доступных решений в направлении Managed databases as a Service были выделены следующие решения для сравнительного анализа: Yandex Cloud[13] MDB, Selectel[14] MDB, KubeDB[15], Percona Everest[17]. Данные продукты представляют из себя решения как от крупных компаний с проприетарными облаками, так и решения для развертывания в среде Kubernetes.

### 1.2.1. Yandex Cloud MDB

Yandex Cloud является облачной платформой от крупной российской компании Яндекс и предоставляет свою реализацию облачной инфраструктуры на платной основе. Yandex Cloud MDB предлагает широкий выбор функционала для построения масштабируемой платформы, однако к ключевому недостатку платформы можно отнести необходимость использования проприетарного облака от компании Яндекс, что не позволяет получить исходный код технологии, вынуждает адаптировать свою инфраструктуру к работе в Yandex Cloud, а так же хранить персональные данные в инфраструктуре Yandex Cloud.

### 1.2.2. Selectel MDB

Selectel MDB от компании Selectel является решением схожим с технологией от Yandex Cloud. Ему также свойственны все недостатки, озвученные для технологии Yandex Cloud.

### 1.2.3. KubeDB

KubeDB является Managed databases платформой от иностранной компании AppCode[16] для развертывания баз данных в облачной инфраструктуре Kubernetes. KubeDB позволяет развернуть решение в персональной инсталляции Kubernetes, однако компания не предоставляет бесплатную версию и исходный код проекта недоступен пользователю. Также не реализован web интерфейс для управления базами данных, что вынуждает оперировать системой через консольную утилиту.

### 1.2.4. Percona Everest

Percona Everest является молодым решением от иностранной компании Percona, разработка которого началась весной 2024 года. Решение позволяет развернуть сервис в собственной инсталляции Kubernetes и имеет web интерфейс для взаимодействия с платформой. Исходный код проекта находится в открытом доступе, однако текущая версия имеет ограниченный функционал с точки зрения управления базой данных MongoDB, автоматизации процессов мониторинга и контроля доступов.



### 1.3. Сравнительный анализ аналогов

Таблица 1 — Сравнительный анализ продуктовых характеристик решений для управления базами данных

Критерий	Платформа DBaaS	Yandex Cloud MDB	Selectel MDB	KubeDB	Percona Everest
Доступно в России	+	+	+	-	+
Развертывание on-premise	+	-	-	+	+
Нет передачи данных третьим лицам	+	-	-	-	+
Открытый исходный код	+	-	-	-	+
Наличие бесплатной версии	+	-	-	-	+

Таблица 2 — Сравнительный анализ функциональных характеристик решений для управления базами данных в бесплатной версии

Критерий	Платформа DBaaS	Yandex Cloud MDB	Selectel MDB	KubeDB	Percona Everest
Горизонтальное масштабирование	+	-	-	-	+
Вертикальное масштабирование	+	-	-	-	+
Резервное копирование	+	-	-	-	+
Восстановление из резервной копии	+	-	-	-	+
Управление сущностями базы данных	+	-	-	-	-
Автоматизация поставки метрик	+	-	-	-	-
Автоматизация поставки логов	+	-	-	-	-
Квотирование ресурсов	+	-	-	-	-
Биллинг ресурсов	+	-	-	-	-

## 1.4. Выводы по главе

Проведя анализ доступных решений на рынке, были выявлены значительные недостатки или ограничения. К ключевым ограничениям можно отнести недоступность сервисов на территории РФ, недостаточность функционала, а так же необходимость использования проприетарного облачного решения от российских компаний с дальнейшей передачей данных третьим лицам.

Таким образом, данная работа призвана предоставить бесплатное решение с открытым исходным кодом для управления базами данных в облачной среде Kubernetes на примере базы данных MongoDB, реализовав востребованный функционал для эксплуатации распределенных баз данных в облачной среде Kubernetes с учетом предоставляемых услуг на рынке.

## 2. Глава 2. Проектирование системы и технологии разработки

### 2.1. Проектирование системы и технологии разработки

#### 2.1.1. Выбор архитектуры системы

В основе DBaaS платформы лежит монолитное приложение (сервис DBaaS), которое является ядром платформы и управляет ее состоянием. В составе сервиса можно выделить 2 больших модуля: MDB и IDM. В рамках данной работы выполняется реализация модуля MDB (Managed databases as a Service).

Выбор монолитной архитектуры главного модуля обусловлен тем, что платформа является инфраструктурным решением, которое не ожидает крайне высокой нагрузки и поставляется для развертывания on-premise. На текущей стадии развития продукта монолитная архитектура для управляющего компонента является наиболее сбалансированным решением для быстрой разработки и упрощенного внедрения в сторонние кластера Kubernetes[11].

В зоне ответственности модуля MDB находится следующий функционал платформы:

- Хранение данных кластеров баз данных под управлением MDB.
- Предоставление API для управления кластерами баз данных под управлением MDB.
- Асинхронная обработка запланированных задач по эксплуатации кластеров баз данных под управлением MDB.
- Квотирование ресурсов кластеров баз данных под управлением MDB.
- Биллинг ресурсов кластеров баз данных под управлением MDB.
- Автоматизация процесса поставки метрик кластеров баз данных под управлением MDB.
- Автоматизация процесса поставки логов кластеров баз данных под управлением MDB.
- Интеграция платформы с системой контроля доступов.
- Интеграция платформы с web интерфейсом платформы.

Помимо монолитного управляющего модуля в рамках данной работы разрабатывается DBaaS агент (MDB агент), который выполняет вспомогательные операции над кластерами баз данных. Также платформа интегрирована со следующими технологиями, обоснование и описание применения которых приведено в разделе 2.2:

- PostgreSQL[21]
- ClickHouse[22]
- Redis[23]
- Minio[41]
- Percona MongoDB operator[42]
- VictoriaMetrics[20]

- VictoriaMetrics operator[44]
- VictoriaMetrics agent[43]
- VictoriaLogs[45]
- Vector agent[46]

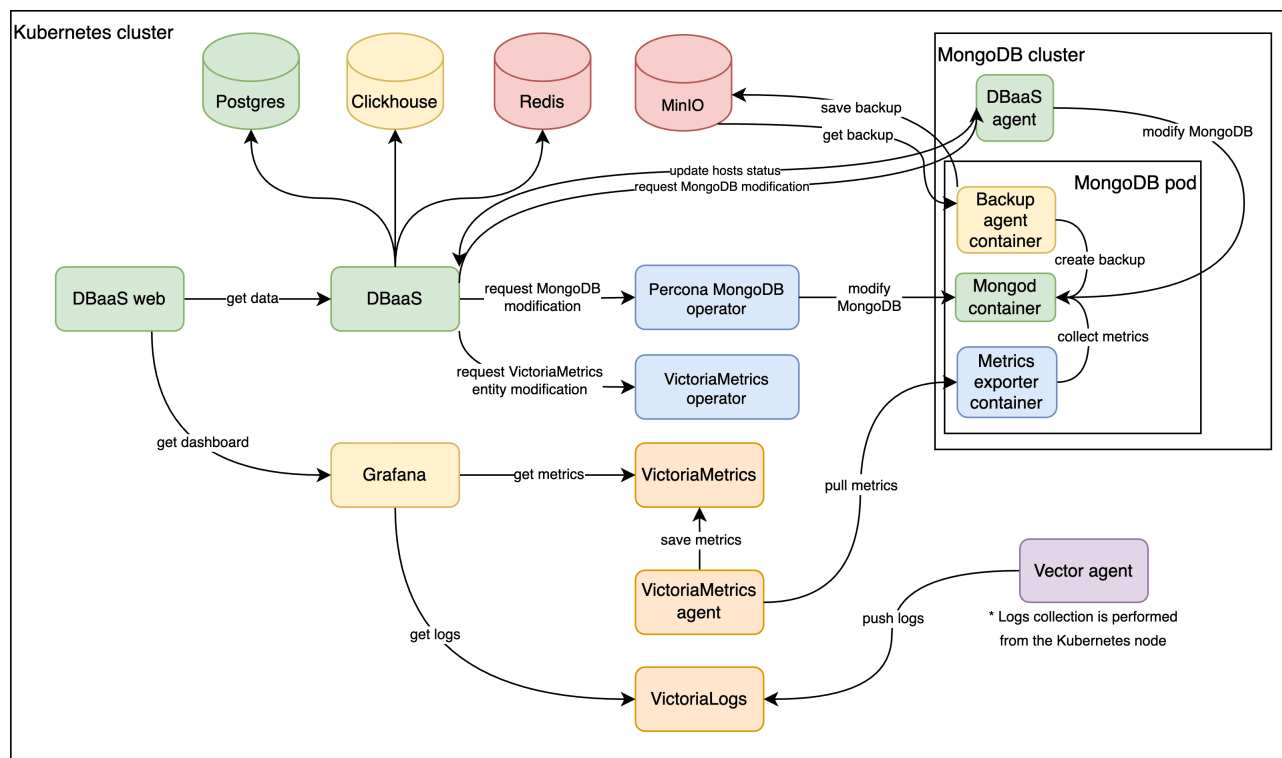


Рис. 1 — Общая архитектура DBaaS платформы

## 2.1.2. Паттерны проектирования и концепции взаимодействия

### 2.1.2.1. Концепция «Control Plane» и «Data Plane»

В основе архитектуры платформы лежит концепция «Control Plane/Data Plane»[47], которая применяется как в сетевых технологиях, так и при проектировании распределенных систем и облачных платформ. В контексте платформы модуль MDB является Control Plane, основная задача которого заключается в управлении инфраструктурой платформы и распределении задач внутри платформы. В свою очередь реализацией Data Plane в рамках платформы являются DBaaS agent и Persona MongoDB operator, которые принимают запросы на модификацию баз данных MongoDB в декларативном виде, а затем выполняют низкоуровневую логику для применения изменений.

Схожая архитектура применяется в Yandex Cloud[13] MDB и описана в статье «Как устроены сервисы управляемых баз данных в Яндекс.Облаке»[67].

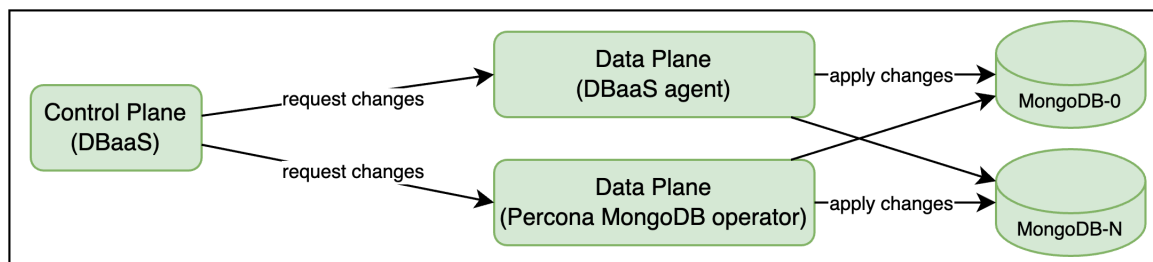


Рис. 2 — Реализация концепции «Control Plane» и «Data Plane» в DBaaS платформе

#### 2.1.2.2. Паттерн проектирования «Operator»

Паттерн проектирования «Operator»[48] является стандартным подходом для автоматизации ресурсов в облачной инфраструктуре Kubernetes. Данный подход предполагает, что оператор ожидает декларативное представление желаемого состояния ресурса от пользователя и выполняет операции согласования ресурса для приведения его в запрашиваемое состояние. Интегрированные технологии Percona MongoDB operator и VictoriaMetrics operator реализованы с использованием описанного выше паттерна.

Примером применения паттерна служит взаимодействие модуля MDB с Percona MongoDB operator. Модуль MDB формирует декларативное представление ресурса Percona MongoDB Server[49], далее обновляет сформированный ресурс через Kubernetes API. В свою очередь, Percona MongoDB operator получает событие обновления ресурса Percona MongoDB Server через Kubernetes API и выполняет модификацию кластера MongoDB, после чего отмечает ресурс, как синхронизированный.

Подобное применение паттерна «Operator» описывается в докладе «Паттерны управления базами данных в multi-cluster Kubernetes среде»[68] от компании Avito[69].

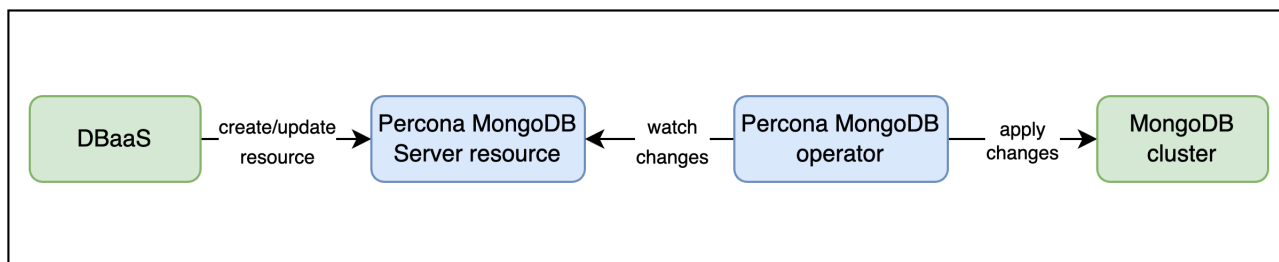


Рис. 3 — Реализация паттерна проектирования «Operator» в DBaaS платформе

#### 2.1.2.3. Паттерн проектирования «Transactional outbox»

Модуль MDB применяет паттерн проектирования «Transactional outbox»[50] при реализации распределенной очереди задач на обслуживание кластеров баз данных. Очередь задач написана с использованием базы данных PostgreSQL и поддерживает асинхронное выполнение ряда задач одновременно. Модуль MDB принимает заявки на обновления кластера, затем формирует список задач, которые требуются для этой операции и планирует их выполнение. Таким образом, модуль MDB предоставляет быстрое API без блокировки на исполнение операций.

Пример реализации паттерна «Transactional outbox» приводится в докладах от компании Т-Банк[58] «Надежно отправляем события в Apache Kafka. От CDC до паттерна Transactional

Outbox»[72] и «Разработка распределенной очереди с отложенными задачами»[71].

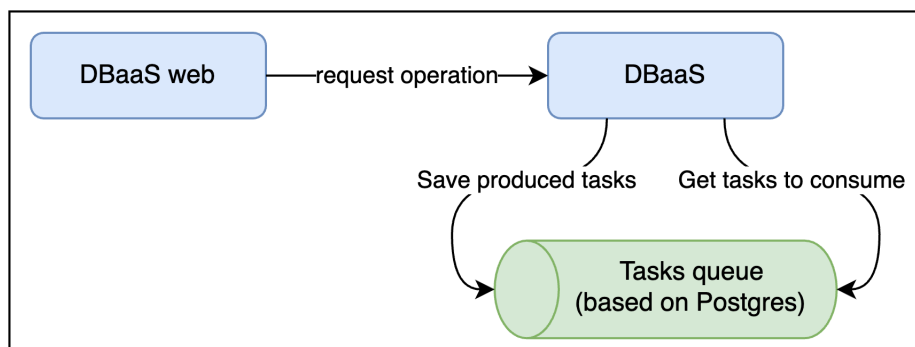


Рис. 4 — Реализация паттерна проектирования «Transactional outbox» в DBaaS платформе

#### 2.1.2.4. Паттерн проектирования «Sidecar»

Облачная инфраструктура Kubernetes позволяет разместить в рамках одного Kubernetes Pod[54] несколько контейнеров с приложениями, среди которых один будет основным, а другие вспомогательными в соответствии с концепцией[51] sidecar контейнеров.

Данный подход используется при разворачивании MongoDB DBaaS платформой. В одном Kubernetes Pod с базой данных запускаются 2 sidecar контейнера: Percona MongoDB metrics exporter[52] для предоставления метрик базы данных MongoDB и Percona MongoDB backup agent[53] для управления резервными копиями базы данных.

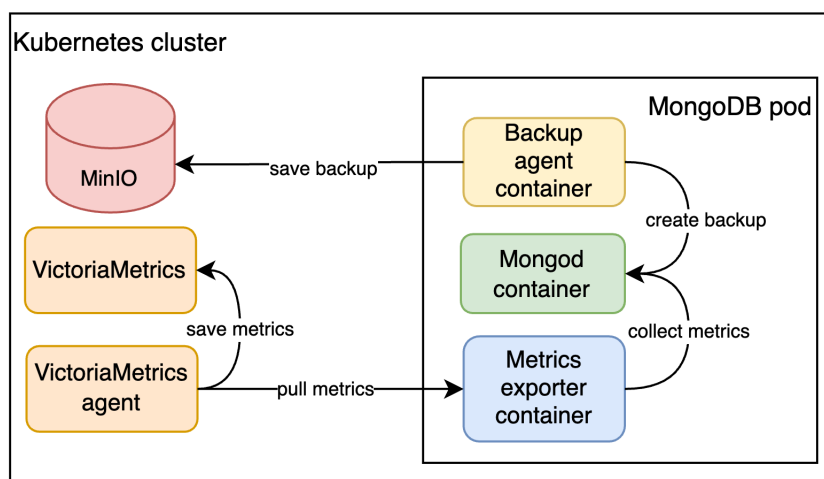


Рис. 5 — Реализация паттерна проектирования «Sidecar» в DBaaS платформе

## 2.2. Выбор технологий реализации

### 2.2.1. Языки программирования и фреймворки

Основным языком для модуля MDB выбран язык Java[26] в паре со Spring Framework[24], так как данное сочетание технологий предоставляет широкий выбор инструментов для разработки комплексных приложений с большим количеством бизнес-логики. Экосистема Spring Framework обладает большим сообществом разработчиков и предлагает различные готовые решения для интеграции со сторонними технологиями.

Выбирая язык Java основным языком для модуля MDB, рассматривались потенциальные недостатки данного решения. К таковым можно отнести нехватку оптимизации в Spring Framework, а также особенность приложений, запускаемых JVM[56], при которой приложение достигает пиковой производительности спустя некоторое время после первоначального запуска. Учитывая предъявляемые требования к системе, данные недостатки не являются критичными, однако они принимаются во внимание и для их решения в перспективе рассматривается использование библиотеки «warmup»[57] для предварительной подготовки Spring Boot приложений к принятию трафика от компании Т-Банк[58], а также переход на фреймворк Quarkus[59]. Подробное применение библиотеки «warmup» описано в докладе «А можно погорячее? Чем и как мы прогреваем Spring-микросервисы»[70] от компании Т-Банк.

В ходе проектирования платформы языки Golang[27] и Python [28] также рассматривался, как основные языки для разработки модуля MDB. Golang является наиболее распространенным языком для сервисов автоматизации облачной инфраструктуры Kubernetes и показывает более высокие показатели производительности по сравнению с Java, однако предпочтение было отдано языку Java в силу экосистемы Java приложений для комплексных enterprise-решений, а также предъявляемых требований к платформе. Язык Python не подошел в качестве языка для модуля MDB по причине низкой производительности и отсутствия строгой типизации типов данных.

Также в рамках данной платформы разрабатывался легковесный агент (MDB agent), который осуществляет наблюдение за кластерами баз данных и выполняет вспомогательные операции над базами данных. В качестве языка программирования был выбран язык Python, так как не предполагается наличие большого количества логики и большой нагрузки на сервис.

### 2.2.2. Базы данных

Для решения различных задач платформа интегрирована со следующими базами данных:

- PostgreSQL[21]
- ClickHouse[22]
- Redis[23]
- VictoriaMetrics[20]
- VictoriaLogs[45]

### 2.2.2.1. PostgreSQL

В качестве основной базы данных для модуля MDB рассматривались реляционные базы данных в силу того, что для платформы важны поддержка транзакционности, организация данных в виде таблиц с возможностью нормализации данных, а также поддержка индексации данных. Среди различных РСУБД была выбрана база данных PostgreSQL.

#### Преимущества PostgreSQL:

- Полная поддержка ACID, которая позволяет выполнять комплексные операции с гарантией целостности данных. Транзакционность особенно актуальна при построении решения для управления базами данных и реализации системы квотирования.
- Поддержка данных в формате JSON/JSONB и массивов, которая значительно упрощает хранение больших конфигураций кластеров баз данных под управлением MDB.
- Поддержка различных индексов таких как, B-tree индексы, Hash индексы, частичные индексы и индексы выражений.
- Реализация параллельного доступа к данным при помощи версионирования данных.

#### Недостатки PostgreSQL:

- Использование индексов влечет дополнительное потребление дискового пространства, а также индексам в PostgreSQL свойственна проблема «распухания», которая обусловлена не мгновенной очисткой обновленных или удаленных строк, которая производится с помощью сборщика мусора VACUUM[64].
- Реализация кластера PostgreSQL с репликацией требует интеграции дополнительных инструментов таких, как Patroni[65] или Stolon[66].

В качестве одного из источников информации об устройстве индексов в PostgreSQL использовался доклад[62] «Индексы в PostgreSQL. Как понять, что создавать» от компании Data Egret[63].

Помимо PostgreSQL рассматривалось использование РСУБД MySQL[60], однако выбор был сделан в пользу PostgreSQL, так как данная база данных ориентирована на выполнение сложных запросов на чтение и запись, когда MySQL показывает более качественные показатели при операциях только на чтение. Также PostgreSQL является объектно-реляционной СУБД, в связи с чем она лучше подходит для хранения сложных типов данных и поддерживает такие типы данных, как JSON, JSONB и массивы. Данные типы данных активно применяются при хранении конфигурации кластеров баз данных и процессе обработки асинхронных задач платформы.

Помимо применения PostgreSQL для хранения стандартной бизнес-логики, данная база данных применяется при реализации процесса обработки комплексных операций, которые состоят из множества задач. Порядок и время выполнения задач строго регламентирован для выполнения сложных операций по эксплуатации баз данных. Кроме того, специфика платформы требует функционала долговременного хранения истории исполнения операций и их перезапуска. Учитывая ограничения, предъявляемые требования к реализации асинхронного выполнения операций, а также необходимость упрощения инфраструктуры для развертывания on-premise, использование уже используемой базы данных PostgreSQL вместо таких технологий, как брокеры сообщений Kafka[98] и RabbitMQ[99]. Подробное описание реализации очереди задач с применением PostgreSQL описано в 3 главе.



#### 2.2.2.2. ClickHouse

В качестве базы данных для хранения истории потребления ресурсов кластерами баз данных рассматривались колоночные базы данных, которые оптимизированы для выполнения аналитических запросов, и была выбрана база данных ClickHouse.

##### **Преимущества ClickHouse:**

- Поддержка работы с временными рядами, что актуально для хранения исторических данных потребления ресурсов кластерами баз данных под управлением модуля MDB.
- Оптимизация агрегации данных, которая требуется при построении отчета по потреблению ресурсов кластерами баз данных под управлением модуля MDB за определенный интервал времени.

##### **Недостатки ClickHouse:**

- Высокие показатели потребления оперативной памяти при развертывании базы данных.
- База данных оптимизирована для вставки большого количества строк в одном запросе, что необходимо учитывать при записи исторических данных потребления ресурсов множеством кластеров баз данных под управлением модуля MDB.
- База данных не оптимизирована для частого обновления или удаления строк, однако данный функционал не предполагается использовать на регулярной основе в рамках модуля MDB.

Кроме ClickHouse также рассматривалась другая колоночная база данных Apache Druid[73], однако она уступает по показателю утилизации ресурсов ClickHouse.

#### 2.2.2.3. Redis

В качестве базы данных для кэширования данных платформы выбрана key-value база данных Redis. Использование key-value базы данных модулем MDB предполагается для хранения статусов реплик кластеров баз данных под управлением модуля MDB, частое обновление которых необходимо для упрощения процесса мониторинга. Также потеря данных о статусе реплики не является критичной, так как обновленный статус вычисляется за короткий промежуток времени на регулярной основе. Важной функцией является поддержка TTL для ключей в Redis, благодаря которому при аварии MDB агента, который предоставляет показатели состояния реплик баз данных, устаревшие статусы будут удалены спустя заданный интервал времени.

##### **Преимущества Redis:**

- Высокая производительность за счет хранения данных в оперативной памяти.
- Поддержка различных типов данных.
- Возможность настройки автоудаления ключей по истечении временного интервала.

##### **Недостатки Redis:**

- Размер базы данных ограничен объемом оперативной памяти.
- Содержимое базы данных будет потеряно при перезапуске базы данных, так как данные хранятся в оперативной памяти.

Среди аналогов Redis рассматривалась база данных KeyDB[74], которая показывает более высокие показатели производительности по сравнению с Redis, однако в рамках проектируемой архитектуры платформы данная производительность является избыточной, поэтому выбор был сделан в пользу более распространенной технологии Redis.

#### 2.2.2.4. VictoriaMetrics

В качестве основного хранилища для метрик кластеров баз данных под управлением модуля MDB используется Time Series база данных VictoriaMetrics. Использование Time Series базы данных обусловлено тем, что метрики поставляются в виде временных рядов, для которых адаптирован данный тип баз данных.

##### **Преимущества VictoriaMetrics:**

- Высокие показатели производительности и утилизации ресурсов.
- Поддержка сжатия данных.
- Адаптирована для длительного хранения метрик.
- Поддержка метрик в формате Prometheus.

##### **Недостатки VictoriaMetrics:**

- Молодая экосистема, которая однако быстро развивается. Также совместимость сторонних систем с VictoriaMetrics решается благодаря поддержке метрик в формате Prometheus.

В качестве базы данных для хранения метрик также рассматривались базы данных Prometheus[29], InfluxDB[78] и TimescaleDB[79], однако согласно статье «Measuring vertical scalability for time series databases in Google Cloud»[77] показатели VictoriaMetrics на запись и чтение превышают показатели InfluxDB и TimescaleDB до 20 раз при масштабировании системы. Также согласно статье «Insert benchmarks with inch: InfluxDB vs VictoriaMetrics»[76] VictoriaMetrics использует до 20 раз меньше оперативной памяти в сравнении с InfluxDB и до 7 раз меньше оперативной памяти в сравнении с Prometheus согласно статье «Prometheus vs VictoriaMetrics benchmark on node\_exporter metrics»[75]. Таким образом, предпочтение было отдано более производительной технологии.

#### 2.2.2.5. VictoriaLogs

В качестве хранилища для логов кластеров баз данных под управлением модуля MDB выбрана колоночная база данных VictoriaLogs.

##### **Преимущества VictoriaLogs:**

- Высокие показатели производительности и утилизации ресурсов.
- Поддержка сжатия данных.
- Поддержка полнотекстового поиска.
- Поддержка различных форматов записи данных, что позволяет интегрировать систему со сторонними распространенными технологиями такими, как Vector[46] и OpenTelemetry[80].

##### **Недостатки VictoriaLogs:**

- Молодая экосистема, которая активно развивается, и уже предоставляет готовые интеграции с наиболее популярными решениями.

Среди аналогов VictoriaLogs также рассматривались базы данных Elasticsearch[30], Grafana Loki[84] и ClickHouse[22].

Согласно статье «How do open source solutions for logs work: Elasticsearch, Loki and VictoriaLogs»[81] VictoriaLogs потребляет до 30 раз меньше оперативной памяти и до 15 раз меньше дискового пространства по сравнению с технологиями Elasticsearch и Grafana Loki.

Согласно документации[82] VictoriaLogs выполняет полнотекстовый поиск по логам до 1000 раз быстрее по сравнению с показателями Grafana Loki. Также Grafana Loki значительно теряет в производительности при обработке полей с высокой кардинальностью, когда VictoriaLogs не подвержена такой проблеме.

Архитектура VictoriaLogs вдохновлена реализацией ClickHouse и ClickHouse также является подходящим решением для хранения логов с высокой производительностью, однако ClickHouse показывает наилучшие результаты, когда известны все поля в логах, под которые можно настроить таблицы и индексирование данных. В свою очередь VictoriaLogs изначально разрабатывалось, как решение для работы с логами, поэтому не имеет подобного ограничения.

### 2.2.3. Объектные хранилища данных

#### 2.2.3.1. MinIO

В качестве основного объектного хранилища для бэкапов кластеров баз данных под управлением модуля MDB выбрана технология MinIO[41].

##### **Преимущества MinIO:**

- Совместимость с S3[86] API.
- Оптимизирован для работы с NVME дисками.
- Поддержка кластерного режима.
- Поддержка шифрования.
- Поддержка разделения контроля доступов.

##### **Недостатки MinIO:**

- Коммерческое использование требует покупки лицензии.
- Отсутствие функционала версионирования файлов.

Помимо MinIO в качестве объектного хранилища рассматривалась технология Ceph[85] и S3[86]. Ceph совместим с S3 API и его лицензия не накладывает ограничения на коммерческое использование, однако Ceph предполагает построение масштабной инфраструктуры, которая не требуется в рамках данной проектной работы. В свою очередь, S3 требует покупки лицензии для использования, в связи с чем в рамках данной работы предпочтение было отдано MinIO с открытым исходным кодом.

Minio, как и другие объектные хранилища разворачиваются на стороне пользователя платформы, поэтому точкой роста проекта является поддержка дополнительных объектных хранилищ, поддерживающих S3 API, для хранения резервных копий баз данных.

## 2.2.4. Системы мониторинга

### 2.2.4.1. Grafana

В качестве основного инструмента для визуализации метрик и логов кластеров баз данных под управлением модуля MDB был выбран сервис Grafana[19].

#### Преимущества Grafana:

- Поддержка различных типов данных и их визуализации.
- Поддержка наиболее распространенных источников данных.
- Наличие интегрированной системы алертинга.

#### Недостатки Grafana:

- Сервис не является базой данных. В связи с этим требуется интеграция стороннего решения для хранения метрик и логов.

## 2.2.5. Облачные операторы

### 2.2.5.1. Percona MongoDB operator

В качестве основной технологии автоматизации баз данных MongoDB[12] используется Percona MongoDB operator[42].

В ходе изучения существующих решений для автоматизации базы данных MongoDB в Kubernetes были выявлены следующие технологии:

- Percona MongoDB operator
- Bitnami Helm Chart for MongoDB[87]
- MongoDB community operator[88]
- MongoDB enterprise operator[89]

Bitnami Helm Chart for MongoDB не подошел на роль средства автоматизации, так как технология Helm Charts в целом не является средством управления сервисами в ходе их работы в облачной среде Kubernetes. Основная задача Helm Charts заключается в применении изменений в соответствии с декларативным представлением сущности без дальнейшего сопровождения развернутого ресурса.

Рассматривая MongoDB community operator, были выявлены следующие недостатки:

- Отсутствие поддержки вертикального масштабирования.
- Отсутствие поддержки резервного копирования и восстановления из резервной копии.
- Ограниченная конфигурация базы данных MongoDB.
- Отсутствие поддержки шардирования базы данных.

Рассматривая MongoDB enterprise operator, были выявлены следующие недостатки:

- Ограниченная конфигурация базы данных MongoDB.

- Необходимость покупки лицензии.

Рассматривая Percona MongoDB operator, были выявлены следующие недостатки:

- Для развертывания используется MongoDB совместимая реализация базы данных MongoDB от компании Percona[18] под названием Percona MongoDB server[49].
- Отсутствие функционала управления MongoDB абстракциями «баз данных» и «пользователей».

Принимая во внимание описанные выше недостатки каждого решения выбор был сделан в пользу Percona MongoDB operator. Для восполнения недостающего функционала управления MongoDB абстракциями «баз данных» и «пользователей» используется реализация собственного MDB агента.

#### 2.2.5.2. VictoriaMetrics operator

Для автоматизации баз данных VictoriaMetrics и VictoriaLogs используется официальный оператор VictoriaMetrics operator[44], который предоставляет функционал динамической конфигурации Kubernetes сущностей VictoriaMetrics и VictoriaLogs. Выбор сделан в пользу оператора так, как данное решение является нативным для облачной среды Kubernetes и рекомендовано разработчиками VictoriaMetrics.

#### 2.2.6. Агенты

##### 2.2.6.1. VictoriaMetrics agent

Для сбора метрик кластеров баз данных под управлением модуля MDB и дальнейшей поставки в VictoriaMetrics используется компонент VictoriaMetrics agent[43], который является частью экосистемы VictoriaMetrics и реализует нативный для VictoriaMetrics процесс сбора метрик.

##### 2.2.6.2. Vector agent

В качестве основного инструмента для поставки логов кластеров баз данных под управлением модуля MDB используется технология Vector в режиме агента.

##### **Преимущества Vector:**

- Высокая производительность за счет реализации на языке программирования Rust.
- Поддержка поставки как логов, так и метрик.
- Поддержка наиболее распространенных источников данных.
- Реализация Vector Remap Language, который позволяет выполнить подготовку логов и метрик перед отправкой в централизованной хранилище. Так, например, VRL используется для разметки логов кластеров баз данных дополнительными лейблами с названием проекта и кластера.
- Реализация инструментов надежной доставки данных.
- Наличие открытого исходного кода.

## Недостатки Vector:

- Vector незначительно уступает технологиям Fluent Bit, Logstash по количеству доступных интеграций.

Помимо Vector рассматривались решения Fluent Bit[91] и Logstash[90]. Ключевым недостатком Logstash является его низкая производительность и низкий показатель утилизации в сравнении с Vector. В свою очередь, Fluent Bit уступает Vector с точки зрения языка трансформации данных: реализация трансформации полученных логов возможна с помощью lua скриптов, написание которых осложняет процесс автоматизации инфраструктуры.

## 2.2.7. Технологии разработки, тестирования и развертывания

### 2.2.7.1. OpenAPI generator

В модуль MDB добавлено использование библиотеки OpenAPI generator[31], которая предоставляет функционал кодогенерации транспортного слоя приложения в соответствии с OpenAPI спецификаций, что значительно упрощает процесс синхронизации взаимодействия между клиентской и серверной частями платформы, а также сокращает время на разработку.

### 2.2.7.2. JOOQ

В модуль MDB добавлено использование библиотеки JOOQ[32] для кодогенерации слоя взаимодействия с базой данных PostgreSQL[21]. Данная библиотека позволяет взаимодействовать с базой данных при помощи сгенерированных объектов со строгой типизацией, что значительно оптимизирует процесс разработки.

Для генерации слоя базы данных реализован собственный Gradle[36] плагин, который при каждой сборке сервиса создает Docker-контейнер с базой данных PostgreSQL при помощи библиотеки Testcontainers[40], затем выполняет миграцию данных для созданной базы данных при помощи Flyway[33], после чего JOOQ подключается к базе данных и генерирует код для взаимодействия с базой данных в соответствии с актуальным состоянием миграций.

### 2.2.7.3. Flyway

В модуль MDB добавлено использование библиотеки Flyway[33], которая позволяет выполнять автоматические миграции с поддержкой версионирования для баз данных PostgreSQL[21] и ClickHouse[22]. Подход с автоматическими миграциями автоматизирует процесс обновления схемы базы данных без необходимости подключения к СУБД в продуктивном окружении и нейтрализует вероятность рассинхронизации состояния базы данных для сервиса одной версии в разных окружениях.

### 2.2.7.4. Testcontainers

В модуль MDB добавлено использование библиотеки Testcontainers[40], которая позволяет создать контейнеры с базами данных PostgreSQL[21], ClickHouse[22] и Redis[23] для выполнения интеграционного тестирования в изолированном окружении. Такой подход к тестированию при правильной конфигурации гарантирует повторное исполнение тестов в идентичном окружении с идентичной конфигурацией.

#### 2.2.7.5. Docker

Для сборки сервисов в изолированном окружении с необходимыми компонентами используется система контейнеризации Docker[39], которая позволяет обеспечить развертывание сервисов в идентичном окружении, что значительно упрощает процессы разработки, тестирования и развертывания.

#### 2.2.7.6. Minikube

Конфигурации компонентов проекта адаптированы для запуска платформы внутри Minikube[35], который позволяет развернуть Kubernetes[11] кластер из одного узла в локальной среде для разработки и тестирования платформы.

#### 2.2.7.7. Github Actions

Для разработки компонентов платформы используется технология Github Actions[92] для автоматизации процесса тестирования и сборки кода перед добавлением новой функциональности в исходный код. Также Github Actions используется для сборки и публикации Docker-образов компонентов платформы.

### 2.3. Вывод по главе

В данной главе произведен разбор архитектуры DBaaS платформы с описанием всех компонентов разрабатываемого программного решения. Описаны паттерны проектирования и концепции взаимодействия, которые были изучены в ходе проектирования и применены при решении поставленных задач, с ссылками на публикации, в рамках которых освещались способы применения данных подходов.

В результате проектирования был определен список технологий, которые должны быть использованы при построении отказоустойчивой и масштабируемой облачной платформы, предоставляющей функционал управления кластерами распределенных баз данных в облачной инфраструктуре Kubernetes. В описании выбранных технологий приведено обоснование выбора того или иного инструмента с обзором различных альтернативных решений, их ключевых достоинств и недостатков.

Также помимо описанных выше сервисов, которые будут вместе составлять единую платформу, описаны технологии и практики, применяющиеся при разработке, тестировании и развертывании программного продукта.

## 3. Глава 3. Программная реализация системы

### 3.1. Описание схемы базы данных и ассоциируемых сущностей

#### 3.1.1. PostgreSQL

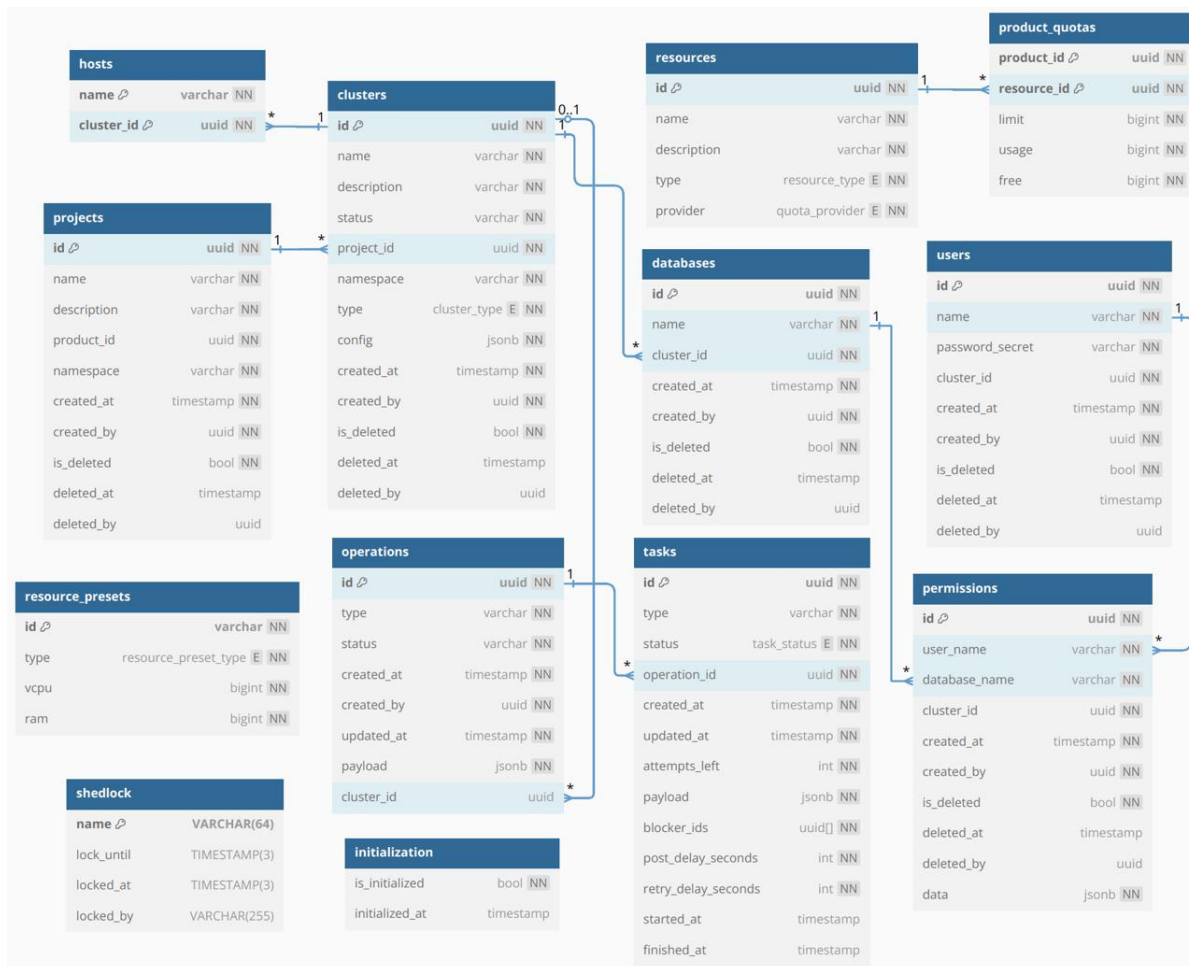


Рис. 6 — Схема базы данных PostgreSQL модуля MDB

##### 3.1.1.1. Таблица «initialization»

###### Список колонок:

- is\_initialized - тип данных bool NOT NULL
- initialized\_at - тип данных timestamp

###### Описание:

Данная таблица используется для первичной инициализации сервиса при его запуске. Во избежание гонки данных используется подход с **SELECT FOR UPDATE**, который не позволяет нескольким репликам сервиса выполнять инициализацию одновременно.



### 3.1.1.2. Таблица «resource\_preset»

#### Список колонок:

- id - тип данных varchar NOT NULL
- type - тип данных public.resource\_preset\_type NOT NULL
- vcpu - тип данных bigint NOT NULL
- ram - тип данных bigint NOT NULL

#### Список ключей:

- PK (id)

#### Описание:

Таблица содержит информацию о предустановленных наборах ресурсов, в рамках которых осуществляется конфигурация кластеров баз данных. Предустановленный набор определяет количество ядер и объем оперативной памяти каждой реплики кластера баз данных.

### 3.1.1.3. Таблица «projects»

#### Список колонок:

- id - тип данных uuid NOT NULL
- name - тип данных varchar NOT NULL
- description - тип данных varchar NOT NULL
- product\_id - тип данных uuid NOT NULL
- namespace - тип данных varchar NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- is\_deleted - тип данных bool NOT NULL DEFAULT false
- deleted\_at - тип данных timestamp
- deleted\_by - тип данных uuid

#### Список ключей:

- PK (id)

#### Описание:

Таблица содержит информацию о проектах платформы, в рамках которых происходит управление кластерами баз данных. Каждый проект привязан к продукту, на основе которого осуществляется квотирование и биллинг ресурсов. Помимо продукта проект привязан к Kubernetes namespace[93], в котором разворачиваются кластеры баз данных.

#### 3.1.1.4. Таблица «clusters»

##### Список колонок:

- id - тип данных uuid NOT NULL
- name - тип данных varchar NOT NULL
- description - тип данных varchar NOT NULL
- status - тип данных varchar NOT NULL
- project\_id - тип данных uuid NOT NULL
- namespace - тип данных varchar NOT NULL
- type - тип данных public.cluster\_type NOT NULL
- config - тип данных jsonb NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- is\_deleted - тип данных bool NOT NULL DEFAULT false
- deleted\_at - тип данных timestamp
- deleted\_by - тип данных uuid

##### Список ключей:

- PK (id)
- FK (project\_id) - projects (id)

##### Описание:

Таблица содержит информацию кластерах баз данных под управлением модуля MDB.

#### 3.1.1.5. Таблица «hosts»

##### Список колонок:

- name - тип данных varchar NOT NULL
- cluster\_id - тип данных uuid NOT NULL

##### Список ключей:

- PK (name, cluster\_id)
- FK (cluster\_id) - clusters (id)

##### Описание:

Таблица содержит связь реплик баз данных с кластерами под управлением модуля MDB.

### 3.1.1.6. Таблица «databases»

#### Список колонок:

- id - тип данных uuid NOT NULL
- name - тип данных varchar NOT NULL
- cluster\_id - тип данных uuid NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- is\_deleted - тип данных bool NOT NULL DEFAULT false
- deleted\_at - тип данных timestamp
- deleted\_by - тип данных uuid

#### Список ключей:

- PK (id)
- FK (cluster\_id) - clusters (id)

#### Описание:

Таблица содержит информацию о сущностях «базы данных», которые созданы в рамках кластера под управлением модуля MDB.

### 3.1.1.7. Таблица «users»

#### Список колонок:

- id - тип данных uuid NOT NULL
- name - тип данных varchar NOT NULL
- password\_secret - тип данных varchar NOT NULL
- cluster\_id - тип данных uuid NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- is\_deleted - тип данных bool NOT NULL DEFAULT false
- deleted\_at - тип данных timestamp
- deleted\_by - тип данных uuid

#### Список ключей:

- PK (id)

#### Описание:

Таблица содержит информацию о сущностях «пользователи», которые созданы в рамках кластера под управлением модуля MDB.

### 3.1.1.8. Таблица «permissions»

#### Список колонок:

- id - тип данных uuid NOT NULL
- user\_name - тип данных varchar NOT NULL
- database\_name - тип данных varchar NOT NULL
- cluster\_id - тип данных uuid NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- is\_deleted - тип данных bool NOT NULL DEFAULT false
- deleted\_at - тип данных timestamp
- deleted\_by - тип данных uuid
- data - тип данных jsonb NOT NULL

#### Список ключей:

- PK (id)
- FK (user\_name) - users (name)
- FK (database\_name) - databases (name)

#### Описание:

Таблица содержит информацию о разрешениях для сущностей «пользователи», которые созданы в рамках кластера под управлением модуля MDB.

### 3.1.1.9. Таблица «operations»

#### Список колонок:

- id - тип данных uuid NOT NULL
- type - тип данных varchar NOT NULL
- status - тип данных varchar NOT NULL
- created\_at - тип данных timestamp NOT NULL
- created\_by - тип данных uuid NOT NULL
- updated\_at - тип данных timestamp NOT NULL
- payload - тип данных jsonb NOT NULL
- cluster\_id - тип данных uuid

#### Список ключей:

- PK (id)
- FK (cluster\_id) - clusters (id)

**Описание:**

Таблица содержит информацию об операциях, которые выполняются в рамках эксплуатации кластеров баз данных под управлением модуля MDB.

### 3.1.1.10. Таблица «tasks»

**Список колонок:**

- id - тип данных uuid NOT NULL
- type - тип данных varchar NOT NULL
- status - тип данных public.task\_status NOT NULL
- operation\_id - тип данных uuid NOT NULL
- created\_at - тип данных timestamp NOT NULL
- updated\_at - тип данных timestamp NOT NULL
- attempts\_left - тип данных int NOT NULL
- payload - тип данных jsonb NOT NULL
- blocker\_ids - тип данных uuid[] NOT NULL
- post\_delay\_seconds - тип данных int NOT NULL
- retry\_delay\_seconds - тип данных int NOT NULL
- started\_at - тип данных timestamp
- finished\_at - тип данных timestamp

**Список ключей:**

- PK (id)
- FK (operation\_id) - operations (id)

**Описание:**

Таблица содержит информацию о задачах, связанных с операциями, которые выполняются в рамках эксплуатации кластеров баз данных под управлением модуля MDB.

### 3.1.1.11. Таблица «resources»

#### Список колонок:

- id - тип данных uuid NOT NULL
- name - тип данных varchar NOT NULL
- description - тип данных varchar NOT NULL
- type - тип данных public.resource\_type NOT NULL
- provider - тип данных public.quota\_provider NOT NULL

#### Список ключей:

- PK (id)
- UNIQUE (type, provider)

#### Описание:

Таблица содержит информацию об уникальных ресурсах платформы, в рамках которых осуществляется квотирование и биллинг ресурсов.

Список ресурсов:

- Количество ядер
- Объем оперативной памяти

### 3.1.1.12. Таблица «product\_quotas»

#### Список колонок:

- product\_id - тип данных uuid NOT NULL
- resource\_id - тип данных uuid NOT NULL
- limit - тип данных bigint NOT NULL
- usage - тип данных bigint NOT NULL
- free - тип данных bigint NOT NULL

#### Список ключей:

- PK (product\_id, resource\_id)

#### Описание:

Таблица содержит состояние квоты продукта по определенному ресурсу из таблицы «resources».

### 3.1.2. ClickHouse

billing_quotas_usage		
product_id		uuid
resource_id		uuid
limit		int64
usage		int64
free		int64
created_at		datetime

Рис. 7 — Схема базы данных ClickHouse модуля MDB

#### 3.1.2.1. Таблица «billing\_quotas\_usage»

**Список полей:**

- product\_id - тип данных uuid
- resource\_id - тип данных uuid
- limit - тип данных int64
- usage - тип данных int64
- free - тип данных int64
- created\_at - тип данных datetime

**Параметры движка:**

- Движок: MergeTree
- Ключ сортировки: (product\_id, resource\_id, created\_at)

**Описание:**

Таблица содержит информацию о состоянии квоты продукта на определенный ресурс в определенный момент времени.

## 3.2. Реализация модуля MDB

### 3.2.1. Реализация асинхронной обработки операций кластеров

При проектировании архитектуры DBaaS платформы рассматривались различные потенциальные ограничения, которые обусловлены эксплуатацией распределенных баз данных. Одной из главных сложностей в автоматизации баз данных является выполнение комплексных операций, которые состоят из ряда задач. При выполнении связанных задач необходимо выполнять их в строгой последовательность и с определенной временной задержкой. Для прозрачности

процесса эксплуатации требуется поддержка сохранения истории выполнения операций и их задач. Также одной из ключевых функций является поддержка перезапуска операции, которая завершилась с ошибкой для минимизации привлечения администраторов платформы.

Для решения данной задачи было принято решение реализовать очередь задач на PostgreSQL на основе паттерна проектирования «Transactional outbox»[50] с возможностью тонкой конфигурации выполнения задач в зависимости от порядка выполнения и типа задач.

Для выполнения операций по эксплуатации кластеров в асинхронном режиме используется несколько ключевых абстракций: Operation, Task, Payload, TaskProducer, TaskConsumer.

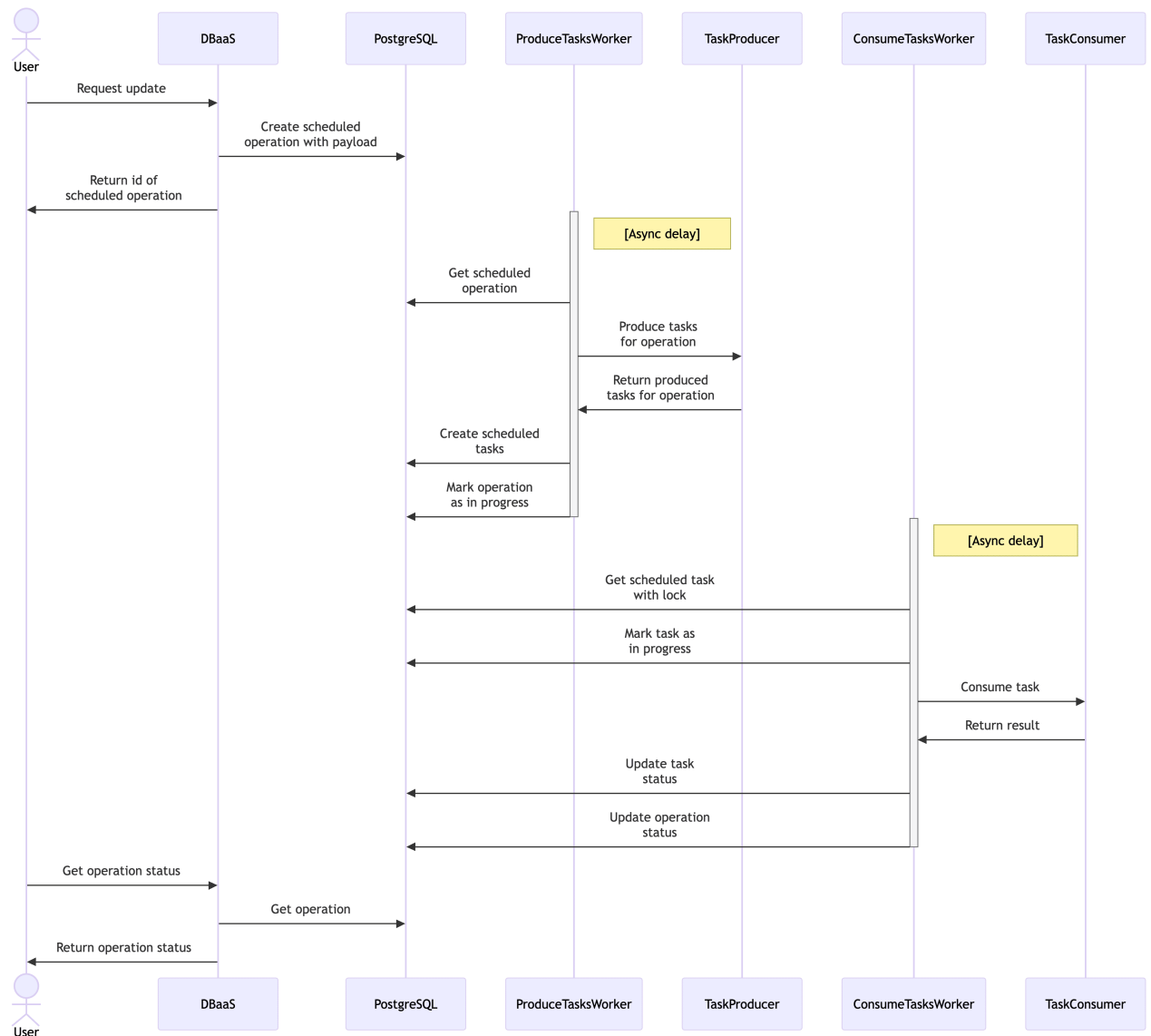


Рис. 8 — Последовательность обработки абстрактной операции обновления DBaaS платформой



### 3.2.1.1. Operation

**Operation** – абстракция над задачами, которая объединяет различные задачи в один единый процесс эксплуатации кластера баз данных.

Листинг 1 — Модель для абстракции Operation

```
public record Operation(  
    UUID id,  
    OperationType type,  
    OperationStatus status,  
    LocalDateTime createdAt,  
    UUID createdBy,  
    LocalDateTime updatedAt,  
    String payload,  
    @Nullable  
    UUID clusterId  
) {}
```

Список реализованных операций:

- MONGO\_CREATE\_CLUSTER – создание кластера MongoDB[12]
- MONGO\_MODIFY\_CLUSTER – изменение кластера MongoDB
- MONGO\_DELETE\_CLUSTER – удаление кластера MongoDB
- MONGO\_CREATE\_DATABASE – создание базы данных MongoDB
- MONGO\_DELETE\_DATABASE – удаление базы данных MongoDB
- MONGO\_CREATE\_USER – создание пользователя MongoDB
- MONGO\_DELETE\_USER – удаление пользователя MongoDB
- MONGO\_CREATE\_BACKUP – создание резервной копии MongoDB
- MONGO\_DELETE\_BACKUP – удаление резервной копии MongoDB
- MONGO\_RESTORE\_CLUSTER – восстановление кластера MongoDB

### 3.2.1.2. Task

**Task** – блок логики, который может быть выполнен автономно и переиспользован в нескольких операциях.

Листинг 2 — Модель для абстракции Task

```
public record Task(  
    UUID id,  
    TaskType type,  
    TaskStatus status,  
    UUID operationId,  
    LocalDateTime createdAt,  
    LocalDateTime updatedAt,  
    int attemptsLeft,  
    String payload,  
    List<UUID> blockerIds,  
    int postDelaySeconds,  
    int retryDelaySeconds,  
    @Nullable  
    LocalDateTime startedAt,  
    @Nullable  
    LocalDateTime finishedAt  
) {}
```

Список реализованных задач:

- MONGO\_APPLY\_PSMDB – применение конфигурации PSMDB[49] (Percona Server for MongoDB)
- MONGO\_UPDATE\_HOSTS – обновление хостов MongoDB кластера
- MONGO\_CHECK\_PSMDB\_READINESS – проверка готовности PSMDB
- MONGO\_APPLY\_ONYXDB\_AGENT – применение конфигурации MDB агента
- MONGO\_APPLY\_ONYXDB\_AGENT\_SERVICE – применение конфигурации Kubernetes Service[55] для MDB агента
- MONGO\_CHECK\_ONYXDB\_AGENT\_READINESS – проверка готовности MDB агента
- MONGO\_APPLY\_EXPORTER\_SERVICE – применение конфигурации Kubernetes Service для Percona MongoDB exporter[52]
- MONGO\_APPLY\_EXPORTER\_SERVICE\_SCRAPE – применение конфигурации VMServiceScrape[94] для сбора метрик из Percona MongoDB exporter
- MONGO\_CREATE\_DATABASE – создание базы данных MongoDB
- MONGO\_CREATE\_USER – создание пользователя MongoDB
- MONGO\_DELETE\_DATABASE – удаление базы данных MongoDB
- MONGO\_DELETE\_USER – удаление пользователя MongoDB
- MONGO\_CREATE\_BACKUP – создание резервной копии кластера MongoDB

- MONGO\_CHECK\_BACKUP\_IS\_READY – проверка готовности резервной копии кластера MongoDB
- MONGO\_DELETE\_BACKUP – удаление резервной копии кластера MongoDB
- MONGO\_CHECK\_BACKUP\_IS\_DELETED – проверка удаления резервной копии кластера MongoDB
- MONGO\_RESTORE\_CLUSTER – восстановление кластера MongoDB из резервной конфигурации
- MONGO\_CHECK\_CLUSTER\_IS\_RESTORED – проверка готовности кластера MongoDB после восстановления из резервной копии
- MONGO\_MARK\_CLUSTER\_READY – перевод MongoDB кластера в статус готового к работе
- MONGO\_MARK\_CLUSTER\_UPDATING – перевод MongoDB кластера в статус обновляемого
- MONGO\_MARK\_CLUSTER\_DELETING – перевод MongoDB кластера в статус удаляемого
- MONGO\_MARK\_CLUSTER\_DELETED – перевод MongoDB кластера в статус удаленного
- MONGO\_DELETE\_EXPORTER\_SERVICE\_SCRAPE – удаление конфигурации VMServiceScrape[94] для сбора метрик из Percona MongoDB exporter
- MONGO\_DELETE\_EXPORTER\_SERVICE – удаление конфигурации Kubernetes Service[55] для Percona MongoDB exporter[52]
- MONGO\_DELETE\_ONYXDB\_AGENT\_SERVICE – удаление конфигурации Kubernetes Service[55] для MDB агента
- MONGO\_DELETE\_ONYXDB\_AGENT – удаление MDB агента
- MONGO\_CHECK\_ONYXDB\_AGENT\_IS\_DELETED – проверка удаления MDB агента
- MONGO\_DELETE\_PSMDB – удаление PSMDB
- MONGO\_CHECK\_PSMDB\_IS\_DELETED – проверка удаления PSMDB
- MONGO\_UPDATE\_QUOTA\_AFTER\_CLUSTER\_DELETION – обновление квоты после удаления кластера MongoDB
- MONGO\_DELETE\_SECRETS – удаление секретов кластера MongoDB
- FINAL\_TASK – завершающая задача для любой операции

### 3.2.1.3. Payload

**Payload** – абстракция, в которой содержатся данные необходимые для исполнения операций и задач.

Листинг 3 — Пример Payload для создания пользователя в кластере MongoDB

```
public interface Payload {  
  
    public record MongoCreateUserPayload(  
        UUID clusterId,  
        String username,  
        String passwordSecretName,  
        String passwordSecretNamespace,  
        List<CreateMongoPermission> permissions  
    ) implements Payload {  
    }  
}
```

### 3.2.1.4. Task producer

**Task producer** – абстракция, отвечающая за генерацию задач для определенного типа операции.

Листинг 4 — Базовый класс TaskProducer

```
public abstract class TaskProducer<PAYLOAD> {  
    protected final ObjectMapper objectMapper;  
  
    protected TaskProducer(ObjectMapper objectMapper) {  
        this.objectMapper = objectMapper;  
    }  
  
    public abstract OperationType getOperationType();  
  
    public abstract List<ProducedTask> produceTasks(Operation operation, PAYLOAD payload);  
  
    public abstract PAYLOAD parsePayload(String payload) throws JsonProcessingException;  
  
    public List<ProducedTask> produceTasks(Operation operation, String payload) {  
        try {  
            return produceTasks(operation, parsePayload(payload));  
        } catch (JsonProcessingException e) {  
            throw new RuntimeException(e);  
        }  
    }  
}
```

Листинг 5 — Пример реализации абстрактного класса TaskProducer для операции создания резервной копии кластера MongoDB

```
public class MongoCreateBackupTaskProducer
    extends TaskProducer<MongoCreateBackupPayload> {
    public MongoCreateBackupTaskProducer(ObjectMapper objectMapper) {
        super(objectMapper);
    }

    @Override
    public OperationType getOperationType() {
        return OperationType.MONGO_CREATE_BACKUP;
    }

    @Override
    public List<ProducedTask> produceTasks(Operation operation, MongoCreateBackupPayload
        payload) {
        UUID operationId = operation.id();

        var clusterPayload = new ClusterPayload(payload.clusterId());

        var markClusterUpdatingTask = ProducedTask.createWithPayload(
            TaskType.MONGO_MARK_CLUSTER_UPDATING,
            operationId,
            List.of(),
            clusterPayload
        );
        var createBackupTask = ProducedTask.createWithPayload(
            TaskType.MONGO_CREATE_BACKUP,
            operationId,
            List.of(markClusterUpdatingTask.id()),
            payload
        );
        var checkBackupIsReadyTask = ProducedTask.createWithPayload(
            TaskType.MONGO_CHECK_BACKUP_IS_READY,
            operationId,
            List.of(createBackupTask.id()),
            payload
        );
        var markClusterReadyTask = ProducedTask.createWithPayload(
            TaskType.MONGO_MARK_CLUSTER_READY,
            operationId,
            List.of(checkBackupIsReadyTask.id()),
            clusterPayload
        );
        var finalTask = ProducedTask.create(
            TaskType.FINAL_TASK,
            operationId,
            List.of(markClusterReadyTask.id())
        );

        return List.of(
```

```

        markClusterUpdatingTask,
        createBackupTask,
        checkBackupIsReadyTask,
        markClusterReadyTask,
        finalTask
    );
}

@Override
public MongoCreateBackupPayload parsePayload(String payload) throws
    JsonProcessingException {
    return objectMapper.readValue(payload, MongoCreateBackupPayload.class);
}
}

```

### 3.2.1.5. Task consumer

**Task consumer** – абстракция, отвечающая за обработку задач с определенным типом.

Листинг 6 — Базовый класс TaskConsumer

```

public abstract class TaskConsumer<PAYLOAD extends Payload> {
    protected final ObjectMapper objectMapper;

    protected TaskConsumer(ObjectMapper objectMapper) {
        this.objectMapper = objectMapper;
    }

    public abstract TaskType getTaskType();

    protected abstract TaskResult internalProcess(Task task, PAYLOAD payload);

    protected abstract PAYLOAD parsePayload(Task task) throws JsonProcessingException;

    public TaskResult process(Task task) {
        try {
            return internalProcess(task, parsePayload(task));
        } catch (JsonProcessingException e) {
            throw new RuntimeException(e);
        }
    }
}
}

```

Листинг 7 — Пример реализации абстрактного класса TaskConsumer для запуска процесса восстановления MongoDB кластера из резервной копии

```
public class MongoRestoreClusterTaskConsumer extends TaskConsumer<
    MongoRestoreClusterPayload> {
    private final ClusterService clusterService;
    private final PsmdbClient psmdbClient;
    private final ProjectRepository projectRepository;

    public MongoRestoreClusterTaskConsumer(
        ObjectMapper objectMapper,
        ClusterService clusterService,
        PsmdbClient psmdbClient,
        ProjectRepository projectRepository
    ) {
        super(objectMapper);
        this.clusterService = clusterService;
        this.psmdbClient = psmdbClient;
        this.projectRepository = projectRepository;
    }

    @Override
    public TaskType getTaskType() {
        return TaskType.MONGO_RESTORE_CLUSTER;
    }

    @Override
    protected TaskResult internalProcess(Task task, MongoRestoreClusterPayload payload) {
        Cluster cluster = clusterService.getClusterOrThrow(payload.clusterId());
        Project project = projectRepository.getProjectOrThrow(cluster.projectId(), false);

        psmdbClient.applyPsmdbRestore(
            cluster.namespace(),
            project.name(),
            cluster.name(),
            payload.backupName()
        );

        return TaskResult.success();
    }

    @Override
    protected MongoRestoreClusterPayload parsePayload(Task task) throws
        JsonProcessingException {
        return objectMapper.readValue(task.payload(), MongoRestoreClusterPayload.class);
    }
}
```

### 3.2.1.6. Поиск задач для исполнения

Для параллельной обработки задач по эксплуатации кластеров баз данных используется подход `SELECT FOR UPDATE SKIP LOCKED`, благодаря которому задача блокируется до окончания обработки, что гарантирует отсутствия конфликтов во время многопоточной реализации.

Листинг 8 — Реализация получения готовых к исполнению задач

```
public List<Task> getTasksToConsume(int limit) {
    Tasks blockers = TASKS.as("blocker");
    return dslContext.select()
        .from(TASKS)
        .where(TASKS.ATTEMPTS_LEFT.gt(0)
            .and(TASKS.STATUS.eq(
                taskMapper.taskStatusToJooqTaskStatus(TaskStatus.SCHEDULED)
            )
            .or(TASKS.STATUS.eq(
                taskMapper.taskStatusToJooqTaskStatus(TaskStatus.RESCHEDULED)
            )
            .and(DSL.localDateTimeAdd(
                TASKS.UPDATED_AT,
                TASKS.RETRY_DELAY_SECONDS,
                DatePart.SECOND
            )
            .lt(DSL.currentLocalDateTime())))
        ))
        .andNotExists(
            DSL.selectOne()
            .from(blockers)
            .where(blockers.ID.eq(DSL.any(TASKS.BLOCKER_IDS)))
            .and(blockers.STATUS.ne(
                taskMapper.taskStatusToJooqTaskStatus(TaskStatus.SUCCESS)
            )
            .or(blockers.FINISHED_AT.isNull())
            .or(DSL.localDateTimeAdd(
                blockers.FINISHED_AT,
                blockers.POST_DELAY_SECONDS,
                DatePart.SECOND
            ).ge(DSL.currentLocalDateTime())))
        )
    )
    .limit(limit)
    .forUpdate()
    .skipLocked()
    .fetchMany()
    .stream()
    .map(result -> result.into(TasksRecord.class))
    .flatMap(List::stream)
    .map(taskMapper::tasksRecordToTask)
    .toList();
}
```



### 3.2.1.7. Единое решение для обработки операций в асинхронном режиме для платформы

Для реализации единого решения по обработке операций в асинхронном режиме в рамках платформы архитектура бизнес-логики была спроектирована с применением абстракций, которые не привязаны к модулю MDB и обслуживанию кластеров баз данных. Таким образом, данное решение позволяет переиспользовать готовое решение внутри других компонентов платформы, что значительно упрощает развитие платформы в будущем.

### 3.2.2. Реализация поставки метрик кластеров MongoDB

Для поставки метрик кластера MongoDB[12] в каждый Kubernetes Pod[54] MongoDB добавлен sidecar-контейнер[51] с Percona MongoDB exporter[52], который предоставляет метрики в формате Prometheus[29]. Далее автоматика создает сущность Kubernetes Service[55] для каждого кластера MongoDB, которая предоставляет точку доступа к репликам кластера MongoDB. Затем для поставки метрик в VictoriaMetrics[20] автоматика создает сущность VMServiceScrape[94], благодаря которой VictoriaMetrics agent[43] обновляет свою конфигурацию. Согласно данной конфигурации VictoriaMetrics agent осуществляет обнаружение сервисов, с которых необходимо собирать метрики. После чего VictoriaMetrics agent опрашивает необходимые сервисы и поставляет полученные метрики в VictoriaMetrics.

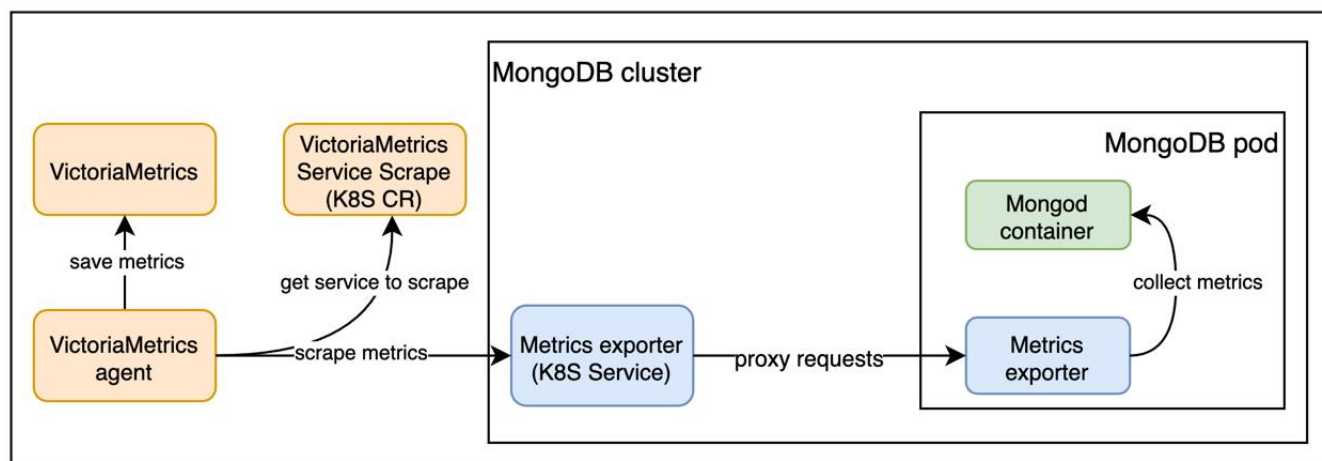


Рис. 9 — Реализация поставки метрик кластеров MongoDB

Листинг 9 — Пример автоматически сгенерированного манифеста Kubernetes Service для Percona MongoDB exporter

```
apiVersion: v1
kind: Service
metadata:
  name: test-1-sandbox-mongo-exporter
  labels:
    app.kubernetes.io/instance: test-1-sandbox-mongo-exporter
spec:
  ports:
    - name: metrics
      port: 9216
      targetPort: 9216
  selector:
    app.kubernetes.io/instance: test-1-sandbox-mongo
```

Листинг 10 — Пример автоматически сгенерированного манифеста VMServiceScrape

```
apiVersion: operator.victoriametrics.com/v1beta1
kind: VMServiceScrape
metadata:
  name: test-1-sandbox-mongo-exporter
spec:
  endpoints:
    - port: metrics
      path: /metrics
      metricRelabelConfigs:
        - action: replace
          targetLabel: onyxdb_project
          replacement: sandbox
        - action: replace
          targetLabel: onyxdb_cluster
          replacement: test-1
  selector:
    matchLabels:
      app.kubernetes.io/instance: test-1-sandbox-mongo-exporter
```

**Примечание:** для корректной разметки метрик в манифест VMServiceScrape добавлено проставление лейблов «onyxdb\_project», «onyxdb\_cluster». Данные лейблы используются при построении Grafana[19] dashboard.

### 3.2.2.1. Визуализация метрик MongoDB

Для визуализации поставляемых метрик подготовлен Grafana[19] dashboard, который адаптирован под метрики, поставляемые платформой.

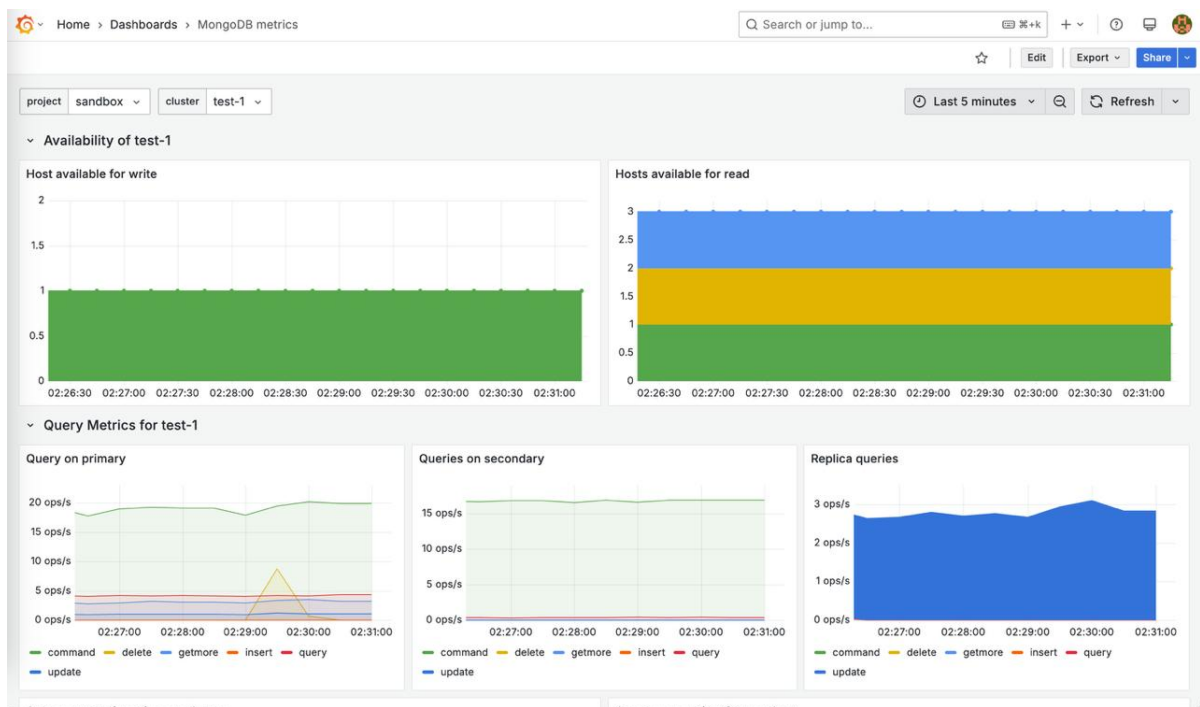


Рис. 10 — Демонстрация Grafana dashboard с метриками MongoDB. Часть 1

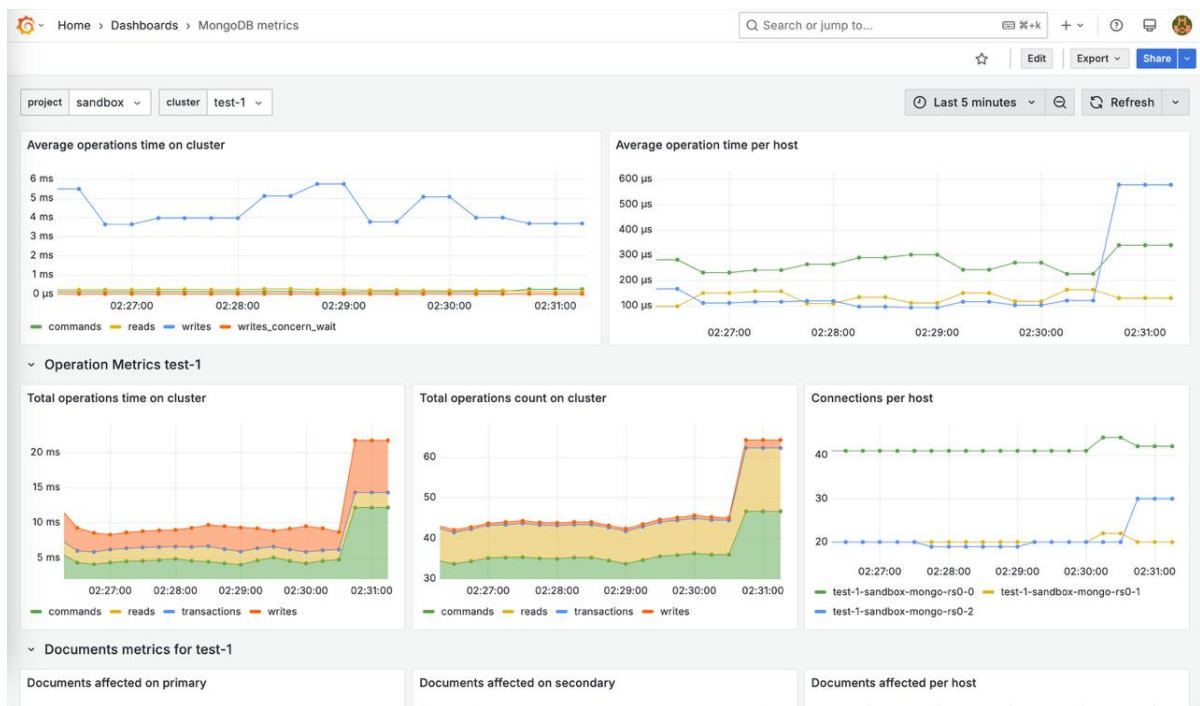


Рис. 11 — Демонстрация Grafana dashboard с метриками MongoDB. Часть 2

### 3.2.3. Реализация поставки логов кластеров MongoDB

Для поставки логов кластеров MongoDB запускается Vector[46] с ролью «Agent», при которой развертывание Vector происходит в качестве Kubernetes DaemonSet[95]. В такой конфигурации реплики Vector размещаются на каждой Kubernetes Node[96].

Vector agent сконфигурирован для сбора логов из источника «kubernetes\_logs»[97], преобразования лейблов логов с добавлением дополнительных «onyxdb\_project», «onyxdb\_cluster», которые используются для агрегации логов при построении Grafana[19] dashboard.

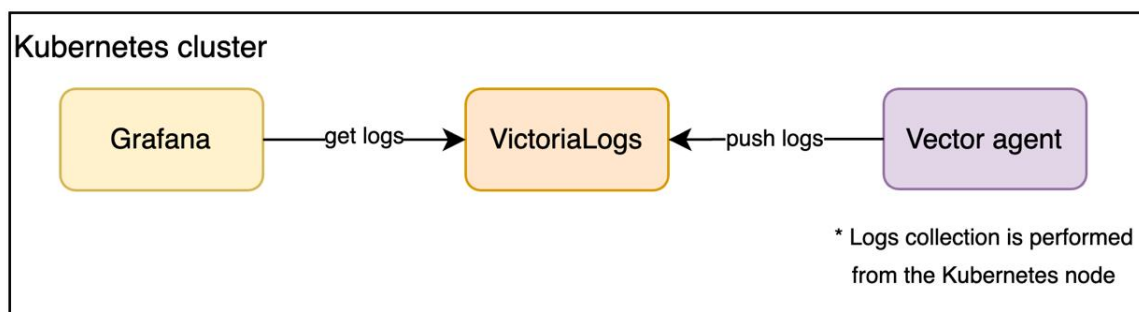


Рис. 12 — Реализация поставки логов кластеров MongoDB

Листинг 11 — Пример минимальной конфигурации Bitnami Helm Chart для Vector

role: "Agent"

```
serviceAccount:
  create: true
  name: vector
```

```
containerPorts:
  - containerPort: 80
    name: http
```

```
service:
  enabled: true
  type: "ClusterIP"
  ports:
    - port: 80
      name: http
```

```
customConfig:
  data_dir: /vector-data-dir
  api:
    enabled: true
  sources:
    kubernetes_logs:
      type: "kubernetes_logs"
      extra_label_selector: "app.kubernetes.io/managed-by=percona-server-mongodb-operator,
        app.kubernetes.io/component=mongod"
  transforms:
    mongod_logs:
```

```

    type: filter
    inputs:
      - kubernetes_logs
    condition: .kubernetes.container_name == "mongod"
  processed_mongod_logs:
    type: remap
    inputs:
      - mongod_logs
    source: |
      .onyxdb_project = .kubernetes.pod_labels."onyxdb.com/project"
      .onyxdb_cluster = .kubernetes.pod_labels."onyxdb.com/cluster"
      .host = .kubernetes.pod_name
  sinks:
    vlogs:
      inputs:
        - processed_mongod_logs
      type: http
      uri: "http://vlogs-onyxdb.onyxdb.svc.cluster.local:9428/insert/jsonline
?_stream_fields=host,container_name&_msg_field=message&_time_field=timestamp"
      compression: gzip
      encoding:
        codec: json
        only_fields:
          - timestamp
          - message
          - onyxdb_project
          - onyxdb_cluster
          - host
      framing:
        method: newline_delimited
      healthcheck:
        enabled: true

```

### 3.2.3.1. Визуализация логов MongoDB

Для визуализации поставляемых логов подготовлен Grafana[19] dashboard, который адаптирован под логи, поставляемые платформой.

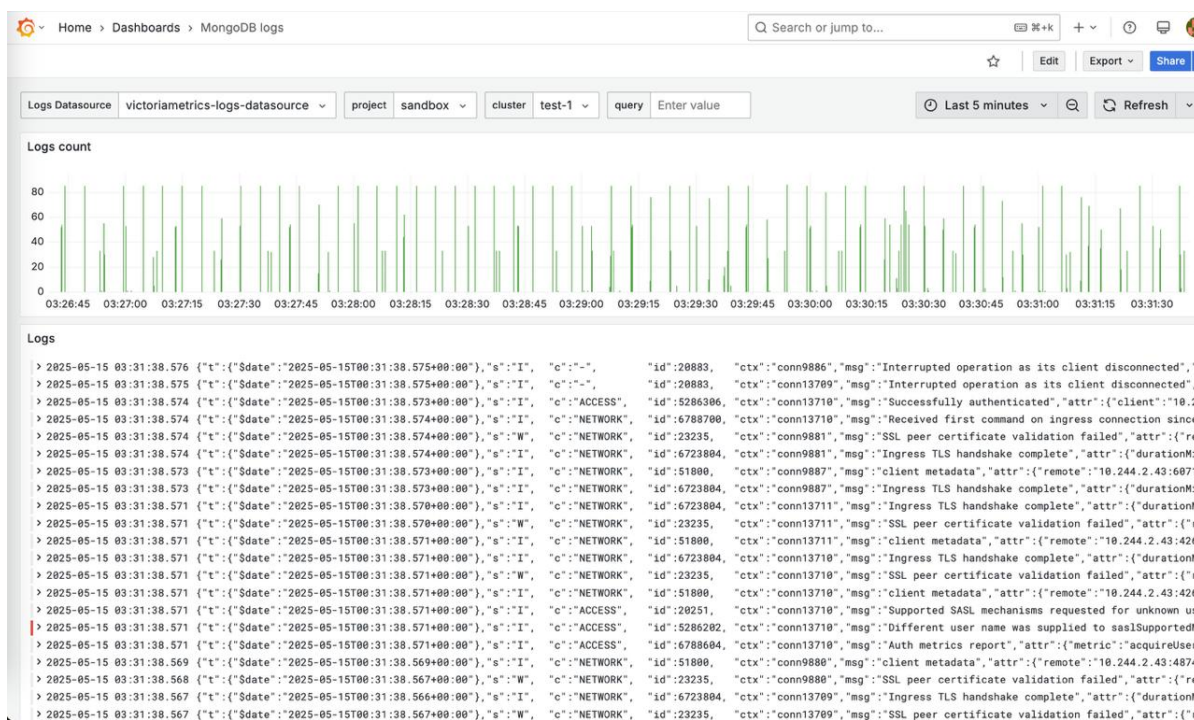


Рис. 13 — Демонстрация Grafana dashboard с логами MongoDB

### 3.2.4. Применение MDB агента

В рамках платформы MDB агент выполняет вспомогательную роль в эксплуатации кластеров баз данных под управлением модуля MDB. В обязанности MDB агента входит предоставление состояния реплик кластеров MongoDB, а также реализация функционала модификации сущностей «баз данных» и «пользователей» в рамках MongoDB, так как данный функционал не реализован в Persona MongoDB operator[42].

Интегрированный Persona MongoDB operator также не поддерживает предоставление подробной информации о состоянии кластера с минимальной задержкой. По этой причине поставка подробного описания состояния реплик кластера осуществляется MDB агентом. Обновление состояния реплик MongoDB через MDB агента реализовано через push-модель, в рамках которой MDB агент с определенной периодичностью опрашивает реплики кластера MongoDB, к которому он привязан. Затем агент конвертирует полученные данные в стандартизированный формат и отправляет результаты преобразования в модуль MDB, который кэширует полученные статусы в Redis. Логика предоставления информации о хостах кластера реализована при помощи обогащения базовой информации о репликах кластера кэшированными статусами, предоставленными MDB агентом. Для хранения данных с актуальным состоянием кластера MongoDB задается TTL, по истечению которого данные удаляются и модуль MDB считает состояние кластера неизвестным.

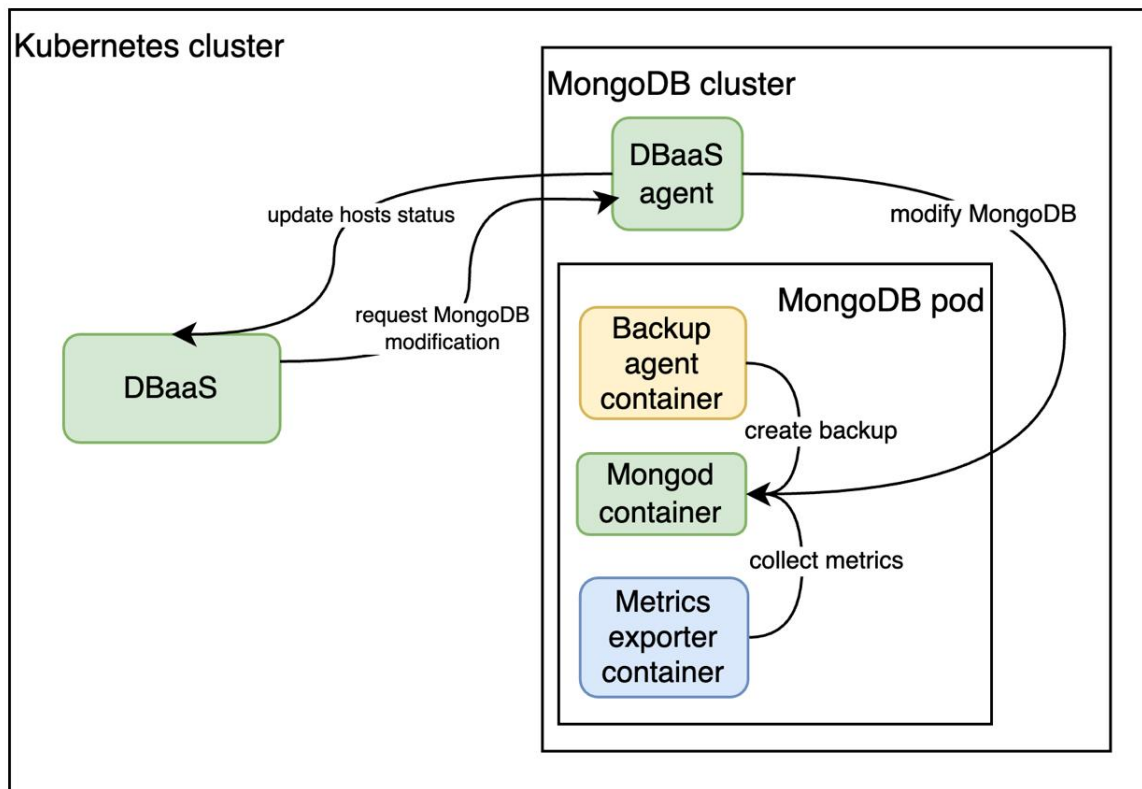


Рис. 14 — Взаимодействие MDB агента с MDB модулем

Листинг 12 — Пример автоматически сгенерированного манифеста Kubernetes Deployment для MDB агента

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-1-sandbox-mongo-agent
  labels:
    app.kubernetes.io/name: onyxdb-agent
    app.kubernetes.io/instance: test-1-sandbox-mongo-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: onyxdb-agent
      app.kubernetes.io/instance: test-1-sandbox-mongo-agent
  template:
    metadata:
      labels:
        app.kubernetes.io/name: onyxdb-agent
        app.kubernetes.io/instance: test-1-sandbox-mongo-agent
    spec:
      serviceAccountName: onyxdb-agent-robot
      containers:
        - name: agent
          image: foxleren/onyxdb-agent:master-feb840b
```

```

imagePullPolicy: Always
ports:
  - containerPort: 9002
resources:
  requests:
    cpu: "0.5"
    memory: "0.4Gi"
  limits:
    cpu: "0.5"
    memory: "0.4Gi"
env:
  - name: ONYXDB_MDB__BASE_URL
    value: http://onyxdb.onyxdb.svc.cluster.local:9001
  - name: ONYXDB_MDB__CLUSTER_ID
    value: 7ddf2617-7745-4b45-bf78-7ea961e3ab25
  - name: ONYXDB_MDB__ACCESS_TOKEN
    valueFrom:
      secretKeyRef:
        name: onyxdb-robot
        key: accessToken
  - name: MONGO_USER
    valueFrom:
      secretKeyRef:
        name: test-1-sandbox-mongo-users
        key: MONGODB_DATABASE_ADMIN_USER
  - name: MONGO_PASSWORD
    valueFrom:
      secretKeyRef:
        name: test-1-sandbox-mongo-users
        key: MONGODB_DATABASE_ADMIN_PASSWORD
  - name: ONYXDB_MONGO__URI
    value: mongodb://$(MONGO_USER):$(MONGO_PASSWORD)@test-1-sandbox-mongo-rs0.
      onyxdb.svc.cluster.local/admin?replicaSet=rs0&ssl=false

```

**Примечание:** запрашиваемые ресурсы в манифесте неокончательные и приведены для демонстрации.

Листинг 13 — Пример автоматически сгенерированного манифеста Kubernetes Service для MDB агента

```

apiVersion: v1
kind: Service
metadata:
  name: test-1-sandbox-mongo-agent
  labels:
    app.kubernetes.io/name: onyxdb-agent-service
    app.kubernetes.io/instance: test-1-sandbox-mongo-agent
spec:
  ports:
    - name: http
      port: 9002
      targetPort: 9002
  selector:

```



`app.kubernetes.io/instance: test-1-sandbox-mongo-agent`

### 3.2.5. Описание конфигурации модуля MDB

#### 3.2.5.1. Описание параметров конфигурации модуля MDB

- `ONYXDB_SELF_URL` — адрес модуля MDB в Kubernetes
- `ONYXDB_SELF_NAMESPACE` — пространство имён, в котором запущен модуль MDB
- `ONYXDB_NAMESPACES` — пространства имён, в которых модуль MDB может разворачивать кластеры баз данных
- `ONYXDB_STORAGE_CLASSES` — Kubernetes storage classes[100], которые можно использовать при развёртывании кластеров баз данных
- `ONYXDB_POSTGRES_URL` — URL подключения к PostgreSQL[21] в формате JDBC[101]
- `ONYXDB_POSTGRES_USER` — пользователь PostgreSQL
- `ONYXDB_POSTGRES_PASSWORD` — пароль PostgreSQL
- `ONYXDB_REDIS_HOST` — хост Redis
- `ONYXDB_REDIS_PORT` — порт Redis
- `ONYXDB_REDIS_USER` — пользователь Redis
- `ONYXDB_REDIS_PASSWORD` — пароль Redis
- `ONYXDB_CLICKHOUSE_URL` — URL подключения к ClickHouse в формате JDBC[101]
- `ONYXDB_CLICKHOUSE_USER` — пользователь ClickHouse
- `ONYXDB_CLICKHOUSE_PASSWORD` — пароль ClickHouse
- `ONYXDB_MINIO_URL` — URL MinIO API
- `ONYXDB_MINIO_SECRET` — название Kubernetes Secret[102], в котором хранится access-ключ для взаимодействия модуля MDB с MinIO
- `ONYXDB_MINIO_BUCKET` — бакет MinIO, в который будут сохраняться резервные копии кластеров баз данных
- `ONYXDB_VLOGS_URL` — URL VictoriaLogs[45]

Листинг 14 — Пример Kubernetes Secret с конфигурацией для модуля MDB

```
apiVersion: v1
kind: Secret
metadata:
  name: onyxdb
  labels:
    app.kubernetes.io/name: onyxdb
    app.kubernetes.io/instance: onyxdb
stringData:
  ONYXDB_SELF_URL: "http://onyxdb.onyxdb.svc.cluster.local:9001"
  ONYXDB_SELF_NAMESPACE: "onyxdb"
  ONYXDB_NAMESPACES: "testing,production"
  ONYXDB_STORAGE_CLASSES: "yc-network-hdd,yc-network-ssd"
  ONYXDB_POSTGRES_URL: "jdbc:postgresql://postgres-onyxdb.onyxdb.svc.cluster.local:5432/
    onyxdb"
  ONYXDB_POSTGRES_USER: "onyxdb"
  ONYXDB_POSTGRES_PASSWORD: "qwerty"
  ONYXDB_REDIS_HOST: "redis-onyxdb.onyxdb.svc.cluster.local"
  ONYXDB_REDIS_PORT: "6379"
  ONYXDB_REDIS_USER: "onyxdb"
  ONYXDB_REDIS_PASSWORD: "qwerty"
  ONYXDB_CLICKHOUSE_URL: "jdbc:clickhouse://clickhouse-onyxdb.onyxdb.svc.cluster.local
    :8123/onyxdb"
  ONYXDB_CLICKHOUSE_USER: "onyxdb"
  ONYXDB_CLICKHOUSE_PASSWORD: "qwerty"
  ONYXDB_MINIO_URL: "http://minio.onyxdb.svc.cluster.local:9000"
  ONYXDB_MINIO_SECRET: "minio-onyxdb"
  ONYXDB_MINIO_BUCKET: "onyxdb"
  ONYXDB_VLOGS_URL: "http://vls-onyxdb.onyxdb.svc.cluster.local:9428"
```

Листинг 15 — Пример Kubernetes Secret с access-ключом для взаимодействия модуля MDB с MinIO

```
apiVersion: v1
kind: Secret
metadata:
  name: minio-onyxdb
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: key
  AWS_SECRET_ACCESS_KEY: value
```

### 3.2.6. Реализация квотирования

В рамках данной проектной работы квотирование осуществляется для ядер процессора и объема оперативной памяти, которые гарантированно выделяются базе данных модулем MDB.

Управление квотами выполняется на основе абстракции «продуктов», в рамках которых происходит распределение бюджета на каждый тип ресурса. Абстракция квоты не является явной частью модуля MDB и выделена в отдельный блок бизнес-логики для возможности использования данной функциональности новыми компонентами DBaaS платформы в будущем.

Функционал квотирования ресурсов тесно интегрирован с основной логикой управления кластерами баз данных модулем MDB, предоставляя возможность гибко настраивать лимиты потребления ресурсов определенным группам пользователей DBaaS платформы. Управление кластерами баз данных модуля MDB осуществляется в пределах абстракции «проект» и для интеграции квот в процессы управления кластерами реализована привязка проектов к продуктам, которые предоставляют квоты для потребления.

**Описание составляющих квоты:**

- limit — максимальное значение квоты, которая выделена продукту
- usage — значение квоты, которая находится в использовании
- free — значение квоты, которая доступна для использования

#### 3.2.6.1. Сценарии применения квот в модуле MDB

- Создание и обновление кластера — проверка наличия квоты на ресурсы, требуемые для работы кластера и применение изменений к состоянию квоты при удовлетворении требованиям
- Удаление кластера — после завершения процесса удаления кластера и его ресурсов выполняется освобождение квоты

### 3.2.7. Реализация биллинга ресурсов

В рамках выполняемой работы биллинг потребления ресурсов осуществляется на основе показателей потребления квоты продуктов. Данный подход обусловлен тем, что текущая реализация функционала управления базами данных модулем MDB гарантирует выделение необходимых ресурсов в пределах доступной квоты. Также данные показатели показывают процент потребления квоты в разрезе всей DBaaS платформы, что помогает администраторам платформы контролировать потребление ресурсов той или иной группой пользователей.

Принимая во внимание, что статистика об утилизации гарантированных ресурсов модулем MDB является важной метрикой при обслуживании большого количества ресурсов, разработка данной функциональности является важной точкой роста проекта, однако в рамкой данной проектной работы не предусмотрена.

Процесс сбора данных биллинга реализован при помощи регулярной синхронизации показателей потребления квоты продуктов платформы и поставки собранных данных в базу данных ClickHouse[22].

### 3.2.8. Мониторинг платформы

Для упрощения процесса администрирования DBaaS платформы в сервис добавлена поставка метрик в формате Prometheus[29].

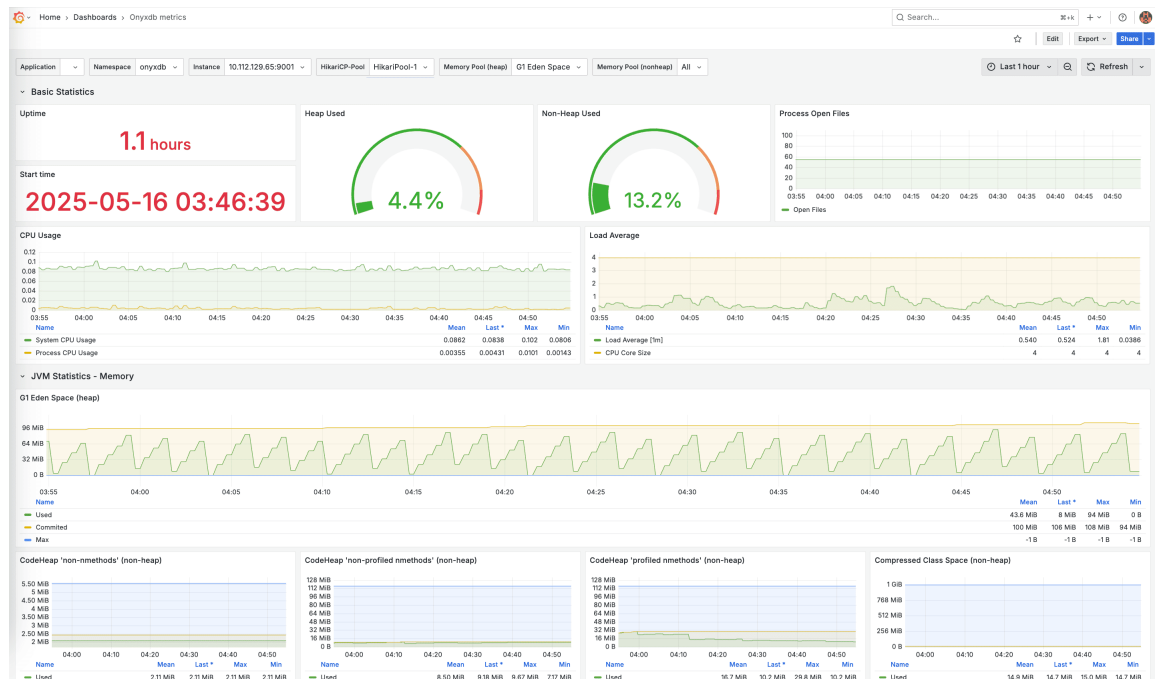


Рис. 15 — Пример Grafana dashboard с метриками сервиса DBaaS

## 3.3. Описание API

### 3.3.1. Управление ресурсными наборами

Resource presets: the Resource presets API			
GET	/api/mdb/resource-presets/{resourcePresetId}	Get resource preset	
PUT	/api/mdb/resource-presets/{resourcePresetId}	Update resource preset	
DELETE	/api/mdb/resource-presets/{resourcePresetId}	Delete resource preset	
GET	/api/mdb/resource-presets	List resource presets	
POST	/api/mdb/resource-presets	Create resource preset	

Рис. 16 — Обработчики для управления ресурсными наборами

#### 3.3.1.1. Получение списка ресурсных наборов

Описание запроса:

Маршрут: GET /api/mdb/resource-presets

ID операции: listResourcePresets

Тело ответа: ListResourcePresetsResponseDTO

### 3.3.1.2. Создание ресурсного набора

**Описание запроса:**

Маршрут: POST /api/mdb/resource-presets

ID операции: createResourcePreset

Тело запроса: CreateResourcePresetRequestDTO

### 3.3.1.3. Получение ресурсного набора

**Описание запроса:**

Маршрут: GET /api/mdb/resource-presets/{resourcePresetId}

ID операции: getResourcePreset

Список параметров:

- resourcePresetId - обязательный path-параметр с типом данных String

Тело ответа: ResourcePresetResponseDTO

### 3.3.1.4. Обновление ресурсного набора

**Описание запроса:**

Маршрут: PUT /api/mdb/resource-presets/{resourcePresetId}

ID операции: updateResourcePreset

Список параметров:

- resourcePresetId - обязательный path-параметр с типом данных String

Тело запроса: UpdateResourcePresetRequestDTO

### 3.3.1.5. Удаление ресурсного набора

**Описание запроса:**

Маршрут: DELETE /api/mdb/resource-presets/{resourcePresetId}

ID операции: deleteResourcePreset

Список параметров:

- resourcePresetId - обязательный path-параметр с типом данных String

## 3.3.2. Управление проектами

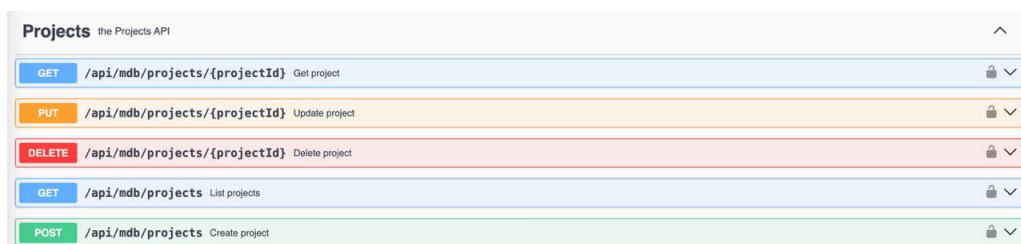


Рис. 17 — Обработчики для управления проектами

#### 3.3.2.1. Получение списка проектов

**Описание запроса:**

Маршрут: GET /api/mdb/projects

ID операции: listProjects

Тело ответа: ListProjectsResponseDTO

#### 3.3.2.2. Создание проекта

**Описание запроса:**

Маршрут: POST /api/mdb/projects

ID операции: createProject

Тело запроса: CreateProjectRequestDTO

Тело ответа: CreateProjectResponseDTO

#### 3.3.2.3. Получение проекта

**Описание запроса:**

Маршрут: GET /api/mdb/projects/{projectId}

ID операции: getProject

Список параметров:

- projectId - обязательный path-параметр с типом данных UUID

Тело ответа: ProjectDTO

#### 3.3.2.4. Обновление проекта

**Описание запроса:**

Маршрут: PUT /api/mdb/projects/{projectId}

ID операции: updateProject

Список параметров:

- projectId - обязательный path-параметр с типом данных UUID

Тело запроса: UpdateProjectRequestDTO

#### 3.3.2.5. Удаление проекта

**Описание запроса:**

Маршрут: DELETE /api/mdb/projects/{projectId}

ID операции: deleteProject

Список параметров:

- projectId - обязательный path-параметр с типом данных UUID

Managed MongoDB clusters the Managed MongoDB clusters API		
GET	/api/mdb/mongodb/clusters/{clusterId}	Get MongoDB cluster
PUT	/api/mdb/mongodb/clusters/{clusterId}	Update MongoDB cluster
DELETE	/api/mdb/mongodb/clusters/{clusterId}	Delete MongoDB cluster
GET	/api/mdb/mongodb/clusters	List MongoDB clusters
POST	/api/mdb/mongodb/clusters	Create MongoDB cluster
POST	/api/mdb/mongodb/clusters/{clusterId}/restore/{backupName}	Restore MongoDB cluster
GET	/api/mdb/mongodb/versions	List MongoDB versions

Рис. 18 — Обработчики для управления кластерами MongoDB

### 3.3.3. Управление кластерами MongoDB

#### 3.3.3.1. Получение списка MongoDB кластеров

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters

ID операции: listClusters

Список параметров:

- projectIds - опциональный query-параметр с типом данных List<String> (UUID)
- isDeleted - опциональный query-параметр с типом данных boolean

Тело ответа: ListMongoClustersResponseDTO

#### 3.3.3.2. Создание MongoDB кластера

**Описание запроса:**

Маршрут: POST /api/mdb/mongodb/clusters

ID операции: createCluster

Тело запроса: CreateMongoClusterRequestDTO

Тело ответа: CreateMongoClusterResponseDTO

#### 3.3.3.3. Получение MongoDB кластера

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters/{clusterId}

ID операции: getCluster

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: MongoClusterDTO

#### 3.3.3.4. Обновление MongoDB кластера

**Описание запроса:**

Маршрут: PUT /api/mdb/mongodb/clusters/{clusterId}

ID операции: updateCluster

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело запроса: UpdateMongoClusterRequestDTO

Тело ответа: ScheduledOperationDTO

### 3.3.3.5. Удаление MongoDB кластера

**Описание запроса:**

Маршрут: DELETE /api/mdb/mongodb/clusters/{clusterId}

ID операции: deleteCluster

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ScheduledOperationDTO

### 3.3.3.6. Восстановление MongoDB кластера из бэкапа

**Описание запроса:**

Маршрут: POST /api/mdb/mongodb/clusters/{clusterId}/restore/{backupName}

ID операции: restoreCluster

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID
- backupName - обязательный path-параметр с типом данных String

Тело ответа: ScheduledOperationDTO

### 3.3.3.7. Получение списка версий MongoDB

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/versions

ID операции: listVersions

Тело ответа: ListMongoVersionsResponseDTO

### 3.3.4. Управление базами данных MongoDB кластеров

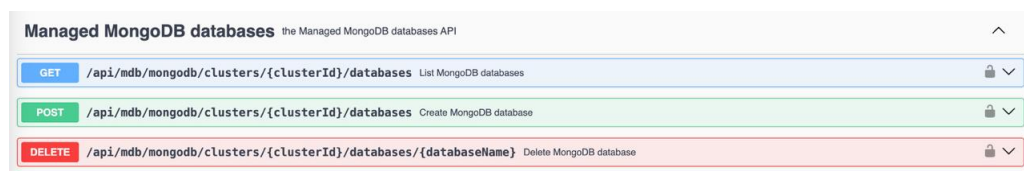


Рис. 19 — Обработчики для управления базами данных MongoDB кластеров



### 3.3.4.1. Получение списка баз данных MongoDB

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters/{clusterId}/databases

ID операции: listDatabases

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ListMongoDatabasesResponseDTO

### 3.3.4.2. Создание базы данных MongoDB

**Описание запроса:**

Маршрут: POST /api/mdb/mongodb/clusters/{clusterId}/databases

ID операции: createDatabase

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело запроса: CreateMongoDatabaseRequestDTO

Тело ответа: ScheduledOperationDTO

### 3.3.4.3. Удаление базы данных MongoDB

**Описание запроса:**

Маршрут: DELETE /api/mdb/mongodb/clusters/{clusterId}/databases/{databaseName}

ID операции: deleteDatabase

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID
- databaseName - обязательный path-параметр с типом данных String

Тело ответа: ScheduledOperationDTO

### 3.3.5. Управление пользователями MongoDB кластеров



Рис. 20 — Обработчики для управления пользователями MongoDB кластеров

#### 3.3.5.1. Получение списка доступных ролей MongoDB

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/users/roles

ID операции: listRoles

Тело ответа: ListMongoRolesResponseDTO

### 3.3.5.2. Получение списка пользователей MongoDB

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters/{clusterId}/users

ID операции: listUsers

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ListMongoUsersResponseDTO

### 3.3.5.3. Создание пользователя MongoDB

**Описание запроса:**

Маршрут: POST /api/mdb/mongodb/clusters/{clusterId}/users

ID операции: createUser

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело запроса: CreateMongoUserRequestDTO

Тело ответа: ScheduledOperationDTO

### 3.3.5.4. Удаление пользователя MongoDB

**Описание запроса:**

Маршрут: DELETE /api/mdb/mongodb/clusters/{clusterId}/users/{userName}

ID операции: deleteUser

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID
- userName - обязательный path-параметр с типом данных String

Тело ответа: ScheduledOperationDTO

### 3.3.6. Управление бэкапами MongoDB кластеров



Рис. 21 — Обработчики для управления бэкапами MongoDB кластеров

#### 3.3.6.1. Получение списка резервных копий MongoDB

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters/{clusterId}/backups

ID операции: listBackups

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ListMongoBackupsResponseDTO

### 3.3.6.2. Создание резервной копии MongoDB

**Описание запроса:**

Маршрут: POST /api/mdb/mongodb/clusters/{clusterId}/backups

ID операции: createBackup

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ScheduledOperationDTO

### 3.3.6.3. Удаление резервной копии MongoDB

**Описание запроса:**

Маршрут: DELETE /api/mdb/mongodb/clusters/{clusterId}/backups/{backupName}

ID операции: deleteBackup

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID
- backupName - обязательный path-параметр с типом данных String

Тело ответа: ScheduledOperationDTO

### 3.3.7. Управление хостами MongoDB кластера

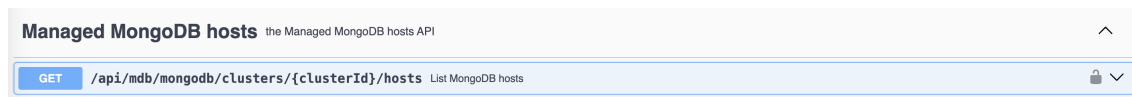


Рис. 22 — Обработчики для управления хостами MongoDB кластера

#### 3.3.7.1. Получение списка хостов MongoDB кластера

**Описание запроса:**

Маршрут: GET /api/mdb/mongodb/clusters/{clusterId}/hosts

ID операции: listHosts

Список параметров:

- clusterId - обязательный path-параметр с типом данных UUID

Тело ответа: ListMongoHostsResponseDTO

### 3.3.8. Внутреннее API MongoDB кластеров

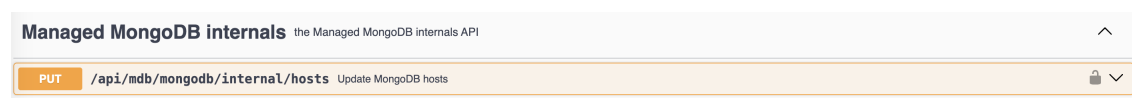


Рис. 23 — Обработчики для внутреннего API MongoDB кластеров

### 3.3.8.1. Обновление хостов MongoDB (внутреннее API)

**Описание запроса:**

Маршрут: PUT /api/mdb/mongodb/internal/hosts

ID операции: updateHosts

Тело запроса: UpdateMongoHostsRequestDTO

### 3.3.9. Общее API для кластеров баз данных

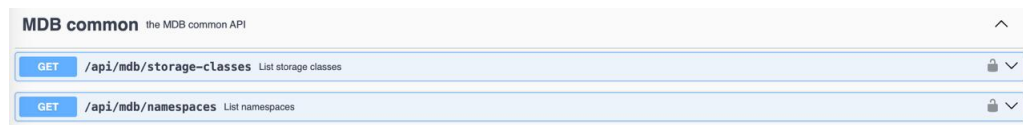


Рис. 24 — Обработчики общего API для кластеров баз данных

#### 3.3.9.1. Получение списка классов хранения

**Описание запроса:**

Маршрут: GET /api/mdb/storage-classes

ID операции: listStorageClasses

Тело ответа: ListStorageClassesResponseDTO

#### 3.3.9.2. Получение списка пространств имён

**Описание запроса:**

Маршрут: GET /api/mdb/namespaces

ID операции: listNamespaces

Тело ответа: ListNamespacesResponseDTO

### 3.3.10. Управление операциями

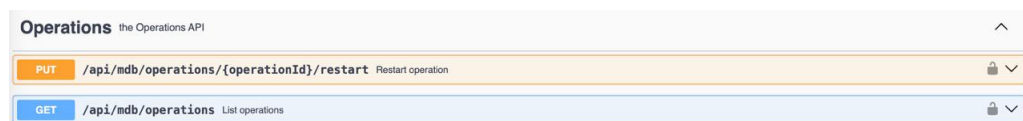


Рис. 25 — Обработчики для управления операциями

#### 3.3.10.1. Получение списка операций

**Описание запроса:**

Маршрут: GET /api/mdb/operations

ID операции: listOperations

Список параметров:

- clusterId - опциональный query-параметр с типом данных UUID

Тело ответа: ListOperationsResponseDTO

### 3.3.10.2. Перезапуск операции

#### Описание запроса:

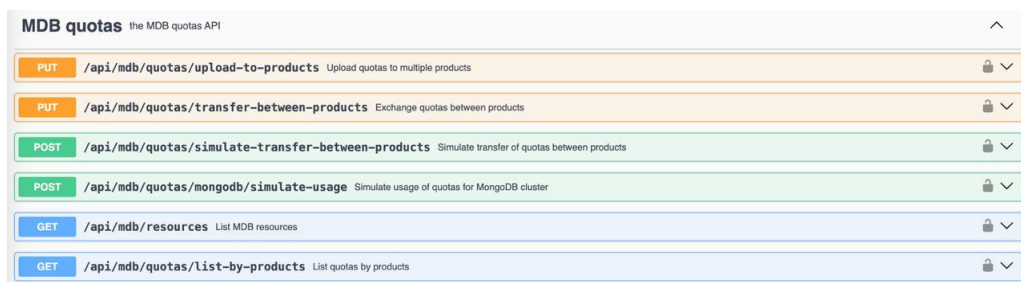
Маршрут: PUT /api/mdb/operations/{operationId}/restart

ID операции: restartOperation

Список параметров:

- operationId - обязательный path-параметр с типом данных UUID

### 3.3.11. Управление квотами продуктов



MDB quotas the MDB quotas API		
PUT	/api/mdb/quotas/upload-to-products	Upload quotas to multiple products
PUT	/api/mdb/quotas/transfer-between-products	Exchange quotas between products
POST	/api/mdb/quotas/simulate-transfer-between-products	Simulate transfer of quotas between products
POST	/api/mdb/quotas/mongodb/simulate-usage	Simulate usage of quotas for MongoDB cluster
GET	/api/mdb/resources	List MDB resources
GET	/api/mdb/quotas/list-by-products	List quotas by products

Рис. 26 — Обработчики для управления квотами продуктов

#### 3.3.11.1. Получение списка квот по продуктам

##### Описание запроса:

Маршрут: GET /api/mdb/quotas/list-by-products

ID операции: listQuotasByProducts

Список параметров:

- productIds - опциональный query-параметр с типом данных List<String> (UUID)

Тело ответа: ListQuotasByProductsResponseDTO

#### 3.3.11.2. Загрузка квот для нескольких продуктов

##### Описание запроса:

Маршрут: PUT /api/mdb/quotas/upload-to-products

ID операции: uploadQuotasToProducts

Тело запроса: UploadQuotasToProductsRequestDTO

#### 3.3.11.3. Обмен квотами между продуктами

##### Описание запроса:

Маршрут: PUT /api/mdb/quotas/transfer-between-products

ID операции: transferQuotasBetweenProducts

Тело запроса: TransferQuotasBetweenProductsRequestDTO

#### 3.3.11.4. Симуляция обмена квотами между продуктами

**Описание запроса:**

Маршрут: POST /api/mdb/quotas/simulate-transfer-between-products

ID операции: simulateTransferQuotasBetweenProducts

Тело запроса: TransferQuotasBetweenProductsRequestDTO

Тело ответа: SimulateTransferQuotasBetweenProductsResponseDTO

#### 3.3.11.5. Получение списка ресурсов MDB

**Описание запроса:**

Маршрут: GET /api/mdb/resources

ID операции: listResources

Тело ответа: ListResourcesResponseDTO

#### 3.3.11.6. Симуляция использования квот для MongoDB кластера

**Описание запроса:**

Маршрут: POST /api/mdb/quotas/mongodb/simulate-usage

ID операции: simulateMongoDbQuotasUsage

Тело запроса: SimulateMongoDBQuotasUsageRequestDTO

Тело ответа: SimulateMongoDBQuotasUsageResponseDTO

#### 3.3.12. Управление биллингом ресурсов

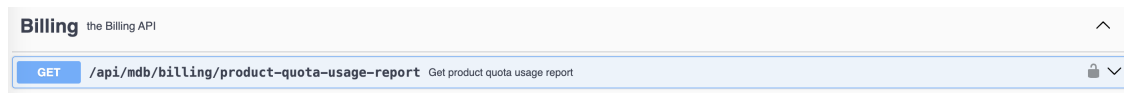


Рис. 27 — Обработчики для управления квотами продуктов

##### 3.3.12.1. Получение отчёта об использовании квот продукта

**Описание запроса:**

Маршрут: GET /api/mdb/billing/product-quota-usage-report

ID операции: getProductQuotaUsageReport

Список параметров:

- productId - обязательный query-параметр с типом данных UUID
- startDate - обязательный query-параметр с типом данных Date (формат YYYY-MM-DD)
- endDate - обязательный query-параметр с типом данных Date (формат YYYY-MM-DD)

Тело ответа: GetProductQuotaUsageReportResponseDTO

MongoDB			^
POST	/api/mongodb/databases	Create MongoDB database	▼
DELETE	/api/mongodb/databases	Delete MongoDB database	▼
POST	/api/mongodb/users	Create MongoDB user	▼
DELETE	/api/mongodb/users	Delete MongoDB user	▼

Рис. 28 — Обработчики MDB агента для модификации MongoDB

### 3.3.13. API MDB агента для MongoDB

#### 3.3.13.1. Создание базы данных MongoDB

**Описание запроса:**

Маршрут: POST /api/mongodb/databases

Тело запроса: CreateMongoDatabaseRequestDTO

Возвращает: HTTP-ответ без тела (200 OK)

#### 3.3.13.2. Удаление базы данных MongoDB

**Описание запроса:**

Маршрут: DELETE /api/mongodb/databases

Тело запроса: DeleteMongoDatabaseRequestDTO

Возвращает: HTTP-ответ без тела (200 OK)

#### 3.3.13.3. Создание пользователя MongoDB

**Описание запроса:**

Маршрут: POST /api/mongodb/users

Тело запроса: CreateMongoUserRequestDTO

Возвращает: HTTP-ответ без тела (200 OK)

#### 3.3.13.4. Удаление пользователя MongoDB

**Описание запроса:**

Маршрут: DELETE /api/mongodb/users

Тело запроса: DeleteMongoUserRequestDTO

Возвращает: HTTP-ответ без тела (200 OK)

## 3.4. Объем выполненной работы

### 3.4.1. Объем реализованного API

В рамках данной работы реализована бизнес-логика и API с 44 обработчиками для различных сущностей.

MDB			^
GET	/api/mdb/resource-presets/{resourcePresetId}	Get resource preset	🔒
PUT	/api/mdb/resource-presets/{resourcePresetId}	Update resource preset	🔒
DELETE	/api/mdb/resource-presets/{resourcePresetId}	Delete resource preset	🔒
PUT	/api/mdb/quotas/upload-to-products	Upload quotas to multiple products	🔒
PUT	/api/mdb/quotas/transfer-between-products	Exchange quotas between products	🔒
GET	/api/mdb/projects/{projectId}	Get project	🔒
PUT	/api/mdb/projects/{projectId}	Update project	🔒
DELETE	/api/mdb/projects/{projectId}	Delete project	🔒
PUT	/api/mdb/operations/{operationId}/restart	Restart operation	🔒
PUT	/api/mdb/mongodb/internal/hosts	Update MongoDB hosts	🔒
GET	/api/mdb/mongodb/clusters/{clusterId}	Get MongoDB cluster	🔒
PUT	/api/mdb/mongodb/clusters/{clusterId}	Update MongoDB cluster	🔒
DELETE	/api/mdb/mongodb/clusters/{clusterId}	Delete MongoDB cluster	🔒
GET	/api/mdb/resource-presets	List resource presets	🔒
POST	/api/mdb/resource-presets	Create resource preset	🔒
POST	/api/mdb/quotas/simulate-transfer-between-products	Simulate transfer of quotas between products	🔒
POST	/api/mdb/quotas/mongodb/simulate-usage	Simulate usage of quotas for MongoDB cluster	🔒
GET	/api/mdb/projects	List projects	🔒
POST	/api/mdb/projects	Create project	🔒
GET	/api/mdb/mongodb/clusters	List MongoDB clusters	🔒
POST	/api/mdb/mongodb/clusters	Create MongoDB cluster	🔒
GET	/api/mdb/mongodb/clusters/{clusterId}/users	List MongoDB users	🔒

Рис. 29 — Обработчики для модуля MDB. Часть 1



POST	/api/mdb/mongodb/clusters/{clusterId}/users	Create MongoDB user	🔒
POST	/api/mdb/mongodb/clusters/{clusterId}/restore/{backupName}	Restore MongoDB cluster	🔒
GET	/api/mdb/mongodb/clusters/{clusterId}/databases	List MongoDB databases	🔒
POST	/api/mdb/mongodb/clusters/{clusterId}/databases	Create MongoDB database	🔒
GET	/api/mdb/mongodb/clusters/{clusterId}/backups	List MongoDB backups	🔒
POST	/api/mdb/mongodb/clusters/{clusterId}/backups	Create MongoDB backup	🔒
GET	/api/mdb/storage-classes	List storage classes	🔒
GET	/api/mdb/resources	List MDB resources	🔒
GET	/api/mdb/quotas/list-by-products	List quotas by products	🔒
GET	/api/mdb/operations	List operations	🔒
GET	/api/mdb/namespaces	List namespaces	🔒
GET	/api/mdb/mongodb/versions	List MongoDB versions	🔒
GET	/api/mdb/mongodb/users/roles	List available MongoDB roles	🔒
GET	/api/mdb/mongodb/clusters/{clusterId}/hosts	List MongoDB hosts	🔒
GET	/api/mdb/billing/product-quota-usage-report	Get product quota usage report	🔒
DELETE	/api/mdb/mongodb/clusters/{clusterId}/users/{userName}	Delete MongoDB user	🔒
DELETE	/api/mdb/mongodb/clusters/{clusterId}/databases/{databaseName}	Delete MongoDB database	🔒
DELETE	/api/mdb/mongodb/clusters/{clusterId}/backups/{backupName}	Delete backup of MongoDB cluster	🔒

Рис. 30 — Обработчики для модуля MDB. Часть 2

MongoDB			^
POST	/api/mongodb/databases	Create MongoDB database	▼
DELETE	/api/mongodb/databases	Delete MongoDB database	▼
POST	/api/mongodb/users	Create MongoDB user	▼
DELETE	/api/mongodb/users	Delete MongoDB user	▼

Рис. 31 — Обработчики для MDB агента

### 3.4.2. Объем кодовой базы

После завершения реализации программного продукта были проведены замеры на объем кода в строках при помощи консольной утилиты «Count Lines of Code»[34]. Объем кода для модуля MDB на Java без учета кодогенерации составляет более 12 тысяч строк и более 35 тысяч строк с учетом кодогенерации. Объем написанного кода для MDB агента на языке Python достигает 500 строк.

**Примечание:** для корректного подсчета количества строк для модуля MDB из репозитория был предварительно удален код модуля IDM.

Листинг 16 — Команда для проведения замеров в корне репозитория  
[github.com/onyxdb/onyxdb](https://github.com/onyxdb/onyxdb)

```
cloc --include-lang=Java,SQL,YAML,YML,JSON,Gradle,Markdown,Properties ./
```

github.com/AlDanial/cloc v 2.04 T=0.37 s (877.4 files/s, 93299.7 lines/s)

Language	files	blank	comment	code
Java	272	2555	1873	12768
SQL	10	55	41	6467
YAML	19	35	31	5430
JSON	2	0	0	4721
Gradle	14	55	29	276
Markdown	1	55	0	152
Properties	7	0	2	13
SUM:	325	2755	1976	29827

Рис. 32 — Замер количества строк кода в репозитории github.com/onyxdb/onyxdb без учета кодогенерации

github.com/AlDanial/cloc v 2.04 T=0.65 s (909.3 files/s, 104984.6 lines/s)

Language	files	blank	comment	code
Java	534	7998	7557	35203
SQL	6	32	29	6290
YAML	19	35	31	5430
JSON	2	0	0	4721
Gradle	14	55	29	276
Text	7	0	0	210
Markdown	1	55	0	152
Properties	7	0	2	13
SUM:	590	8175	7648	52295

Рис. 33 — Замер количества строк кода в репозитории github.com/onyxdb/onyxdb с учетом кодогенерации

Листинг 17 — Команда для проведения замеров в корне репозитория  
github.com/onyxdb/onyxdb-agent

```
cloc --include-lang=Python,SQL,YAML,YML,JSON,Markdown,Text ./
```

```
github.com/AlDanial/cloc v 2.04 T=0.02 s (988.5 files/s, 40282.8 lines/s)
```

Language	files	blank	comment	code
Python	15	176	14	535
YAML	2	10	0	43
Markdown	2	8	0	16
Text	1	0	0	13
SUM:	20	194	14	607

Рис. 34 — Замер количества строк кода в репозитории github.com/onyxdb/onyxdb-agent

### 3.5. Вывод по главе

В данной главе выполнен подробный разбор реализации модуля MDB, разрабатываемого в рамках DBaaS платформы в Kubernetes. Предоставлены схемы баз данных, которые используются сервисом, с описанием ассоциируемых с ними абстракций.

Разобрана реализация процесса асинхронной обработки операций по эксплуатации кластеров баз данных с описанием ключевых компонентов, благодаря которым происходит обслуживание нескольких кластеров баз данных одновременно. Данный функционал является основополагающим при разработке облачной платформы, которая должна выполнять множество задач единовременно, и значительно оптимизирует процесс эксплуатации распределенных баз данных. Также следует отметить, что архитектура процесса обработки задач реализована без привязки к абстракциям, связанным с базами данных, что позволяет переиспользовать разработанный инструмент в рамках платформы. Такой подход позволяет унифицировать ключевые процессы облачной платформы и облегчить ее эксплуатацию.

Также подробно описана автоматизация мониторинга баз данных, которая предлагает готовое решение для поставки и визуализации как логов, так и метрик. Помимо этого приведено обоснование разработки собственного агента, который дополняет функционал интегрированного оператора Persona MongoDB operator[42] и предоставляет возможность отслеживать доступность реплик кластера с минимальной задержкой для пользователя.

Кроме того, добавлена информация об устройстве системы квотирования и биллинга, которая значительно упрощает управление облачной платформой с автоматизацией баз данных. Также приведено описание конфигурации модуля MDB для развертывания в инфраструктуре Kubernetes и описание спецификации реализованного API.

## Заключение

Подводя итоги выполненной работы, следует отметить, что на данный момент облачные технологии являются одной из наиболее распространенных сред для построения масштабируемых платформ, однако эксплуатация большой инфраструктуры требует значительной автоматизации процессов.

В ходе анализа существующих решений были найдены различные подходы к автоматизации распределенных баз данных. Однако доступные решения имеют ряд ограничений, которые могут быть неприемлемыми для ряда пользователей в силу организационных ограничений или личных предпочтений. Среди выявленных потенциальных недостатков на рынке следует выделить следующие: недоступность продукта на территории РФ, предоставление проприетарных технологий, ограниченность функционала в открытых решениях или узкая направленность подобных проектов.

В рамках данной работы была разработана масштабируемая архитектура платформы, которая интегрирована с различными технологиями. При проектировании платформы были применены различные паттерны проектирования и концепции взаимодействия, которые способствуют построению адаптивного и устойчивого решения для автоматизации процессов в облачной инфраструктуре. Отдельное внимание было уделено выбору технологий для применения с описанием преимуществ и недостатков. Для совершенствования процесса эксплуатации распределенных баз данных был реализован различный функционал, среди которого асинхронное API для обслуживания баз данных, автоматизация поставки метрик и логов баз данных, поставка данных о доступности баз данных с минимальной задержкой, автоматизация управления базой данных MongoDB, а также функционал квотирования и биллинга ресурсов. Кроме того, при разработке проекта большой акцент делался на унификации компонентов платформы, благодаря чему платформа предоставляет единый функционал выполнения комплексных операций, а также функционал управления ресурсами, что значительно упрощает как пользовательский опыт, так и опыт разработки продукта в будущем.

Таким образом, выполненная работа полностью соответствует предъявляемым требованиям и решает поставленные задачи.

Дальнейшее развитие проекта предполагается по следующим ключевым направлениям:

- Расширение API для более детальной конфигурации баз данных MongoDB
- Реализация поддержки новых типов баз данных
- Реализация поддержки новых типов ресурсов системой квотирования
- Развитие системы биллинга для детализации предоставляемой статистики
- Увеличение автоматизации мониторинга путем интеграции с системой уведомлений
- Реализация динамического отслеживания реплик различных кластеров баз данных агентом для оптимизации инфраструктуры платформы

## Список используемой литературы

- [1] ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [2] ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [3] ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [4] ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [5] ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [6] ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [7] ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [8] ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [9] ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [10] ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
- [11] Kubernetes [Электронный ресурс] – URL: <https://kubernetes.io/> (дата обращения: 19.11.2024).
- [12] MongoDB [Электронный ресурс]. URL: <https://www.mongodb.com/> (дата обращения: 12.11.2024).
- [13] Yandex.Cloud [Электронный ресурс]. URL: <https://yandex.cloud/ru/> (дата обращения: 13.11.2024).
- [14] Selectel [Электронный ресурс]. URL: <https://selectel.ru/> (дата обращения: 13.11.2024).
- [15] KubeDB [Электронный ресурс]. URL: <https://kubedb.com/> (дата обращения: 14.11.2024).
- [16] AppsCode [Электронный ресурс]. URL: <https://appscode.com/> (дата обращения: 14.11.2024).

- [17] Percona Everest [Электронный ресурс]. URL: <https://github.com/percona/everest> (дата обращения: 14.11.2024).
- [18] Percona [Электронный ресурс]. URL: <https://percona.com> (дата обращения: 14.11.2024).
- [19] Grafana [Электронный ресурс] – URL: <https://grafana.com/> (дата обращения: 19.11.2024).
- [20] VictoriaMetrics [Электронный ресурс] – URL: <https://victoriametrics.com/> (дата обращения: 15.12.2024).
- [21] PostgreSQL [Электронный ресурс] – URL: <https://www.postgresql.org/> (дата обращения: 19.11.2024).
- [22] Clickhouse [Электронный ресурс] – URL: <https://clickhouse.com/> (дата обращения: 22.11.2024).
- [23] Redis [Электронный ресурс] – URL: <https://redis.io/> (дата обращения: 25.11.2024).
- [24] Spring Framework [Электронный ресурс] – URL: <https://spring.io/projects/spring-framework> (дата обращения: 18.12.2024).
- [25] Java JDK 21 [Электронный ресурс] – URL: <https://jdk.java.net/21/> (дата обращения: 19.11.2024).
- [26] Java [Электронный ресурс] – URL: <https://jdk.java.net/> (дата обращения: 19.11.2024).
- [27] Go/Golang [Электронный ресурс] – URL: <https://go.dev/> (дата обращения: 10.11.2024).
- [28] Python [Электронный ресурс] – URL: <https://www.python.org//> (дата обращения: 11.11.2024).
- [29] InfluxDB [Электронный ресурс] – URL: <https://prometheus.io/> (дата обращения: 25.11.2024).
- [30] Elastic [Электронный ресурс] – URL: <https://www.elastic.co/> (дата обращения: 19.11.2024).
- [31] OpenAPI Generator [Электронный ресурс]. URL: <https://github.com/OpenAPITools/openapi-generator> (дата обращения: 15.12.2024)
- [32] jOOQ (Java) [Электронный ресурс]. URL: <https://www.jooq.org/> (дата обращения: 15.12.2024)
- [33] Flyway (Java) [Электронный ресурс]. URL: <https://github.com/flyway/flyway> (дата обращения: 15.12.2024)
- [34] Count Lines of Code [Электронный ресурс]. URL: <https://github.com/AlDanial/cloc/> (дата обращения: 20.04.2025)
- [35] Minikube [Электронный ресурс]. URL: <https://minikube.sigs.k8s.io/docs/> (дата обращения: 15.12.2024)
- [36] Gradle [Электронный ресурс] – URL: <https://gradle.org/> (дата обращения: 19.11.2024).
- [37] Git [Электронный ресурс] – URL: <https://git-scm.com/> (дата обращения: 19.11.2024).

- [38] Swagger [Электронный ресурс] – URL: <https://swagger.io/> (дата обращения: 19.11.2024).
- [39] Docker [Электронный ресурс] – URL: <https://www.docker.com/> (дата обращения: 19.11.2024).
- [40] Testcontainers [Электронный ресурс] – URL: <https://testcontainers.com/> (дата обращения: 19.11.2024).
- [41] MinIO [Электронный ресурс] – URL: <https://min.io/> (дата обращения: 02.12.2024).
- [42] MongoDB operator by Percona [Электронный ресурс]. URL: <https://docs.percona.com/percona-operator-for-mongodb> (дата обращения: 10.11.2024).
- [43] VictoriaMetrics agent [Электронный ресурс] – URL: <https://docs.victoriametrics.com/victoriametrics/vmagent/> (дата обращения: 16.12.2024).
- [44] VictoriaMetrics operator [Электронный ресурс] – URL: <https://docs.victoriametrics.com/operator/> (дата обращения: 15.12.2024).
- [45] VictoriaLogs [Электронный ресурс] – URL: <https://docs.victoriametrics.com/victorialogs/> (дата обращения: 12.12.2024).
- [46] Vector [Электронный ресурс] – URL: <https://vector.dev/> (дата обращения: 01.12.2024).
- [47] Control planes and data planes [Электронный ресурс] – URL: <https://docs.aws.amazon.com/whitepapers/latest/aws-fault-isolation-boundaries/control-planes-and-data-planes.html> (дата обращения: 27.11.2024).
- [48] Operator pattern [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/> (дата обращения: 23.11.2024).
- [49] Percona MongoDB Server [Электронный ресурс] – URL: <https://docs.percona.com/percona-server-for-mongodb> (дата обращения: 10.11.2024).
- [50] Transactional outbox pattern [Электронный ресурс] – URL: <https://microservices.io/patterns/data/transactional-outbox.html> (дата обращения: 05.12.2024).
- [51] Kubernetes sidecar containers [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/workloads/pods/sidecar-containers/> (дата обращения: 09.11.2024).
- [52] Percona MongoDB exporter [Электронный ресурс] – URL: [https://github.com/percona/mongodb\\_exporter](https://github.com/percona/mongodb_exporter) (дата обращения: 02.12.2024).
- [53] Percona MongoDB backup agent [Электронный ресурс] – URL: <https://docs.percona.com/percona-backup-mongodb/details/pbm-agent.html> (дата обращения: 03.12.2024).
- [54] Kubernetes Pod [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/workloads/pods/> (дата обращения: 03.12.2024).
- [55] Kubernetes Service [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/services-networking/service/> (дата обращения: 03.12.2024).
- [56] JVM [Электронный ресурс] – URL: <https://docs.oracle.com/javase/specs/jvms/se21/html/> (дата обращения: 03.12.2024).

- [57] warmup [Электронный ресурс] – URL: <https://opensource.tbank.ru/top-core-libs/warmup> (дата обращения: 04.12.2024).
- [58] Т-Банк [Электронный ресурс] – URL: <https://tbank.ru> (дата обращения: 04.12.2024).
- [59] Quarkus [Электронный ресурс] – URL: <https://quarkus.io/> (дата обращения: 25.11.2024).
- [60] MySQL [Электронный ресурс] – URL: <https://www.mysql.com/> (дата обращения: 20.11.2024).
- [61] InnoDB [Электронный ресурс] – URL: <https://dev.mysql.com/doc/refman/8.4/en/innodb-storage-engine.html> (дата обращения: 04.12.2024).
- [62] Андрей Сальников. Индексы в PostgreSQL. Как понять, что создавать [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=ju9F80vnL4E> (дата обращения: 01.11.2024).
- [63] Data Egret [Электронный ресурс] – URL: <https://dataegret.com/> (дата обращения: 05.12.2024).
- [64] PostgreSQL VACUUM [Электронный ресурс] – URL: <https://www.postgresql.org/docs/current/sql-vacuum.html> (дата обращения: 23.11.2024).
- [65] Patroni [Электронный ресурс] – URL: <https://github.com/patroni/patroni> (дата обращения: 23.11.2024).
- [66] Stolon [Электронный ресурс] – URL: <https://github.com/sorintlab/stolon> (дата обращения: 10.11.2024).
- [67] Владимир Бородин. Как устроены сервисы управляемых баз данных в Яндекс.Облаке [Электронный ресурс] – URL: <https://habr.com/ru/companies/yandex/articles/477860/> (дата обращения: 01.11.2024).
- [68] Полина Кудрявцева. Паттерны управления базами данных в multi-cluster Kubernetes среде [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=HC229E8tisg> (дата обращения: 25.11.2024).
- [69] Avito [Электронный ресурс] – URL: <https://www.avito.ru> (дата обращения: 10.11.2024).
- [70] Сергей Балдин. А можно погорячее? Чем и как мы прогреваем Spring-микросервисы [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=V7bXSZzjWoY> (дата обращения: 23.11.2024).
- [71] Максим Иванов. Разработка распределенной очереди с отложенными задачами на основе PostgreSQL [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=tw1wJmln-nM> (дата обращения: 10.11.2024).
- [72] Алексей Кашин. Надежно отправляем события в Apache Kafka. От CDC до паттерна Transactional Outbox [Электронный ресурс] – URL: [https://www.youtube.com/watch?v=b42gkda\\_6s](https://www.youtube.com/watch?v=b42gkda_6s) (дата обращения: 05.12.2024).
- [73] Apache Druid [Электронный ресурс] – URL: <https://druid.apache.org/> (дата обращения: 04.12.2024).



- [74] KeyDB [Электронный ресурс] – URL: <https://docs.keydb.dev/> (дата обращения: 01.11.2024).
- [75] Aliaksandr Valialkin. Prometheus vs VictoriaMetrics benchmark on node\_exporter metrics [Электронный ресурс] – URL: <https://valyala.medium.com/prometheus-vs-victoriametrics-benchmark-on-node-exporter-metrics-4ca29c75590f> (дата обращения: 23.11.2024).
- [76] Aliaksandr Valialkin. Insert benchmarks with inch: InfluxDB vs VictoriaMetrics [Электронный ресурс] – URL: <https://valyala.medium.com/insert-benchmarks-with-inch-influxdb-vs-victoriametrics-e31a41ae2893> (дата обращения: 23.11.2024).
- [77] Aliaksandr Valialkin. Measuring vertical scalability for time series databases in Google Cloud [Электронный ресурс] – URL: <https://valyala.medium.com/measuring-vertical-scalability-for-time-series-databases-in-google-cloud-92550d78d8> (дата обращения: 05.12.2024).
- [78] InfluxDB [Электронный ресурс] – URL: <https://www.influxdata.com/> (дата обращения: 25.11.2024).
- [79] TimescaleDB [Электронный ресурс] – URL: <https://www.timescale.com/> (дата обращения: 25.11.2024).
- [80] OpenTelemetry [Электронный ресурс] – URL: <https://opentelemetry.io/> (дата обращения: 05.12.2024).
- [81] Aliaksandr Valialkin. How do open source solutions for logs work: Elasticsearch, Loki and VictoriaLogs [Электронный ресурс] – URL: <https://itnext.io/how-do-open-source-solutions-for-logs-work-elasticsearch-loki-and-victorialogs-9f70> (дата обращения: 23.11.2024).
- [82] VictoriaLogs FAQ [Электронный ресурс] – URL: <https://docs.victoriametrics.com/victorialogs/faq/> (дата обращения: 23.11.2024).
- [83] Grafana Loki [Электронный ресурс] – URL: <https://grafana.com/oss/loki/> (дата обращения: 23.11.2024).
- [84] Grafana Loki [Электронный ресурс] – URL: <https://grafana.com/oss/loki/> (дата обращения: 23.11.2024).
- [85] Ceph [Электронный ресурс] – URL: <https://ceph.io> (дата обращения: 23.11.2024).
- [86] S3 [Электронный ресурс] – URL: <https://aws.amazon.com/ru/s3/> (дата обращения: 23.11.2024).
- [87] Bitnami Helm Chart for MongoDB [Электронный ресурс] – URL: <https://artifacthub.io/packages/helm/bitnami/mongodb> (дата обращения: 01.11.2024).
- [88] MongoDB community operator [Электронный ресурс] – URL: <https://github.com/mongodb/mongodb-kubernetes-operator> (дата обращения: 23.11.2024).
- [89] MongoDB enterprise operator [Электронный ресурс] – URL: <https://www.mongodb.com/docs/kubernetes-operator/current/> (дата обращения: 23.11.2024).

- [90] Logstash [Электронный ресурс] – URL: <https://www.elastic.co/logstash> (дата обращения: 20.11.2024).
- [91] Fluent Bit [Электронный ресурс] – URL: <https://fluentbit.io/> (дата обращения: 04.12.2024).
- [92] Github Actions [Электронный ресурс] – URL: <https://docs.github.com/en/actions> (дата обращения: 23.11.2024).
- [93] Kubernetes namespace [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/> (дата обращения: 05.12.2024).
- [94] VMServiceScrape [Электронный ресурс] – URL: <https://docs.victoriametrics.com/operator/resources/vmservicescrape/> (дата обращения: 07.12.2024).
- [95] Kubernetes DaemonSet [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/> (дата обращения: 01.12.2024).
- [96] Kubernetes Node [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/architecture/nodes/> (дата обращения: 23.11.2024).
- [97] Vector Kubernetes Logs source [Электронный ресурс] – URL: [https://vector.dev/docs/reference/configuration/sources/kubernetes\\_logs/](https://vector.dev/docs/reference/configuration/sources/kubernetes_logs/) (дата обращения: 22.11.2024).
- [98] Kafka [Электронный ресурс] – URL: <https://kafka.apache.org/> (дата обращения: 20.11.2024).
- [99] RabbitMQ [Электронный ресурс] – URL: <https://www.rabbitmq.com/> (дата обращения: 20.11.2024).
- [100] Kubernetes Storage Class [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/storage/storage-classes/> (дата обращения: 20.11.2024).
- [101] JDBC [Электронный ресурс] – URL: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> (дата обращения: 22.11.2024).
- [102] Kubernetes Secret [Электронный ресурс] – URL: <https://kubernetes.io/docs/concepts/configuration/secret/> (дата обращения: 22.11.2024).