

Федеральное государственное образовательное бюджетное  
учреждение высшего образования  
«Финансовый университет при Правительстве Российской Федерации»  
(Финансовый университет)

Факультет информационных технологий и анализа больших данных  
Департамент информационной безопасности

## ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему «Создание и эксплуатация стрессо- и отказоустойчивых  
кластеров баз хранения данных на базе отечественного ПО и решений с  
открытым исходным кодом с последующим тестированием целостности  
системы и ее анализ

Направление подготовки 10.03.01 «Информационная безопасность»  
Профиль «Безопасность автоматизированных систем в финансово-  
банковской сфере»

Выполнил: студент группы ИБ19-3  
Зотов Андрей Сергеевич \_\_\_\_\_  
(подпись)

Руководитель: ассистент,  
Петров Иван Андреевич \_\_\_\_\_  
(подпись)

ВКР соответствует предъявленным  
требованиям

Руководитель департамента  
\_\_\_\_\_ А.В. Царегородцев  
(подпись)

« \_\_\_\_ » \_\_\_\_\_ 2023 г.

## **СОДЕРЖИМОЕ**

<b>ВВЕДЕНИЕ .....</b>	<b>3</b>
<b>ГЛАВА 1. ПРИНЦИПЫ РАБОТЫ КЛАСТЕРОВ БАЗ ДАННЫХ .....</b>	<b>7</b>
1.1.    Описание принципов работы кластеров баз данных .....	7
1.2.    Обзор методов и технологий обеспечения стойкости к стрессу и отказам и законодательные требования .....	14
1.3.    Основные требования к созданию кластера баз данных .....	19
<b>ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ ДАННЫХ И ЕЕ ТЕСТИРОВАНИЕ .....</b>	<b>24</b>
2.1.    Выбор ПО и решений для создания кластера баз данных, теоретический разбор работы утилит .....	24
2.2.    Реализация кластера баз данных .....	34
2.3.    Проверка работоспособности и надежности кластера баз данных .....	42
<b>ГЛАВА 3. ТЕСТИРОВАНИЕ ОБРАБОТКИ ДАННЫХ И ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ .....</b>	<b>46</b>
3.1.    Тестирование на потоке данных .....	46
3.2.    Выявление проблем и недостатков в кластерах и их решение .....	51
3.3.    Выявление общих проблем и способы повышения эффективности .....	55
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>58</b>
<b>СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ .....</b>	<b>60</b>

## ВВЕДЕНИЕ

В современном мире информационные технологии играют важную роль в повседневной жизни людей и деятельности организаций. Количество данных, генерируемых и обрабатываемых информационными системами, постоянно увеличивается, и поэтому необходимо обеспечивать надежное и безопасное хранение этих данных. В связи с этим возрастает значимость кластеров баз данных, которые позволяют обеспечить высокую доступность данных и обеспечить их сохранность при возможных отказах оборудования или программного обеспечения.

Одной из основных проблем при создании кластеров баз данных является обеспечение их стабильной работы и отказоустойчивости. Кластер баз данных - это группа компьютеров, объединенных в единую систему с целью увеличения производительности, надежности и доступности. Однако, несмотря на все преимущества, кластеры баз данных сталкиваются со множеством проблем, связанных с нестабильностью оборудования, неправильной настройкой программного обеспечения, ошибками в работе операционной системы и другими факторами, которые могут привести к отказам системы и потере данных.

Проблема обеспечения надежности кластеров баз данных является очень актуальной в настоящее время. В условиях постоянно растущей нагрузки на информационные системы и повышенных требований к надежности, доступности и безопасности данных, создание отказоустойчивых кластеров баз данных становится особенно важным. Кроме того, существует большое количество различных программных и аппаратных решений для создания кластеров баз данных, что усложняет выбор наиболее оптимального и эффективного решения.

**Целью данной работы является** создание и эксплуатация стрессо- и отказоустойчивых кластеров баз хранения данных на базе отечественного ПО и решений с открытым исходным кодом с последующим тестированием

целостности системы и ее анализ. Для достижения этой цели были поставлены следующие задачи:

1. Изучить существующие технологии и методы создания кластеров баз данных на базе отечественного ПО и решений с открытым исходным кодом.
2. Разработать архитектуру кластеров баз данных с учетом требований к отказоустойчивости и стрессоустойчивости.
3. Провести настройку и установку программного и аппаратного обеспечения для создания кластеров баз данных.
4. Провести тестирование кластеров баз данных на стабильность, отказоустойчивость и стрессоустойчивость.
5. Провести анализ результатов тестирования и выявить проблемы, связанные с отказами и ошибками в работе кластеров баз данных.
6. Разработать методы и рекомендации по улучшению надежности и стабильности кластеров баз данных.
7. Провести сравнительный анализ различных программных и аппаратных решений для создания кластеров баз данных на базе отечественного ПО и решений с открытым исходным кодом.
8. Сформулировать выводы и рекомендации по созданию и эксплуатации отказоустойчивых кластеров баз данных на базе отечественного ПО и решений с открытым исходным кодом.

Реализация данных задач позволит достичь поставленной цели и обеспечить надежность и стабильность кластеров баз данных на базе отечественного ПО и решений с открытым исходным кодом.

**Существует множество решений** для создания кластеров баз данных, как на коммерческой основе, так и на основе открытых исходных кодов. В данной работе рассматриваются решения на базе отечественного ПО и решений с открытым исходным кодом.

Одним из самых популярных и широко используемых решений для создания кластеров баз данных на базе открытого исходного кода является

PostgreSQL. PostgreSQL - это объектно-реляционная система управления базами данных (СУБД), которая обеспечивает высокую надежность, безопасность и производительность. Одним из преимуществ PostgreSQL является возможность создания многомастерных репликаций, что позволяет создавать кластеры баз данных с высокой отказоустойчивостью и доступностью.

Еще одним популярным решением для создания кластеров баз данных является MySQL Cluster. MySQL Cluster представляет собой распределенную базу данных с открытым исходным кодом, которая обеспечивает высокую отказоустойчивость, масштабируемость и производительность. MySQL Cluster имеет ряд функций, таких как автоматическое разделение данных, автоматический балансировщик нагрузки, репликация и партиционирование.

Одним из популярных отечественных решений для создания кластеров баз данных является СУБД Yandex ClickHouse. ClickHouse - это колоночная СУБД с открытым исходным кодом, которая была разработана для анализа данных. ClickHouse обладает высокой скоростью выполнения запросов, поддерживает масштабирование и репликацию. ClickHouse используется в таких компаниях, как Яндекс, Mail.ru, CERN и других.

Также существуют решения, такие как Couchbase, Riak, Apache Cassandra, которые позволяют создавать кластеры баз данных с высокой отказоустойчивостью и доступностью.

В ходе обзора литературы и анализа существующих решений было выявлено, что создание отказоустойчивых и стрессоустойчивых кластеров баз данных является актуальной и важной задачей в области информационной безопасности и сетевых технологий. Это связано с тем, что в современном мире существует множество угроз и рисков, которые могут привести к сбоям и потере данных в кластере баз данных. Такие проблемы могут привести к серьезным последствиям, таким как потеря денежных средств, нарушение конфиденциальности данных, потеря доверия клиентов и др.

Создание отказоустойчивых и стрессоустойчивых кластеров баз данных позволяет решить эти проблемы. Такие кластеры баз данных могут обеспечить высокую доступность, надежность и производительность, что позволяет уменьшить риски и обеспечить сохранность данных в любых условиях.

## ГЛАВА 1. ПРИНЦИПЫ РАБОТЫ КЛАСТЕРОВ БАЗ ДАННЫХ

### 1.1. Описание принципов работы кластеров баз данных

Кластер баз данных представляет собой группу из двух или более независимых компьютеров, которые работают вместе как единая система. Каждый компьютер в кластере называется узлом, и каждый узел может быть как физическим, так и виртуальным. Кластер баз данных может работать как в локальной, так и в глобальной сети, объединяя узлы, находящиеся на различных удаленных местах.

Основными принципами работы кластеров баз данных являются:

**1. Репликация данных.** Каждый узел в кластере хранит копию данных, которые доступны всем пользователям кластера. Репликация данных может осуществляться как синхронно, так и асинхронно. При синхронной репликации данные передаются на другой узел непосредственно в момент изменения, что обеспечивает максимальную надежность и целостность данных, но может замедлять производительность системы. Асинхронная репликация позволяет ускорить производительность системы, но может приводить к потере данных в случае отказа узла.

**2. Шардинг данных.** Шардинг — это процесс разделения базы данных на отдельные фрагменты, которые хранятся на разных узлах кластера. Это позволяет распределить нагрузку на кластер и ускорить работу с данными. При шардинге данные могут быть разделены как по горизонтали, так и по вертикали. При горизонтальном шардинге данные разделяются по строкам, а при вертикальном - по столбцам.

**3. Балансировка нагрузки.** Балансировка нагрузки позволяет распределять запросы к базе данных между узлами кластера для обеспечения максимальной производительности и доступности данных. Балансировка нагрузки может осуществляться на уровне приложения, на уровне операционной системы, или на уровне кластера базы данных.

**Кластеризация.** Механизм репликации данных в кластерах баз данных предназначен для обеспечения надежности и целостности данных путем

создания копий данных на нескольких узлах кластера. Когда данные изменяются на одном узле кластера, эти изменения должны быть автоматически синхронизированы на всех остальных узлах, где хранятся копии данных. Подробная схема изложена на рисунке 1:

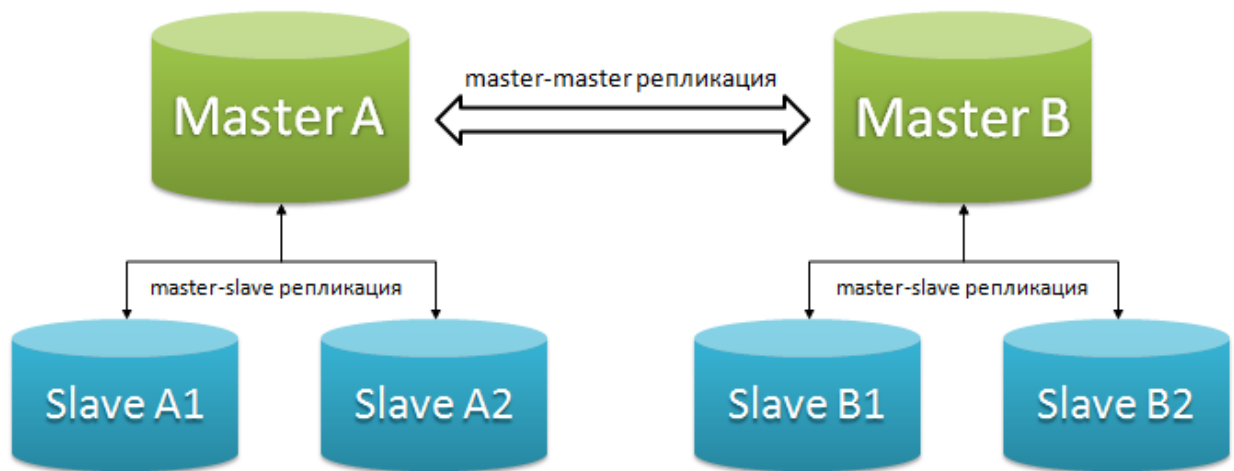


Рисунок 1. Базовая схема работы репликации

Существует несколько типов механизмов репликации данных:

1. Мастер-мастер репликация. В этом случае каждый узел является мастером, который может принимать запросы на запись данных. Когда один узел получает запрос на изменение данных, он автоматически синхронизирует изменения на других узлах кластера. Это обеспечивает высокую доступность данных и масштабируемость системы.

2. Мастер-слейв репликация. В этом случае один узел является мастером (главным), который принимает запросы на запись данных, а все остальные узлы являются слейвами (подопечным), которые хранят копии данных. Когда мастер получает запрос на изменение данных, он автоматически синхронизирует изменения на всех слейвах. Это обеспечивает высокую доступность данных и более простую архитектуру системы.

3. Репликация на основе журнала транзакций. В этом случае изменения данных записываются в журнал транзакций, который затем передается на другие узлы кластера для выполнения тех же операций. Этот механизм репликации обеспечивает максимальную целостность данных, но может замедлять производительность системы.



4. Репликация на основе читателя. В этом случае узлы кластера разделяются на две группы: группу мастеров и группу читателей. Мастеры принимают запросы на изменение данных, а читатели хранят копии данных для чтения. Когда данные изменяются на мастере, они автоматически синхронизируются на всех читателях. Этот механизм репликации обеспечивает высокую доступность данных и масштабируемость системы.

Кроме того, механизмы репликации могут быть реализованы как синхронно, так и асинхронно. При синхронной репликации изменения на узле кластера передаются на другие узлы непосредственно в момент изменения, что обеспечивает максимальную надежность и целостность данных, но может замедлять производительность системы. При асинхронной репликации изменения передаются на другие узлы кластера после завершения операции изменения данных. Это позволяет ускорить производительность системы, но может приводить к потере данных в случае отказа узла.

В целом, механизмы репликации данных в кластерах баз данных являются важным механизмом для обеспечения надежности и целостности данных. Они позволяют создавать масштабируемые и отказоустойчивые системы, которые способны обрабатывать большие объемы данных и поддерживать работу в условиях высоких нагрузок. Однако, для правильной реализации механизмов репликации необходимо учитывать особенности конкретной системы и выбирать соответствующие технологии и решения.

**Шардинг данных.** Данный механизм в кластерах баз данных предназначен для распределения данных между несколькими узлами кластера. Это позволяет ускорить доступ к данным и обеспечить более эффективное использование ресурсов кластера. Схема изложена на рисунке 2:

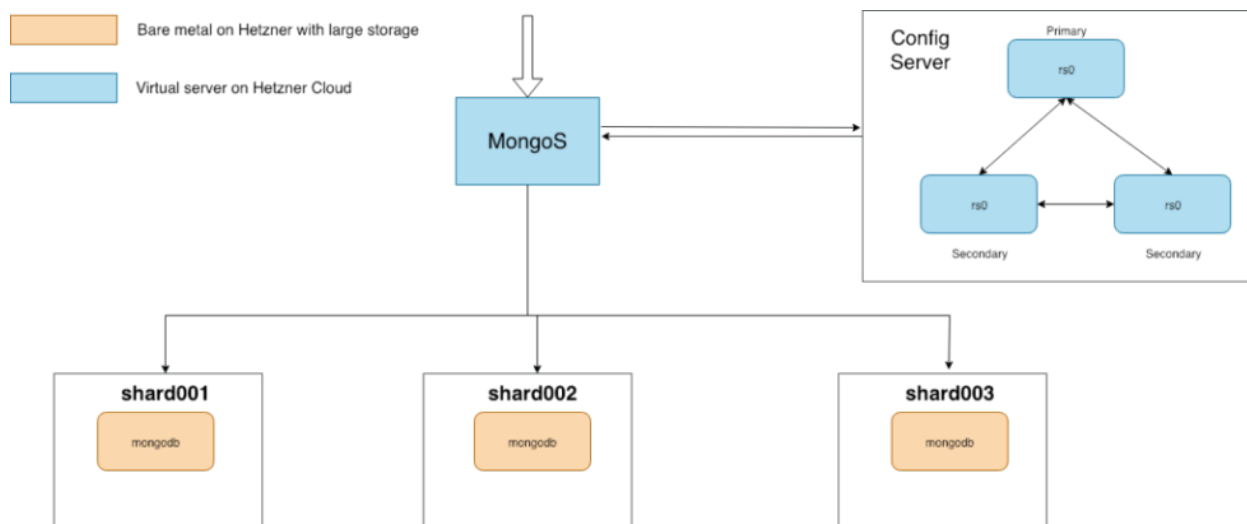


Рисунок 2. Реализации шардинга на примере БД MongoDB

Шардинг данных может быть реализован по двум основным типам: горизонтальному и вертикальному.

1. Горизонтальный шардинг данных предполагает разбиение данных на отдельные фрагменты по строкам таблицы. Таким образом, каждый фрагмент данных хранится на отдельном узле кластера. Например, если у нас есть таблица "Пользователи", мы можем разделить ее на несколько фрагментов, каждый из которых будет содержать записи пользователей с определенными диапазонами идентификаторов. Эти фрагменты данных затем будут распределены по узлам кластера, что позволит снизить нагрузку на каждый узел и ускорить доступ к данным.

2. Вертикальный шардинг данных предполагает разбиение данных на отдельные фрагменты по столбцам таблицы. Таким образом, каждый фрагмент данных будет содержать только определенные столбцы из таблицы. Например, если у нас есть таблица "Пользователи", мы можем разделить ее на два фрагмента: первый фрагмент будет содержать только данные об имени, фамилии и адресе пользователей, а второй фрагмент будет содержать данные об их электронной почте и телефонных номерах. Эти фрагменты данных затем будут распределены по узлам кластера, что позволит снизить объем данных, хранимых на каждом узле, и ускорить доступ к данным.

Шардинг данных может быть реализован как на уровне приложения, так и на уровне базы данных. Например, на уровне приложения можно

использовать схемы и префиксы таблиц для разделения данных между разными узлами кластера. На уровне базы данных можно использовать специальные программные решения, такие как *MongoDB Sharding*, которые автоматически распределяют данные между узлами кластера.

Однако, при реализации механизмов шардинга необходимо учитывать особенности конкретной системы и выбирать соответствующие технологии и решения. Например, горизонтальный шардинг может быть более эффективным для систем, в которых объем данных растет очень быстро, тогда как вертикальный шардинг может быть более эффективным для систем, в которых количество пользователей и запросов на чтение и запись данных высоко.

Кроме того, при реализации механизмов шардинга необходимо учитывать возможность балансировки нагрузки между узлами кластера. Это может быть реализовано через специальные программные решения, которые автоматически перераспределяют данные между узлами кластера в зависимости от текущей нагрузки.

В целом, механизмы шардинга данных в кластерах баз данных являются важным механизмом для обеспечения эффективного использования ресурсов кластера и ускорения доступа к данным. Однако, при реализации механизмов шардинга необходимо учитывать особенности конкретной системы и выбирать соответствующие технологии и решения, а также обеспечивать балансировку нагрузки между узлами кластера.

**Балансировка нагрузки.** Механизм балансировки нагрузки в кластерах баз данных предназначен для распределения запросов на чтение и запись данных между несколькими узлами кластера. Это позволяет увеличить производительность системы, обеспечить более эффективное использование ресурсов и повысить отказоустойчивость системы (см. рисунок 3).

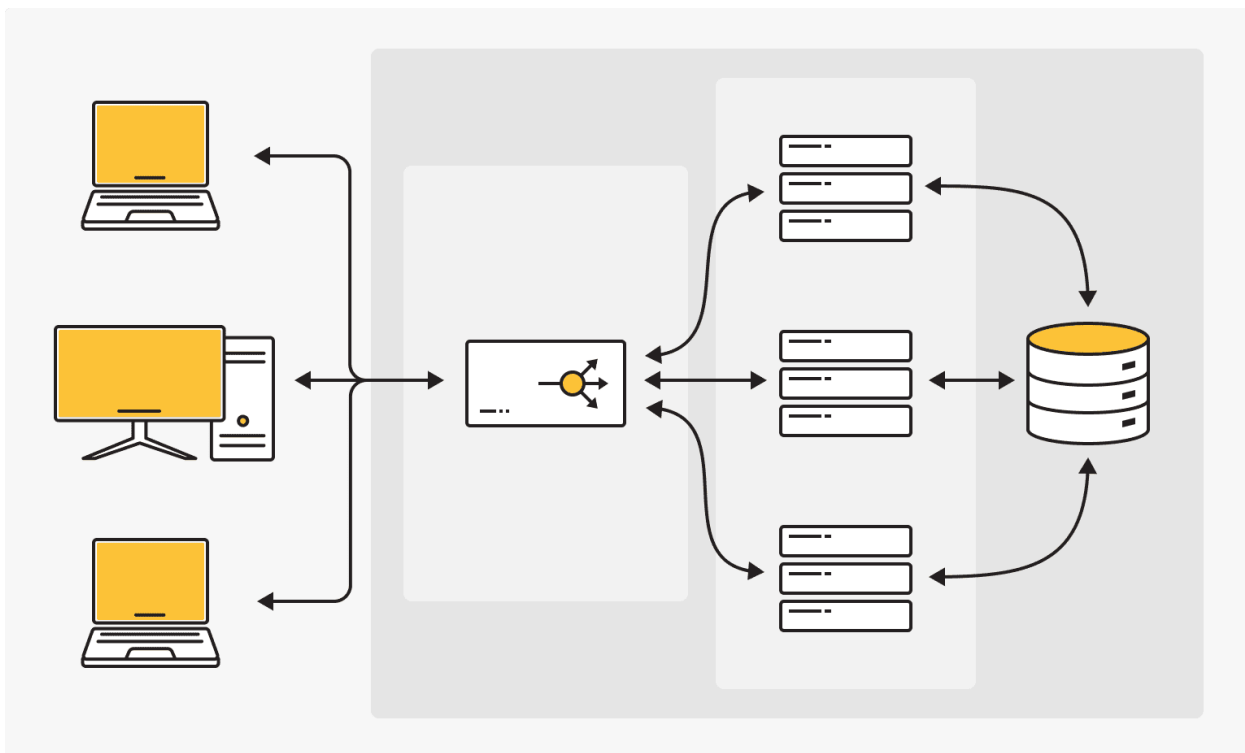


Рисунок 3. Схема работы балансировщика нагрузки на трех серверах

Механизмы балансировки нагрузки могут быть реализованы по разным принципам, включая следующие:

1. Распределение нагрузки на основе алгоритма ***Round-Robin***. В этом случае запросы по очереди отправляются на каждый узел кластера. Этот алгоритм прост в реализации и может быть эффективным для систем с небольшим количеством узлов кластера и однородной нагрузкой на систему.

2. Распределение нагрузки на основе алгоритма ***Least-Connections***. В этом случае запросы отправляются на узел кластера с наименьшим количеством активных соединений. Этот алгоритм более эффективен для систем с неоднородной нагрузкой на систему.

3. Распределение нагрузки на основе алгоритма ***IP-Hash***. В этом случае запросы отправляются на узел кластера на основе хэш-значения IP-адреса клиента. Этот алгоритм обеспечивает более равномерное распределение нагрузки в системе.

4. Распределение нагрузки на основе алгоритма ***Content-Hash***. В этом случае запросы отправляются на узел кластера на основе хэш-значения содержимого запроса. Этот алгоритм позволяет обеспечить более

эффективное использование ресурсов кластера, когда запросы имеют различный уровень нагрузки на систему.

Кроме того, механизмы балансировки нагрузки могут быть реализованы как на уровне приложения, так и на уровне сети. Например, на уровне приложения можно использовать специальные программные решения, такие как *HAProxy* или *Patroni*, которые позволяют балансировать нагрузку между разными узлами кластера. На уровне сети можно использовать специальное оборудование, такое как балансировщики нагрузки или контент-переключатели, которые позволяют балансировать нагрузку между разными узлами кластера на уровне сети.

При реализации механизмов балансировки нагрузки необходимо учитывать особенности конкретной системы и выбирать соответствующие технологии и решения. Например, алгоритм *Round-Robin* может быть более эффективным для систем с небольшим количеством узлов кластера, тогда как алгоритм *Least-Connections* может быть более эффективным для систем с большим количеством узлов и неоднородной нагрузкой на систему.

Кроме того, при реализации механизмов балансировки нагрузки необходимо учитывать возможность отказов узлов кластера и обеспечивать механизмы репликации данных и механизмы автоматического перенаправления запросов на другие узлы кластера в случае отказа. Это позволит обеспечить высокую отказоустойчивость системы и сохранить целостность данных.

В целом, механизмы балансировки нагрузки в кластерах баз данных являются важным механизмом для обеспечения эффективного использования ресурсов кластера и увеличения производительности системы. Однако, при реализации механизмов балансировки нагрузки необходимо учитывать особенности конкретной системы и выбирать соответствующие технологии и решения, а также обеспечивать отказоустойчивость системы и сохранение целостности данных.

**Вывод:** были проанализированы и изучены существующие методики повышения доступности, уровня защиты консистентности данных, обрабатываемых системой. Далее они будут применены на практике.

## **1.2. Обзор методов и технологий обеспечения стойкости к стрессу и отказам и законодательные требования**

Обеспечение стойкости к стрессу и отказам в кластерах баз данных является важной задачей, которая требует использования различных методов и технологий. В данном параграфе будет рассмотрен обзор некоторых методов и технологий, которые могут быть использованы для обеспечения стойкости к стрессу и отказам.

**1. Резервное копирование данных.** Резервное копирование данных является одним из основных методов обеспечения стойкости к отказам и защиты от потери данных. Этот метод предполагает создание дублированных копий данных, которые могут быть использованы для восстановления в случае сбоя или потери основных данных. Резервное копирование может быть реализовано на разных уровнях, включая уровень приложения или уровень операционной системы и базы данных. Регулярное выполнение резервного копирования данных и их хранение на надежных носителях играет важную роль в обеспечении сохранности и доступности информации.

**2. Репликация данных.** Репликация данных представляет собой механизм, при помощи которого создаются дубликаты данных на нескольких узлах в кластере. Это позволяет достичь стойкости к отказам и обеспечить непрерывную доступность данных даже в случае сбоя или отказа одного или нескольких узлов. В процессе репликации данные синхронизируются между узлами, что гарантирует их консистентность и надежность. Если один из узлов становится недоступным, клиентские запросы автоматически перенаправляются на другие доступные узлы, где уже существуют копии данных. Это обеспечивает непрерывную работу системы и минимизирует потенциальное влияние отказов на доступность и целостность данных.

**3. Кластеризация.** Кластеризация баз данных представляет собой процесс объединения нескольких узлов в единый кластер для обеспечения стойкости к отказам и повышения доступности данных. В этом методе несколько физических или виртуальных серверов объединяются в кластер, который функционирует как единая сущность для обработки запросов и хранения данных. При такой конфигурации, если один из узлов испытывает сбой или недоступен, другие узлы в кластере автоматически принимают на себя его функции, обеспечивая непрерывность работы системы. Кластеризация позволяет распределить нагрузку между узлами, предоставить высокую доступность данных и улучшить общую производительность системы.

**4. Мониторинг состояния системы.** Мониторинг состояния системы является важным методом обеспечения стойкости к отказам и непрерывной работой кластера баз данных. Этот метод включает постоянное отслеживание состояния каждого узла в кластере, анализ ключевых метрик и параметров производительности, а также обнаружение возможных проблем или неисправностей. Мониторинг состояния системы позволяет оперативно выявлять потенциальные проблемы, такие как высокая загрузка узлов, сбои в работе, недоступность или деградация производительности. При нахождении таких проблем мониторинг системы генерирует уведомления и предупреждения, что позволяет операторам принять необходимые меры для предотвращения отказов или восстановления работы системы. Мониторинг состояния системы также позволяет собирать данные о производительности и использовании ресурсов, анализировать тренды и уровень нагрузки. Это позволяет операторам системы принимать информированные решения о масштабировании, оптимизации и улучшении производительности кластера баз данных. В целом, мониторинг состояния системы является важным инструментом для обеспечения непрерывной работы и стабильности кластера баз данных. Он позволяет оперативно реагировать на проблемы, устранять их и предотвращать возможные сбои, обеспечивая надежность и доступность данных.

**5. Балансировка нагрузки.** Балансировка нагрузки является важным методом обеспечения стойкости к отказам и оптимальной производительности кластера баз данных. Этот метод заключается в распределении запросов и нагрузки между несколькими узлами кластера с целью достижения более равномерной нагрузки на каждый узел. Балансировка нагрузки позволяет предотвратить перегрузку одного или нескольких узлов, что может привести к снижению производительности и возникновению проблем. Путем равномерного распределения запросов и нагрузки на узлы кластера достигается оптимальное использование ресурсов и повышается отказоустойчивость системы. Существуют различные методы и алгоритмы балансировки нагрузки, такие как раунд-робин, IP-сессии, алгоритмы с учетом нагрузки и другие. Они позволяют распределять запросы таким образом, чтобы каждый узел получал примерно равное количество запросов, а загрузка на них была равномерной. Балансировка нагрузки также может включать механизмы мониторинга состояния узлов, чтобы определить и реагировать на изменения в нагрузке или доступности. Это позволяет автоматически адаптировать балансировку нагрузки и перенаправлять запросы на доступные и менее загруженные узлы.

**6. Использование специальных инструментов и технологий.** Для обеспечения стойкости к отказам могут быть использованы различные инструменты и технологии, такие как контейнеризация, виртуализация, микросервисная архитектура и другие. Контейнеризация и виртуализация позволяют создавать изолированные среды, которые могут быть легко переносимы между различными узлами кластера и обеспечивают возможность быстрого восстановления в случае отказов. Микросервисная архитектура позволяет разделять приложение на отдельные сервисы, что позволяет обеспечивать более гибкое управление и масштабирование системы.

Тестирование и анализ системы. Тестирование и анализ системы являются важными методами для обеспечения стойкости к стрессу и отказам. Тестирование позволяет выявлять проблемы и уязвимости системы, а анализ



позволяет оптимизировать работу системы и обнаруживать проблемы до их появления.

В целом, использование различных методов и технологий обеспечения стойкости к стрессу и отказам в кластерах баз данных позволяет обеспечить бесперебойную работу системы и сохранность данных в случае возникновения проблем. При выборе методов и технологий необходимо учитывать особенности конкретной системы и обеспечивать максимальную отказоустойчивость и сохранность данных.

**В России существует ряд законодательных требований, которые относятся к кластерам баз хранения данных. Некоторые из основных требований, которые могут быть применены к кластерам баз данных в России, включают в себя следующее:**

1. Федеральный закон "О персональных данных". Данный закон устанавливает требования к обработке персональных данных в России, в том числе и при их хранении в кластерах баз данных. Согласно этому закону, персональные данные должны храниться на территории России, если они получены при оказании услуг на территории России, за исключением случаев, когда законодательством Российской Федерации установлены иные требования.

2. Федеральный закон "О связи". Данный закон устанавливает требования к защите конфиденциальности связи. Кластеры баз данных, которые хранят данные связи, такие как электронные письма и текстовые сообщения, должны соответствовать требованиям этого закона.

3. Федеральный закон "О защите прав потребителей". Данный закон устанавливает требования к качеству услуг, которые предоставляются потребителям. Кластеры баз данных, которые используются для предоставления услуг потребителям, должны соответствовать требованиям этого закона.

4. Федеральный закон "Об информации, информационных технологиях и о защите информации". Данный закон устанавливает требования к защите

информации, которая хранится в кластерах баз данных. В соответствии с этим законом, организации должны обеспечить защиту информации от несанкционированного доступа, изменения и уничтожения.

5. Постановление Правительства РФ "Об утверждении требований к защите персональных данных при их обработке в информационных системах персональных данных". Данное постановление устанавливает требования к защите персональных данных при их обработке в кластерах баз данных.

В целом, при использовании кластеров баз данных в России необходимо учитывать законодательные требования, которые могут применяться к этим системам, и обеспечивать соответствие требованиям законодательства Российской Федерации. Кроме того, в России существуют нормативные документы, которые регулируют конкретные отрасли и области деятельности, где используются кластеры баз данных. Например, в банковской сфере могут применяться требования к хранению данных и обеспечению их конфиденциальности, в медицинской сфере - требования к обработке медицинских данных и т. д.

Для обеспечения соответствия законодательным требованиям в России можно использовать различные методы и технологии, такие как шифрование данных, аутентификация пользователей, контроль доступа, резервное копирование данных и т.д. Кроме того, при проектировании и эксплуатации кластеров баз данных в России необходимо учитывать особенности законодательства и обеспечивать максимальную стойкость к стрессу и отказам, чтобы минимизировать риски нарушения требований законодательства и потери данных.

В целом, обеспечение соответствия законодательным требованиям является важным аспектом проектирования и эксплуатации кластеров баз данных в России. При выборе методов и технологий необходимо учитывать особенности конкретной системы и требования законодательства Российской Федерации, чтобы обеспечить максимальную защиту данных и бесперебойную работу системы.

**Вывод:** были проанализированы существующие методы и технологии обеспечения стойкости к стрессу данных с сохранением их консистентности, а также были рассмотрены законодательные требования относительно систем обработки данных. Это позволит создать конкретную модель системы, которую можно использовать на практике в организации.

### **1.3. Основные требования к созданию кластера баз данных и разбор архитектуры**

Основные требования к созданию кластера баз данных включают в себя множество аспектов, которые обеспечивают эффективную работу системы, ее стойкость к стрессу и отказам, безопасность и соответствие законодательным требованиям. Некоторые из ключевых требований к созданию кластера баз данных описаны ниже.

1. Стойкость к стрессу и отказам. Кластер баз данных должен быть стойким к стрессу и отказам, чтобы обеспечивать бесперебойную работу системы и сохранность данных в случае возникновения проблем. Для этого необходимо предусмотреть механизмы репликации, балансировки нагрузки, резервного копирования и восстановления данных.

2. Безопасность данных. Кластер баз данных должен быть защищен от несанкционированного доступа, изменения и уничтожения данных. Для этого необходимо предусмотреть механизмы аутентификации и авторизации пользователей, шифрование данных, контроль доступа и мониторинг безопасности системы.

3. Высокая производительность. Кластер баз данных должен обеспечивать высокую производительность при обработке больших объемов данных и запросов. Для этого необходимо оптимизировать запросы, предусмотреть механизмы кэширования данных, масштабируемость системы и оптимизацию хранения данных.

4. Соответствие законодательным требованиям. Кластер баз данных должен соответствовать законодательным требованиям, которые могут применяться к хранению и обработке данных. Для этого необходимо

предусмотреть защиту персональных данных, соответствие нормативным требованиям отраслей и областей деятельности, где используется кластер баз данных.

5. Гибкость и масштабируемость. Кластер баз данных должен быть гибким и масштабируемым, чтобы соответствовать изменяющимся требованиям бизнеса и растущим объемам данных. Для этого необходимо предусмотреть возможность добавления и удаления узлов кластера, масштабирование системы и возможность интеграции с другими системами.

6. Простота управления и мониторинга. Кластер баз данных должен быть прост в управлении и мониторинге, чтобы обеспечить высокую доступность и качество обслуживания системы. Для этого необходимо предусмотреть механизмы мониторинга производительности и надежности системы, удобный интерфейс управления и механизмы управления конфигурацией системы.

7. Поддержка открытых стандартов и совместимость. Кластер баз данных должен поддерживать открытые стандарты и быть совместимым с другими системами, что обеспечит удобство интеграции и обмена данными между различными системами.

8. Обеспечение поддержки и обслуживания. Кластер баз данных должен быть обеспечен поддержкой и обслуживанием, чтобы обеспечить эффективность и бесперебойную работу системы. Для этого необходимо предусмотреть механизмы поддержки и обновления системы, круглосуточную техническую поддержку и гарантии качества обслуживания.

Основные требования к созданию кластера баз данных должны определяться в соответствии с бизнес-требованиями организации и учитывать особенности конкретной системы. Также важно учитывать законодательные требования, отраслевые стандарты и лучшие практики в области проектирования и эксплуатации кластеров баз данных. При разработке требований к созданию кластера баз данных необходимо тесно сотрудничать

с командой разработчиков и экспертами по базам данных, чтобы обеспечить эффективность и качество реализации системы.

**Основными компонентами кластера баз данных** включают в себя управляющей узел, реплики данных и механизмы коммуникации между ними (абстрактная схема приведена на рис. 4):

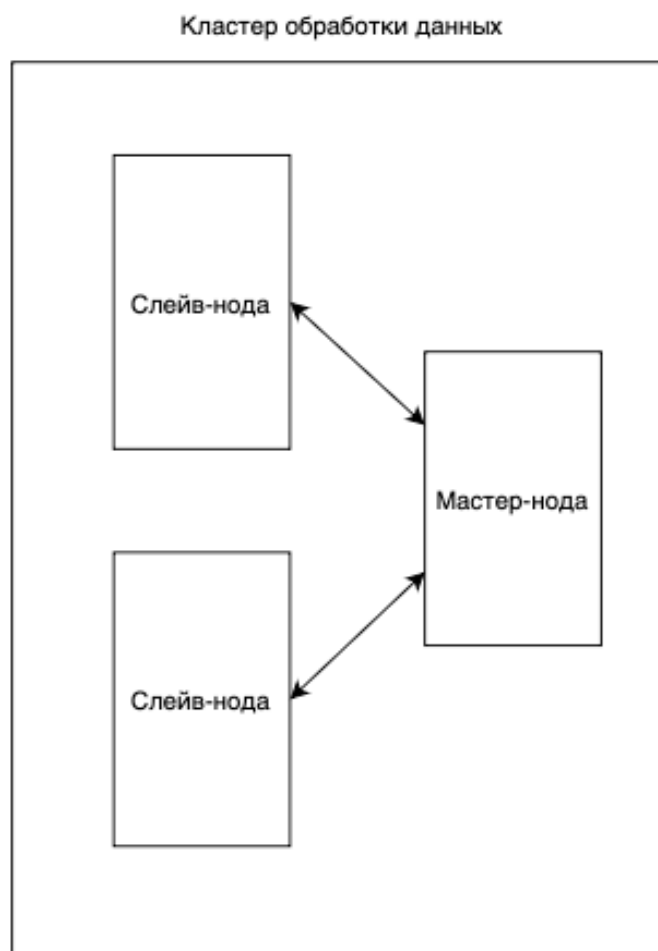


Рисунок 4. Абстрактное представление взаимосвязи нод кластера

1. **Управляющий узел (Master):** Управляющий узел является центральным элементом кластера баз данных. Он отвечает за координацию работы всех узлов, выполнение транзакций и обработку запросов от клиентов. Управляющий узел принимает записи и обновления данных от клиентов, а затем распространяет их на реплики данных в кластере. Он также обеспечивает контроль за целостностью данных, согласовывает конфликты и принимает решения относительно репликации и распределения данных в кластере.

2. Реплики данных (Slaves): Реплики данных представляют собой копии данных, которые хранятся на различных узлах кластера. Они служат для повышения отказоустойчивости и обеспечения доступности данных. Реплики актуализируются путем репликации изменений, внесенных в управляющий узел, и применяют эти изменения к своим локальным копиям данных. Каждая реплика может выполнять чтение и обработку запросов от клиентов, что позволяет балансировать нагрузку и повышать производительность системы.

3. Механизмы коммуникации: Для обеспечения взаимодействия между управляющим узлом и репликами данных используются различные механизмы коммуникации. Это может быть репликация журналов транзакций, сетевые протоколы или специальные инструменты коммуникации, разработанные для конкретного кластера баз данных. Механизмы коммуникации позволяют передавать изменения данных с управляющего узла на реплики, синхронизировать состояние кластера и обеспечивать согласованность данных.

**Вывод:** посредством литературного анализа были изучены существующие требования к актуальным систем обработки потоков данных с обеспечением высокой доступности, безопасности, масштабируемости и сохранения консистентности. Далее это будет использовано для определение требований к разработке практической модели.

**Вывод по главе:** определены основные требования к созданию кластеров баз данных, которые включают в себя стойкость к стрессу и отказам, безопасность данных, высокую производительность, соответствие законодательным требованиям, гибкость и масштабируемость, простоту управления и мониторинга, поддержку открытых стандартов и обеспечение поддержки и обслуживания.

Существует множество методов и технологий, которые могут быть использованы для обеспечения доступности, стойкости к стрессу и отказам, безопасности и эффективности работы кластеров баз данных. Среди них можно выделить механизмы репликации, шаринга данных, балансировки

нагрузки, а также различные инструменты и методы мониторинга и управления кластерами баз данных.

При разработке кластеров баз данных необходимо учитывать особенности конкретной системы, бизнес-требования организации, а также законодательные требования и отраслевые стандарты. Только тесное сотрудничество команды разработчиков и экспертов по базам данных позволит обеспечить эффективность и качество реализации системы.

Таким образом, создание и эксплуатация стрессо- и отказоустойчивых кластеров баз хранения данных на базе отечественного ПО и решений с открытым исходным кодом является важной задачей, требующей комплексного подхода и использования современных технологий и методов.

## ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ ОБРАБОТКИ ДАННЫХ И ЕЕ ТЕСТИРОВАНИЕ

### 2.1. Выбор ПО и решений для создания кластера баз данных, теоретический разбор работы утилит

В рамках данной работы выборы технологий и методов обусловлены их актуальностью, достижимости требований, объеме используемых ресурсов и сложности эксплуатации.

Изначально требуется обозначить точки входа и выхода (*entry point* и *end point*) в систему обработки данных, а также требования. В первой главе были описаны общие требования для подобных систем. Они и были взяты в качестве основных правил построения.

Для понятности используемых интерфейсов была использована абстрактную схему распространенной актуальной системы агрегации данных (рис. 4):

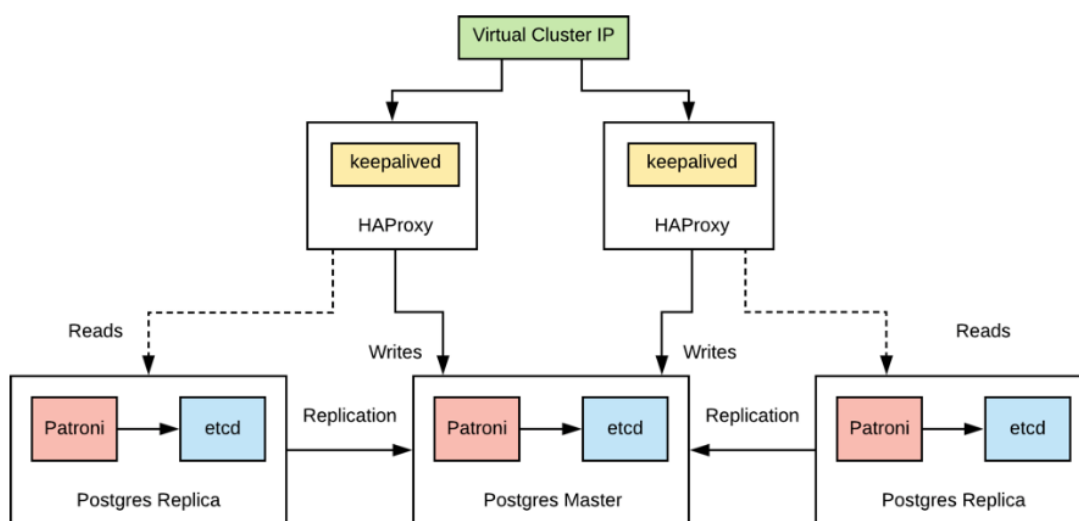


Рисунок 5. Абстрактное представление архитектуры кластера БД

- **Entry-point.** Точка входа в систему от конечного пользователя. В рамках данной работы будет использоваться интерфейс системы управления базы данных.
- **СУБД.** В роли реляционной базы данных была выбрана СУБД *PostgreSQL*, ввиду его высокой надежности, стабильности, актуальности, популярности и учета выставленных требований.



- **ОС.** В качестве операционной системы была выбрана Astra Linux (дистрибутив Смоленск. Astra Linux Smolensk - это российская операционная система на базе Linux, разработанная специально для государственных и корпоративных систем безопасности и защиты информации. Она была создана с учетом требований законодательства России по защите государственной тайны и является одной из наиболее популярных операционных систем в сфере государственной безопасности.

- **Кластеризатор.** В качестве инструмента управления кластерами PostgreSQL был выбран Python-асинхронный скрипт *Patroni*. Patroni - это программное обеспечение с открытым исходным кодом для управления высокодоступными кластерами PostgreSQL. Он обеспечивает автоматическое обнаружение отказов и переключение между узлами кластера, а также поддерживает функции балансировки нагрузки и репликации данных. Patroni использует протокол репликации PostgreSQL (streaming replication) для синхронизации данных между узлами кластера. Каждый узел кластера может быть настроен как мастер (primary) или реплика (replica), и Patroni автоматически переключает мастера в случае отказа текущего мастера. При этом, Patroni использует etcd для хранения конфигурационных данных кластера и координации между узлами.

- **Балансировщик нагрузки.** Keepalived — это программное обеспечение с открытым исходным кодом для обеспечения высокой доступности сетевых сервисов и приложений в распределенных системах. Keepalived работает на уровне сетевого протокола и обеспечивает механизмы отказоустойчивости и балансировки нагрузки между узлами сети. Keepalived используется в различных сценариях, включая кластеры баз данных, веб-сервера, почтовые серверы, а также другие сервисы и приложения. Он работает в паре с балансировщиком нагрузки и использует протоколы VRRP и OSPF для обеспечения высокой доступности сервисов и приложений. Основным принципом работы Keepalived заключается в том, что он проверяет состояние сетевых интерфейсов и сервисов на узлах кластера и автоматически

переключает трафик на другой узел в случае отказа текущего узла. Это обеспечивает непрерывную работу сервисов и приложений даже в случае отказа одного из узлов. Keeralived также поддерживает балансировку нагрузки между узлами кластера, что позволяет равномерно распределять трафик между узлами и предотвращать перегрузки. Он использует различные алгоритмы балансировки нагрузки, такие как round-robin, least-connections и IP-hash.

- **Безопасность.** В рамках построения данной системы вопрос безопасности не поднимался, т.к. все запускалось в локальном контуре. Так что начальным уровнем безопасности назначен белый список адресов IP-адресов в настройках штатного брандмауэра.

- **Распределенное хранилище.** Для обмена данными между узлами кластеров была выбрана система ETCD. ETCD - это распределенное хранилище данных с открытым исходным кодом, которое используется для хранения конфигурационных данных, координации и управления состоянием в распределенных системах. Он используется в кластерных приложениях, таких как Kubernetes, для хранения данных конфигурации, обмена данными между узлами кластера и обеспечения координации между сервисами. Основной принцип работы ETCD заключается в том, что он хранит данные в виде ключ-значение и обеспечивает доступ к этим данным через API. ETCD использует протокол Raft для достижения консенсуса между узлами кластера и обеспечения надежности данных. ETCD позволяет создавать, изменять и удалять данные в режиме реального времени и обеспечивает механизмы уведомления (watch) для оповещения о изменениях в данных. Он также поддерживает различные методы аутентификации и шифрования для обеспечения безопасности данных.

Все выбранные технологии имеют открытый исходный код (за исключением Astra Linux – это российский дистрибутив) и их использование сильно распространены в мировом сообществе, что позволяет находить

ответы на конкретные вопросы о работе или происходящих ошибках в процессе работы.

Кластер баз данных на основе PostgreSQL, Patroni, etcd и Keepalived состоит из четырех серверов баз данных, каждый из которых имеет свой экземпляр PostgreSQL. Кластер работает на основе принципа мастер-нода и слейв-нода.

В этом кластере мастер-нода – это сервер, который обрабатывает все транзакции базы данных, а слейв-нода – это серверы, которые используются для резервного копирования данных. При отказе мастер-ноды, одна из слейв-нод будет переключена в режим мастер-ноды и начнет обрабатывать транзакции базы данных.

Принцип работы кластера баз данных на основе PostgreSQL, Patroni, etcd и Keepalived включает в себя следующие шаги:

1. Установка и настройка PostgreSQL: на каждой машине в кластере устанавливается PostgreSQL и настраивается для работы в кластере.

2. Установка и настройка Patroni: Patroni устанавливается на каждой машине в кластере и настраивается для обнаружения отказов мастер-ноды и автоматического переключения на работающий узел кластера. Patroni также отвечает за управление репликацией данных между узлами кластера.

3. Установка и настройка ETCD: ETCD используется для хранения конфигурации кластера и состояния узлов. Он устанавливается на каждой машине в кластере и настраивается для обмена информацией о состоянии узлов.

4. Установка и настройка Keepalived: Keepalived используется для балансировки нагрузки между узлами кластера. Он устанавливается на отдельной машине, которая не является частью кластера, и настраивается для мониторинга состояния узлов и перенаправления запросов к работающей мастер-ноде.

5. Настройка балансировки нагрузки: Keepalived настраивается для балансировки нагрузки между узлами кластера. Он использует алгоритм round-robin для равномерного распределения нагрузки между узлами.

6. Настройка мониторинга и управления: ETCD, Patroni и Keepalived используются для управления конфигурацией кластера, мониторинга состояния узлов и обнаружения отказов в режиме поточной записи.

7. Репликация данных: Данные хранятся на каждом узле кластера в отдельных экземплярах PostgreSQL. Репликация данных между узлами происходит с помощью механизма репликации, который обеспечивает высокую доступность и отказоустойчивость данных.

8. Шардинг данных: Шардинг происходит на уровне приложения и может быть реализован с помощью различных технологий. Например, можно использовать репликацию данных на основе логического декомпозиции данных или физического разделения данных на разные узлы кластера.

9. Обеспечение безопасности: Кластер может быть защищен с помощью механизмов безопасности, таких как аутентификация и авторизация, шифрование данных, защита от атак на основе сетевого протокола и т.д.

Кластер баз данных на основе PostgreSQL, Patroni, etcd и Keepalived обеспечивает высокую доступность, отказоустойчивость и масштабируемость для обработки большого объема данных. Эта система может использоваться в различных сферах, таких как финансы, здравоохранение, телекоммуникации и т.д., где высокая доступность и отказоустойчивость данных являются критически важными факторами.

В итоге получилась данная архитектура проекта:

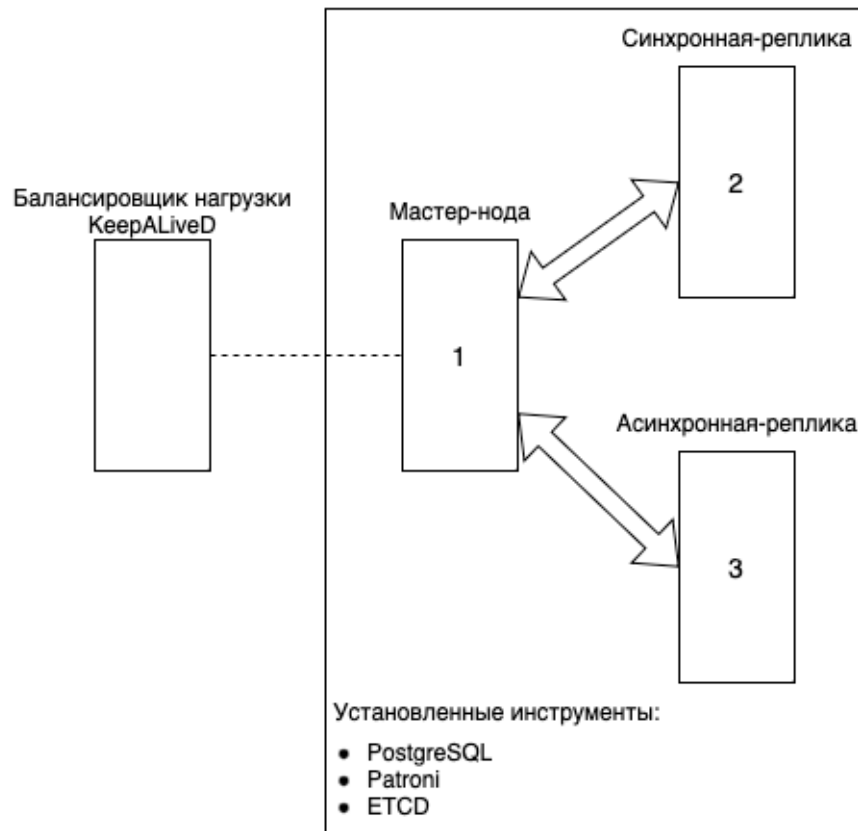


Рисунок 6. Архитектура проекта в схематике

Точкой входа в данную модель является сервер балансировщика нагрузки KeepALiveD, который, используя штатный интерфейс PostgreSQL подключается к кластеру и выполняет все запросы на агрегацию данных.

Машина внутреннего контура под номер 1 является мастер-нодой. При выполнении записи/изменении/удалении данных на которой изменения проецируются на репликах. Машина под номером 2 – синхронная реплика. Все данные, которые появляются в первой машине в параллели, отправляются и на вторую. Машина под номером 3 – асинхронная реплика, изменения на которой появляются во время простоя основных машин.

В случае, если мастер-нода выходит из строя по любым рода ошибкам и Patroni не может получить ответ от нее, то роль мастера переходит к синхронной реплике, что сохраняет актуальность данных в системе без ущерба по времени отклика. Если вторая машина выходит из строя, то асинхронная реплика становится синхронной и подтягивает актуальные данные. Если третья

машина теряет связь с общей системой, то общая целостность модели ничего не теряет, требуется лишь исправить поломку:

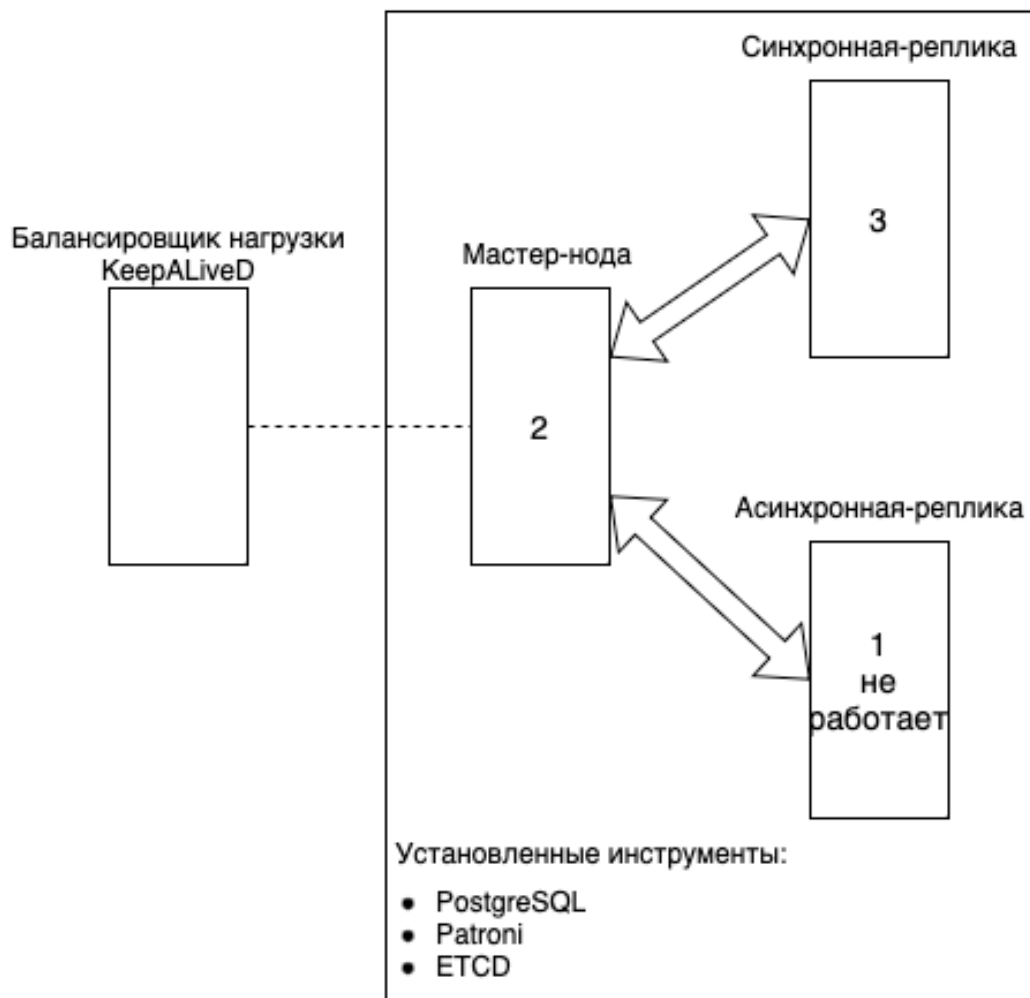


Рисунок 7. Смена ролей в случае выхода из строя мастер-ноды

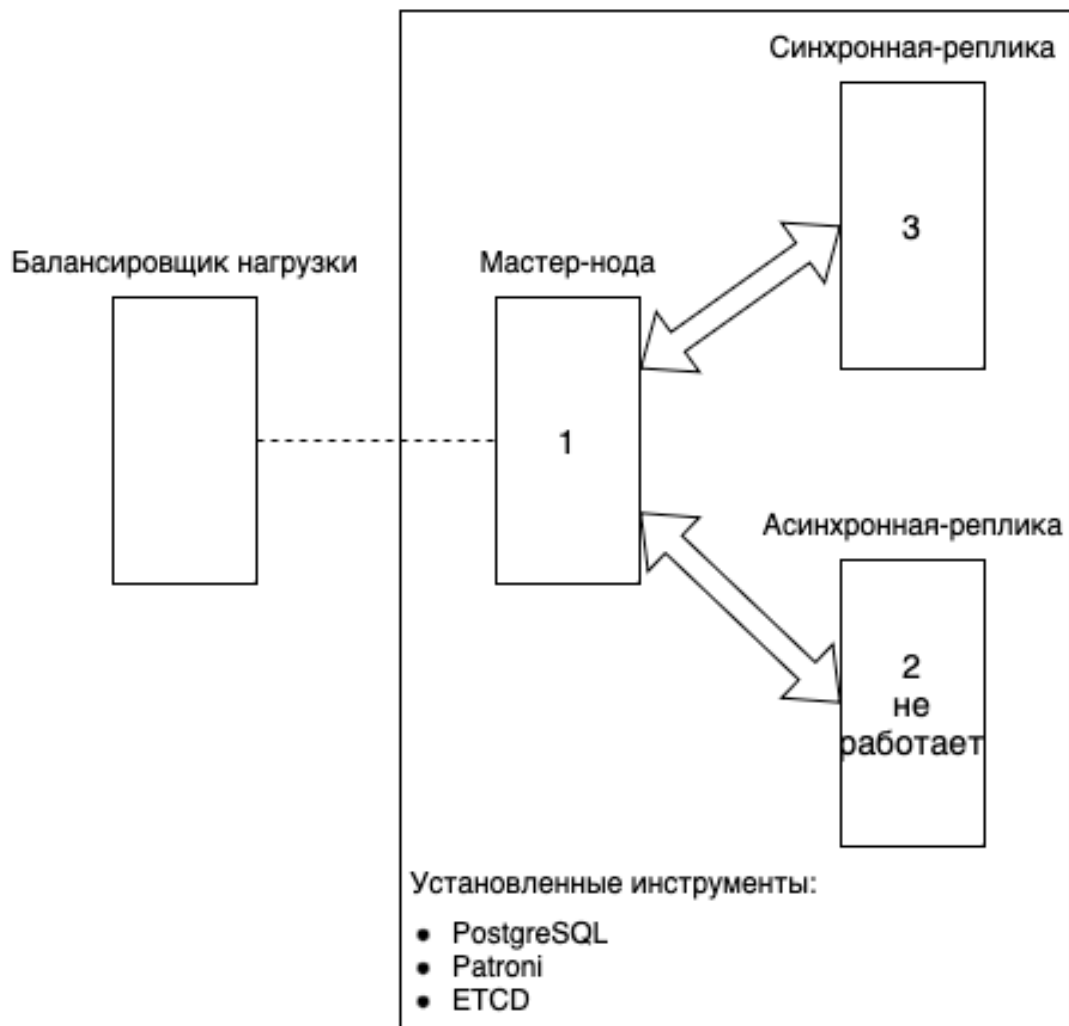


Рисунок 8. Смена ролей в случае выхода из строя слейв-нод

Keepalived использует протокол Virtual Router Redundancy Protocol (VRRP), который позволяет объединить несколько серверов в группу и представить их как единый виртуальный сервер. В рамках этой группы один сервер назначается главным (master), а остальные - резервными (backup). Главный сервер выполняет функции балансировщика нагрузки и обрабатывает запросы клиентов, а резервные серверы следят за работоспособностью главного сервера и готовы взять на себя его функции в случае его отказа. В целом, Keepalived позволяет обеспечить высокую доступность и балансировку нагрузки для кластера серверов и может быть полезен для обеспечения стабильности работы приложений и сервисов.

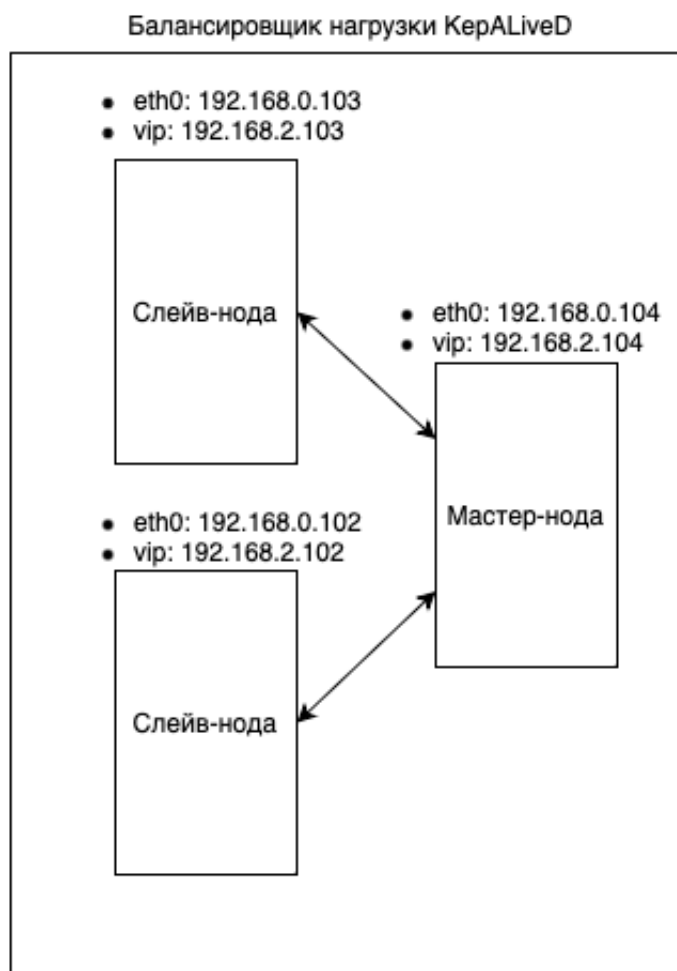


Рисунок 9. Схема работы балансировщика нагрузки

У каждой из трех машин есть свой физический адрес в локальной сети и есть виртуальный, который предоставляет KeepALiveD на отдельном виртуальном интерфейсе. KeepALiveD работает на протоколе VRRP (Virtual Router Redundancy Protocol). Общий алгоритм работы определяется двумя этапами:

1. Master сервер с заданным интервалом отправляет VRRP пакеты на зарезервированный адрес multicast (многоадресной) рассылки 224.0.0.18, а все slave сервера слушают этот адрес. Multicast рассылка — это когда отправитель один, а получателей может быть много.

2. Если Slave сервер не получает пакеты, он начинает процедуру выбора Master и если он переходит в состояние Master по приоритету, то активирует VIP и отправляет gratuitous ARP. Gratuitous ARP — это особый вид ARP ответа, который обновляет MAC таблицу на подключенных коммутаторах, чтобы



проинформировать о смене владельца виртуального IP адреса и mac адреса для перенаправления трафика и смене виртуальных адресов. При возвращении в активный режим бывшего мастера получает роль слейва.

Безопасность в данном случае требуется обеспечить в трех местах:

- Подключение к серверу балансировщика нагрузки. Вариантов для активной и пассивной защиты очень много. Например: брандмауэры, VPN, IPSec, аутентификация и авторизация, периодическое мониторингирование и аудита.
- Подключение сервера балансировщика и кластера. Для защиты канала связи рекомендуется использовать активные и пассивные варианты защиты для единократной настройки и уменьшения шансов на проникновение злоумышленником. Для этого подойдет протоколы SSL/TLS, VPN, IPsec, использование токенов аутентификации и авторизации автоматизированного режима с тщательным протоколированием.
- Защитный контур кластера обработки данных. В данном случае требуется исключить возможность получения физического доступа системе обработки данных злоумышленниками и реализовать белый список (разрешенных) IP-адресов для подключения.

Ввиду решений с открытым исходным кодом и простоты общей архитектуры системы возможны любые модификации системы с интеграцией новых слоев защиты, агрегации или шифрования данных, реализации внешних программных интерфейсов взаимодействия (API).

Таким образом система является масштабируемой системой с максимально простой возможностью интеграцией новых модулей любого уровня. Развертывание данной системы можно заключить в плейбуки менеджера задач Ansible при должной сноровке и умении. Безопасность данной системы заключается в трех контрольных точках и может быть реализована на любом из этапов разработки. Так же в системе выделено место для балансировки нагрузки в поточном режиме, что позволяет

**Вывод:** были обоснованы выборы тех или иных технологий, на базе которых будет построена практическая модель. Данная архитектура удовлетворяет всем поставленным требованиям и позволяет масштабировать модель системы в любом исполнении.

## **2.2. Реализация кластера баз данных**

Установка всех компонентов происходит с помощью пакетного менеджера Astra Linux – **APT** (advanced packet tool). В виду того, что Astra Linux основан на Debian Buster и разработанный Astra пакетный менеджер **APM** (Astra Linux Package Manager) не использовался в процессе написания данной работы.

Для полноценной реализации системы агрегации данных потребуется минимум 4 машин с операционной системой Astra Linux:

- Три машины для самого кластера с установленными PostgreSQL, Patroni, etcd (etcd не рекомендуется устанавливать на тех же машинах, где находится Patroni, но в рамках теста системы это допустимо) и KeepALiveD (Patroni работает лучше всего на кластерах с кратным 3 количеством машин, так как один – мастер, второй – синхронная репликация, третий – асинхронная репликация);
- Одна машина для установления прокси-сервера (мастер-роли KeepALiveD) и решения проблемы плавающей точки входа.

В рамках работы все машины являются виртуальными благодаря специализированному ПО для виртуализации второго типа (изолированный запуск ОС на одном устройстве на основной ОС) Oracle VM VirtualBox.

**Установка дистрибутива Astra Linux Smolensk** происходит штатным образом с помощью официального дистрибутива ОС:

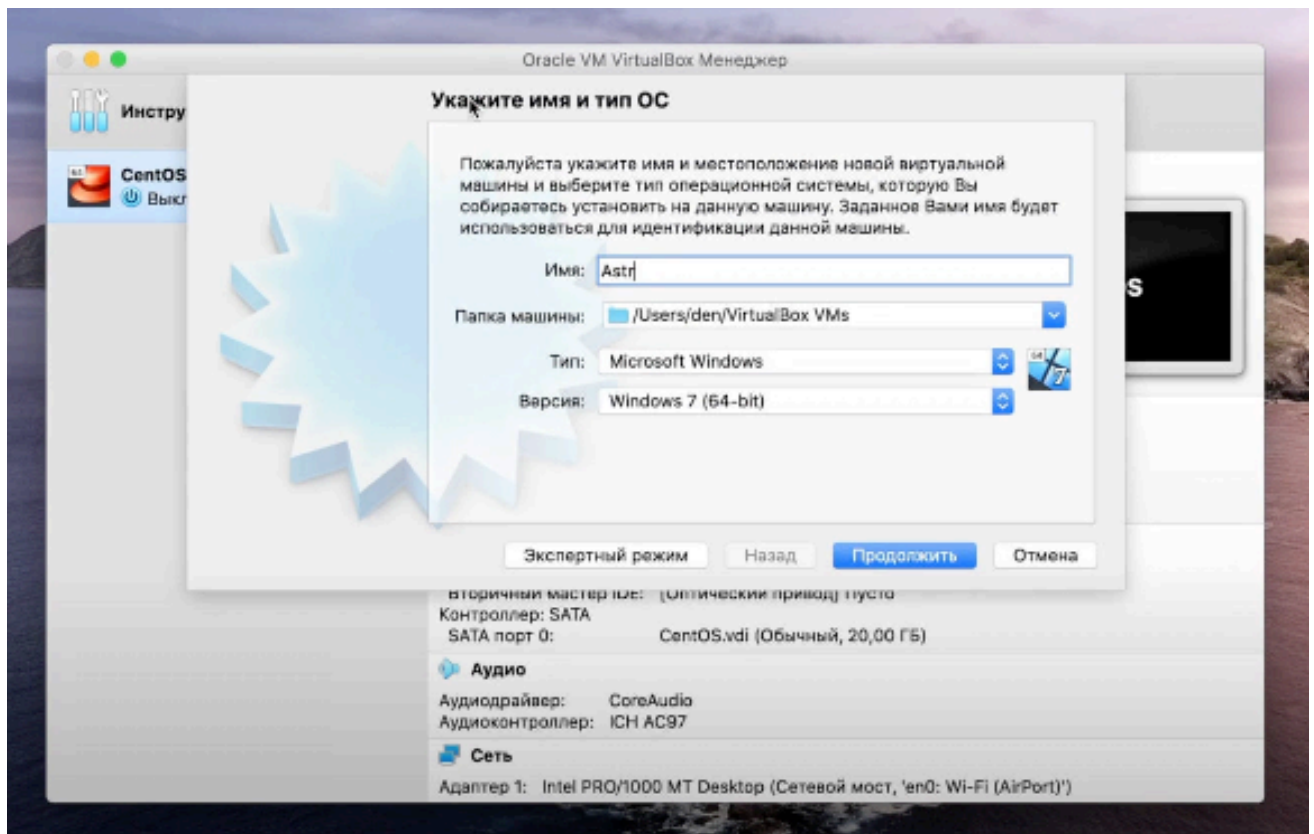


Рисунок 10. Создание ВМ и указание имени

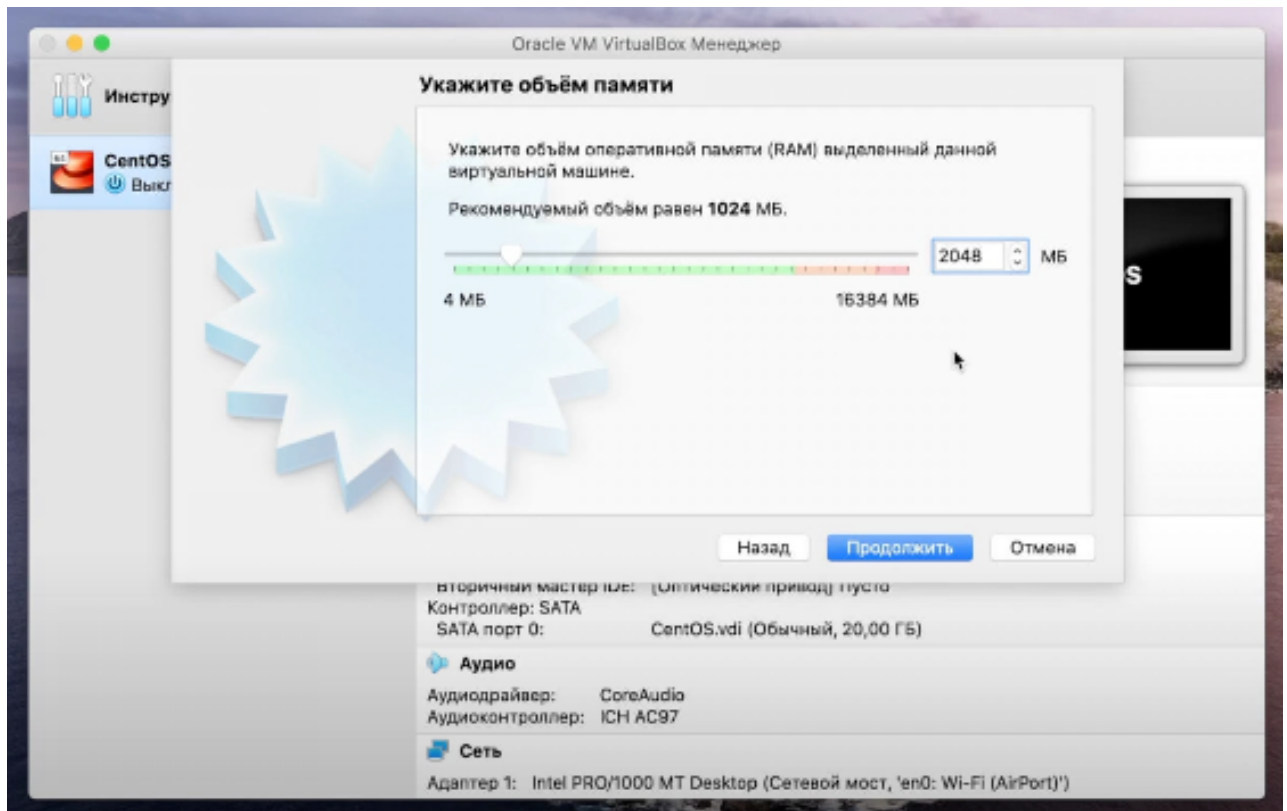


Рисунок 11. Выбор объема ОП (рекомендуемый объем - 2ГБ)

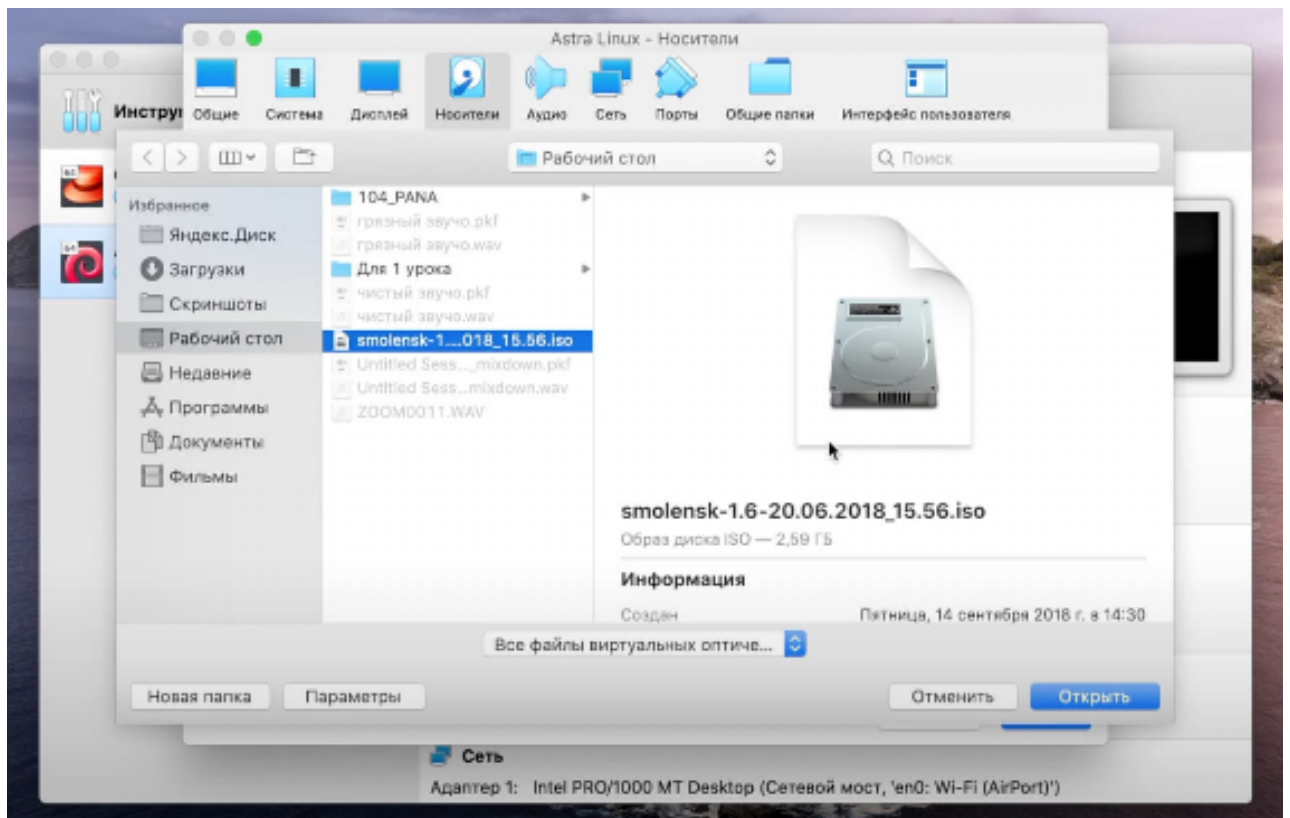


Рисунок 12. Выбор дистрибутива для установки

Далее ВМ запускается и выбираются все рекомендованные настройки, а имена машин выбраны в соответствии с их ролью в архитектуре (Postgres1, Postgres2, Postgres3, etcdHub, proxy).

### **Установка, настройка инструментов и требуемого ПО:**

- **ETCD.** Для установки etcd был использован пакетный менеджер apt:

apt-get update

apt-get install etcd

Настройка происходит в файле /etc/default/etcd:

### [member]

```
ETCD_NAME=datanode1 # hostname  
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
```

**ALL IP ADDRESSES SHOULD BE VALID. LISTER PEER, CLIENT etc SHOULD BE SET TO IP ADDRESS OF HOST**

```
ETCD_LISTEN_PEER_URLS="http://192.168.0.143:2380" # адрес машины  
ETCD_LISTEN_CLIENT_URLS="http://192.168.0.143:2379,http://127.0.0.1:2379" # адрес машины
```

### [cluster]

```
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://192.168.0.143:2380" # адрес машины  
ETCD_INITIAL_CLUSTER="datanode1=http://192.168.0.143:2380,datanode2=http://  
192.168.0.144:2380,datanode3=http://192.168.0.145:2380" # адреса всех машин в кластере etcd  
ETCD_INITIAL_CLUSTER_STATE="new"  
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster-1"  
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.0.143:2379" # адрес машины
```

Рисунок 13. Пример настройки файла

Затем требуется запустить сервис-демон etcd:

```
systemctl start etcd
```

```
systemctl restart etcd
```

- **PostgreSQL и Patroni**

Для установки и настройки данного ПО был использован менеджер задач Ansible. Ansible - система автоматизации управления конфигурацией и развертывания приложений. Она позволяет описывать инфраструктуру в виде кода и автоматизировать процессы установки, настройки и управления системами, используя простой и понятный язык описания инфраструктуры. Ansible основан на модели клиент-сервер, где клиентом является управляющий узел (узел, на котором запускаются команды Ansible), а серверами - узлы, которые необходимо управлять. Управляющий узел может управлять любым количеством серверов, используя SSH-подключение. Однако, в данном случае плейбуки будут выполнены на каждой из машин без централизованного удаленного запуска для упрощения. Предварительная установка и частичная настройка была реализована благодаря bash-скрипту:

```
#!/bin/bash
apt-get install gnupg -y
echo "deb http://apt.postgresql.org/pub/repos/apt/ buster-pgdg main" >> /etc/apt/sources.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt-get update
apt-get install postgresql-9.6 python3-pip python3-dev libpq-dev -y
systemctl stop postgresql
pip3 install --upgrade pip
pip install psycpg2
pip install patroni[etcd]
echo "\
[Unit]
Description=Runners to orchestrate a high-availability PostgreSQL
After=syslog.target network.target

[Service]
Type=simple

User=postgres
Group=postgres

ExecStart=/usr/local/bin/patroni /etc/patroni.yml

KillMode=process

TimeoutSec=30

Restart=no

[Install]
WantedBy=multi-user.target" > /etc/systemd/system/patroni.service
mkdir -p /data/patroni
chown postgres:postgres /data/patroni
chmod 700 /data/patroni
touch /etc/patroni.yml
```

Рисунок 14. Код bash-скрипта

Затем в созданный файл `etc/patroni.yml` был добавлен скрипт плейбука (IP-адреса были заменены на собственные для каждой машины):

```

scope: postgresql # должно быть одинаковым на всех нодах
namespace: /cluster/ # должно быть одинаковым на всех нодах
name: postgres1 # должно быть разным на всех нодах

restapi:
  listen: 192.168.0.143:8008 # адрес той ноды, в которой находится этот файл
  connect_address: 192.168.0.143:8008 # адрес той ноды, в которой находится этот файл

etcd:
  hosts: 192.168.0.141:2379,192.168.0.142:2379,192.168.0.143:2379

# this section (bootstrap) will be written into Etcd: /<namespace>/<scope>/config after initializing new cluster
# and all other cluster members will use it as a 'global configuration'
bootstrap:
  dcs:
    ttl: 100
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
  postgresql:
    use_pg_rewind: true
    use_slots: true
    parameters:
      wal_level: replica
      hot_standby: "on"
      wal_keep_segments: 5120
      max_wal_senders: 5
      max_replication_slots: 5
      checkpoint_timeout: 30

  initdb:
    - encoding: UTF8
    - data-checksums
    - locale: en_US.UTF8
    # init pg_hba.conf должен содержать адреса ВСЕХ машин, используемых в кластере
  pg_hba:
    - host replication postgres ::1/128 md5
    - host replication postgres 127.0.0.1/8 md5
    - host replication postgres 192.168.0.143/24 md5
    - host replication postgres 192.168.0.144/24 md5
    - host replication postgres 192.168.0.145/24 md5
    - host all all 0.0.0.0/0 md5

  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb

  postgresql:
    listen: 0.0.0.0:5432 # адрес той ноды, в которой находится этот файл
    connect_address: 0.0.0.0:5432 # адрес той ноды, в которой находится этот файл
    data_dir: /data/patroni
    bin_dir: /usr/lib/postgresql/9.6/bin
    pgpass: /tmp/pgpass
    authentication:
      replication:
        username: postgres
        password: postgres
      superuser:
        username: postgres
        password: postgres
    create_replica_methods:
      basebackup:
        checkpoint: 'fast'
    parameters:
      unix_socket_directories: ''

  tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false

```

Рисунок 15. Код плейбука

Адрес 0.0.0.0 означает всевозможные адреса в диапазоне маски локальной сети. После выполнения предварительной конфигурации требуется запустить сервисы:

```
systemctl start patroni  
systemctl start postgresql
```

**KeepALiveD** был установлен и настроен стандартным вопросом в виду простоты. Установка с помощью пакетного менеджера:

```
apt-get install keepalived
```

После установки потребовалась конфигурация в файле `etc/keepalived/keepalived.conf`:

```
! Configuration File for keepalived  
global_defs {  
    router_id uMASTER  
}  
  
vrrp_instance VI_1 {  
    state MASTER  
    interface agge  
    virtual_router_id 230  
    priority 101  
    advert_int 1  
    authentication {  
        auth_type PASS  
        auth_pass SecPassWord  
    }  
  
    virtual_ipaddress {  
        192.168.0.102/32 dev agge label agge:0  
        192.168.0.103/32 dev agge label agge:0  
        192.168.0.104/32 dev agge label agge:0  
    }  
}
```

Рисунок 16. Конфигурация на мастер-ноде



```

! Configuration File for keepalived
global_defs {
    router_id uBACKUP
}

vrrp_instance VI_1 {
    state BACKUP
    interface agge
    virtual_router_id 230
    priority 100          # PAY ATTENTION ON PRIORITY!!
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass SecPassWord      #change pass if need
    }

    virtual_ipaddress {
        192.168.0.102/32 dev agge label agge:0
        192.168.0.103/32 dev agge label agge:0
        192.168.0.104/32 dev agge label agge:0
    }
}

```

Рисунок 17. Конфигурация на слейв-нодах

Обязательно указать корректный интерфейс и конечный ip-адрес для корректной работы. После сохранения конфигурационных файлов демон был включен и проверен:

```
systemctl start keepalived
```

Проверка сетевых интерфейсов с помощью утилиты ifconfig:

```

# ifconfig
en0: flags=5187<UP,BROADCAST,RUNNING,MASTER,MULTICAST> mtu 1500
    inet 192.168.0.104 netmask 255.255.255.255 broadcast 0.0.0.0
    ether 192:168:0:47:e4:3e txqueuelen 1000 (Ethernet)

```

Рисунок 18. Результат работы утилиты ifconfig

Теперь есть два интерфейса – ethernet и inet.

**Вывод:** были инициализированы 4 виртуальных машин в одной локальной сети, установлено все необходимое ПО, включая требуемые зависимости. Далее эта система будет протестирована после тщательной настройки.

### 2.3. Проверка работоспособности и надежности кластера баз данных

Проверка работоспособности кластера заключается в тестировании штатных механизмов и сервисов таких как Patronictl. Так как все ПО установлено и сконфигурировано, сервисы запущены и подключены к автозапуску, то был произведен тест работы на передачу данных между нодами кластера.

Patronictl - это утилита командной строки, которая используется для управления кластером баз данных PostgreSQL, запущенным с помощью Patroni. Она предоставляет простой и удобный интерфейс для управления кластером и выполнения различных операций, таких как создание, удаление, настройка, перезапуск и масштабирование кластера. Patronictl поддерживает различные команды, которые могут быть использованы для управления кластером, включая команды для настройки конфигурации кластера, мониторинга состояния узлов, выполнения обслуживания баз данных, управления репликацией данных и другие функции. Она также поддерживает автоматическую проверку состояния кластера и автоматическую установку необходимых параметров конфигурации.

В данном случае потребовалась команда `patronictl list`:

```
test@Test01:~$ patronictl list
+ Cluster: TestScopePatroni (7088376499457688247) ----+-----+
| Member | Host           | Role           | State   | TL | Lag in MB |
+-----+-----+-----+-----+---+-----+
| Test01  | 10.31.138.46   | Replica        | running | 11 |          0 |
| Test02  | 10.31.138.60   | Sync Standby   | running | 11 |          0 |
| Test03  | 10.31.138.55   | Leader         | running | 11 |          |
+-----+-----+-----+-----+---+-----+
test@Test01:~$
```

Рисунок 19. Результат работы утилиты `patronictl`

На рисунке 17 видны все три ноды кластера с их ролями. В качестве IP адреса отображен виртуальный IP адрес в общей архитектуре, а не физический. Member – имя ноды для консольного взаимодействия извне, state – актуальный статус ноды. TL – timeline (временная линия). В PostgreSQL Timeline – специальный механизм, который используется для отслеживания изменений в базе данных, связанных с репликацией данных. Когда происходит изменения на основе узла кластера, они реплицируются на другие узлы в кластере, и Timeline используется для отслеживания версий этих изменений. Другими словами, один из механизмов отслеживания актуальности данных. Значение TL должно быть одинаково у лидера и синхронной реплики, а у асинхронной может незначительно отличаться, однако в ближайшее время придти к тому же значению. Lag in MB отображает на сколько реплики отличаются от актуальной версии в мегабайтах информации.

После введения изменений в БД значение TL изменилось на один пункт на всех нодах, что отображает о корректной работе установленного ПО и сервисов и соблюдает поставленную задачу:

```

test@Test02: ~
Every 2,0s: patronictl list

+ Cluster: TestScopePatroni (7088376499457688247) ----+-----+
| Member | Host           | Role           | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| Test01 | 10.31.138.46   | Sync Standby   | running | 12 | 0          |
| Test02 | 10.31.138.60   | Replica        | running | 12 | 0          |
| Test03 | 10.31.138.55   | Leader         | running | 12 |           |
+-----+-----+-----+-----+-----+-----+
  
```

Рисунок 20. Изменение TL в patronictl

Следующим этапом тестирования является аварийное выключение одной из нод для проверки механизма смены мастера в иерархии кластера. Для этого так же была использована утилита patronictl с командой switchover:

+Cluster: TestScopePatroni (7088376499457688247) -----+						
Member	Host	Role	State	TL	Lag in MB	
Test01	10.31.138.46	Sync Standby	running	12	0	
Test02	10.31.138.60	Replica	running	12	0	
Test03	10.31.138.55	Leader	running	12	0	

Рисунок 21. До ручной аварийной смены ролей

+Cluster: TestScopePatroni (7088376499457688247) -----+						
Member	Host	Role	State	TL	Lag in MB	
Test01	10.31.138.46	Replica	running	13	0	
Test02	10.31.138.60	Leader	running	13	0	
Test03	10.31.138.55	Sync Standby	running	13	0	

Рисунок 22. После ручной аварийной смены ролей

Из рисунков 19 и 20 видна корректная отработка смена ролей и изменение TL в общей архитектуре patroni. Это говорит о корректности настройки, правильной работе установленного ПО и удовлетворению требований. Фактическое физическое отключение одной из нод или отключение от сети не требуется ввиду принципа работы patronictl failover – данный алгоритм временно выключает доступ к локальной сети у выбранной ноды и затем работает штатный режим смены роли. Таким образом это доказывает корректную отработку в любых аварийных происшествиях (за исключением одновременного отключения всех трех машин).

Для проверки корректности работы KeepALiveD достаточно отправлять запросы на плавающий IP-адрес с помощью утилиты ping, а затем остановить демон keepalived на мастер-ноде с помощью команды:

```
systemctl stop keepalived
```

```
ping 192.168.2.102

64 bytes from 192.168.2.102: icmp_seq=59 ttl=53 time=260.582 ms
64 bytes from 192.168.2.102: icmp_seq=60 ttl=53 time=314.723 ms
64 bytes from 192.168.2.102: icmp_seq=62 ttl=53 time=498.176 ms
64 bytes from 192.168.2.102: icmp_seq=63 ttl=53 time=167.139 ms
Request timeout for icmp_seq 64
64 bytes from 192.168.2.102: icmp_seq=65 ttl=53 time=597.598 ms
64 bytes from 192.168.2.102: icmp_seq=66 ttl=53 time=172.361 ms
64 bytes from 192.168.2.102: icmp_seq=67 ttl=53 time=240.334 ms
```

Рисунок 23. Результат работы утилиты ping

На рисунке 21 видна задержка (строка Request timeout for icmp\_seq 64). В этот момент на мастер-ноде был отключен сервис KeepALiveD и в следствие изменена машина обрабатывающие запросы. Соответственно отключение сервиса показало корректную отработку смены сервера, обрабатывающего запросы. То есть подобное произойдет при любой причине отключения мастер-сервера и остановки отправки запросов на слейв-нод.

**Вывод:** были проверены инициализированные ноды кластера, штатные алгоритмы установленных инструментов отказоустойчивости кластера и проверена смена точки входа балансировщика нагрузки. Все утилиты и инструменты удовлетворяют поставленным условиям и требованиям.

**Вывод по главе:** в данной главе были обоснованы причины тех или иных технологий, разобраны принципы работы данных утилит. Был инициализирован кластер обработки данных с поддержкой виртуальных IP-адресов и отказоустойчивости. Так же были рассмотрены возможные механизмы обеспечения защиты и возможные методы интегрирования дополнительных модулей.

## ГЛАВА 3. ТЕСТИРОВАНИЕ ОБРАБОТКИ ДАННЫХ И ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ

### 3.1. Тестирование на потоке данных

Для проведения тестирования обработки данных можно использовать 4 варианта:

1. Генерация тестовых данных: с помощью инструментов генерации тестовых данных (например, `pgbench`) можно создать большое количество записей в базе данных PostgreSQL. Это позволит оценить производительность базы данных при обработке большого потока данных.

2. Нагрузочное тестирование: можно создать скрипты, которые будут генерировать запросы на чтение и запись данных в базу данных PostgreSQL, и запустить их в нескольких потоках. Это позволит оценить производительность базы данных при работе с множеством запросов и пользователей одновременно.

3. Тестирование репликации данных: можно создать несколько реплик базы данных PostgreSQL и запустить нагрузочное тестирование на основном узле. Затем можно проверить, что данные успешно реплицируются на все реплики и что они остаются доступными для чтения и записи.

4. Тестирование восстановления данных: можно имитировать сбой в основном узле базы данных PostgreSQL и проверить, как быстро и успешно происходит восстановление данных на других узлах кластера.

В рамках работы авторским предпочтением является нагрузочное тестирование собственным скриптом для создания популяции тестовых данных. Это позволит выявить недостатки в пропускной способности сети, производительность базы данных и корректности работы кластера на большом потоке данных.

Скрипт написан на языке программирования Python 3.8 и использует встроенные модули и `psycopg2` – низкоуровневое решение для обработки запросов SQL в коде Python:

```

#!/usr/bin/env python
#
# Usage: ./pgpoke.py <dsn>
#
# Edit MaxTries to configure to change the maximum amount of fail attempts

import psycopg2
from sys import argv
from time import sleep
import datetime
import logging

MaxTries = 5
logging.basicConfig(format='%(asctime)s %(message)s')

def main():
    seqno = 0
    counter = 0
    while True:
        try:
            counter = 0
            conn = psycopg2.connect(argv[1], connect_timeout=5)
            while True:
                tstamp = datetime.datetime.utcnow()
                cur = conn.cursor()
                cur.execute("SET statement_timeout = 5")
                cur.execute("SELECT seqno, tstamp FROM poke_table ORDER BY id DESC LIMIT 1")
                rows = cur.fetchall()
                if len(rows) == 0:
                    logging.error("No rows returned")
                else:
                    prev_seqno, prev_tstamp = rows[0]
                    if seqno - prev_seqno != 1:
                        logging.error(f"Sequence number mismatch - current: {seqno}, previous: {prev_seqno}")
                    logging.error(f"    Trying to insert sequence number {seqno}")
                    #cur.execute("set synchronous_commit = off")
                    try:
                        a = datetime.datetime.now()
                        cur.execute("INSERT INTO poke_table(seqno, tstamp) VALUES (%(seqno)s, %(tstamp)s)", {
                            'seqno': seqno,
                            'tstamp': tstamp
                        })
                        b = datetime.datetime.now()
                        logging.error(f"[{b.microsecond - a.microsecond} ms] Sucessfully inserted sequence number {seqno}")
                    except psycopg2.Error as error:
                        counter += 1
                        if counter == MaxTries:
                            return error
                        logging.error(f"Error occured: {error}")
                    conn.commit()

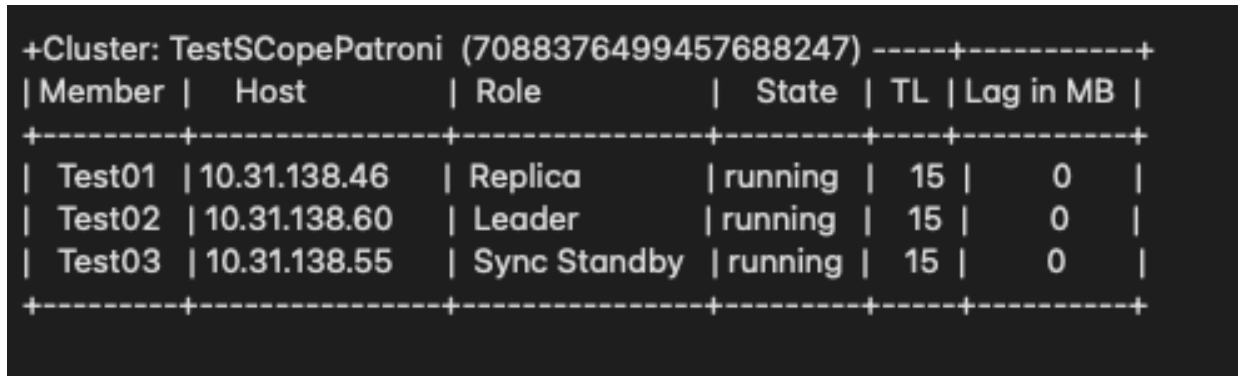
                    sleep(1)
                    seqno += 1
            except psycopg2.Error as exc:
                logging.error(f"Exception occured: {exc}")
                counter += 1
                if counter == MaxTries:
                    return exc
                sleep(2)

```

Рисунок 24. Скрипт для нагрузочного тестирования

Данный скрипт создает таблицу для популирования тестовыми данными и отправляет запрос на добавление данных каждую секунду. Данные отправляются на точку входа в архитектуру проекта и переадресуются в интерфейс входа в кластер. Скрипт может работать постоянно и отключаются лишь по требованию пользователя комбинацией клавиш для прекращения выполнения. В случае, прекращения обработки данных соответствующее уведомление получит пользователь и с интервалом в 2 секунды будет

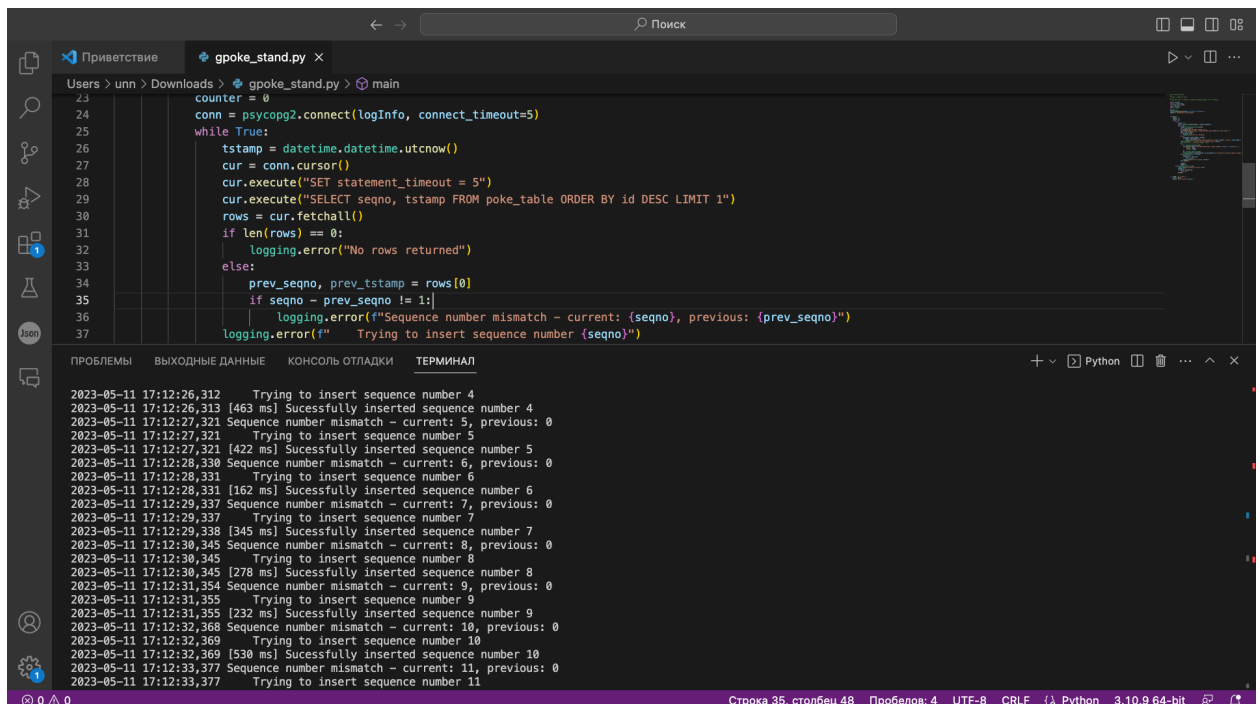
осуществляться попытка повторного популирования таблицы. Проверка результатов нагрузочного тестирования с помощью утилиты `patronictl list`:



```
+Cluster: TestSCOpePatroni (7088376499457688247) -----+-----+
| Member | Host           | Role           | State | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| Test01 | 10.31.138.46   | Replica        | running | 15 | 0          |
| Test02 | 10.31.138.60   | Leader         | running | 15 | 0          |
| Test03 | 10.31.138.55   | Sync Standby   | running | 15 | 0          |
+-----+-----+-----+-----+-----+-----+
```

Рисунок 25. Результат работы утилиты до начала тестирования

Для тщательной проверки скрипт был запущен ровно на час и отслеживались появляющиеся проблемы:



```
Users > unh > Downloads > gpoke_stand.py > main
23 counter = 0
24 conn = psycopg2.connect(logInfo, connect_timeout=5)
25 while True:
26     tstamp = datetime.datetime.utcnow()
27     cur = conn.cursor()
28     cur.execute("SET statement_timeout = 5")
29     cur.execute("SELECT seqno, tstamp FROM poke_table ORDER BY id DESC LIMIT 1")
30     rows = cur.fetchall()
31     if len(rows) == 0:
32         logging.error("No rows returned")
33     else:
34         prev_seqno, prev_tstamp = rows[0]
35         if seqno - prev_seqno != 1:
36             logging.error(f"Sequence number mismatch - current: {seqno}, previous: {prev_seqno}")
37         logging.error(f"Trying to insert sequence number {seqno}")

ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ
2023-05-11 17:12:26,312 Trying to insert sequence number 4
2023-05-11 17:12:26,313 [463 ms] Successfully inserted sequence number 4
2023-05-11 17:12:27,321 Sequence number mismatch - current: 5, previous: 0
2023-05-11 17:12:27,321 Trying to insert sequence number 5
2023-05-11 17:12:27,321 [422 ms] Successfully inserted sequence number 5
2023-05-11 17:12:28,330 Sequence number mismatch - current: 6, previous: 0
2023-05-11 17:12:28,331 Trying to insert sequence number 6
2023-05-11 17:12:28,331 [162 ms] Successfully inserted sequence number 6
2023-05-11 17:12:29,337 Sequence number mismatch - current: 7, previous: 0
2023-05-11 17:12:29,337 Trying to insert sequence number 7
2023-05-11 17:12:29,338 [345 ms] Successfully inserted sequence number 7
2023-05-11 17:12:30,345 Sequence number mismatch - current: 8, previous: 0
2023-05-11 17:12:30,345 Trying to insert sequence number 8
2023-05-11 17:12:30,345 [278 ms] Successfully inserted sequence number 8
2023-05-11 17:12:31,354 Sequence number mismatch - current: 9, previous: 0
2023-05-11 17:12:31,355 Trying to insert sequence number 9
2023-05-11 17:12:31,355 [232 ms] Successfully inserted sequence number 9
2023-05-11 17:12:32,368 Sequence number mismatch - current: 10, previous: 0
2023-05-11 17:12:32,369 Trying to insert sequence number 10
2023-05-11 17:12:32,369 [530 ms] Successfully inserted sequence number 10
2023-05-11 17:12:33,377 Sequence number mismatch - current: 11, previous: 0
2023-05-11 17:12:33,377 Trying to insert sequence number 11
```

Рисунок 26. Процесс работы скрипта

Далее на изображениях процесса работы будут выводы работы скрипта.

Спустя час наблюдения ошибок не было, а объем записанных данных составил почти 3 гигабайта (по гигабайту на каждую ноду). Утилита `patronictl` показывала следующее:



+Cluster: TestScopePatroni (7088376499457688247) -----+						
Member	Host	Role	State	TL	Lag in MB	
+-----+						
Test01	10.31.138.46	Replica	running	370	0	
Test02	10.31.138.60	Leader	running	370	0	
Test03	10.31.138.55	Sync Standby	running	370	0	
+-----+						

Рисунок 27. Результат работы утилиты после начала тестирования

Данный результат показал безотказное прохождение нагрузочного тестирования. В реальном мире поток данных может быть больше по объему и сложности, однако это исключение. Задача состояла в проверке устойчивости длительном потоке данных, что и было доказано. Далее был проведен такой же тест с выводом их строя мастер-ноды и смены сервера, обрабатывающего запросы.

Кластер должен отработать выключение мастер-ноды, прервать получения потока данных до момента завершения работы механизма переназначения ролей в кластере и продолжить штатную обработку запросов.

+Cluster: TestScopePatroni (7088376499457688247) -----+						
Member	Host	Role	State	TL	Lag in MB	
+-----+						
Test01	10.31.138.46	Replica	running	12	0	
Test02	10.31.138.60	Leader	running	12	0	
Test03	10.31.138.55	Sync Standby	running	12	0	
+-----+						

Рисунок 28. Начальное состояние кластера в отображении patronictl list



На рисунке 30 видно восстановление значения TL и восстановление актуальных данных в бывшем мастере кластера. Таким образом была доказана стрессо- и отказоустойчивость данной конфигурации кластера.

**Вывод:** было произведено нагрузочное тестирование, доказавшее пригодность использования данной конфигурации в особо стрессовых условиях, и сохраняет актуальность данных во всех нодах кластера.

### **3.2. Выявление проблем и недостатков в кластерах и их решение**

Ввиду многогранности возможных проблем в любых конфигурациях кластера требуется жесткий разбор и регулирование всех аспектов. Есть два класса проблем:

**1. Физические нарушения целостности системы.** Этот класс можно разбить на несколько пунктов:

- Уничтожение или повреждение оборудования: Физическое нарушение целостности системы может привести к уничтожению или повреждению серверов, хранилищ данных или другого оборудования, которое используется в кластере. Это может привести к потере данных или недоступности кластера.
- Перехват или кража оборудования: Несанкционированный доступ к физическому оборудованию кластера может привести к краже или перехвату серверов или хранилищ данных. Это может привести к потере данных или доступности кластера, а также к возможному несанкционированному доступу к конфиденциальной информации.
- Уничтожение или повреждение сетевой инфраструктуры: Физическое нарушение целостности системы может также привести к уничтожению или повреждению сетевой инфраструктуры, включая коммутаторы, маршрутизаторы или кабельные соединения. Это может вызвать проблемы в связи между узлами кластера и привести к недоступности или нестабильности работы кластера.
- Несанкционированный доступ к физическим ресурсам: Физическое нарушение целостности системы может создавать риск

несанкционированного доступа к физическим ресурсам кластера, таким как серверные помещения или центры обработки данных. Это может привести к возможному вмешательству в работу кластера или угрозе конфиденциальности данных.

- Повреждение или изменение конфигурации: Физическое нарушение целостности системы может привести к непредвиденным изменениям или повреждению конфигурационных файлов или настроек кластера. Это может привести к некорректной работе кластера или потере данных.

Однако данную угрозу невозможно решить в корне, но возможно минимизировать. Для максимального снижения риска возникновения угроз требуется следующее:

- Физическая безопасность помещений: Обеспечение физической безопасности серверных помещений и центров обработки данных, где находятся серверы и другое оборудование кластера. Установка мер контроля доступа, такие как замки, системы видеонаблюдения, системы контроля доступа и пропускной системы.

- Ограничение физического доступа: Ограничение физический доступ к серверам и другому оборудованию кластера только авторизованному персоналу. Установка меры идентификации, такие как пропускные системы и биометрические системы, для контроля доступа к помещениям.

- Шифрование данных: Использование шифрования данных на уровне хранения и передачи. Шифрование данных, хранящихся на дисках или в файловой системе, а также данные, передаваемые по сети между узлами кластера. Это поможет защитить данные от несанкционированного доступа в случае физического доступа к оборудованию.

- Контроль состояния оборудования: Введение правил по периодическому контролю состояния оборудования исправит проблему неожиданной поломки технических устройств серверов обработки данных.

Данные мероприятия требуются для организаций любого размера, так как все подвержены рискам физического нарушения целостности системы.

Независимо от того, является ли организация крупной корпорацией или небольшим предприятием, наличие соответствующих мер безопасности является неотъемлемой частью обеспечения надежности, конфиденциальности и доступности данных в кластере баз данных.

**2. Нарушение целостности цифрового пространства.** Во второй главе были разобраны возможные точки входа в цифровое пространство системы. Для удобства разбора была подготовлена подробная абстрактная схема архитектуры проекта:

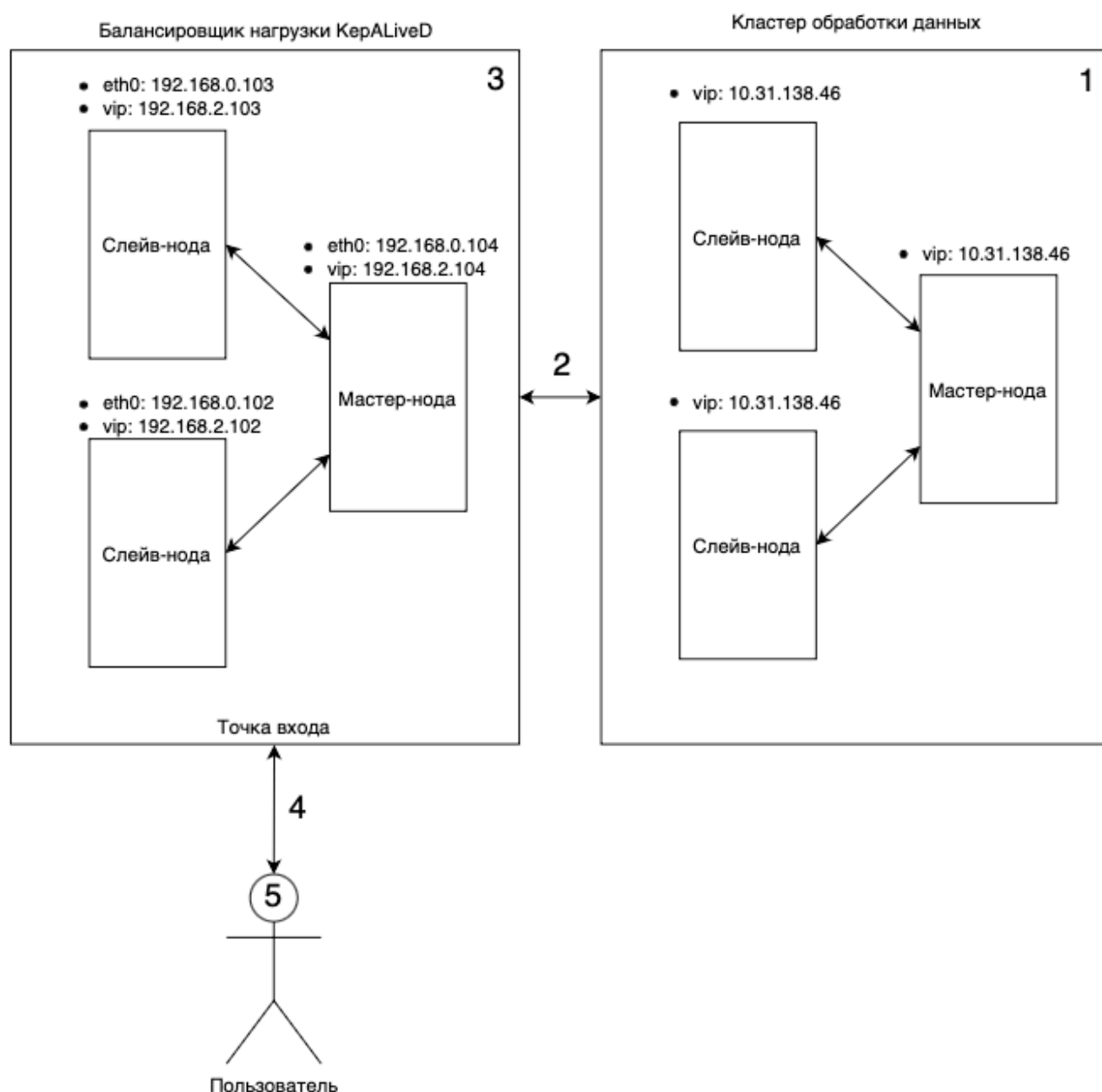


Рисунок 32. Полная схема архитектуры

В схеме, изображенной на рисунке 31, существуют 5 возможных мест проникновения в цифровое пространство злоумышленниками:

1. Нарушения целостности защитного контура кластера. Предотвращение доступа злоумышленниками на физическом уровне был разобран в предыдущем пункте и не брался в расчет, остается точка входа. Так как подразумевается, что кластер получает поток данных только от балансировщика нагрузки, то достаточно настроить брандмауэр на белый список IP-адресов только на IP-адрес балансировщика и нескольких администраторов системы.

2. Нарушение целостности канала связи кластера и балансировщика нагрузки. Для обхода брандмауэра контура кластера возможна подмена эмулированием IP-адреса. Данную проблему решает введение авторизации, аутентификации и токенов авторизации. Достаточно ввести алгоритм генераций паролей по одному хеш-ключу и канал связи будет надежно защищен.

3. Нарушение целостности защитного контура балансировщика нагрузки. Схема защиты схожа с контуром кластера, однако адресов подключения может быть множество. Для минимизации шанса возникновения данной угрозы требуется введение всевозможных мер защиты. Например, VPN, IPS, туннелирование, аутентификация и авторизация, сессионное использование системы.

4. Канал связи между конечным пользователем и точкой входа в систему обеспечивается общим контуром балансировщика нагрузки, однако возможна дополнительно требуется введение журналирование историй подключений и трафика для последующего анализа и выявления проблем.

5. Конечный пользователь – самая уязвимая часть системы и для минимизации шансов возникновения угроз существует два способа – сессионное подключение по VPN протоколу к системе и выдача конечному пользователю АРМ с жестким протоколированием установленного ПО и возможных модификаций ОС.

Общим требованием к обрабатываемой информации требуется введение криптографических алгоритмов шифрования информации в системе с момента поступления, до момента выхода из нее, что повышает уверенность в защите при нарушении любого из контуров защиты.

**Вывод:** были разобраны возможные уязвимости в общей архитектуре как физического типа, так и цифрового. Были предложены методы и технологии для их ликвидации и повышения общего уровня безопасности системы.

### **3.3. Способы выявления общих проблем и способы повышения эффективности**

Ввиду асинхронности и синхронности используемых технологий и утилит вопрос скорости обработки данных исключительно в физических показателях системы: частоте и количестве ядер процессора, типа кабелей, объеме и частоте оперативной памяти, скорости накопителей данных и т.д.

На практике непреодолимость скорости света является серьезным камнем преткновения — например, между двумя ЦОД, связанных оптоволоком, находящихся на расстоянии 100 километров, время задержки составит 0.3 миллисекунды при условии отсутствия иных причин задержки сигнала. Поэтому существует корреляция между объемом и частотой обрабатываемой информацией и качеством используемого оборудования во всех мелочах и из-за этого требуется тщательный подбор требуемого технического оборудования.

Одной из общих проблем, с которой можно столкнуться, является недостаточная производительность кластера. Это может быть вызвано множеством факторов, включая неправильную конфигурацию, неэффективные запросы, отсутствие оптимизации индексов или недостаточную аппаратную мощность. Для повышения эффективности кластера можно применять следующие подходы:

1. Оптимизация запросов: Анализ выполняемых запросов и оптимизация их для повышения производительности. Рассмотрение использования индексов,

переписывание запросов, устранение дублирующихся или ненужных операций. Использование инструментов профилирования и трассировки запросов для идентификации медленных или неэффективных запросов.

2. Конфигурация базы данных: Проверка настройки конфигурации кластера баз данных. Оптимизация размер буферного кэша, количество одновременных соединений, параметры репликации и другие параметры, чтобы соответствовать требованиям производительности и нагрузки.

3. Масштабирование: Рассмотрение возможности масштабирования кластера баз данных для увеличения пропускной способности и обработки большего объема данных. Это может включать добавление новых узлов кластера или использование вертикального или горизонтального масштабирования.

4. Мониторинг и настройка: Инициализация системы мониторинга и алертинга для отслеживания производительности и доступности кластера. Использование средства мониторинга, таких как Prometheus, Grafana или Nagios, для сбора и анализа метрик, чтобы раньше выявлять проблемы и принимать соответствующие меры.

5. Регулярное обслуживание и обновление: Регулярное обслуживание и обновление кластера баз данных. Это включает в себя установку патчей безопасности, обновление версии базы данных, анализ и оптимизацию структуры базы данных и прочего ПО, а также регулярное резервное копирование и проверку целостности данных.

6. Тестирование и анализ производительности: Регулярное тестирование производительности для выявления узких мест и определения областей, требующих оптимизации. Используйте инструменты для нагрузочного тестирования и анализа производительности, чтобы получить данные о производительности кластера и принять соответствующие меры.

В целом, выявление общих проблем и применение способов повышения эффективности кластера баз данных позволяет оптимизировать работу системы, обеспечить высокую производительность, доступность и



надежность. Регулярное мониторинг, обслуживание и анализ позволяют выявлять проблемы на ранних стадиях и принимать меры для их решения, что способствует эффективной и надежной работе кластера баз данных.

**Вывод:** была изучена проблематика всех аспектов построения кластеров обработки информации, выявлены проблемы и уязвимые точки в общей архитектуре и предложены варианты минимизации рисков и защиты критических точек.

**Вывод по главе:** в данной главе была протестирована инициализированная система обработки данных на стрессо- и отказоустойчивость, путем нагрузочного тестирования и аварийного выключения мастера кластера. Были разобраны методики повышения уровня безопасности в общей архитектуре и конкретных критических точках. Так же были приведены способы повышения эффективности системы обработки данных для каждого аспекта в архитектуре.

## ЗАКЛЮЧЕНИЕ

В рамках исследования были рассмотрены теоретические основы создания и эксплуатации стрессо- и отказоустойчивых кластеров баз хранения данных. Были изучены принципы работы кластеров, механизмы репликации данных, шаринг данных, балансировку нагрузки и обеспечение высокой доступности. Это позволило уяснить важность использования таких решений и технологий для обеспечения надежности и эффективности работы баз данных.

Одним из ключевых результатов исследования является выявление подходящих технологий и инструментов для создания кластера баз данных. Были изучены PostgreSQL, Patroni, etcd и KeepALiveD. Описаны принципы их работы, а также способы их взаимодействия для создания надежного и гибкого кластера.

Были предложены рекомендации по обнаружению и устранению общих проблем, а также способы повышения эффективности кластера баз данных. Это включает оптимизацию запросов, настройку базы данных, масштабирование, мониторинг, обновление и резервное копирование данных. Применение этих рекомендаций позволяет достичь оптимальной производительности и надежности кластера.

В процессе исследования было выявлено, что использование отечественного ПО и решений с открытым исходным кодом предоставляет ряд преимуществ. Они обеспечивают высокую производительность, возможность масштабирования, отказоустойчивость и простоту в настройке и управлении. Также были идентифицированы некоторые ограничения, которые требуют дополнительного внимания и решения.

Важным аспектом данного исследования является подчеркивание значимости обеспечения физической безопасности и защиты от угроз, связанных с физическим нарушением целостности системы. Реализация соответствующих мер безопасности, таких как контроль доступа к серверным

помещениям, шифрование данных и мониторинг физической безопасности, является важной составляющей обеспечения безопасности кластера баз данных. Это помогает предотвратить несанкционированный доступ к данным, минимизировать риски физического нарушения целостности системы и обеспечить сохранность важной информации.

В заключение, данное исследование позволило получить глубокое понимание принципов работы кластеров баз данных и ознакомиться с современными технологиями и инструментами, такими как PostgreSQL, Patroni, ETCD и KeepALiveD. Были выявлены общие проблемы, с которыми можно столкнуться при создании и эксплуатации кластера, и предложены способы их решения для повышения эффективности и надежности системы.

Однако, необходимо отметить, что реализация и успешное функционирование кластера баз данных требует комплексного подхода и учета индивидуальных особенностей организации. Кроме того, технологии и инструменты, описанные в данном исследовании, могут быть подвержены изменениям и развитию, поэтому рекомендуется обновлять свои знания и следить за новыми тенденциями в области баз данных и управления кластерами.

В целом, данное исследование способствует более глубокому пониманию создания и эксплуатации стрессо- и отказоустойчивых кластеров баз данных. Оно предоставляет базовые знания и рекомендации, которые могут быть использованы при проектировании и развертывании кластеров в реальных организациях. Эти знания помогут повысить надежность, производительность и доступность системы, обеспечивая более эффективное управление данными и поддержку бизнес-процессов.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Официальная документация PostgreSQL: <https://postgrespro.ru/docs>
2. Официальный сайт Patroni: <https://github.com/zalando/patroni>
3. Официальный сайт etcd: <https://etcd.io/>
4. Официальная документация Keepalived: <https://www.keepalived.org/documentation.html>
5. Журнал "Открытые системы": <http://www.osjournal.ru/>
6. Журнал "Хакер": <https://xakep.ru/>
7. Книга "PostgreSQL. Администрирование и разработка" (Александр Коротков, Максим Богданов): <https://www.ozon.ru/product/postgresql-administrirovanie-i-razrabotka-19209569>
8. Книга "Хранение данных на базе PostgreSQL" (Дмитрий Жуков, Александр Зайцев): <https://www.ozon.ru/product/khranenie-dannyh-na-baze-postgresql-123791371>
9. Сайт отечественной компании Postgres Professional: <https://postgrespro.ru/>
10. Блог Ruslan Fatkhullin, эксперта в области PostgreSQL: <https://fatkhullin.info/>