**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ**

**ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ**

**(АКТ (ф) СПбГУТ)**

Допущен к защите

Зав. отделением

**\_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись) (И.О. Фамилия)

«\_\_» \_\_\_\_\_ 20\_\_ г.

**ДИПЛОМНЫЙ ПРОЕКТ**

### НА ТЕМУ

|  |
| --- |
| Проектирование и развертывание кластера PostgreSQL на базе Patroni |
| на предприятии ООО «ТГК-2 ЭНЕРГОСБЫТ» |
| Л114. 25ДП00. 003 ПЗ |

(Обозначение документа)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Рецензент |  | |  |  | А.М. Ахметшина |
|  |  | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Руководитель |  | |  |  | К.С. Ефремова |
|  |  | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Дипломник |  | |  |  | В.К. Кузьмин |
|  | |  | (Подпись) | (Дата) | (И.О. Фамилия) |
| Консультант по оформлению | |  |  |  | К.С. Ефремова |
|  | | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Консультант по экономической части | | |  |  | С.Н. Короткова |
|  | | | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2025

**СОДЕРЖАНИЕ**

Перечень сокращений и обозначений

Введение

1 Анализ и разработка требований к модернизации кластера

1.1 Постановка актуальности и основной задачи проекта

1.2 Анализ организации ООО «ТГК-2 Энергосбыт»

1.3 Определение кластера PostgreSQL

1.4 Аппаратно-программные требования к модернизации кластера PostgreSQL

1.5 Выбор программных средств для модернизации

1.6 Составление плана работ

2 Настройка отказоустойчивости кластера

2.1 Установка и конфигурация HAProxy

2.2 Установка и конфигурация PgBouncer

2.3 Установка и конфигурация менеджера Keepalived

3 Настройка мониторинга ресурсов кластера

3.1 Установка и конфигурация платформы для сбора метрик и мониторинга систем Prometheus

3.2 Конфигурация сервиса оповещений на основе Prometheus

3.3 Установка и конфигурация инструмента визуализации временных рядов данных Grafana

4 Автоматизация развёртывания средств сбора метрик и мониторинга кластера

4.1 Определение инвентаря и плейбуков сервиса Anisble

4.2 Конфигурация роли для установки экспортеров метрик

4.3 Конфигурация роли для установки мониторинга

6 Технико-экономическое обоснование модернизации PostgreSQL кластера

7 Техника безопасности при работе с ПК

7.1 Общие требования безопасности

7.2 Требования охраны труда перед началом работы

7.3 Требования охраны труда во время работы

7.4 Требования охраны труда в аварийных ситуациях

7.5 Требования безопасности по окончании работы

Заключение

Список использованных источников

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящем дипломном проекте применяются следующие сокращения и обозначения:

БД – база данных

ВМ – виртуальная машина

ОС – операционная система

ОУК – отказоустойчивый кластер СУБД Postgres Pro

СУБД – система управления базами данных

API – интерфейс программирования приложения

APT – расширенный инструмент для работы с пакетами

DCS – распределенная система конфигурации

DNS – система доменных имен

FAQ – часто задаваемые вопросы

IP – протокол Интернета

SSH – безопасная оболочка

TCP – транспортный протокол для передачи данных

VIP – виртуальный IP

**ВВЕДЕНИЕ**

В современном мире информационных технологий и высокоскоростного обмена данными, использование баз данных стало неотъемлемой частью жизни и работы многих организаций. Обеспечение эффективного проектирования и внедрения оптимизированных баз данных выступает ключевым фактором повышения конкурентоспособности и общей продуктивности предприятий.

Местом прохождения практики стало ООО «ТГК-2 Энергосбыт», специализирующемся на поставке электроэнергии населению Архангельска и АО.

Целью дипломной работы является реализация модернизации кластера PostgreSQL на базе платформы Patroni на указанном предприятии.

В ходе проекта проведен детальный анализ текущего состояния инфраструктуры базы данных предприятия, выявлены ключевые характеристики и сформулированы необходимые требования к модернизации. Проект представляет собой комплексную задачу, включающую в себя ряд этапов, таких как анализ требований к кластеру, выбор и обоснование схемы работы с базами, разработку и реализацию всех необходимых компонентов и сервисов.

Результатом реализации проекта станет создание высокоэффективного и надежного кластера PostgreSQL, удовлетворяющего требованиям предприятия. Кластеризация баз данных обеспечивает значительное повышение надежности, гарантируя бесперебойный доступ к ресурсам даже при выходе из строя отдельных узлов. Помимо этого, подобная архитектура способствует повышению уровня масштабируемости инфраструктуры путем равномерного распределения вычислительной нагрузки между компонентами кластера, что ведет к увеличению скорости обработки запросов и улучшению общего качества обслуживания клиентов.

Преимущества перехода на кластеризацию СУБД включают в себя следующие параметры:

* высокая устойчивость к сбоям оборудования за счёт избыточности ресурсов;
* непрерывность доступа к СУБД при условии работоспособности хотя бы одного узла кластера;
* возможность расширять инфраструктуру горизонтально, улучшая пропускную способность системы;
* ускорение обработки запросов посредством балансировки нагрузок между серверами;
* централизованное управление инфраструктурой и облегчение администрирования кластера.

1 Анализ и разработка требований к модернизации кластера

1.1 Постановка основной задачи проекта

В настоящее время PostgreSQL благодаря своей надежности, производительности и открытости исходного кода является одной из наиболее популярных реляционных СУБД, однако для крупных предприятий и высоконагруженных проектов одного экземпляра PostgreSQL недостаточно, а кластеризация не предоставляет достаточной степени масштабирования: имеет проблемы с переключением мастер/реплика и требует практически постоянного ручного вмешательства со стороны системного администратора. Предприятиям требуются масштабируемые решения с высокой доступностью и надежностью, которые позволят автоматизировать процесс управления кластером и облегчить его обслуживание.

Именно таким решением является Patroni – сервис для управления PostgreSQL-кластерами с автоматическим восстановлением и переключением между узлами кластера. Patroni обеспечивает высокий уровень доступности баз данных, минимизируя риск потери данных и снижения качества обслуживания пользователей даже при отказе отдельных серверов.

Несмотря на очевидные преимущества, внедрение Patroni требует модернизации существующей инфраструктуры. Именно этот аспект становится предметом текущего дипломного проекта. Проект направлен на разработку комплексной системы, включающей механизмы отказоустойчивости, балансировку нагрузки и мониторинг состояния кластеров PostgreSQL. Это позволит существенно повысить производительность и надежность корпоративной информационной системы, минимизировать риски потерь данных и обеспечить непрерывность работы предприятия.

Настройка подобного кластера представляет особую важность для организаций, стремящихся сохранить конкурентоспособность и стабильность в условиях растущих объемов данных и требований к инфраструктуре предприятий.

Для выполнения модернизации кластера необходимо сначала провести анализ мест потенциального улучшения кластера, определить требования для дальнейшей модернизации и объединить все используемые сервисы с кластером.

Задачи проекта:

* проведение анализа текущего состояния кластера PostgreSQL;
* оценивание существующих ограничений и выявление возможностей для совершенствования;
* модификацию инфраструктуры кластера для увеличения доступности и устойчивости к сбоям.

**1.2 Анализ предприятия ООО «ТГК-2 Энергосбыт»**

Дипломный проект выполняется на предприятии ООО «ТГК-2 Энергосбыт». Данное предприятие является крупнейшим субъектом оптового рынка электрической энергии (мощности) и гарантирующим поставщиком электрической энергии в Архангельской области []. Предприятие осуществляет деятельность в качестве независимой энергосбытовой компании на территории Архангельской, Ярославской, Вологодской и Новгородской областей.

Как крупный поставщик электроэнергии, ООО «ТГК-2 Энергосбыт» постоянно сталкивается с большими потоками данных, включая информацию о клиентах, договорных обязательствах, финансовых операциях, учетных системах и производственных показателях. Эти данные требуют надежного хранения, быстрого доступа и постоянной защиты от возможных технических проблем и угроз безопасности. Важнейшей частью такой инфраструктуры являются современные системы управления базами данных. Поскольку объемы обрабатываемой информации огромны, традиционные подходы к организации хранилищ данных становятся неэффективными. Поэтому компания решила перейти на использование кластерных решений на базе PostgreSQL, поддерживаемых инструментом Patroni.

**1.3 Определение кластера PostgreSQL**

Перед началом изучения способов модернизации кластеров PostgreSQL, необходимо разобраться с основными понятиями и технологиями, лежащими в основе существующего на предприятии кластера. Ключевыми элементами его архитектуры выступают три компонента:

1. сервис согласования состояний Etcd;
2. кластер PostgreSQL;
3. система управления кластером Patroni.

Понимание особенностей каждого вышеперечисленного элемента позволит лучше осознать принципы построения решений на базе PostgreSQL. Далее будет рассмотрено, что представляют собой кластеры этой СУБД, какова роль инструмента Patroni в управлении ими, а также какую пользу приносит использование распределенного сервиса хранения данных Etcd.

Etcd представляет собой распределённое отказоустойчивое хранилище ключей-значений, предназначенное для координации процессов и обмена состоянием между компонентами сложных распределённых систем (вычислительная система, где компоненты распределены между несколькими узлами или компьютерами, взаимодействующими через сеть). Оно используется для реализации высокодоступных решений, таких как Patroni, для хранения важной информации о структуре и состояниях кластера PostgreSQL.

Сервис «Etcd» применяется в роли координатора, хранящего информацию о кластере, текущих ролях узлов, изменениях топологии кластера и статусе каждой машины. Если какой-то узел выходит из строя, Etcd сигнализирует другим узлам, что состояние изменилось, и запускается процесс автоматического восстановления или переключения.

Кластер PostgreSQL представляет собой группу серверов, объединённых вместе для совместного выполнения одной задачи, такой как хранение и обработка данных. Основная цель кластера - повышение надёжности и доступности данных, обеспечение высокой производительности и устойчивости к отказам аппаратного и программного обеспечения. Кластер должен включать в себя от двух узлов (нод): один ведущий узел (далее - мастер) и один или больше подчинённых узлов (далее - реплика). Ведущий узел отвечает за запись новых данных и выполнение основных функций обновления БД, тогда как реплики отвечают за чтение данных и создание резервных копий, улучшая общую производительность и уменьшая нагрузку на основной сервер.

Patroni был создан как инструмент автоматизации управления кластерами PostgreSQL, разработанный для обеспечения высокой доступности, устойчивости и удобного управления кластером. Его задача - поддержка согласованности между всеми узлами кластера, автоматически обнаруживать неисправности и производить необходимые изменения конфигурации (пример: автоматическое повышение статуса реплики до мастера, при недоступности сервиса на текущей мастер ноде) для поддержания нормальной работы системы.

Таким образом, сочетание распределённого хранилища Etcd, кластера PostgreSQL и сервиса Patroni позволяет создать мощную, устойчивую и производительную инфраструктуру для работы с данными в промышленных условиях. Данный набор инструментов оптимален для эксплуатации в корпоративных проектах, таких как проекты энергоснабжения, банковские приложения и веб-сервисы высокого уровня доступности.

На предприятии реализован кластер со следующими параметрами: три виртуальные машины, на каждой установлен сервис Etcd, вместе они образованы в кластер для хранения данных о кластере PostgreSQL. СУБД и сервис Patroni установлены на двух из трёх машин, в следствие чего образована инфраструктура следующего плана (рисунок 1, 2).

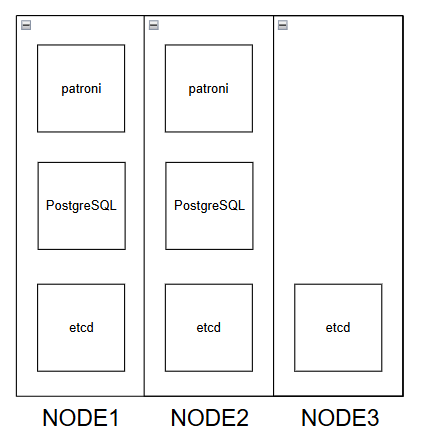


Рисунок 1 – Вид внешний инфраструктуры кластера PostgreSQL на предприятии

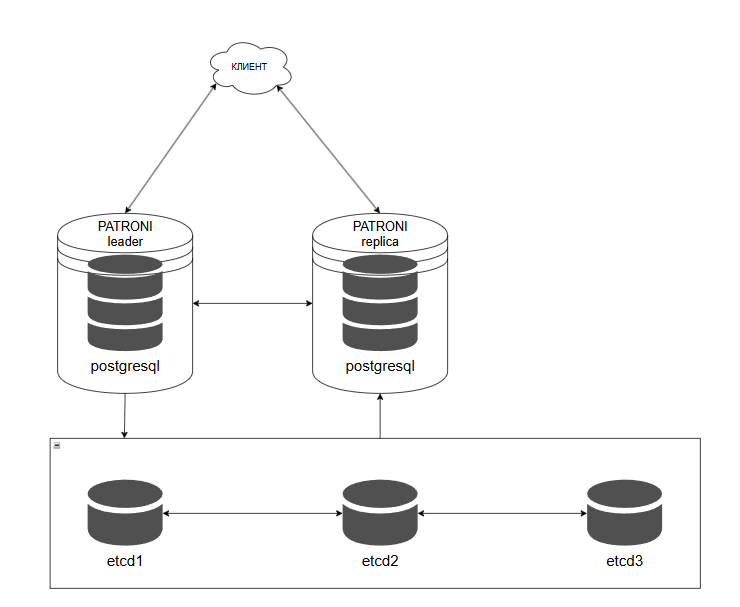


Рисунок 2 – Вид внешний схемы взаимодействий сервисов кластера

1.4 Аппаратно-программные требования к модернизации кластера PostgreSQL

Выявление системных требований к процессу модернизации кластера требует предварительного анализа существующей инфраструктуры СУБД на предмет числа ядер процессора, объёма оперативной памяти и дискового пространства на машину (таблица 1).

Таблица 1 – Анализ аппаратных ресурсов, используемых на кластере PostgreSQL

|  |  |
| --- | --- |
| **Сервер СУБД PostgreSQL** | |
| Количество ядер процессора | 4 ядра |
| Объём оперативной памяти | 8 гигабайт |
| Дисковое пространство | 500 гигабайт |

По результатам оценивания системных требований принято решение организовать тестовую среду, состоящую из виртуальных машин со следующими параметрами:

- 3 ядра процессора;

- 4 гигабайта оперативной памяти;

- 40 гигабайт динамически заполняемого дискового пространства.

Данных характеристик должно быть достаточно для проверки работоспособности созданного кластера, перед внедрением изменений в производственную среду основного кластера предприятия.

1.4 Выбор программных средств для модернизации

Выбор программных средств для модернизации кластера начинается с определения направлений потенциальных усовершенствований, на основании которых осуществляется подбор соответствующих программных решений, устраняющих недостатки и повышающих надежность кластера. Проведенный анализ применяемых на действующем кластере PostgreSQL программных средств (таблица 2), зон потенциального улучшения, а также техническое задание, представленное предприятием, послужили основой для формирования перечня программного обеспечения, предназначенного для оснащения тестового стенда PostgreSQL:

* система управления соединениями;
* система балансировки нагрузки;
* сервис виртуального IP для обеспечения единой точки входа в кластер PostgreSQL;
* системы экспорта метрик СУБД, прокси и статуса использования ресурсов виртуальных машин;
* система мониторинга, сбора и хранения метрик;
* система визуализации временных рядов данных;
* сервис оповещений об сбоях в программном и аппаратном обеспечении.

Таблица 2 – Анализ программных средств, применяемых на кластере PostgreSQL

|  |  |
| --- | --- |
| **Анализ программных средств** | |
| Операционная система | Debian 12 bookworm |
| СУБД | PostgreSQL 16 |
| Управление кластером | Patroni 4.0.5 |
| Хранилище данных | Etcd 3.5.9 |

По итогам анализа зон потенциальных улучшений была составлена таблица, содержащая решения, обеспечивающие нужную степень модернизации кластера СУБД предприятия (таблица 3).

Таблица 3 – Уязвимости кластера и сервисы, способные это исправить

|  |  |
| --- | --- |
| **Проблема** | **Решение** |
| Неравномерная нагрузка на отдельные узлы PostgreSQL при обработке клиентских запросов | HAProxy |
| Отсутствие единой точки входа | Keepalived |
| Избыточная нагрузка на кластер из-за открытия новых соединений | Pgbouncer |
| Отсутствие мониторинга использования ресурсов | Prometheus  Grafana |
| Отсутствие системы уведомлений о критических сбоях в инфраструктуре | Prometheus |

1.4 Составление плана работ

Учитывая невысокую степень надёжности развёртывания решений непосредственно на машинах с установленной СУБД, целесообразно включить в кластер четвёртую виртуальную машину. Четвёртая ВМ совместно с третьей будет выполнять функции балансировки нагрузки, сбора метрик, мониторинга и системы оповещений. Подобный подход снижает вероятность отказа кластера за счет рационального распределения нагрузки и физического разделения служб поддержки СУБД от серверов базы данных.

Итоговая схема предусматривает формирование четырехмашинного кластера, где две единицы предназначены для размещения СУБД и менеджера соединений PgBouncer, тогда как оставшиеся две используются для запуска сервисов, поддерживающих функционирование кластера (рисунок 3).

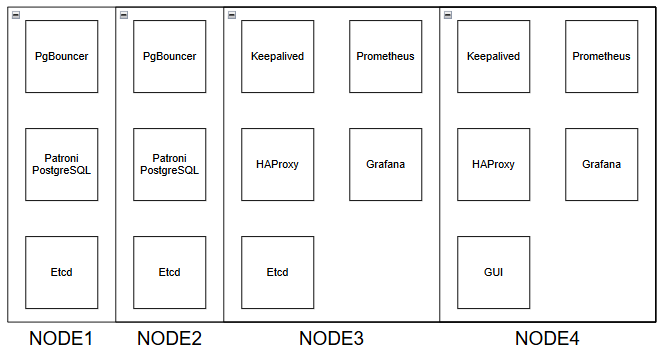


Рисунок 3 – Вид внешний тестовой инфраструктуры кластера PostgreSQL

План настройки и установки:

1. HAProxy – сервис балансировки нагрузки. Прокси равномерно нагружает обе ноды СУБД;
2. PgBouncer – сервис балансировки нагрузки, для новых запросов использует уже существующие подключения к базе данных;
3. Keepalived – сервис предоставляет виртуальный IP адрес как единую точку входа в базу данных (при падении мастер ноды кластера пользователю не придётся обращаться к бд по другому адресу);
4. Prometheus (платформа для сбора метрик и мониторинга систем);
5. сервис оповещений на основе Prometheus;
6. Grafana (инструмента визуализации временных рядов данных);
7. Автоматизация развёртывания средств сбора метрик и мониторинга кластера.

2 Настройка отказоустойчивости кластера

2.1 Установка и конфигурация HAProxy

HAProxy представляет собой универсальный высокопроизводительный прокси-сервер, ориентированный на распределение сетевого трафика и повышение производительности приложений. Несмотря на изначальную ориентацию продукта на обработку только HTTP-запросов, поддержка протоколов TCP делает возможным применение HAProxy для балансировки запросов к реляционным базам данных, включая PostgreSQL.

В проекте HAProxy применяется для эффективного распределения нагрузки между несколькими серверами СУБД следующим образом:

- разделение входящего трафика на запись и чтение;

- вывод трафика записи на мастер ноду Patroni;

- вывод трафика чтения на ноду реплики Patroni.

Описанный метод балансировки нагрузки обеспечивает оптимальное распределение входящих запросов между серверами PostgreSQL, значительно снижая рабочую нагрузку на ведущий сервер и позволяя увеличить общую эффективность кластера СУБД.

Сервис HAProxy устанавливается посредством APT - стандартного менеджера пакетов Debian на две виртуальные машины node3 и node4 без изменений в конфигурации. Следующим этапом является настройка конфигурации, осуществляемая путем внесения изменений в файл «haproxy.cfg», расположенный в директории «/etc/haproxy/».

Внешний вид файла конфигурации HAProxy представлен на листинге А.1. В данном файле конфигурации указаны следующие ключевые параметры для корректной балансировки нагрузки [1]:

1. параметр «mode» устанавливает режим передачи данных. Установка значения «tcp» активизирует работу сервиса «HAProxy» в режиме трансляции трафика протокола TCP. Данный режим обеспечивает совместимость с различными видами реляционных баз данных, включая PostgreSQL;
2. интерфейс «frontend patroni\_master» служит точкой входа для клиентских запросов, поступающих на порт 5000, обеспечивая направление обращений к интерфейсу обработки «pg\_master\_backend», ответственному за обработку главного узла базы данных;
3. интерфейс «frontend patroni\_replica» обрабатывает запросы, полученные на порту 5001, передавая их на интерфейс обработки «pg\_replica\_backend» ответственному за узлы базы данных, выполняющие роль реплик;
4. компонент «backend pg\_master\_backend» формирует группу, определяющую основной узел PostgreSQL, на который в последствии направляются входящие запросы на запись и (или) чтение данных;
5. компонент «backend pg\_replica\_backend» реализует распределение нагрузки на инфраструктуру PostgreSQL посредством переадресации запросов чтения на доступные копии базы данных;
6. опция «option httpchk GET (master/replica)» совместно с «http-check» предназначена для мониторинга состояния серверов путем регулярного осуществления HTTP-запросов на их IP адреса позволяя определить доступность сервера на основании возвращаемого статуса (мастер или реплика). Backend использует тот сервер, который в результате проверки доступности возвращает статус 200 и параметр «master» или «replica» в зависимости от ожидаемого значения в поле «GET». Данная настройка гарантирует фильтрацию входящих запросов между ведущим узлом и репликами PostgreSQL;
7. опция «http-check expect status 200» определяет работоспособность сервера, если отклик на проверку вернулся с опрашиваемого сервера с состоянием HTTP 200;
8. опция «balance» настраивает стратегию распределения нагрузки между серверами группы «backend». В представленном проекте использован алгоритм «Round-Robin», обеспечивающий равномерную последовательную передачу запросов каждому активному серверу группы в циклическом порядке. В зависимости от требований к прокси серверу можно выбрать метод «leastconn» (выбор наименее нагруженного сервера) или «source» (перенаправление клиентов на один и тот же сервер на протяжении сеанса связи).

Проверка функциональности службы после завершения настройки возможна следующими способами:

* тестирование посредством утилиты командной строки PostgreSQL «psql». Для этого выполняется подключение к СУБД по IP адресу прокси-сервера с использованием порта 5000;
* применение стороннего инструмента анализа производительности «Hatester» (не поставляется совместно с сервисом HAProxy).

В проекте, для более эффективного тестирования, был выбран второй способ. Загрузка производится путём копирования файла HAtester.py с веб-сайта «github», после чего файлу назначаются права на исполнение. Далее требуется внести изменения в конфигурационный файл HAtester.py. Следует обновить следующие поля:

* IP-адрес сервера HAProxy;
* название тестируемой базы данных;
* идентификатор пользователя;
* пароль учетной записи пользователя базы данных.

Изменённая для работы с кластером PostgreSQL конфигурация HAtester представлена на рисунке 4.

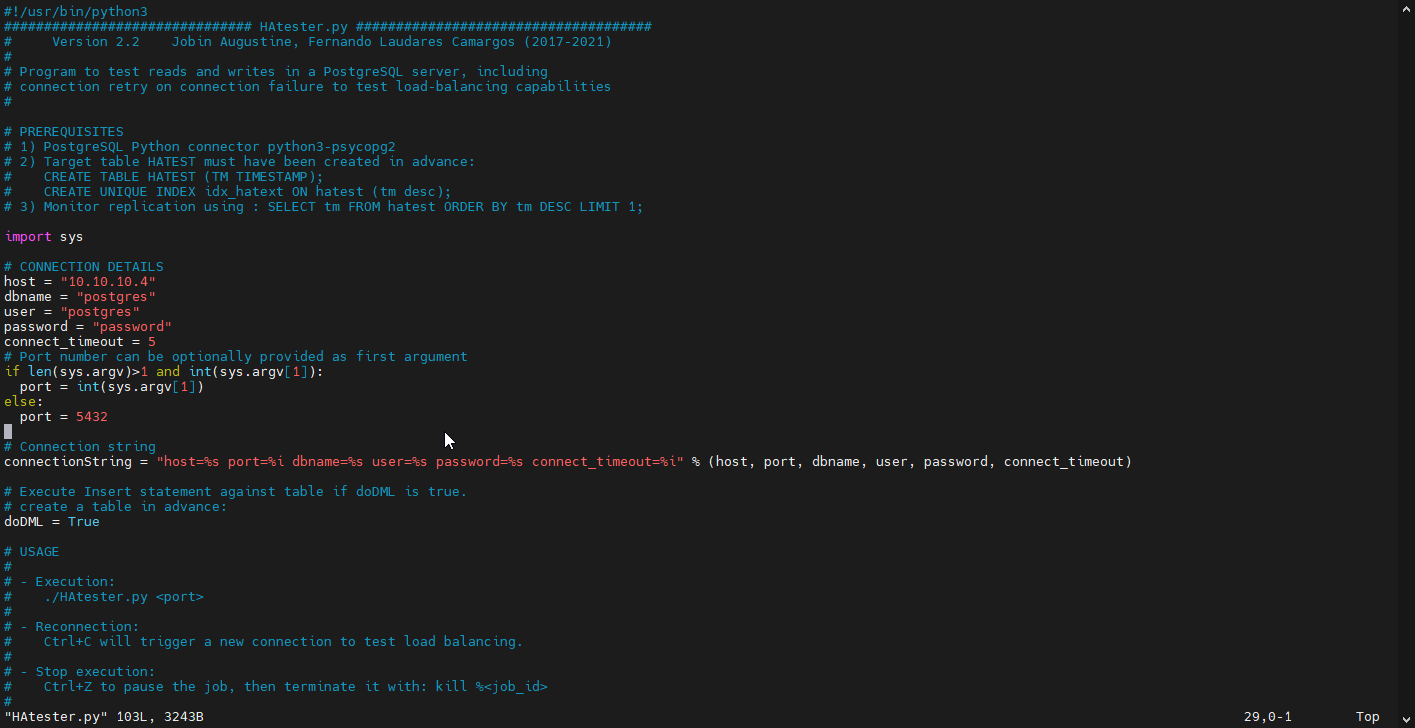


Рисунок 4 – Вид внешний конфигурации HAtester.py

Для запуска скрипта после внесения изменений в конфигурацию применяется команда «./HAtester.py» с указанием порта сервера.

При успешной настройке HAProxy и Hatester, выполнение вышеуказанной команды приведет к следующим результатам (рисунок 3, 4).

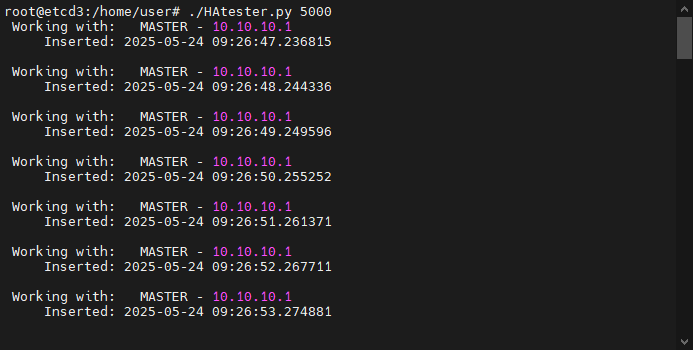


Рисунок 3 – Вид внешний проверки работоспособности HAProxy

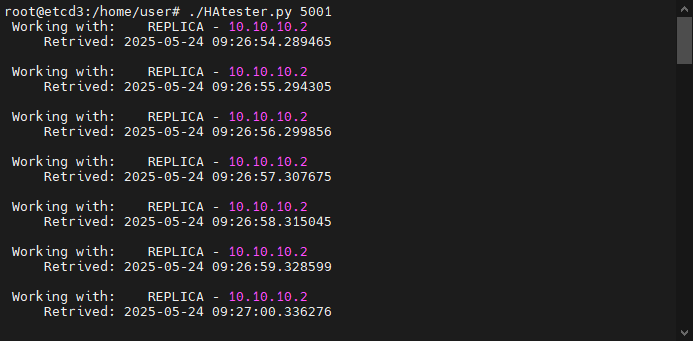


Рисунок 4 – Вид внешний проверки работоспособности HAProxy

Проверить правильное распределение трафика по нодам можно выполнив команду «patronictl». Выполнение указанной команды выводит таблицу, содержащую список всех задействованных хостов в инфраструктуре Patroni с отображением текущих ролей каждого узла относительно структуры кластера (рисунок 5).

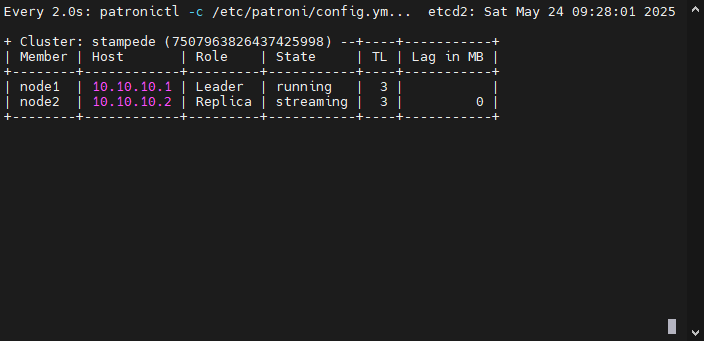


Рисунок 5 – Вид внешний состояния кластера Patroni

Сервис HAProxy дополнительно оснащён графическим web-интерфейсом для наблюдения за состоянием системы и потоками трафика. В представленной конфигурации web-интерфейс был назначен на порт 7000 адреса прокси-сервера. Для дополнительной безопасности при входе предусмотрена процедура авторизации пользователей с учётными данными admin password. администратор получает возможность наблюдать детальную статистику по трафику, проходящему через интерфейсы backend-сервисов HAProxy (рисунок 6).

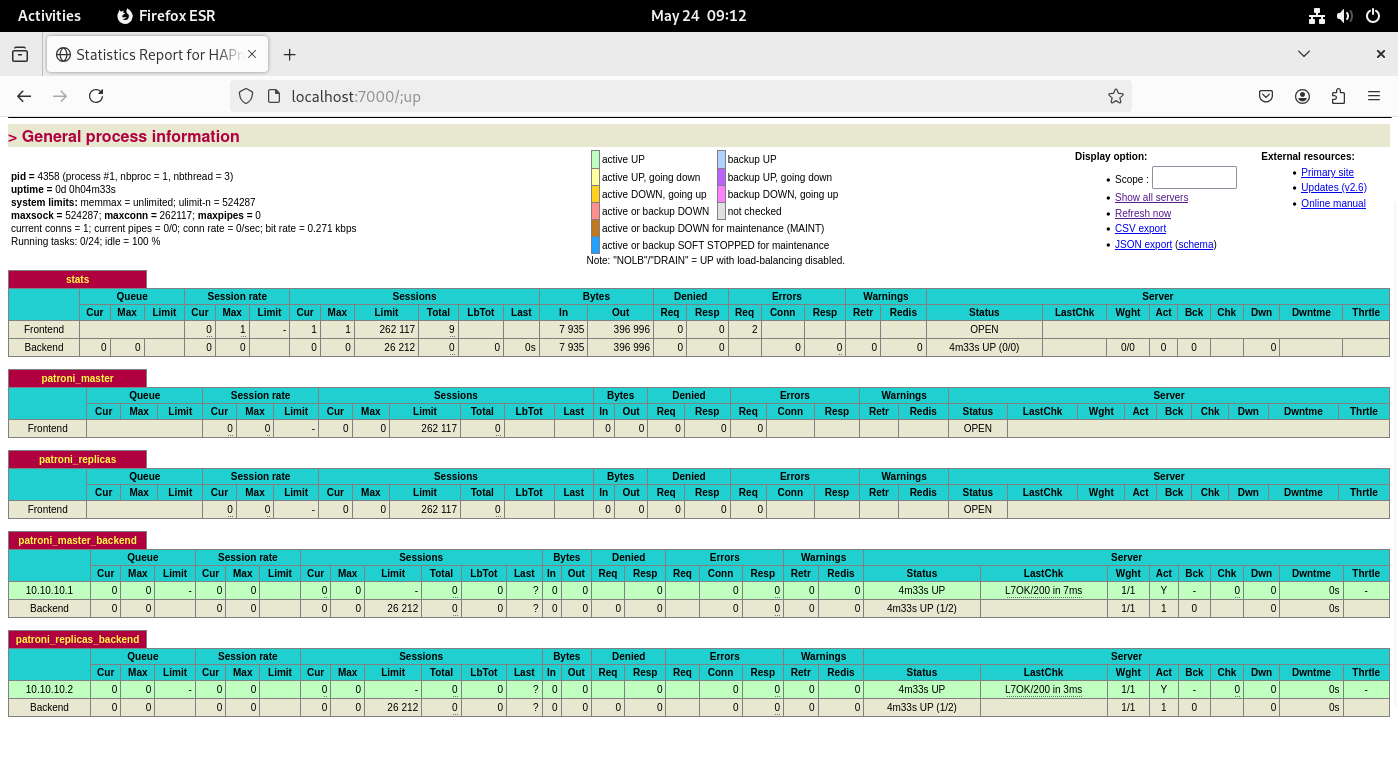


Рисунок 6 – Вид внешний веб-интерфейса HAProxy

2.2 Установка и конфигурация PgBouncer

PgBouncer представляет собой специализированный балансировщик соединений, предназначенный для оптимизации взаимодействия клиентов с базой данных PostgreSQL. Основная цель заключается в снижении числа одновременно используемых соединений с базой данных, что становится критически важным фактором при превышении оптимального лимита активных сессий в СУБД.

В рамках проекта сервис «PgBouncer» применяется для управления соединениями к базе данных в режиме «transaction». Данный режим предоставляет клиенту подключение только на период выполнения обращения к базе данных, после завершения которого, подключение возвращается в пул открытых соединений. Такой подход минимизирует потребление вычислительных ресурсов и память сервера, предоставляя наиболее эффективное использование пропускной способности базы данных, особенно при обработке множества кратковременных транзакций Этот режим показал свою особую эффективность в сценариях высокой загрузки базы данных и широко применяется при проектировании высоконагруженных веб-приложений и решений, использующих облачные технологии.

Установка PgBouncer выполнена стандартным способом через систему управления пакетами APT, применяемую на Debian. Процесс установки охватывает две виртуальных машины (node1 и node2) с незначительными отличиями в настройках на каждой. Завершающим этапом является изменение содержимого файлов конфигурации (листинг А.2).

Конфигурация PgBouncer на виртуальной машине node2 претерпела следующие изменения: значение «host» группы «databases» теперь соответствует значению 10.10.10.2.

В конфигурации были указаны следующие ключевые значения:

1. параметр «listen\_addr = 10.10.10.3,10.10.10.4» отражает готовность PgBouncer принимать соединения от указанных адресов, соответствующих серверам HAProxy;
2. настройка «listen\_port = 6432» указывает на прослушивание указанного порта для приема запросов к PgBouncer;
3. опция «postgres = host=10.10.10.1 port=5432 dbname=postgres» задаёт параметры подключения к целевой базе данных, включая адрес сервера, номер порта и имя базы данных.

Для проверки доступа к базе данных через «PgBouncer» требуется выполнить команду «psql» с указанием порта PgBouncer. На рисунке 7 показан процесс проверки доступа к БД командой без указания пароля (данные для аутентификации сохранены в файле pgpass).

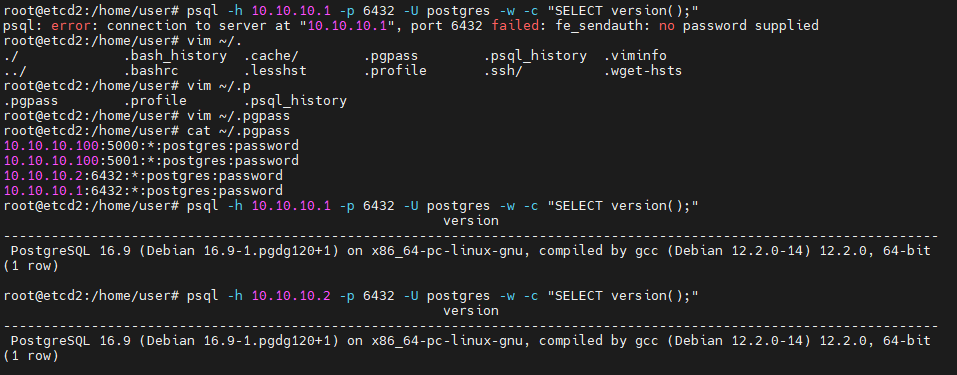


Рисунок 7 – Вид внешний проверки работоспособности Pgbouncer

После успешной настройки PgBouncer следует заменить порты серверов, к которым обращаются backend интерфейсы в конфигурационном файле HAProxy на прокси серверах на порты PgBouncer. Таким образом, вся обработка входящего трафика от клиента будет перенаправляться непосредственно на службу PgBouncer. Сервис HAProxy будет выступать посредником, перенаправляющим весь входящий трафик от клиентов на PgBouncer, обеспечивая эффективную разгрузку базы данных за счёт динамического распределения соединений.

2.3 Установка и конфигурация Keepalived

Сервис Keepalived в кластере PostgreSQL представляет собой механизм обеспечения высокой доступности и бесшовного перехода между основными и резервными нодами в случае аварии или технического обслуживания одной из них. В контексте кластеров PostgreSQL, Keepalived служит следующим целям:

1. обеспечение возможности пользователям взаимодействовать с кластером PostgreSQL через постоянный виртуальный IP-адрес. Таким образом, если один из узлов становится недоступным, пользователи продолжают обращаться к другому активной узлу без изменений в IP адресе;
2. в случае отказа ведущего узла Keepalived, виртуальный IP адрес автоматически перемещается на резервный узел, продолжающий обслуживать запросы вместо ведущего;
3. переключения между нодами практически не влияет на клиентов, поскольку виртуальный IP адрес остается неизменным.

Установка Keepalived производится через APT – стандартный пакетный менеджер Debian на две виртуальные машины node3 и node4 с минимальными различиями в конфигурациях. Установка сопровождается обязательным редактированием соответствующего конфигурационного файла. Конфигурация Keepalived представлена на листинге А.3.

В конфигурации сервиса Keepalived на виртуальной машине node4 значение «priority» группы «vrrp\_instance» было изменено на 99, а значение «state» той же группы на «BACKUP».

В конфигурации были указаны следующие ключевые значения:

1. опция «vrrp\_script haproxy\_check» определяет скрипт, который при обнаружении ошибок в работе сервиса HAProxy, передаёт «virtual\_ipaddress» резервной ноде Keepalived;
2. опция «state (MASTER/BACKUP)» указывает статус ноды Keepalived, ведущая она или резервная;
3. настройка «interface» определяет имя интерфейса, выходящего в сеть;
4. опция «priority» используется для назначения приоритета выделения виртуального IP адреса. В первую очередь IP назначается на узлы с самым высоким приоритетом;
5. поле «virtual\_ipaddress» хранит сведения о виртуальном IP адресе, через который клиенты получают доступ к системе PostgreSQL.

По завершении процесса конфигурации сервис Keepalived мгновенно вступает в рабочий режим, предлагая клиентам единый виртуальный IP-адрес для взаимодействия с HAProxy, а следовательно, и с PostgreSQL. Это делает возможным прозрачный переход между ведущими и резервными узлами, не вызывая простоя для клиента. Для проверки работоспособности Keepalived достаточно вывести команду для просмотра IP адресов. У интерфейса, назначенного в конфигурации выше, будет два IP-адреса: личный и виртуальный (рисунок 8).

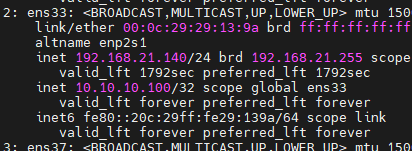


Рисунок 8 – Вид внешний работоспособности Keepalived

3 Настройка мониторинга ресурсов кластера

3.1 Установка и конфигурация платформы для сбора метрик и мониторинга систем Prometheus

Prometheus представляет собой открытое программное решение для сбора метрик и мониторинга систем, первоначально разработанное компанией SoundCloud. Со временем этот проект обрёл популярность среди крупных компаний и организаций, применяющих стратегии контейнеризации и архитектуры микросервисов. Изначально ориентированный на мониторинг среды Kubernetes, сфера применения данного сервиса постепенно была расширена, после чего Prometheus стал универсальным инструментом для отслеживания различных типов сервисов и приложений, в том числе – PostgreSQL.

В данном проекте Prometheus используется для централизованного сбора показателей активности и рабочих характеристик HAProxy, PostgreSQL, и общих метрик потребления аппаратных ресурсов серверов. Также сервис используется для получения настраиваемых уведомлений, для повышения скорости реакции на ошибки и отклонения в работе систем.

Для работы Prometheus требуется установка сервисов экспорта метрик. Чтобы упростить и ускорить процесс развертывания необходимых инструментов, установка сервисов экспортирования метрик Node Exporter и PostgreSQL Exporter была автоматизирована средствами Ansible. Их установка рассмотрена в главе 4.

После установки Prometheus, необходимо настроить конфигурационный файл для обнаружения экспортеров метрик (листинг А.4).

Завершив установку и настройку Prometheus, предоставляется возможность доступа к веб-интерфейсу сервиса через браузер при указании адреса сервера и порта 9090. Подключившись к указанному интерфейсу (рисунок В.1), при условии корректности всей конфигурации, пользователь может оперативно получать следующую информацию:

1. состояние целевых объектов (мониторинг состояний объектов наблюдения, отображающий активные и неактивные компоненты инфраструктуры);
2. метрики (подробные данные о накопленных показателях эффективности, учитывая последние замеры, вплоть до времени последнего сбора и состояния каждой метрики);
3. оповещения (управление и проверка настроенных пользователем предупреждений о событиях в инфраструктуре кластера);
4. запросы (возможность выполнения запросов к метрикам для анализа данных).

3.2 Конфигурация сервиса оповещений на основе Prometheus

Для настройки сервиса оповещений на основе Prometheus сперва требуется обновить конфигурационный файл Prometheus параметрами: «rule\_files» и «alert.rules».

Данная конфигурация определяет файл «alert.rules» содержащим правила уведомлений Prometheus. Предприятием была поставлена задача по установке уведомлений для следующих параметров:

* недоступность сервиса Patroni (node1, node2) в течение 5 минут;
* недоступность сервиса HAProxy (node3, node4) в течение 5 минут;
* загруженность процессора на 80% и больше в течение 5 минут;
* загруженность оперативной памяти на 90% и больше в течение 5 минут;

После указания файла уведомлений следует установка правил в вышеуказанный файл «alert.rules» (листинг А.5).

Данные правила определяют уведомления, приходящие в API Prometheus, в них указаны следующие ключевые фрагменты:

* «expr» (условие отправки уведомления);
* «for» (время, которое должно пройти для отправки уведомления);
* «severity» (степень критичности уведомления. Данная опция определяется вручную, исходя из понимания тяжести ошибки для системы администратором);
* «description» (описание уведомления).

Среди существующих решений для организации системы уведомлений широкое распространение получил сервис Prometheus Alertmanager, предлагающий обширные возможности для кастомизации правил рассылки сообщений о событиях и инцидентах. Среди вариантов доставки уведомлений выделяются электронные письма и интеграция с популярными мессенджерами, такими как Telegram, или уже заблокированный сервис «Discord». Однако выбор системы оповещения не ограничивается одним решением. Если предлагаемый инструментарий отвечает поставленным задачам, возможно применение альтернативных подходов. Например, в рамках этого проекта рассматривается создание собственного Python-бота для направления уведомлений в мессенджер «Telegram» (листинг Б.1).

Данное решение позволяет гибко настроить получаемое сообщение, до получения желаемого вывода уведомления. Исходя из специфики уведомлений Prometheus, была определена следующая концепция: Бот запрашивает API сервиса по адресу «localhost:9090/api/v1/alerts», извлекая JSON файл, содержащий подготовленные заранее предупреждения. Когда система Prometheus регистрирует сбой, обработчик Python-скрипта генерирует уведомление на основе JSON файла и отправляет его конечному пользователю через мессенджер Telegram.

При обнаружении ошибки, в мессенджер «Telegram» отправляется сообщение, следующего вида (рисунок Б.1). В отсутствие ошибок система отображает сообщение о нормальном функционировании инфраструктуры (рисунок Б.2).

В уведомлении об ошибке мы получаем следующие данные:

* чёткое определение причины срабатывания;
* IP адрес сервера, на котором зарегистрирован сбой;
* информация о текущем состоянии инцидента (промежуточный этап – pending, активное событие – firing), позволяющая оценить динамику развития ситуации;
* описание подробностей происшествия;
* время выявления ошибки, необходимое для оценки длительности проблемы.

Эти данные предоставляют достаточный объем информации, позволяющий системному администратору оперативно отреагировать на проблему, предприняв необходимые меры по её устранению. За счет постоянного мониторинга и оперативной системы отправки уведомлений, удаётся минимизировать риск критических сбоев и сохранить стабильность функционирования кластера PostgreSQL.

3.3 Установка и конфигурация инструмента визуализации временных рядов данных Grafana

Grafana представляет собой свободную платформу с открытым исходным кодом для визуализации, анализа и мониторинга различных метрик и журналов системы. Она позволяет создавать интерактивные панели инструментов (далее - дэшборды), графики и отчёты, помогающие анализировать данные в режиме реального времени. Интеграция Grafana с Prometheus представляет собой мощное сочетание инструмента для сбора метрик и визуализационной платформы. Grafana подключается к источнику данных Prometheus и помогает визуально представлять собранные метрики, предоставляя следующие преимущества:

* визуализация данных;
* фильтрация и агрегирование метрик;
* интерактивность и кастомизация;
* удобство восприятия.

Grafana дополняет и усиливает потенциал Prometheus, позволяет превратить необработанные данные в наглядную аналитику, играющую ключевую роль в поддержке непрерывного мониторинга и эффективного управления производительностью информационных систем. Это решение широко используется в современных IT-инфраструктурах для оперативного контроля производительности и своевременного выявления проблем.

Установка сервиса Grafana осуществляется через стандартный менеджер пакетов – APT. Однако предварительно необходимо добавить официальный репозиторий Grafana. Последовательность действий по добавлению репозитория подробно изложена в листинге А.6.

После установки сервиса требуется обновить конфигурационный файл Grafana, конфигурация представлена на листинге 7.

Листинг 7 – Конфигурационный файл Grafana

[server]

http\_port = 3000

[security]

admin\_user = admin

admin\_password = password

[auth.anonymous]

enabled = false

[dashboards]

versions\_to\_keep = 20

Предложенная конфигурация обеспечивает надежную организацию рабочей среды Grafana, предусматривая ряд важных аспектов:

* определение открытого порта, через который возможен последующий доступ к веб-интерфейсу Grafana;
* настройка параметров аутентификации, гарантирующей безопасность доступа к системе;
* создание механизма сохранения предыдущих версий дэшбордов, позволяющего восстанавливать рабочие конфигурации в случае непредвиденных сбоев или неправильных обновлений.

После конфигурации пользователь получает возможность войти в административный интерфейс Grafana, пройдя процедуру аутентификации. Здесь находятся ссылки на ознакомление с документацией, справочными материалами, новостями сообщества и обновлениями блога проекта (рисунок В.2).

Дэшборд представляет из себя специальную среду для представления метрик и данных об использовании ресурсов виртуальными машинами в наглядной форме. Он объединяет различные типы визуализаций (графики, таблицы, диаграммы), объединённые общей темой или задачей. Панели позволяют представить совокупность всех метрик одинаковой группы в компактном и интуитивно-понятном виде. Дэшборды находятся на определённой странице в административной панели Grafana (рисунок В.3).

На начальной стадии эксплуатации Grafana, интерфейс панели мониторинга пуст, не содержит ни одного дэшборда. Существует несколько методов добавления панелей мониторинга в систему:

1. создание дэшбордов вручную (самостоятельная разработка панелей мониторинга, учитывающих уникальные требования бизнеса и особенности имеющейся инфраструктуры);
2. импорт готовых конфигураций (использование шаблонов из сообщества пользователей Grafana. Такие панели охватывает большинство потребностей любого предприятия и обеспечивают достаточный функционал без необходимости переработки).

Для импорта дэшбордов требуется зайти на соответствующую страницу (рисунок В.4), после чего ввести ID панели и нажать кнопку «Load». Идентификатор дэшборда Grafana можно получить из адресной строки. Пример: идентификатором панели находящейся по адресу «https://grafana.com/grafana/dashboards/1860-node-exporter-full/» будет 1860.

После тщательного отбора, для мониторинга были выбраны и подготовлены к импорту с серверов Grafana следующие интерактивные панели инструментов:

* Node Exporter Full (рисунок В.5);
* HAProxy Servers | HAProxy (рисунок В.6);
* PostgreSQL Database (рисунок В.7).

После импорта дэшбордов, они становятся доступными в разделе панели управления Grafana. На этом этапе появляется возможность отслеживать состояния сервисов HAProxy и PostgreSQL, а также общее потребление ресурсов серверами кластера. Функционал Grafana позволяет как наблюдение за каждым отдельным узлом кластера, так и мониторинг нескольких серверов одновременно, что даёт целостное представление о работе всей инфраструктуры и позволяет быстрее выявить возможные проблемы.

4 Автоматизация развёртывания средств сбора метрик и мониторинга кластера

Поскольку развертывание средств сбора метрик и мониторинга кластера характеризуется многократным выполнением одних и тех же шагов, требуется автоматизация данного процесса. Это актуально как для масштабирования существующей инфраструктуры, так и для расширения охвата новыми машинами сотрудников или виртуальными машинами.

В рамках проекта был разработан специальный конфигурационный модуль для инструмента Ansible. Эта структура предусматривает задание перечня серверов или виртуальных машин в специальном инвентаре (inventory.ini), разделение устройств по ролям и запуск сценария развертывания, охватывающего всю инфраструктуру [].

Структура проекта автоматизированного развёртывания системы мониторинга кластера указана на рисунке 15.

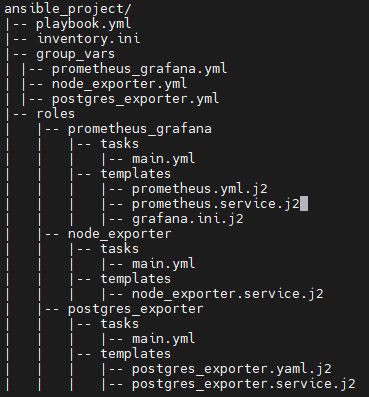


Рисунок 15 - Структура проекта автоматизированного развёртывания системы мониторинга кластера

* 1. Определение инвентаря и плейбуков сервиса Anisble

Инвентарь Ansible представляет собой файл, служащий для определения набора управляемых хостов и групп, составляющих инфраструктуру кластера. Основное назначение этого файла состоит в организации и структурировании серверов, принадлежащих проекту, посредством группировки и назначения специфичных переменных для каждого отдельного устройства или группы устройств (листинг 8).

Листинг 8 – файл инвентаря «inventory.ini»

[node\_exporter]

node1 ansible\_host=10.10.10.1 ansible\_user=user ansible\_ssh\_pass=4789

node2 ansible\_host=10.10.10.2 ansible\_user=user ansible\_ssh\_pass=4789

node3 ansible\_host=10.10.10.3 ansible\_user=user ansible\_ssh\_pass=4789

node4 ansible\_host=10.10.10.4 ansible\_user=user ansible\_ssh\_pass=4789

[postgres\_exporter]

node1 ansible\_host=10.10.10.1 ansible\_user=user ansible\_ssh\_pass=4789

node2 ansible\_host=10.10.10.2 ansible\_user=user ansible\_ssh\_pass=4789

[prometheus\_grafana]

prometheus\_grafana1 ansible\_host=10.10.10.3 ansible\_user=user ansible\_ssh\_pass=4789

Инвентарь Ansible организован таким образом, чтобы чётко структурировать различные категории серверов в соответствии с исполняемой ими функцией в инфраструктуре. Представленная конфигурация выделяет три главные группы, каждая из которых предназначена для установки и настройки определенного типа сервисов:

* node exporter;
* postgresql exporter;
* prometheus + grafana.

Такая организация инвентаря Ansible обеспечивает прозрачность и простоту управления большими группами серверов, позволяя легко управлять инфраструктурой, снижая риски ошибок и увеличивая производительность администрирования.

Плейбук является набором инструкций, описывающих последовательность шагов для автоматической настройки и развертывания программного обеспечения, приложений или инфраструктуры на удалённых серверах. Чтобы сделать плейбук модульным, переиспользуемым и легко поддерживаемым, в нём были определены роли под установку каждого сервиса (листинг 9).

Листинг 9 – конфигурационный файл главного плейбука

---

- name: Установка экспортера общих метрик

hosts: node\_exporter

become: true

roles:

- node\_exporter

- name: Установка экспортера метрик HAProxy

hosts: haproxy\_exporter

become: true

roles:

- haproxy\_exporter

- name: Установка экспортера метрик PostgreSQL

hosts: postgres\_exporter

become: true

roles:

- postgres\_exporter

- name: Установка сервиса мониторинга

hosts: prometheus\_grafana

become: true

roles:

- prometheus\_grafana

Роли в сервисе «Ansible» объединяют несколько компонентов: задачи, шаблоны, переменные и зависимости между ними. Использование ролей особенно полезно, когда вам нужно многократно применять одни и те же наборы изменений на разных серверах или группах хостов. Вместо громоздких плейбуков с большим количеством повторяющихся задач, при применении ролей получаются небольшие, независимые роли, которые потом легко подключаются к основным сценариям.

4.2 Конфигурация роли для установки экспортеров метрик

Первая роль, определённая в плейбуке выше – node exporter. С её помощью сервис устанавливает экспортеры общих метрик (данные о загруженности дисков, оперативной памяти или процессора) для узлов кластера. Конфигурация файла задач представлена на листинге 10.

Листинг 10 – конфигурационный файл плейбука node\_exporter

- name: Create node\_exporter system user

ansible.builtin.user:

name: node\_exporter

system: true

shell: /sbin/nologin

create\_home: false

- name: Download and install node\_exporter

block:

- name: Download node\_exporter

ansible.builtin.get\_url:

url: "https://github.com/prometheus/node\_exporter/releases/download/v{{ node\_exporter\_version }}/\

node\_exporter-{{ node\_exporter\_version }}.{{ arch }}.tar.gz"

dest: /tmp/node\_exporter.tar.gz

mode: '0644'

- name: Extract node\_exporter

ansible.builtin.unarchive:

src: /tmp/node\_exporter.tar.gz

dest: /tmp

remote\_src: true

- name: Copy node\_exporter binary

ansible.builtin.copy:

src: "/tmp/node\_exporter-{{ node\_exporter\_version }}.{{ arch }}/node\_exporter"

dest: "/usr/local/bin/node\_exporter"

mode: '0755'

remote\_src: true

- name: Create systemd unit for node\_exporter

ansible.builtin.template:

src: node\_exporter.service.j2

dest: /etc/systemd/system/node\_exporter.service

mode: '0644'

- name: Start and enable exporters

ansible.builtin.systemd:

name: "{{ item }}"

state: started

enabled: true

daemon\_reload: true

with\_items:

- node\_exporter

- name: Check node\_exporter metrics endpoint

ansible.builtin.uri:

url: "http://localhost:{{ node\_exporter\_port }}/metrics"

status\_code: 200

return\_content: true

register: node\_exporter\_metrics

- name: Validate node\_exporter metrics

ansible.builtin.debug:

msg: "Node exporter metrics collected successfully: {{ node\_exporter\_metrics.content | regex\_search('node\_cpu\_seconds\_total') is not none }}"

Также к каждому файлу задач приложены один или больше шаблонов настроек устанавливаемых сервисов, в листинге выше есть один такой шаблон, под названием «node\_exporter.service.j2» (листинг B.1)

После установки экспортеров общей информации, идёт процесс установки экспортеров данных PostgreSQL (листинг 11)

Листинг 11 – конфигурационный файл плейбука postgres\_exporter

- name: Download and install postgres\_exporter

block:

- name: Download postgres\_exporter

ansible.builtin.get\_url:

url: "https://github.com/prometheus-community/postgres\_exporter/releases/download/\

v{{ postgres\_exporter\_version }}/postgres\_exporter-{{ postgres\_exporter\_version }}.{{ arch }}.tar.gz"

dest: /tmp/postgres\_exporter.tar.gz

mode: '0644'

- name: Extract postgres\_exporter

ansible.builtin.unarchive:

src: /tmp/postgres\_exporter.tar.gz

dest: /tmp

remote\_src: true

- name: Copy postgres\_exporter binary

ansible.builtin.copy:

src: "/tmp/postgres\_exporter-{{ postgres\_exporter\_version }}.{{ arch }}/postgres\_exporter"

dest: "/usr/local/bin/postgres\_exporter"

mode: '0755'

remote\_src: true

- name: Ensure postgres\_exporter config directory exists

ansible.builtin.file:

path: /etc/postgres\_exporter

state: directory

mode: '0755'

owner: postgres

group: postgres

- name: Create postgres\_exporter config

ansible.builtin.template:

src: postgres\_exporter.yaml.j2

dest: /etc/postgres\_exporter/postgres\_exporter.yaml

mode: '0640'

owner: postgres

group: postgres

- name: Create systemd unit for postgres\_exporter

ansible.builtin.template:

src: postgres\_exporter.service.j2

dest: /etc/systemd/system/postgres\_exporter.service

mode: '0644'

- name: Start and enable exporters

ansible.builtin.systemd:

name: "{{ item }}"

state: started

enabled: true

daemon\_reload: true

with\_items:

- postgres\_exporter

- name: Check postgres\_exporter metrics endpoint

ansible.builtin.uri:

url: "http://localhost:{{ postgres\_exporter\_port }}/metrics"

status\_code: 200

return\_content: true

register: postgres\_exporter\_metrics

- name: Validate postgres\_exporter metrics

ansible.builtin.debug:

msg: "Postgres exporter metrics collected successfully: {{ postgres\_exporter\_metrics.content | regex\_search('pg\_database\_size') is not none }}"

Файлы шаблонов для экспортеров метрик PostgreSQL (листинги В.2, В.3) в листинге выше определены как «postgres\_exporter.yaml.j2» (шаблон конфигурации экспортеров) и «postgres\_exporter.service.j2» (шаблон файла сервиса экспорта PostgreSQL).

4.3 Конфигурация роли для установки мониторинга

Последней ролью в плейбуке является установка систем мониторинга. Данные системы были определены в плейбук из соображений будущего расширения нод наблюдения за кластером. Файл задачи установки Prometheus и конфигурации Grafana указан в листинге 12.

Листинг 12 – конфигурационный файл prometheus-grafana

- name: Create prometheus user

ansible.builtin.user:

name: prometheus

system: true

shell: /sbin/nologin

- name: Create prometheus directories

ansible.builtin.file:

path: "{{ item }}"

state: directory

owner: prometheus

group: prometheus

mode: '0755'

with\_items:

- "/var/lib/prometheus"

- "/etc/prometheus"

- name: Download Prometheus

ansible.builtin.get\_url:

url: "https://github.com/prometheus/prometheus/releases/download/v3.4.1/\

prometheus-3.4.1.linux-{{ prometheus\_arch }}.tar.gz"

dest: /tmp/prometheus.tar.gz

mode: '0644'

- name: Extract Prometheus

ansible.builtin.unarchive:

src: /tmp/prometheus.tar.gz

dest: /tmp

remote\_src: true

- name: Copy Prometheus binaries

ansible.builtin.copy:

src: "/tmp/prometheus-3.4.1.linux-{{ prometheus\_arch }}/{{ item }}"

dest: "/usr/local/bin/"

mode: '0755'

remote\_src: true

with\_items:

- prometheus

- promtool

- name: Verify prometheus binary after installation

ansible.builtin.command: /usr/local/bin/prometheus --version

register: prometheus\_version\_check

changed\_when: false

ignore\_errors: true

- name: Display prometheus version

ansible.builtin.debug:

var: prometheus\_version\_check.stdout\_lines

when: prometheus\_version\_check.rc == 0

- name: Fail if prometheus binary check failed

ansible.builtin.fail:

msg: "Prometheus binary verification failed"

when: prometheus\_version\_check.rc != 0

- name: Create Prometheus config

ansible.builtin.template:

src: prometheus.yml.j2

dest: "/etc/prometheus/prometheus.yml"

owner: prometheus

group: prometheus

mode: '0644'

- name: Create Prometheus systemd service

ansible.builtin.template:

src: prometheus.service.j2

dest: /etc/systemd/system/prometheus.service

mode: '0644'

- name: Configure Grafana

ansible.builtin.template:

src: grafana.ini.j2

dest: /etc/grafana/grafana.ini

mode: '0644'

- name: Start and enable services

ansible.builtin.systemd:

name: "{{ item }}"

state: restarted

enabled: true

daemon\_reload: true

with\_items:

- prometheus

- grafana-server

Файлы шаблонов конфигураций в данной роли (листинги Г.1, Г.2, Г.3) определены как «prometheus.yml.j2» (конфигурация Prometheus), «prometheus.service.j2» (сервис Prometheus) и «grafana.ini.j2» (Конфигурация Grafana).

6 Технико-экономическое обоснование модернизации PostgreSQL кластера

Дипломный проект направлен на модернизацию для PostgreSQL кластера на базе сервиса patroni.

Для подготовки экономического обоснования модернизации необходимо выполнить расчет трудозатрат и финансовых вложений, затраченных на реализацию проекта.

, (1)

где – затраты на оплату машинного времени, руб.;

– общие затраты, руб.

Трудоёмкость разработки приложения , чел.ч, определяется по формуле

, (2)

где – затраты труда на подготовку описания задачи, чел.ч;

– затраты труда на исследование алгоритма решения задачи, чел.ч;

– затраты труда на разработку алгоритма, чел.ч;

– затраты труда на разработку диаграмм алгоритма, чел.ч;

– затраты труда на программирование по готовой диаграмме, чел.ч;

– затраты труда на отладку программы ЭВМ, чел.ч;

– затраты труда на подготовку документации, чел.ч.

Составление затрат вычисляется при помощи условного числа операторов. Условное число операторов , ед, в программе определяется по формуле

, (3)

где q – число операторов (исходных команд), ед;

с – коэффициент, учитывающий новизну и сложность программы;

р – коэффициент коррекции программы в ходе разработки, зависит от точности и корректности поставленной задачи (0.05-0.10).

В разработанной программе число операторов примем равным 400 (q=400).

Коэффициент, учитывающий новизну и сложность программы, определяется на пересечении групп сложности и степени новизны.

Работа по степени новизны относится к одной из четырёх групп:

1. группа А – разработка принципиально новых задач;
2. группа Б – разработка оригинальных программ;
3. группа В – разработка программ с использованием типовых решений;
4. группа Г – разовая типовая задача.

По степени сложности работа относится к одной из трёх групп:

1. алгоритмы оптимизации и моделирования систем;
2. задачи учёта, отчётности и статистики;
3. стандартные алгоритмы;

В таблице Е.1 представлены значения коэффициентов c и B.

Модернизация кластера по степени новизны относится к разработке программ с использованием типовых решений (группа В), а по степени сложности алгоритма – алгоритмы оптимизации и моделирования систем (группа 2). Из таблицы Д.1 следует, что коэффициент c=1,15, коэффициент B=1,20.

С учётом того, что задача была поставлена достаточно чётко, коэффициент p принимается равным 0,07 (p=0,07).

Условное число операторов, согласно формуле (3), составляет

Затраты труда на подготовку описания задачи , чел.ч, точно определить невозможно, т.к. это связано с творческим характером работы. С учетом этого можно принять данное значение равным 47 чел.ч ().

Затраты труда на изучение описания задачи с учётом уточнения описания и квалификации системного администратора , чел.ч, определяются по формуле

, (4)

где В – коэффициент увеличения затрат труда вследствие недостаточного описания задачи, уточнений и некоторой недоработки;

К – коэффициент квалификации работника.

В таблице Е.2 представлены коэффициенты квалификации работника. По таблице, для работающих до двух лет, коэффициент К = 0,80.

Затраты труда на разработку алгоритма решения задачи , чел.ч, определяются по формуле

, (5)

Затраты труда на разработку диаграмм решения задачи , чел.ч, определяются по формуле

, (6)

Затраты труда на составление программы по готовой диаграмме , чел.ч, определяются по формуле

, (7)

Затраты труда на отладку программы на ЭВМ при комплексной отладке , чел.ч, определяются по формуле

, (8)

где – затраты труда на отладку программы на ЭВМ при автономной отладке одной задачи, чел.ч.

Затраты труда на отладку программы на ЭВМ при автономной отладке одной задачи , чел.ч, определяются по формуле

, (9)

Далее требуется рассчитать затраты труда на отладку программы на ЭВМ при комплексной отладке по формуле (8)

Затраты труда на подготовку документации по задаче , чел.ч, определяются по формуле

, (10)

где – затраты труда на подготовку материалов рукописи, чел.ч;

– затраты на редактирование, печать и оформление документации, чел.ч.

Затраты труда на подготовку материалов рукописи , чел.ч, определяются по формуле

, (11)

Затраты на редактирование, печать и оформление документации , чел.ч, определяются по формуле

, (12)

Далее требуется рассчитать затраты труда на подготовку документации по задаче по формуле (10)

Трудоёмкость модернизации кластера, согласно формуле (2), составляет

При пятидневной рабочей неделе и недельной норме в 40 часов это составляет 13 рабочих дней.

Таким образом, рассчитав все трудозатраты, определили общую трудоемкость проекта, то есть время, затраченное разработчиком на создание программного продукта. Для определения структуры трудоемкости необходимо построить диаграмму трудозатрат (рисунок Е.1).

Максимальная доля трудозатрат (47%) приходится на этап подготовки описания задачи.

Затраты на оплату машинного времени при отладке программы , руб., определяются по формуле

, (13)

где – цена машино–часа арендного времени, руб/ч;

– фактическое время отладки программы на ЭВМ, чел.ч.

Фактическое время отладки , чел.ч, определяется по формуле:

, (14)

Цена машино-часа , руб/ч, определяется по формуле

, (15)

где – действительный месячный фонд времени ЭВМ, ч.

Действительный месячный фонд времени ЭВМ , ч, определяется по формуле

, (16)

где – общее количество дней в период написания проекта;

– количество праздничных и выходных дней в месяце;

– время простоя в профилактических работах, ч.

Общее количество дней = 28, число праздничных и выходных дней = 12 [произв календарь].

Время простоя в профилактических работах определяется как еженедельная профилактика по 4 часа.

Далее требуется рассчитать действительный месячный фонд времени ЭВМ, рассчитывается по формуле (16)

Затраты на эксплуатацию ЭВМ , руб., определяются по формуле

, (17)

где – издержки на амортизацию, руб.;

– издержки на электроэнергию, потребляемую ЭВМ, руб..

Компьютер, на котором выполнена разработка программы, приобретён по рыночной цене руб. С учётом того, что рыночная цена компьютера менее 100 тыс. руб., компьютер не является амортизируемым имуществом в соответствии со ст.256-257 НК РФ ч. 2 [1], следовательно = 0 руб.

, (18)

где – суммарная мощность ЭВМ, кВт;

– стоимость 1 кВт∙ч электроэнергии, руб.

Согласно техническому паспорту ЭВМ, потребление электроэнергии составляет 0,63 кВт. Стоимость электроэнергии в г. Архангельске, где проходила разработка, составляет 6,59 руб/кВт·ч [5].

С учетом этого стоимость электроэнергии, потребляемой за месяц, по формуле (18)

Затраты на эксплуатацию ЭВМ, согласно формуле (17), составляют

Цена машино-часа, согласно формуле (15), составляет

Затраты на оплату машинного времени при отладке программы, согласно формуле (13), составляют

Общие затраты , руб., определяются по формуле

(19)

где – издержки на заработную плату, руб.;

– издержки на отчисления в страховые взносы, руб.;

– издержки на прочие и накладные расходы, руб.

Заработная плата работников , руб., определяется по формуле

, (20)

где – коэффициент, учитывающий северную надбавку для работающих в местах, приравненных к Крайнему Северу;

– коэффициент, учитывающий районную надбавку на территориях, приравненных к Крайнему Северу.

Районный коэффициент в г. Архангельске составляет 20% от основной заработной платы, а выплаты за выслугу лет, проработанных на территории, приравненной к территории Крайнего Севера – 50% от основной заработной платы, в соответствии с законом «О государственных гарантиях и компенсациях для лиц, работающих и проживающих в районах Крайнего Севера и приравненных к ним местностях», Федеральный закон № 4520-1 [2], [4].

Примем оклад системного администратора равным 22440,00 руб [3].

Месячная заработная плата работников по формуле (20) составляет

В связи с тем, что на предприятии было проведено 13 дней, а заработная плата считается за рабочий месяц с количеством 21-23 рабочих дня, [произв календарь] фактическая заработная плата сотрудника в период создания дипломного проекта рассчитывается путём деления заработной платы на усреднённое значение 22 рабочих дня и умножается на количество фактически отработанных на предприятии дней (по фактически отработанным дням Ззпф) Фактическая заработная плата работников по формуле (20) составляет

Страховые взносы с оплаты труда , руб., определяются по формуле

, (21)

где – фонд оплаты труда, руб.;

– размер страховых взносов с оплаты труда, %.

В силу того, что число работников соответствует одному, то можно принять равным (=), а составляют 30% от суммы заработной платы без учета вносов на травматизм в соответствии со ст. 425 НК РФ ч. 2 [1].

Страховые взносы с оплаты труда по формуле (21) составляют

Прочие затраты , руб., принимаются в размере 10% в общей сумме затрат и определяются по формуле

(22)

Общие затраты, согласно формуле (19), составляют

На основании данных о затратах построена диаграмма затрат (рисунок Е.2).

Основная часть затрат приходится на оплату труда – 68,70%, что свидетельствует о высокой трудоемкости процесса разработки приложения.

Затраты на разработку приложения, согласно формуле (1), составляют

В процессе модернизации кластера могут возникнуть различные риски, способные негативно повлиять на ход работы. Их своевременная идентификация позволяет системному администратору выявить потенциальные слабые места и уязвимости проекта. Это, в свою очередь, дает возможность внести необходимые корректировки и улучшения, повышая тем самым качество и надежность конечного продукта.

Наиболее значимые риски определены в таблице Е.3 и пути их минимизации, не увеличивающие стоимость проекта, установлены.

Наиболее имеющие значение риски определены и пути их минимизации установлены. Действия по минимизации рисков не приведут к повышению денежных затрат.

В результате произведенных расчетов трудоемкость создания ИС составила 100,91 чел. ч., издержки на создание данного программного продукта составили 33192,15 руб.

Приведённая в проекте модернизация кластера PostgreSQL принесёт предприятию следующие преимущества:

* увеличенная доступность кластера (клиенты смогут подключаться к СУБД по единой точке входа);
* улучшение контроля и аналитики (настроенная система мониторинга позволяет собирать и анализировать данные об использовании кластером ресурсов сервера за разные временные промежутки);
* повышение качества и скорости обслуживания (система уведомлений позволяет быстро получать информацию о сбоях в кластере и ускоряет исправление ошибок);
* снижение затрат (автоматизация и цифровизация процессов уменьшит расходы на бумажную документацию и сократит необходимость в дополнительном персонале);
* повышение безопасности данных (запрос к СУБД проходит через два сервиса балансировки нагрузки, сервис «PgBouncer» не позволяет открыть множество соединений к бд одновременно, что сильно снижает вероятность кибер атак типа «DOS»).

7 Техника безопасности при работе с ПК

Техника безопасности сформирована в соответствии с документом санитарные правила СП 2.4.3648-20 «Санитарно-эпидемиологические требования к организациям воспитания и обучения, отдыха и оздоровления детей и молодежи» от 28 сентября 2020 года N 28.

7.1 Общие требования безопасности

К работе на персональном компьютере допускаются лица, прошедшие обучение безопасным методам труда, вводный инструктаж, первичный инструктаж на рабочем месте. При эксплуатации персонального компьютера на работника могут оказывать действие следующие опасные и вредные производственные факторы:

* повышенный уровень электромагнитных излучений;
* повышенный уровень статического электричества;
* пониженная ионизация воздуха;
* статические физические перегрузки;
* перенапряжение зрительных анализаторов.

Работник обязан:

* выполнять только ту работу, которая определена его должностной инструкцией;
* содержать в чистоте рабочее место;
* соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности;
* соблюдать меры пожарной безопасности.

7.2 Требования охраны труда перед началом работы

Перед началом работ необходимо:

* подготовить рабочее место;
* отрегулировать освещение на рабочем месте;
* проверить правильность подключения оборудования к электросети;
* проверить исправность проводов питания и отсутствие оголенных участков проводов;
* проверить правильность установки стола, стула, подставки для ног, пюпитра, угла наклона экрана, положение клавиатуры, положение "мыши" на специальном коврике, при необходимости произвести регулировку рабочего стола и кресла, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительных напряжений тела.

7.3 Требования охраны труда во время работы

Работнику при работе на ПК запрещается:

* прикасаться к задней панели системного блока (процессора) при включенном питании;
* переключать разъемы интерфейсных кабелей периферийных устройств при включенном питании;
* допускать попадание влаги на поверхность системного блока (процессора), монитора, рабочую поверхность клавиатуры и других устройств;
* производить самостоятельное вскрытие и ремонт оборудования;
* работать на компьютере при снятых кожухах;
* отключать оборудование от электросети и выдергивать электровилку, держась за шнур.

Продолжительность непрерывной работы с компьютером без регламентированного перерыва не должна превышать двух часов. Во время регламентированных перерывов с целью снижения нервно - эмоционального напряжения, утомления зрительного анализатора, устранения влияния гиподинамии и гипокинезии, предотвращения развития познотонического утомления выполнять комплексы упражнений.

7.4 Требования охраны труда в аварийных ситуациях

В случае аварийной ситуации необходимо:

* во всех случаях обрыва проводов питания, неисправности заземления и других повреждений, появления гари, немедленно отключить питание и сообщить об аварийной ситуации руководителю;
* не приступать к работе до устранения неисправностей;
* при получении травм или внезапном заболевании немедленно известить своего руководителя, организовать первую доврачебную помощь или вызвать скорую медицинскую помощь.

7.5 Требования безопасности по окончании работы

По окончании работы необходимо:

* отключить питание компьютера;
* привести в порядок рабочее место.

ЗАКЛЮЧЕНИЕ

В ходе проекта был проведен детальный анализ состояния инфраструктуры базы данных предприятия, выявлены ключевые характеристики и сформулированы необходимые требования к модернизации. Проведён выбор и обоснование схемы работы с базами данных, разработка и реализация всех необходимых компонентов и сервисов, таких как: Pgbouncer, HAProxy, Keepalived, Prometheus и Grafana. Для обеспечения процесса доставки уведомлений о сбоях в инфраструктуре было выполнено написание бота на высокоуровневом языке программирования «Python», с использьзованием Prometheus API.

Результатом дипломного проекта стал полностью высокоэффективный и надежный кластер PostgreSQL, удовлетворяющий требованиям предприятия на предприятии ООО «ТГК-2 Энергосбыт». В рамках этой работы был проанализирован существующий кластер базы данных на предприятии, а также определены основные требования и особенности, предъявляемые к базе данных и последующая модернизация. Был создан высокопроизводительный и отказоустойчивый кластер PostgreSQL, способный удовлетворить потребности предприятия в постоянном доступе к хранимой информации, обеспечении её резервного копирования и отказоустойчивости.

В проекте соблюдены следующие требования:

* настроена высокая устойчивость к сбоям оборудования;
* обеспечена непрерывность доступа к СУБД при условии работоспособности хотя бы одного узла кластера;
* учтена возможность горизонтального расширения инфраструктуры;
* увеличена скорость обработки запросов посредством балансировки нагрузок между серверами;
* обеспечено централизованное управление инфраструктурой и облегчённое администрирования кластера;
* инфраструктура обеспечена сервисами мониторинга и уведомлений;
* настроена единая точка входа в кластер посредством сервиса VIP.

Каждая из этих задач была направлена на создание надёжной системы, способной улучшить работоспособность кластера PostgreSQL в условиях высоких нагрузок и возможных сбоев.

Внедрённое решение способствует:

* снижению времени, затрачиваемого на ручной мониторинг кластера;
* улучшению пользовательского опыта при использовании средств кластеризации PostgreSQL;
* оптимизации использования серверных ресурсов за счет перераспределения мощностей между нодами кластера.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Российская Федерация. Законы. Налоговый Кодекс Российской Федерации. Часть вторая : Федеральный закон № 117-ФЗ : [принят Государственной думой 19 июля 2000 года : одобрен Советом Федерации 26 июля 2000 года]. – Москва : ЦЕНТРМАГ , 2025. – 256-257 с.
2. Российская Федерация. Законы. Налоговый Кодекс Российской Федерации. Часть вторая : Федеральный закон № 117-ФЗ : [принят Государственной думой 19 июля 2000 года : одобрен Советом Федерации 26 июля 2000 года]. – Москва : ЦЕНТРМАГ , 2025. – 425 с.
3. Российская Федерация. Законы. Трудовой Кодекс Российской Федерации : Федеральный закон № 197-ФЗ : [принят Государственной думой 21 декабря 2001 года : одобрен Советом Федерации 26 декабря 2001 года]. – Москва : Проспект , 2025. – 133 с.
4. Российская Федерация. Законы. Трудовой Кодекс Российской Федерации : Федеральный закон № 197-ФЗ : [принят Государственной думой 21 декабря 2001 года : одобрен Советом Федерации 12 декабря 2001 года]. – Москва : Проспект , 2025. – 316 с.
5. Российская Федерация. Законы. Об электроэнергетике. Российской Федерации : Федеральный закон № 35-ФЗ : [принят Государственной думой 21 февраля 2003 года : одобрен Советом Федерации 12 марта 2003 года]. Москва : Норматика , 2025. – 24 с.
6. ГОСТ Р 53246-2019. Информационные технологии. Системы кабельные структурированные. Проектирование основных узлов системы: дата введения 2010-10-10. – Москва: Стандартпроект, 2017.
7. ГОСТ Р 59502-2021 Единая система условных обозначений в области информационно-телекоммуникационных систем: дата введения 2021-11-30. – Москва: Стандартинформ, 2021.
8. ТГК-2 Энергосбыт : [сайт] – Архангельск, 2025. – Обновляется ежедневно. – URL: https://tgc2-energo.ru (дата обращения: 24.05.2025). Текст : электронный.
9. Etcd : [сайт]. – Роли, 2025. – Обновляется ежемесячно. – URL: https://etcd.io (дата обращения: 24.05.2025). – Текст : электронный.
10. Создание кластера баз данных. – Текст : электронный // Postgres Professional [сайт]. – 2023. – URL: https://postgrespro.ru/docs/postgrespro/9.5/creating-cluster (дата обращения: 24.05.2025).
11. Patroni : Introduction : [сайт] – Германия, 2025. – Обновляется ежемесячно. – URL: https://patroni.readthedocs.io/en/latest (дата обращения: 24.05.2025). – Текст : электронный.
12. Minimum requirements to set up a highly available PostgreSQL cluster with Patroni. – Текст : электронный // Percona community forum [сайт]. – 2024. – URL: https://forums.percona.com/t/minimum-requirements-to-set-up-a-highly-available-postgresql-cluster-with-patroni/30212 (дата обращения: 24.05.2025).
13. How HAProxy works. – Текст : электронный // HAProxy Documentation [сайт]. – 2025. – URL: https://docs.haproxy.org/3.2/intro.html#3.2 (дата обращения: 24.05.2025).
14. Высокодоступные кластеры PostgreSQL с Patroni: Ваша очередь протестировать сценарии отказов. – Текст : электронный // Хабр. – 2025. – URL: https://habr.com/ru/companies/otus/articles/753294 (дата обращения: 24.05.2025).
15. Руководство по использованию PGBouncer. – Текст : электронный // Postgres Professional [сайт]. – 2025. – URL: https://postgrespro.ru/docs/postgrespro/10/pgbouncer (дата обращения: 24.05.2025).
16. Configuration. – Текст : электронный // PGBouncer [сайт]. – 2025. – URL: https://www.pgbouncer.org/config.html (дата обращения: 24.05.2025).
17. Инструкции по настройке Keepalived и VIP. – Текст : электронный // Selectel [сайт]. – 2025. – URL: https://selectel.ru/blog/tutorials/what-is-keepalived-and-vip (дата обращения: 24.05.2025).
18. Getting started. – Текст : электронный // Prometheus [сайт]. – 2025. – URL: https://prometheus.io/docs/prometheus/latest/getting\_started (дата обращения: 24.05.2025).
19. Prometheus Alerting 101: Rules, Recording Rules, and Alertmanager. – Текст : электронный // VictoriaMetrics [сайт]. – 2023. – URL: https://victoriametrics.com/blog/alerting-recording-rules-alertmanager/index.html (дата обращения: 24.05.2025).
20. HTTP API. – Текст : электронный // Prometheus [сайт]. – 2023. – URL: https://prometheus.io/docs/prometheus/latest/querying/api (дата обращения: 24.05.2025).
21. Основы мониторинга (обзор Prometheus и Grafana). – Текст : электронный // Хабр [сайт]. – 2025. – URL: https://habr.com/ru/articles/709204 (дата обращения: 24.05.2025).
22. Dashboards. – Текст : электронный // Grafana Labs [сайт]. – 2025. – URL: https://grafana.com/docs/grafana/latest/dashboards (дата обращения: 24.05.2025).
23. Roles. – Текст : электронный // Ansible Community Documentation [сайт]. – 2023. – URL: https://docs.ansible.com/ansible/latest/playbook\_guide/playbooks\_reuse\_roles.html (дата обращения: 24.05.2025).

**Приложение А**

**(обязательное)**

В данном листинге представлены конфигурационные файлы сервисов, используемых в процессе выполнения дипломного проекта.

Листинг А.1 – конфигурация haproxy.cfg

global

log /dev/log local0

chroot /var/lib/haproxy

user haproxy

group haproxy

daemon

maxconn 8000

stats socket ipv4@:9001 level admin

defaults

mode tcp

timeout connect 5s

timeout client 5m

timeout server 5m

frontend patroni\_master

bind \*:5000

mode tcp

default\_backend pg\_master\_backend

frontend patroni\_replicas

bind \*:5001

mode tcp

default\_backend pg\_replica\_backend

backend pg\_master\_backend

mode tcp

option httpchk GET /master

http-check expect status 200

default-server inter 3s fall 3 rise 2

server db1 10.10.10.1:5432 check port 8008

server db2 10.10.10.2:5432 check port 8008

backend pg\_replica\_backend

mode tcp

balance roundrobin

option httpchk GET /replica

http-check expect status 200

default-server inter 3s fall 3 rise 2

server db1 10.10.10.1:5432 check port 8008

server db2 10.10.10.2:5432 check port 8008

listen stats

bind :7000

mode http

stats enable

stats uri /

stats realm haproxy\ Statistics

stats auth admin:password

Листинг А.2 – Конфигурация сервиса «PgBouncer»

[databases]

postgres = host=10.10.10.1 port=5432 dbname=postgres

[pgbouncer]

listen\_addr = 10.10.10.3,10.10.10.4

listen\_port = 6432

auth\_type = md5

pool\_mode = transaction

max\_client\_conn = 100

default\_pool\_size = 20

reserve\_pool\_size = 5

server\_reset\_query = DISCARD ALL

stats\_period = 60

#logfile = /var/log/pgbouncer/pgbouncer.log

pidfile = /run/pgbouncer/pgbouncer.pid

admin\_users = postgres

auth\_file = /etc/pgbouncer/userlist.txt

ignore\_startup\_parameters = extra\_float\_digits

Листинг А.3 – конфигурация Keepalived

global\_defs {

router\_id lb1

log\_file /var/log/keepalived/keepalived.log

}

vrrp\_script haproxy\_check {

script "/usr/bin/systemctl -is-active --quiet HAProxy"

interval 2

weight 2

}

vrrp\_instance VI\_1 {

state MASTER

interface ens33

virtual\_router\_id 51

priority 100

advert\_int 1

authentication {

auth\_type PASS

auth\_pass password

}

virtual\_ipaddress {

10.10.10.100 dev ens33

}

}

Листинг А.4 – конфигурация Prometheus

global:

scrape\_interval: 10s

scrape\_timeout: 5s

evaluation\_interval: 10s

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'node'

static\_configs:

- targets:

- '10.10.10.1:9100'

- '10.10.10.2:9100'

- '10.10.10.3:9100'

- '10.10.10.4:9100'

- job\_name: 'postgres\_exporter'

static\_configs:

- targets:

- '10.10.10.1:9187'

- '10.10.10.2:9187'

- job\_name: 'patroni\_postgresql'

static\_configs:

- targets:

- '10.10.10.1:8008'

- '10.10.10.2:8008'

- job\_name: 'haproxy\_exporter'

static\_configs:

- targets:

- '10.10.10.3:9101'

- '10.10.10.4:9101'

Листинг А.5 – файл правил отправки уведомлений

groups:

- name: patroni\_alert

rules:

- alert: Patroni

expr: up{job="patroni\_postgresql"} == 0

for: 5m

labels:

severity: critical

annotations:

summary: "Server {{ $labels.instance }}."

description: "Patroni недоступна по адресу {{ $labels.instance }}."

- name: node\_alert

rules:

- alert: CPU

expr: 100 - (avg by(instance) (irate(node\_cpu\_seconds\_total{mode='idle',job="node"}[1m])) \* 100) > 80

for: 5m

labels:

severity: critical

annotations:

summary: "Server {{ $labels.instance }}."

description: "Сервер {{ $labels.instance }} использует {{ $value }}% процессора"

- name: memory\_alert

rules:

- alert: RAM

expr: 100 - (node\_memory\_MemAvailable\_bytes{job="node"} / node\_memory\_MemTotal\_bytes{job="node"} \* 100) > 90

for: 5m

labels:

severity: critical

annotations:

summary: "Server($labels.instance)"

description: Уровень загрузки оперативной памяти превысил 90% на сервере {{ $labels.instance }}.

- name: haproxy\_alert

rules:

- alert: haproxy

expr: haproxy\_up{job="haproxy\_exporter"} == 0

for: 5m

labels:

severity: critical

annotations:

summary: "Server {{ $labels.instance }}."

description: "HAProxy недоступен по адресу {{ $labels.instance }}."

Листинг А.6 – процесс добавления репозитория Grafana

sudo apt-get install -y apt-transport-https software-properties-common wget

sudo mkdir -p /etc/apt/keyrings/

wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null

echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://apt.grafana.com stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list

**Приложение Б**

**(обязательное)**

В листинге Б.1 представлена конфигурация телеграм бота для получения и отправки уведомлений Prometheus.

Листинг Б.1– конфигурация телеграм бота для получения и отправки уведомлений от сервиса «Prometheus»

import urllib.request

import json

import time

import requests

import hashlib

prometheus\_url = 'http://127.0.0.1:9090'

bot\_token = "8057975296:AAHpTYrdxeRSBoYHvDSLCZnMWsLsDdNL0oM"

bot\_chatID = "1762114550"

last\_message\_id = None

def tg\_bot(bot\_message):

""" Отправляет сообщение и возвращает его id. """

send\_text = f"https://api.telegram.org/bot{bot\_token}/sendMessage?chat\_id={bot\_chatID}&parse\_mode=HTML&text={bot\_message}"

response = requests.get(send\_text)

result = response.json()

if result['ok']:

return result['result']['message\_id']

else:

raise ValueError("Ошибка отправки сообщения.")

def delete\_last\_message(message\_id):

""" Удаляет последнее отправленное сообщение. """

del\_url = f"https://api.telegram.org/bot{bot\_token}/deleteMessage?chat\_id={bot\_chatID}&message\_id={message\_id}"

requests.post(del\_url)

#def tg\_bot(bot\_message):

# send\_text = "https://api.telegram.org/bot" + bot\_token + "/sendMessage?chat\_id=" + bot\_chatID + "&parse\_mode=HTML&text=" + bot\_message

# response = requests.get(send\_text)

# return response.json()

def get\_data(api):

with urllib.request.urlopen(prometheus\_url + '/api/v1/' + api) as url:

data = json.load(url)

return data

def get\_alerts\_message():

data = get\_data('alerts')

msg = '<b>Актуальное состояние инфраструктуры:</b>\n'

if data['data']['alerts']:

for alert in data['data']['alerts']:

msg += f'\n=====================================\n' \

f'<b>⚠ Проблема:</b> {alert["labels"].get("alertname")}\n' \

f'<b>💻 Машина:</b> {alert["labels"].get("instance")}\n' \

f'<b>🔴 Статус: критический - </b>{alert["state"]}\n' \

f'<b>ℹ Описание:</b> {alert["annotations"].get("description")}\n' \

f'\n<b>⏳ Время:</b> {alert["activeAt"]}\n' \

f'=====================================\n'

else:

msg += f'\n<b>Система работает в штатном режиме</b>\n' \

f'☕ У вас есть время попить кофе\n'

return msg

last\_alerts = None

while True:

try:

current\_alerts\_msg = get\_alerts\_message()

current\_alerts = current\_alerts\_msg

if last\_alerts is None or current\_alerts != last\_alerts:

new\_message\_id = tg\_bot(current\_alerts\_msg)

# time.sleep(1)

if last\_message\_id is not None:

delete\_last\_message(last\_message\_id)

last\_message\_id = new\_message\_id

last\_alerts = current\_alerts

time.sleep(5)

except Exception as e:

print(f'Ошибка подключения к Prometheus: {e}')

time.sleep(30)

При обнаружении ошибки системой «Prometheus», в мессенджере «Telegram» отображается сообщение следующего вида (рисунок А.1). В отсутствие ошибок система отображает сообщение о нормальном функционировании инфраструктуры (рисунок Б.2).

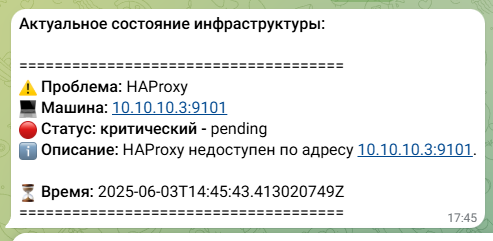


Рисунок Б.1 – Вид внешний уведомления об ошибке

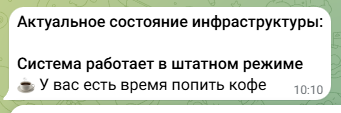


Рисунок Б.2 - Вид внешний уведомления об отсутствии ошибок

**Приложение В**

**(обязательное)**

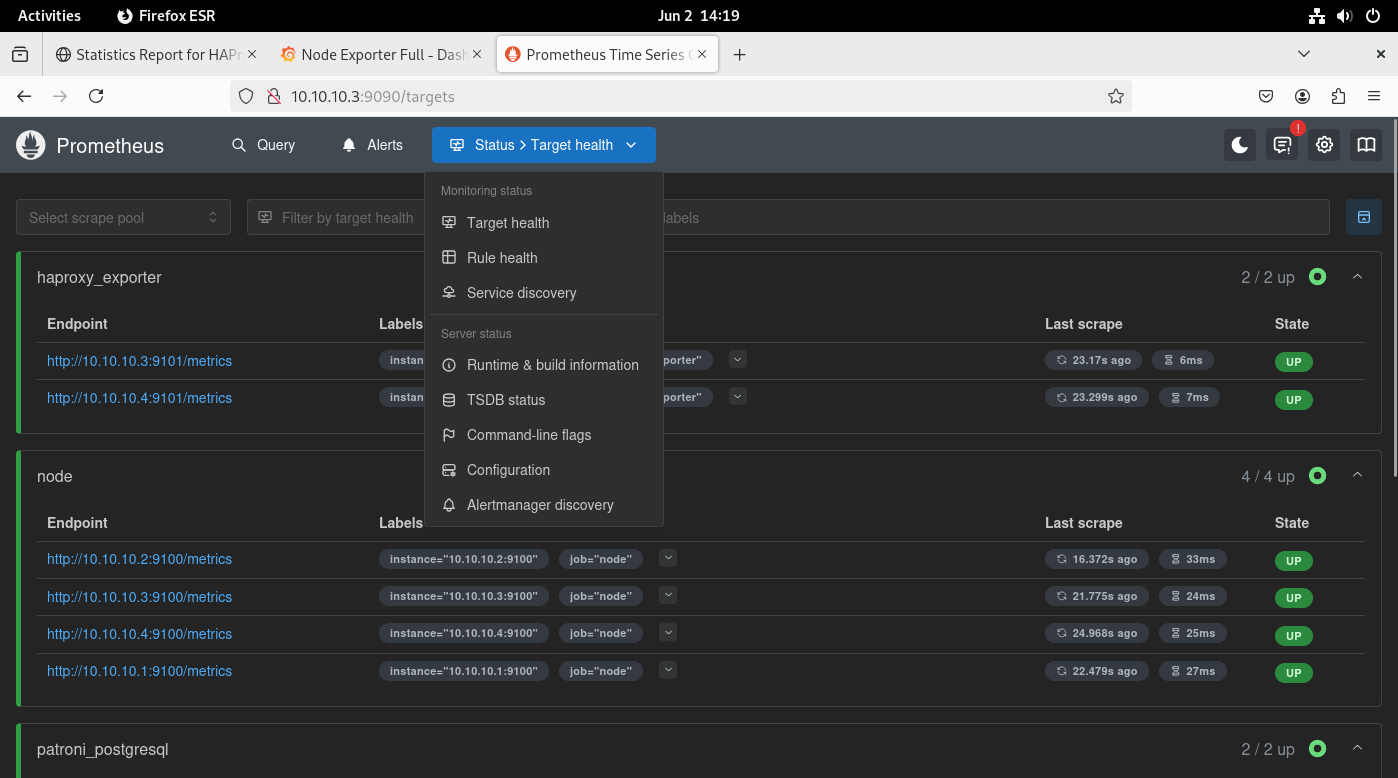


Рисунок В.1 – Вид внешний веб-интерфейса Prometheus

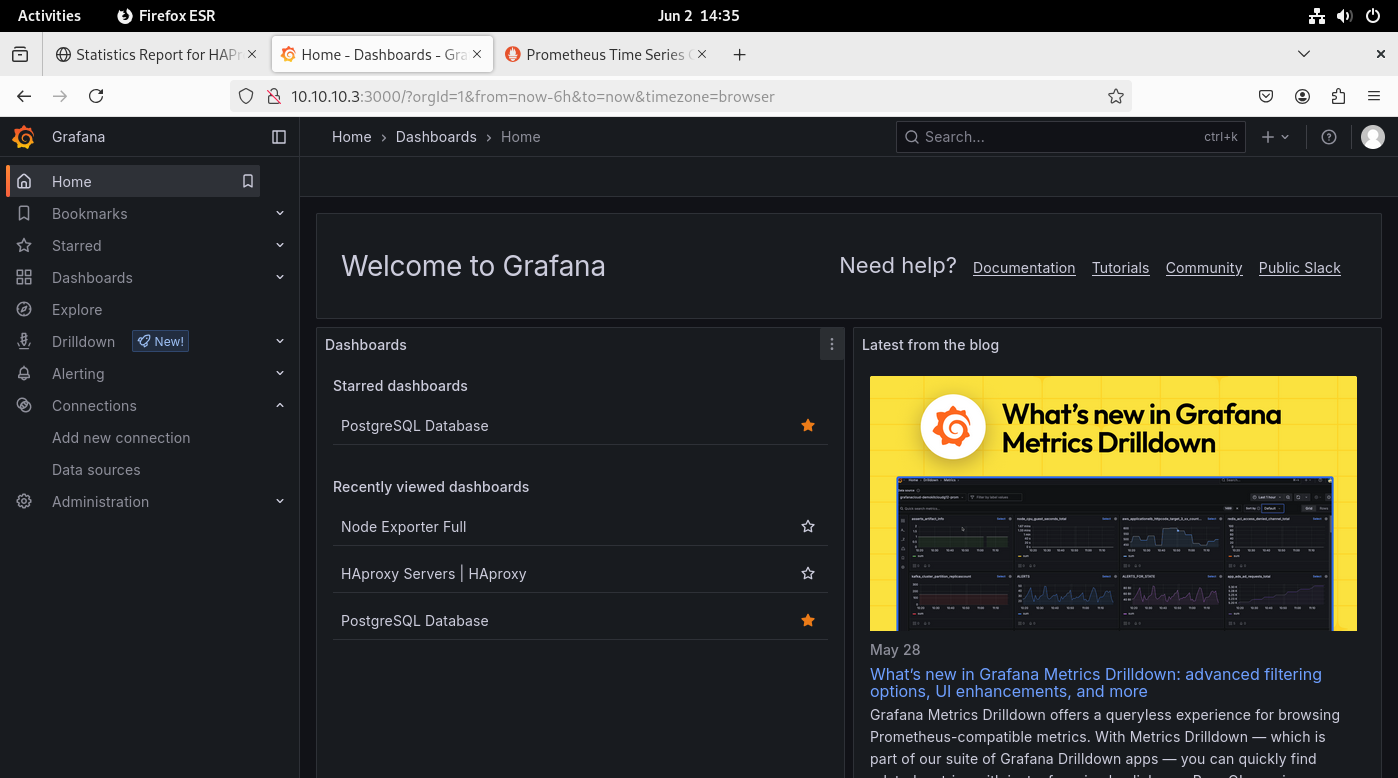


Рисунок В.2 – Вид внешний главной страницы Grafana

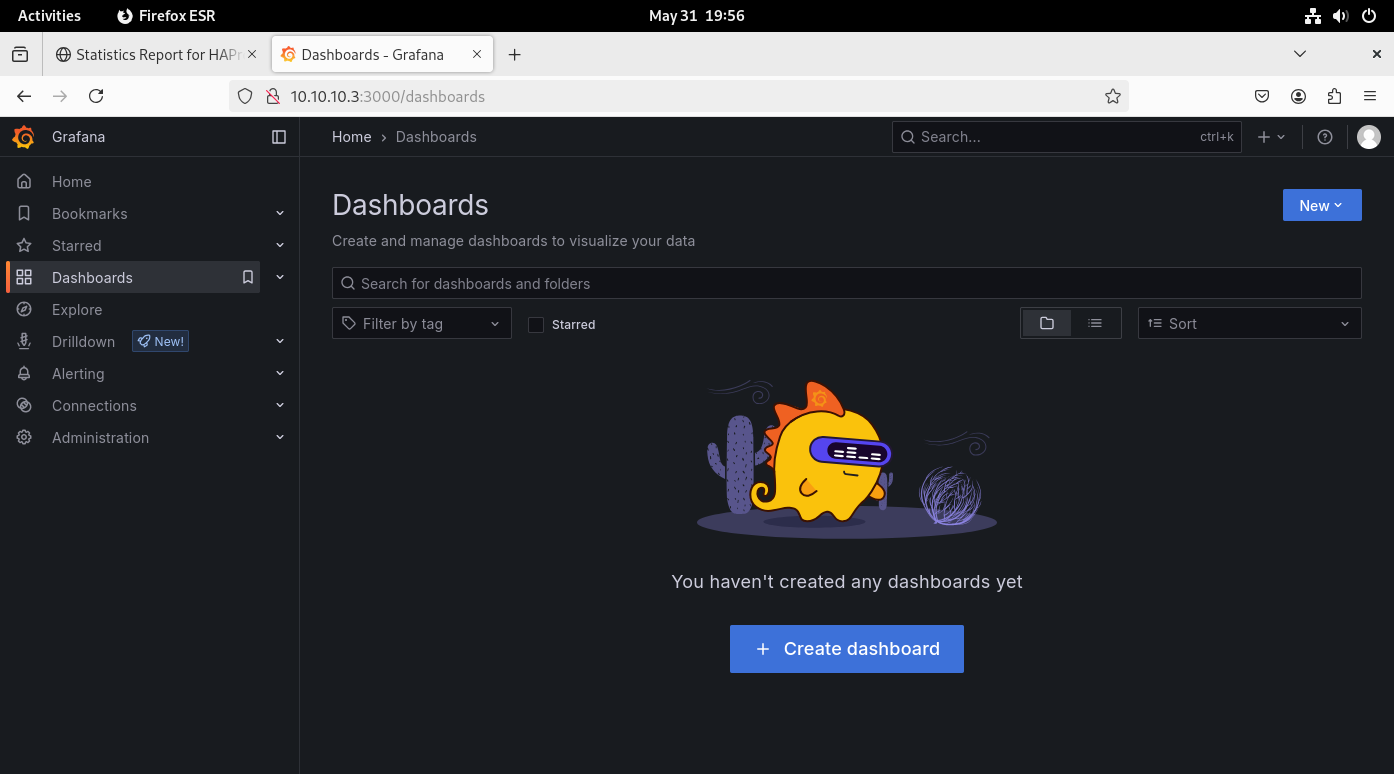


Рисунок В.3 – Вид внешний дэшбордов Grafana

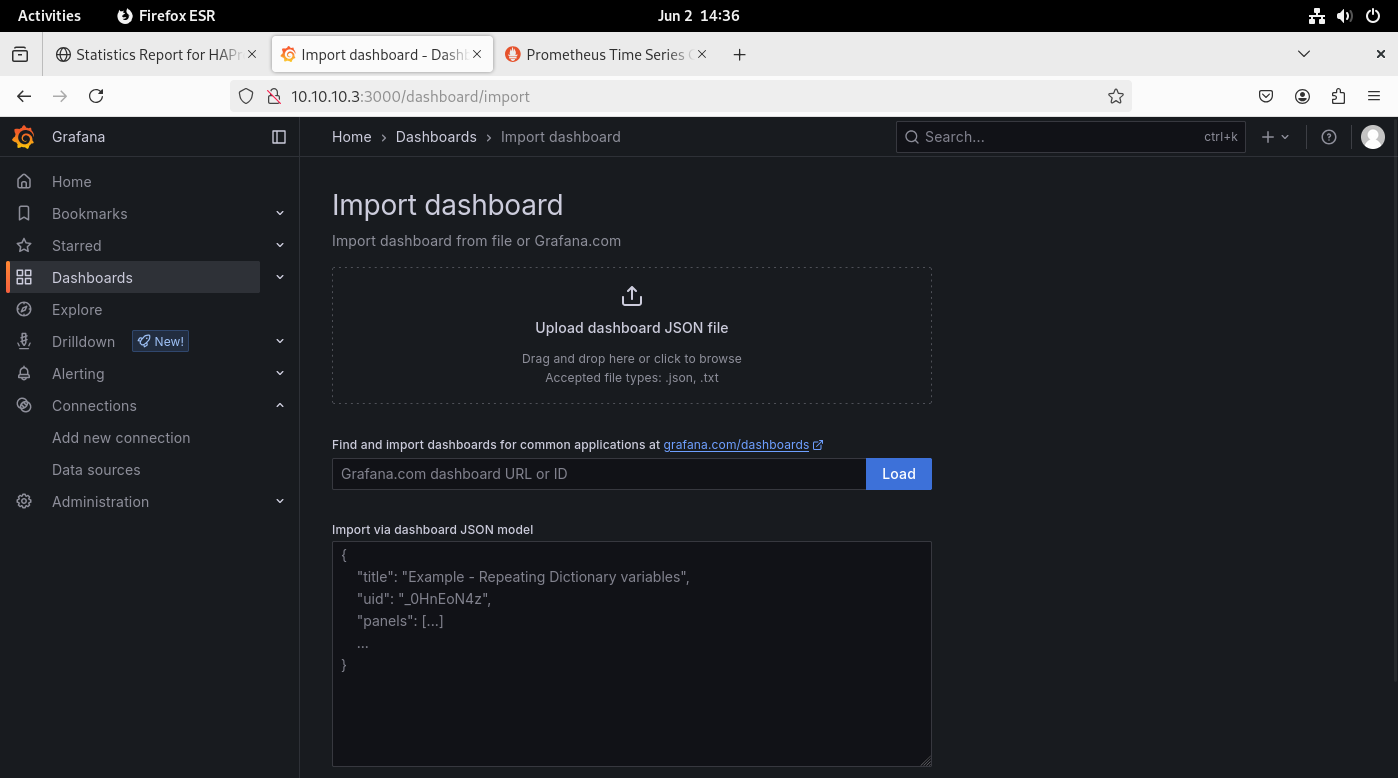


Рисунок В.4 – Вид внешний страницы импорта дэшбордов Grafana

На рисунке В.5 представлен дэшборд Node Exporter Full.

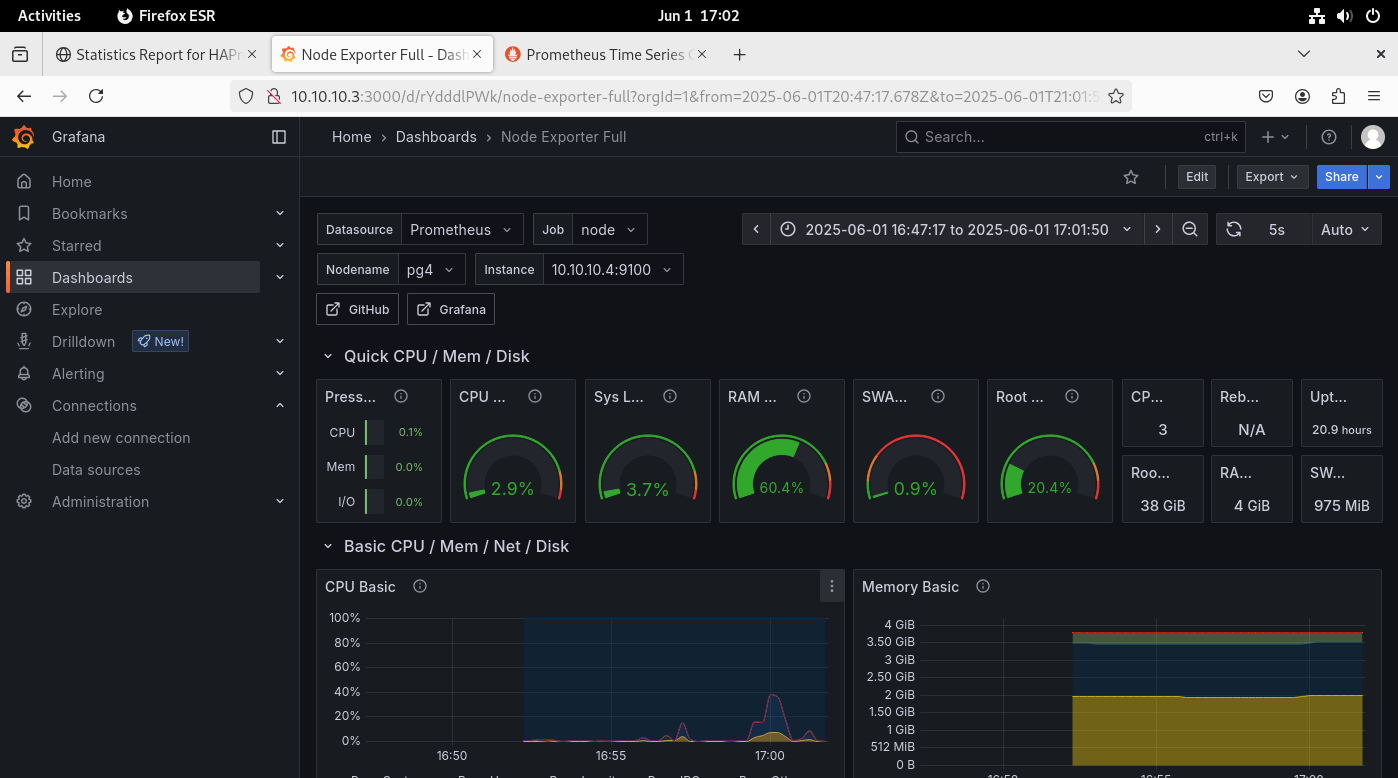


Рисунок В.5 – Вид внешний дэшборда Node Exporter Full

На рисунке В.6 представлен дэшборд HAProxy Servers | HAProxy.

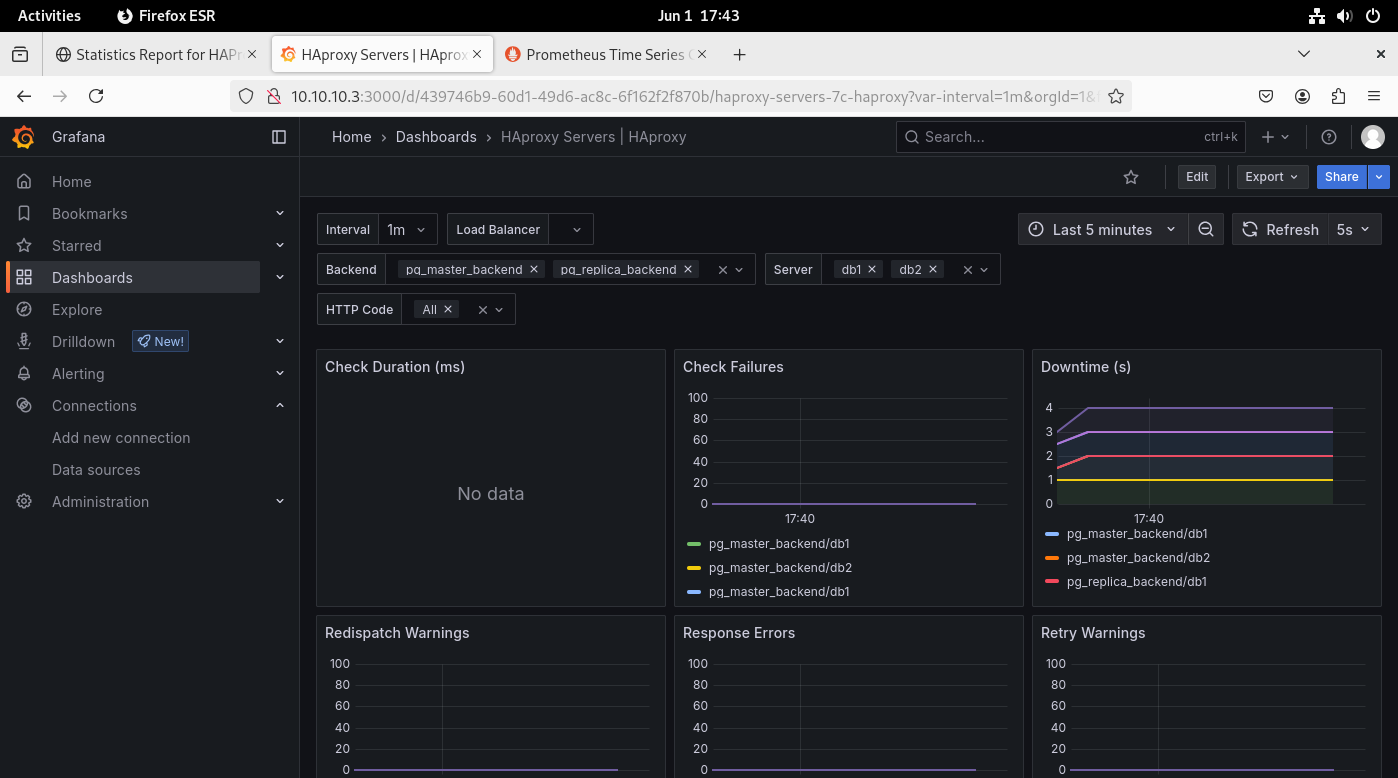


Рисунок В.6 – Вид внешний дэшборда HAProxy Servers | HAProxy

На рисунке В.7 представлен дэшборд PostgreSQL Database.

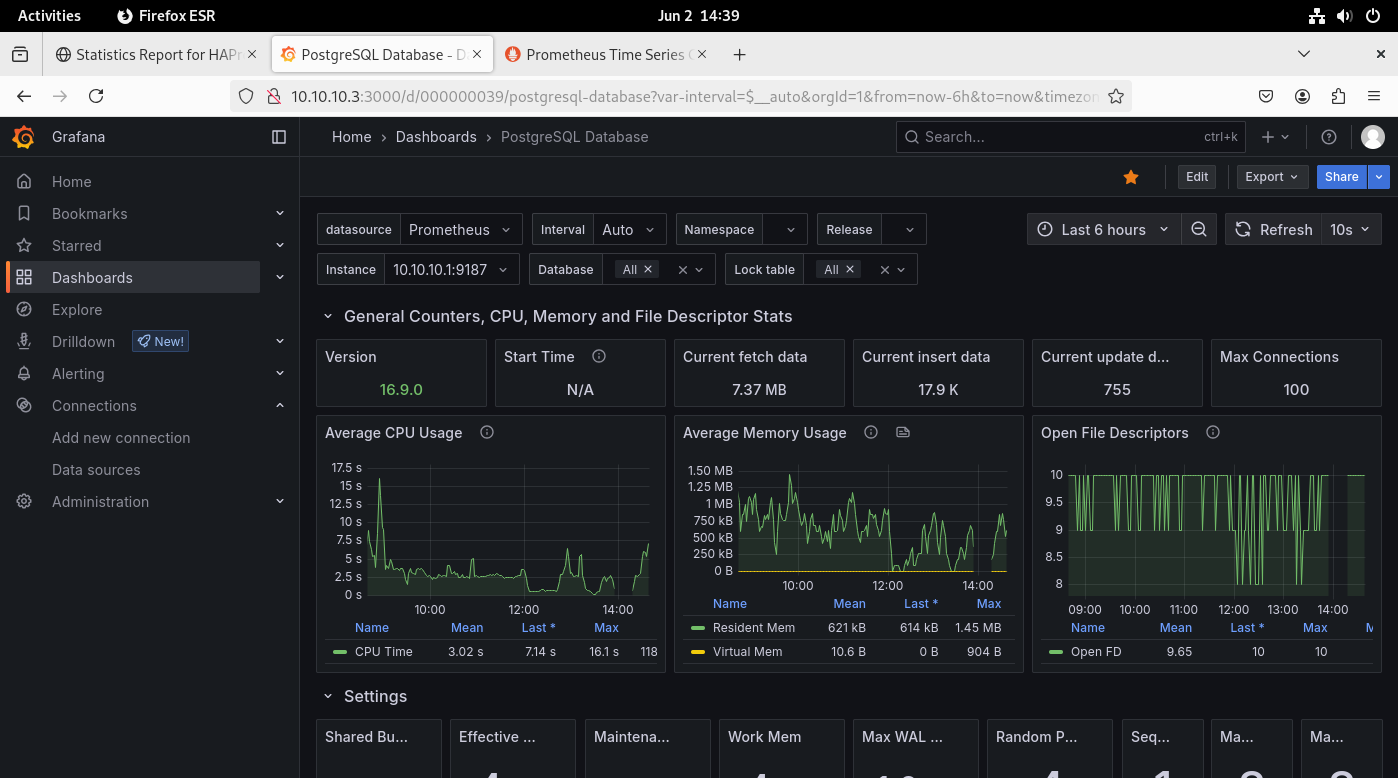


Рисунок В.7 – Вид внешний дэшборда PostgreSQL Database

**Приложение Г**

**(обязательное)**

В данном приложении представлены файлы шаблонов конфигураций node exporter (листинг В.1) и postgres exporter (листинги B.2 В.3)

Листинг Г.1 – шаблон node\_exporter.service.j2

[Unit]

Description=Node Exporter

After=network.target

[Service]

User=node\_exporter

Group=node\_exporter

Type=simple

ExecStart=/usr/local/bin/node\_exporter

Restart=always

[Install]

WantedBy=multi-user.target

Листинг Г.2 – шаблон postgres\_exporter.yaml.j2

pg\_setting:

query: "SELECT name, setting, unit, short\_desc FROM pg\_settings"

metrics:

- name: value

usage: "GAUGE"

description: "Value of pg\_settings parameter"

pg\_replication:

query: >

SELECT CASE WHEN pg\_is\_in\_recovery() THEN 0 ELSE 1 END as is\_master,

CASE WHEN pg\_is\_in\_recovery() THEN 1 ELSE 0 END as is\_replica,

COALESCE(EXTRACT(EPOCH FROM (now() - pg\_last\_xact\_replay\_timestamp()))::int, 0) as lag

metrics:

- name: is\_master

usage: "GAUGE"

description: "Instance is master"

- name: is\_replica

usage: "GAUGE"

description: "Instance is replica"

- name: lag\_seconds

usage: "GAUGE"

description: "Replication lag in seconds"

pg\_stat\_activity:

query: >

SELECT count(\*) as count,

state,

COALESCE(wait\_event\_type, '') as wait\_event\_type

FROM pg\_stat\_activity

GROUP BY state, wait\_event\_type

metrics:

- name: connections

usage: "GAUGE"

description: "Number of connections in this state"

labels:

- "state"

- "wait\_event\_type"

Листинг В.3 – шаблон postgres\_exporter.service.j2

[Unit]

Description=Prometheus PostgreSQL Exporter

After=network.target postgresql.service

[Service]

User=postgres

Group=postgres

Type=simple

Environment=DATA\_SOURCE\_NAME="user=postgres host=localhost sslmode=disable"

ExecStart=/usr/local/bin/postgres\_exporter --config.file=/etc/postgres\_exporter/postgres\_exporter.yaml

Restart=always

[Install]

WantedBy=multi-user.target

**Приложение Д**

**(обязательное)**

В данном приложении представлены шаблоны файлов конфигурации «prometheus.yml.j2» (листинг Г.1) «prometheus.service.j2» (листинг Г.2) «grafana.ini.j2» (листинг Г.3)

Листинг Д.1 – prometheus.yml.j2

global:

scrape\_interval: 10s

evaluation\_interval: 10s

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:{{ prometheus\_port }}']

- job\_name: 'node\_exporter'

static\_configs:

- targets:

{% for host in groups['promoters'] %}

- '{{ hostvars[host].ansible\_host }}:{{ node\_exporter\_port }}'

{% endfor %}

- job\_name: 'postgres\_exporter'

static\_configs:

- targets:

{% for host in groups['promoters'] %}

- '{{ hostvars[host].ansible\_host }}:{{ postgres\_exporter\_port }}'

{% endfor %}

Листинг Д.2 - prometheus.service.j2

[Unit]

Description=Prometheus

Wants=network-online.target

After=network-online.target

[Service]

User=prometheus

Group=prometheus

Type=simple

ExecStart=/usr/local/bin/prometheus \

--config.file={{ prometheus\_config\_path }}/prometheus.yml \

--storage.tsdb.path={{ prometheus\_storage\_path }} \

--storage.tsdb.retention.time={{ prometheus\_retention }} \

--web.console.templates=/etc/prometheus/consoles \

--web.console.libraries=/etc/prometheus/console\_libraries \

--web.enable-lifecycle

[Install]

WantedBy=multi-user.target

Листинг Д.3 - grafana.ini.j2

[server]

http\_port = {{ grafana\_port }}

[security]

admin\_user = admin

admin\_password = {{ grafana\_admin\_password }}

[auth.anonymous]

enabled = false

[dashboards]

versions\_to\_keep = 20

**Приложение Е**

**(обязательное)**

В данном приложении представлены таблицы и рисунки, используемые для экономического обоснования проекта.

Таблица Е.1 – Значение коэффициентов «c» и «В»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Язык программирования** | **Группа сложности** | **Степень новизны** | | | | **Коэффициент В** |
| **А** | **Б** | **В** | **Г** |
| Высокого уровня | 1 | 1,38 | 1,26 | 1,15 | 1,20 | 1,20 |
| 2 | 1,30 | 1,19 | 1,08 | 0,65 | 1,35 |
| 3 | 1,20 | 1,10 | 1,00 | 0,60 | 1,50 |
| Низкого уровня | 1 | 1,58 | 1,45 | 1,32 | 0,79 | 1,20 |
| 2 | 1,49 | 1,37 | 1,24 | 0,74 | 1,35 |
| 3 | 1,38 | 1,26 | 1,15 | 0,69 | 1,50 |

Таблица Е.2 – Коэффициент квалификации разработчика

|  |  |  |
| --- | --- | --- |
| **Опыт работы** | **Коэффициент квалификации** | |
| До двух лет | 0,80 | |
| 2-3 года | 1,00 | |
| 3-5 лет | | 1,10 – 1,20 |
| 5-7 лет | | 1,30 – 1,40 |
| Более 7 лет | | 1,50 – 1,60 |

Таблица Е.3 – определение рисков проекта

|  |  |
| --- | --- |
| **Риск** | **Способ минимизации риска** |
| Неясные требования к проекту | Проведение детальных встреч с заказчиком для уточнения всех требований и регулярное их обновление |
| Несоответствие ожиданий заказчика конечному продукту | Постоянное взаимодействие с заказчиком и демонстрация промежуточных результатов |
| Потеря данных из-за сбоя системы | Регулярное создание резервных копий, опись проделанных действий и использование облачных сервисов для хранения данных |
| Нарушение сроков разработки продукта | Правильная организация процесса создания программного продукта |
| Выход ПК из строя | Проведение своевременного техобслуживания применяемого оборудования |

Рисунок Е.1

диаграмма

Рисунок Е.2

Диаграмма