**Изображение выглядит как логотип

Автоматически созданное описаниеМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**(национальный исследовательский университет)»

**Институт (Филиал)** № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

**Группа** М8О-410Б-20 **Направление подготовки** 02.03.02 Фундаментальная информатика и информационные технологии

**Профиль** Информатика и компьютерные науки

**Квалификация: бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

на тему: «Масштабирование вычислительных мощностей автоматизированной системы мониторинга расхода топлива»

Автор ВКРБ: Забелин Никита Алексеевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Руководитель: Чернова Татьяна Александровна (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант:  (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант: (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Рецензент: (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**К защите допустить**

Заведующий кафедрой № 806 «Вычислительная математика  
и программирование» Крылов Сергей Сергеевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

*ХХ* мая 2024 года

Москва 2024

**Изображение выглядит как логотип

Автоматически созданное описаниеМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**(национальный исследовательский университет)»

**Институт (Филиал)** № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

**Группа** М8О-410Б-20 **Направление подготовки** 02.03.02 Фундаментальная информатика и информационные технологии

**Профиль** Информатика и компьютерные науки

**Квалификация**  **бакалавр**

**УТВЕРЖДАЮ**

Заведующий кафедрой № 806 «Вычислительная   
математика и программирование» (\_\_\_\_\_\_\_\_\_\_\_\_) С.С. Крылов

14 октября 2022 года

**ЗАДАНИЕ   
на выпускную квалификационную работу бакалавра**

**Обучающийся** Забелин Никита Алексеевич

**Руководитель** Чернова Татьяна Александровна*,* д.т.н, доцент, профессор кафедры 806

**1. Наименование темы:** Масштабирование вычислительных мощностей автоматизированной системы мониторинга расхода топлива

**2. Срок сдачи обучающимся законченной работы:** 24 апреля 2024 года

**3. Задание и исходные данные к работе**

* Обеспечить возможность автоматизированной системе мониторинга расхода топлива масштабироваться в режиме реального времени в соответствии с входящим потоком данных.

**4. Перечень подлежащих разработке разделов и этапы выполнения работы:**

| № п/п | Наименование раздела или этапа | Трудоемкость в % от полной трудо­емкости квалифи­кационной работы | Срок выполнения | Примечание |
| --- | --- | --- | --- | --- |
| 1 | Изучение и анализ литературы о современных системах автоматизированного контроля топлива их возможностях к масштабированию | 15 | 18.12.2024 |  |
| 2 | Проектирование архитектуры разрабатываемого решения и непосредственная его реализация, с использованием выбранных технологий | 35 | 13.02.2024 |  |
| 3 | Оценка полноты охвата поставленных задач разработанной системой и анализ эффективности ее работы | 25 | 27.03.2024 |  |
| 4 | Введение | 15 | 10.04.2024 |  |
| 5 | Заключение (краткие выводы и перспективы дальнейшей разработки темы) | 10 | 17.04.2024 |  |

**5. Исходные материалы и пособия:**

– Список литературы (основные работы, 25-30% от приведенного в работе списка)

**6. Дата выдачи задания:** 14 октября 2024 года

Руководитель (\_\_\_\_\_\_\_\_\_\_\_\_\_) Т.А. Чернова

Задание принял к исполнению (\_\_\_\_\_\_\_\_\_\_\_\_\_) Н.А. Забелин

# **РЕФЕРАТ**

Выпускная квалификационная работа бакалавра состоит из *00* страниц, *00* рисунков, 1 таблицы, *00* использованных источников, 1 приложения.

Ключевые слова: YANDEX CLOUD, АВТОМАСШТАБИРОВАНИЕ, KUBERNETES, СИСТЕМЫ КОНТРОЛЯ ТОПЛИВА, PROMETHEUS, ОБЛАЧНЫЕ ТЕХНОЛОГИИ, ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ.

Объектом разработки в данной работе является программная система, которая будет в себе заключать автоматизированную систему контроля топлива и модуль, контролирующий выделенные ей мощности.

Цель работы – создание программной системы, которая в соответствии с входящим потоком данных будет изменять количество выделенных для нее мощностей для обеспечения минимального времени отклика и экономии средств в случаях его повышения и понижения интенсивности трафика соответственно*.*

Для достижения поставленной цели были проведены исследования в области современных подходов к разворачиванию распределенных высоконагруженных автоматизированных систем, интеграции облачных решений в инфраструктуры компаний и алгоритмов масштабирования программных систем*.* Основное содержание работы состояло в разработке двух программных модулей*.* Первый из них отвечает за прием данных от удаленных источников, а второй производит мониторинг нагрузки на него и, в зависимости от заданной конфигурации, увеличивает или уменьшает выделенные для него мощности.

Основными результатами работы, полученными в процессе разработки, являются развернутые в облачной среде компоненты программной системы, которые в режиме реального времени в состоянии функционировать, то есть подстраиваться под входящий поток данных*.*

Данные результаты разработки предназначены для использования компаниями, которые по природе своей деятельности предполагают наличие систем с существенно изменяющейся на протяжении времени интенсивностью получаемых запросов, в частности топливо снабжающие организации.

Применение результатов данной работы позволяет организациям снизить свои затраты на обслуживание мощностей, которые размещены локально, и использовать вместо них облачную платформу, а также оптимизировать выделяемые средства за счет варьирования запрашиваемых для корректного функционирования ресурсов.

Перспективы развития решения заключаются в совершенствовании модели по вычислению необходимых ей на данный момент мощностей и развития web-консоли администратора.

СОДЕРЖАНИЕ

[РЕФЕРАТ 4](#_Toc167227528)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 7](#_Toc167227529)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 8](#_Toc167227530)

[ВВЕДЕНИЕ 9](#_Toc167227531)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 13](#_Toc167227532)

[1.1 Анализ предметной области и изучение объекта исследования, а также оценка изученности проблемы 13](#_Toc167227533)

[1.2 Выбор облачной платформы, ее преимущества и недостатки 18](#_Toc167227534)

[1.3 Формирование требований к программной системе и концепция программного продукта 22](#_Toc167227535)

[1.4 Выбор платформы и языка программирования для разработки системы 24](#_Toc167227536)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 30](#_Toc167227537)

[2.1 Разработка архитектурных решений и реализация собственной системы на языке программирования Python 3 и TypeScript 30](#_Toc167227538)

[2.2 Настройка облачной среды Yandex Cloud для разворачивания приложения 36](#_Toc167227539)

[2.3 Разворачивание приложения в облачной среде Yandex Cloud 41](#_Toc167227540)

[2.4 Представление результатов исследования, связанных с способностью программной системы масштабировать вычислительные ресурсы приложения 49](#_Toc167227541)

[ЗАКЛЮЧЕНИЕ 53](#_Toc167227542)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 56](#_Toc167227543)

[ПРИЛОЖЕНИЕ А 61](#_Toc167227544)

# **ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

|  |  |
| --- | --- |
| Кластер | *–* объединение однородных элементов в единое целое, в данном случае объединение вычислительных мощностей. |
| Вычислительная мощность | *–* объект, который в состоянии выполнять вычисления и обрабатывать данные. |
| Docker-образ | *–* файл специального формата, в котором содержится среда исполнения, утилиты и полезная нагрузка. |
| Под | *–* минимальная размещаемая сущность системы Kubernetes, включающая в себя один или больше контейнеров с общими ресурсами. |
| REST-API | *–* технология, позволяющая общаться с приложением с использованием HTTP протокола. |
| Web-сервер | *–* сервер, принимающий HTTP-запросы и отдающий HTTP-ответы. |
| Транзакция | – единица работы, которая выполняется в системе управления базами данных и обрабатывается согласованным и надёжным способом независимо от других транзакций. |
|  |  |
|  |  |

# **ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

|  |  |
| --- | --- |
| СУБД | *–* система управления базами данных |
| HTTP | *–* HyperText Transfer Protocol (протокол передачи гипертекста) |
| API | *–* Application Programming Interface (интерфейс программирования приложения) |

# 

# **ВВЕДЕНИЕ**

Актуальность данной работы связана с тем, что в наше время распределенные системы стали основой любой крупной компании, работающей с большим потоком входных данных. Главной задачей таких компаний является обработка нагрузки с максимальной скоростью и с минимальными затратами на ресурсы. В процессе ее выполнения возникает проблема: каким образом добиться необходимого результата? Если выделить большое количество вычислительных мощностей, то это увеличит затраты на аренду или эксплуатацию, а в случае минимизации их числа вызовет увеличение времени ответа пользователям, или вовсе отказ всей системы. Из чего можно сделать вывод, что реализация алгоритма динамического масштабирования системы, и последующее его применение для реальных потребителей, жизненно необходимо в рамках современных реалий.

Таким образом, выполненная работа актуальна и с теоретической, и с практической точек зрения.

Для достижения поставленной цели в работе были решены следующие задачи:

* + Сравнительный анализ эффективности работы существующих автоматизированных систем мониторинга расхода топлива при существенном изменении объёма и характера нагрузки;
  + Установление зависимости стабильности функционирования серверного узла распределенной вычислительной системы и его состояния от входных параметров и определение оптимальной вычислительной мощности при заданной нагрузке;
  + Обеспечение оптимального распределения нагрузки на сервера распределенной вычислительной системы при условии изменения характера и интенсивности входящего потока заявок с помощью разработанного программного модуля;
  + Проведение нагрузочного тестирования разработанной системы и оценка полученных показателей эффективности.

Цель работы – развёртывание в облачном сервисе распределенной вычислительной системы, обеспечение отказоустойчивости при изменении интенсивности и объёма информационного потока, оптимизируя вычислительную мощность используемых аппаратных ресурсов.

Работа основывалась на следующих инструментах и методах: облачная платформа Yandex Cloud, программное обеспечение для автоматизации и развертывания и упаковки приложений в виде исполняемых контейнеров Docker, программное обеспечение для оркестровки контейнеризировнных приложений Kubernetes, высокоуровневый язык программирования с динамической строгой типизацией Python3, система мониторинга и оповещения Prometheus, HTTP-сервер и обратный прокси-сервер, почтовый прокси-сервер, а также TCP/UDP прокси-сервер Nginx, javascript фреймворк, основанный на React – NextJS, мультипарадигменный язык программирования TypeScript, библиотека компонентов для фреймворка React ­– Material UI и пакетный менеджер для JavaScript – Yarn.

Основными результатами, полученными в работе, являются:

* Создание и реализация алгоритма динамического масштабирования системы в зависимости от заданной конфигурации и текущего состояния программной системы;
* Настройка облачной среды для размещения в ней системы в целях дальнейшей эксплуатации в отказоустойчивом экономном режиме;
* Демонстрация возможностей спроектированной и разработанной системы реагировать на изменение интенсивности и объема входящего потока информации уменьшением или увеличением используемых мощностей.

Результаты работы предназначены для применения компаниями, предоставляющими услуги по обслуживанию пользовательского трафика или собирающими его со своих конечных точек, у которых численность данных в единицу времени может разниться от времени года, суток или других факторов, таких как погодные условия, праздники, маркетинговые кампании или социальные события. Эти вариации в объёме данных требуют гибких и масштабируемых решений, которые могут адаптироваться к изменяющимся нагрузкам. Использование разработанных решений позволяет компаниям эффективно управлять этими изменениями, обеспечивая стабильную и высокопроизводительную работу своих систем независимо от внешних условий.

Использование данной разработки позволяет существенно уменьшить расходы на инфраструктуру, что особенно важно в случае её наличия у компании. Это снижение затрат достигается за счёт оптимизации существующих ресурсов и сокращения необходимости в дополнительном оборудовании или программном обеспечении. Кроме того, внедрение такой разработки обеспечивает непрерывную и стабильную работу всех сервисов компании, что критично для выполнения обязательств перед как внутренними, так и внешними заказчиками. Компании, использующие эту разработку, могут более эффективно управлять своими ресурсами, что ведёт к повышению производительности и снижению эксплуатационных расходов. Это, в свою очередь, позволяет направить сэкономленные средства на другие стратегические направления или инновационные проекты, что способствует общему росту и развитию бизнеса. Добросовестное выполнение обязательств перед заказчиками обеспечивается за счёт повышения надежности и скорости работы систем, что минимизирует время простоя и снижает риски связанных с этим убытков. Кроме того, минимальные затраты на эксплуатацию системы означают, что компания может предложить конкурентоспособные цены на свои услуги, что привлекает больше клиентов и увеличивает долю рынка. Таким образом, использование данной разработки способствует не только улучшению внутренних процессов, но и повышению общей конкурентоспособности компании на рынке.

# **1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

## **1.1 Анализ предметной области и изучение объекта исследования, а также оценка изученности проблемы**

Продолжительное время в России повышаются цены на топливо. Это подтверждают данные на сайте Росстат [2]. На рисунке 1 приведена статистика, как вели себя цены на топливо за последнее десятилетие. Продолжительный рост на графике очевиден.

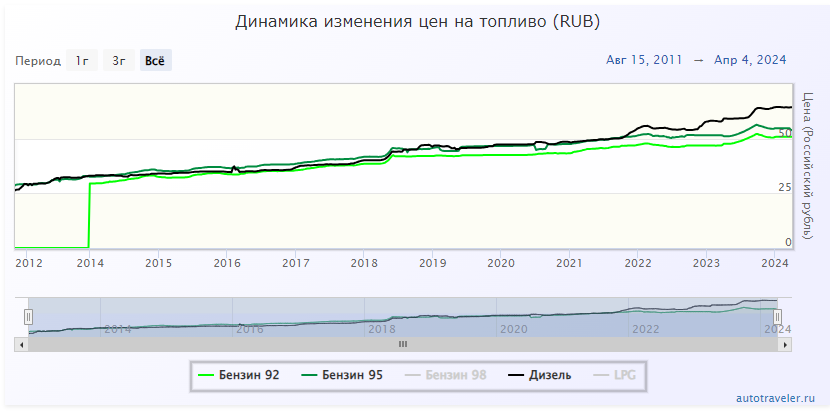


Рис. 1 – Динамика цен на 92, 95 бензин и дизель

Рост цены на топливо оказывает непосредственное влияние на автотранспортные предприятия и организации, чья деятельность осуществляется в основном автомобильными грузоперевозками, в статье Плотниковой О. В. [3] подробно рассмотрен этот вопрос и приведена статистика, изучена зависимость товаров и услуг от цены на топливо. Очевидно, что для обычного человека это повышение также будет заметно, но не только из-за потребительских цен на топливо на автозаправочных станциях для личного транспорта. В цену каждого товара на полке магазина или оказываемой услуги, вроде поездки на автобусе, вложена стоимость горюче-смазочных материалов, поэтому цена огромного числа товаров и услуг находится в прямой зависимости от стоимости топлива.

Рассмотрим крупную топливоснабжающую организацию, имеющую автозаправочные станции. Она обслуживает клиентов: автобазы, логистические и автопредприятия, отдельных перевозчиков и т. д., в общем, организации, чья деятельность построена на автомобильном транспорте. Её клиенты при вышеуказанных условиях начинают активнее использовать услугу этой топливоснабжающей организации. Услуга заключается в отслеживании расхода топлива транспортных средств, а также их мониторинга местоположения и передвижения, ведении реестра топливных карт и отслеживания наличия средств на этих картах. Она позволяет оптимизировать потребление топлива, исключить несанкционированные траты, об этом подробно рассказано в статьях [6, 7]. Так, например, по графику расхода топлива от времени легко заметить сливы топлива, необычные расходы, к тому же есть возможность наблюдать местоположение в данное время, это исключает возможность подобных топливных непредвиденных трат. В статье Кошевого Н. Д. [4] подробно рассмотрено внутреннее устройство такой системы для двигателя внутреннего сгорания, это желательно понимать для дальнейшей работы с подобной системой. Однако для данного исследования мы будем работать над информационной системой, то есть работа не будет касаться блока сбора данных.

Помимо того, что наша топливоснабжающая организация предоставляет все нужные датчики, необходимые для работы данной системы, она имеет ранее упомянутую информационную систему, то есть некоторое программное обеспечение, в состав которого обычно входит программное решение в виде web или desktop приложения, как на рисунке 2, и база данных с серверной частью. Пример таких программ приведён на сайтах [5, 9, 10], здесь же рассмотрены все виды предоставления услуги и их преимущества. Этой услугой как раз и пользуются клиенты. Обычно, как самым популярным способом, для размещения программного обеспечения организации используют сервера (свои или арендованные). В чём же состоит проблема такого подхода? Здесь мы подходим к сути.

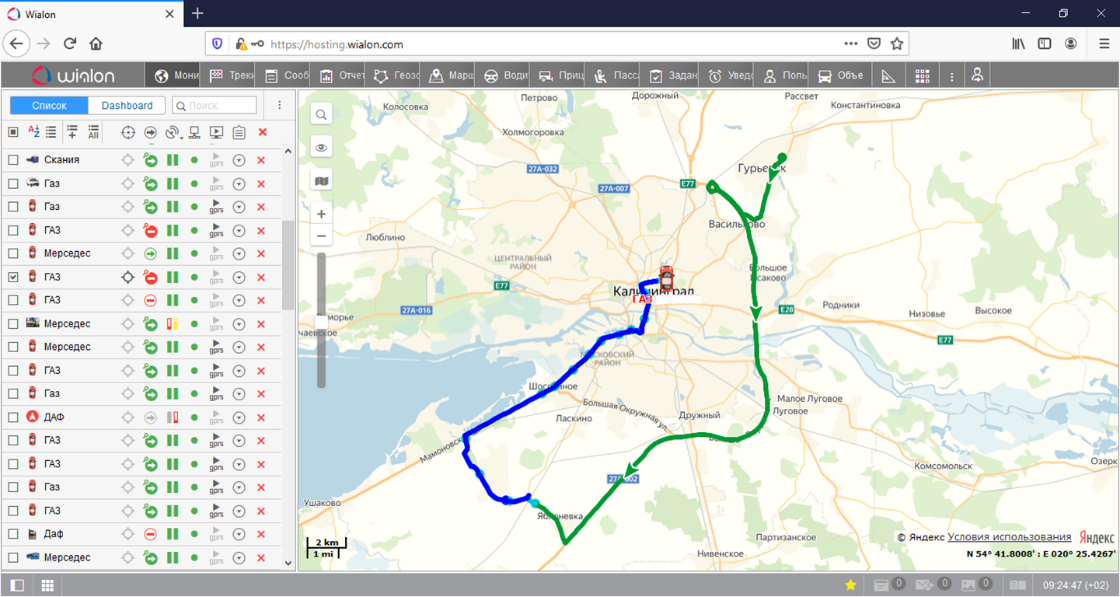


Рис. 2 – Пример внешнего вида программы

Сервер – это, по сути, отдельный компьютер для выполнения сервисного программного обеспечения. Мощность сервера строго ограничена его комплектующими, память –жёсткими дисками, оперативная память – блоками оперативных запоминающих устройств, вычислительная мощность – процессором. Клиенты в заданных ранее условиях пользуются услугой – отправляют системе запросы, причём, делают это в одни и те же промежутки времени, наблюдается повышение и снижение активности, суточное и сезонное. Пример графика периодической нагрузки (зависимости загрузки ядер центральных процессоров от времени) на рисунке 3.

Рассмотрим обе ситуации с точки зрения сервера. Если запросов приходит слишком много, то возникает нехватка вычислительных ресурсов сервера, запросы обрабатываются очень долго и, как правило, клиенты не дожидаются своей очереди на обслуживание, если же запросы не приходят или приходит мало, а сервер работает, происходит недогрузка и владелец его платит, по сути, за его работу впустую. Возникшая проблема появилась из-за невозможности масштабировать мощность сервера.

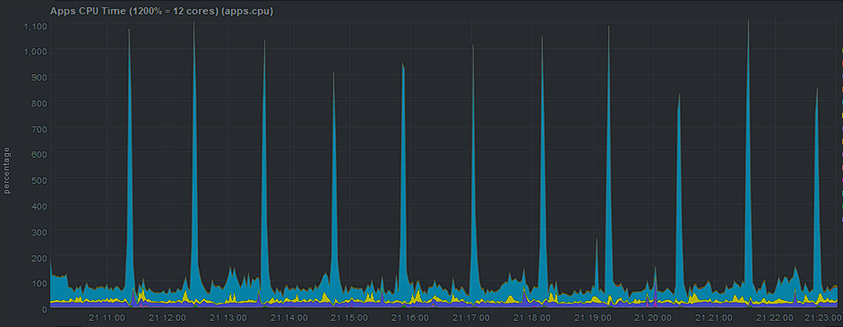


Рис. 3 – Периодическая нагрузка на сервер

Что же можно сделать? Можно масштабировать его мощность двумя способами: повысить максимальную вычислительную мощность сервера посредствам улучшения компонентов системы (вертикальное масштабирование), то есть заменить процессоры на более современные, с большим числом потоков, оперативных запоминающих устройств с большим объёмом и частотой памяти, (по большому счёту увеличение максимальной вычислительной мощности сервера, см. рис. 5), либо увеличить количество виртуальных машин, между которыми распределяется нагрузка (горизонтальное масштабирование, см. рис. 7), требуется заранее учесть, что не всё программное обеспечение подходит для работы с таким типом расширения производительности, придется его либо модернизировать либо заменять на другое. Подробно про масштабирование рассказано на сайтах [6, 8], также мы более подробно рассмотрим масштабирование в облаке в следующем параграфе. Преимущества и недостатки одного или множества серверов приведено на рисунке 4, единственный камень преткновения в использовании множества — это его сложность.

Во время пиковой нагрузки мы не можем вертикально масштабировать сервер без его перезагрузки, только заранее предугадать ожидаемую нагрузку, а горизонтально достаточно сложно, для этого как минимум потребуется специалист, которому нужно будет платить зарплату, в обоих случаях проблема с недогрузкой тоже есть. Решение данной задачи лежит далее.

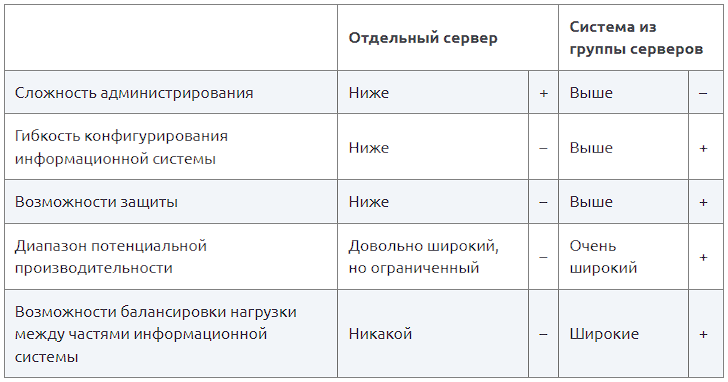


Рис. 4 - Сравнение способов облачного развёртывания

Предлагается использовать для данных целей облачные сервисы, такие как Yandex Cloud, VK Cloud и другие. Они включают в себя технологии распределённых вычислений, облачных хранилищ, и иных решений для IT-сферы компаний. Предоставляют возможности разработки, выполнения приложений и хранения данных на серверах, расположенных в распределённых дата-центрах. Одним из основных преимуществ этих систем, как отдельная услуга, является динамическое масштабирование, вертикальное и горизонтальное.

## **1.2 Выбор облачной платформы, ее преимущества и недостатки**

На данный момент на отечественном рынке облачных технологий лидируют решения 3 различных крупных компаний: Яндекса, ВКонтакте и Сбербанка. Их предложения Yandex Cloud, VK Cloud и Sber Cloud в первую очередь нацелены на конкурирование с зарубежными аналогами, такими как Amazon Web Services, Google Cloud, Microsoft Azure. Каждая из 3 представленных платформ предоставляет ряд типовых для данной сферы возможностей: хранение файлов, управление сервисом для оркестровки контейнеризированных приложений, развертывания СУБД, хранение кэша и временных данных и размещение брокера сообщений.

Таблица 1 – Функционал, отечественных облачных платформ

|  |  |  |  |
| --- | --- | --- | --- |
|  | Yandex Cloud | VK Cloud | Sber Cloud |
| Размещение статических файлов | Yandex Object Storage | VK Cloud Solutions Storage | Object Storage |
| Управление сервисом Kubernetes | Yandex Managed Service for Kubernetes | VK Solutions Containers | Cloud Container Engine |
| Развертывание БД | Yandex Managed Service for PostgreSQL | VK Cloud Solutions Databases | Relational Database Service |
| Хранение кэша или временных данных | Yandex Managed Service for Redis | Distributed Cache Service for Redis |
| Размещение брокера сообщений | Managed Service for Apache Kafka | VK Cloud Solutions Queue Queues | Distributed Message Service for Kafka |

Изучив отечественный рынок, а также приняв во внимание популярность решения от Yandex, проанализируем облачную платформу Yandex Cloud.

В рамках данной работы нас интересуют, в основном, возможности платформы по масштабированию приложений. Рассмотрим то, как реализуется вертикальное и горизонтальное масштабирование приложения в облачном сервисе. Существует порядка десятка планов под любые нужды бизнеса и предприятий в вертикальном варианте. Таким образом можно выбрать подходящее число ядер процессора, самих центральных процессоров, памяти и т. д. В любой момент можно это план поменять. В горизонтальном варианте работает так: выбирается минимальное и максимальное число подключаемых виртуальных машин, а далее платформа сама регулирует их необходимое количество, оптимальное для владельца.

Рассмотрим все имеющиеся варианты с помощью графика на рисунке 3. В первом случае мы поднимаем максимальную мощность (условно красная линия) системы (изображено на рисунке 5), тем самым пики нагрузки далее помещаются под красной линией, это значит, что решилась первая проблема недостаточности вычислительных ресурсов. Вторая проблема остаётся, ведь наибольший промежуток времени остается минимальная загрузка, для которой совсем не требуется вся мощность. Нужно исключить избыточные траты, чтобы мощности не простаивали. Сервис предоставляет возможность создания графика увеличения мощности. То есть если наша автоматизированная система сталкивается с перегрузкой, условно, каждый будний день в 8:00, либо каждое начало сезона, то легко спрогнозировать будущую нагрузку и дополнительно выделять в нужные интервалы ресурсы.

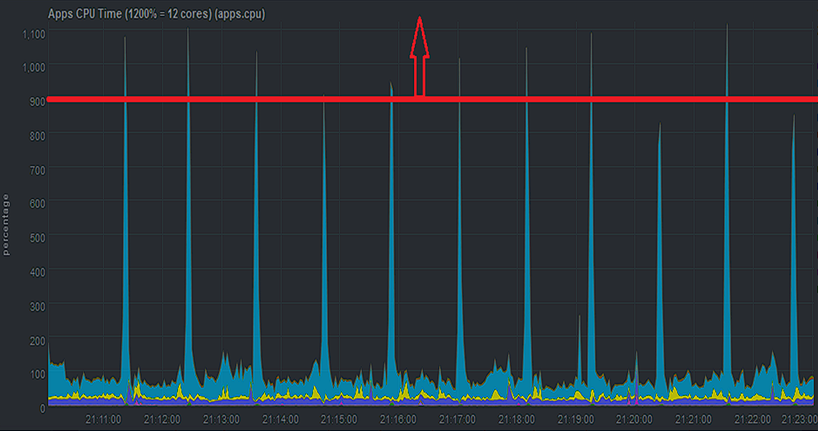


Рис. 5 – Вертикальное масштабирование в облаке

Во втором случае (горизонтальное масштабирование или автомасштабирование), как ранее было сказано, выбираем минимальное и максимальное число виртуальных машин и изображаем красными линиями (см. рис. 7). В данном примере минимально - 1 рабочая машина, максимально - 5, см. рис. 6. На большей части графика требуется только одна машина, её мощности достаточно для нормальной работы облачного сервера. В периоды повышения нагрузки машины динамически подключаются при достижении n-ной загрузки. То есть при достижении определённого процента от максимальной текущей загрузки текущая прямая поднимается на уровень выше (довыделяется дополнительная рабочая машина).

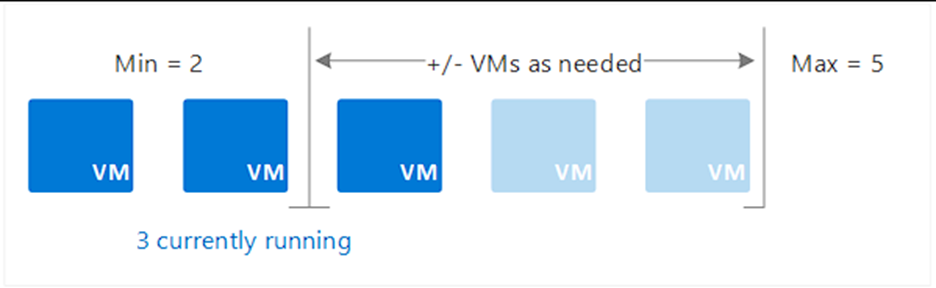


Рис. 6 - Диапазон количества выделяемых ВМ для автомасштабирования

Этот подход решает обе проблемы, во-первых, нет избыточности привлекаемых ресурсов, владелец платит только за фактически применяемую вычислительную мощность, во-вторых, в случае недостаточности ресурсов, они динамически довыделяются (как виртуальные машины). Для владельца это значит, что такой тип масштабирования очень удобно применять в случае неизвестных периодов повышения нагрузки, то есть нагрузка может повыситься внезапно. К тому же можно использовать автомасштабирование просто потому, что всё делается само, и принимать участие в довыделении мощности не нужно как в первом случае. Возможно, для крупного бизнеса или предприятия это обойдётся даже дешевле.

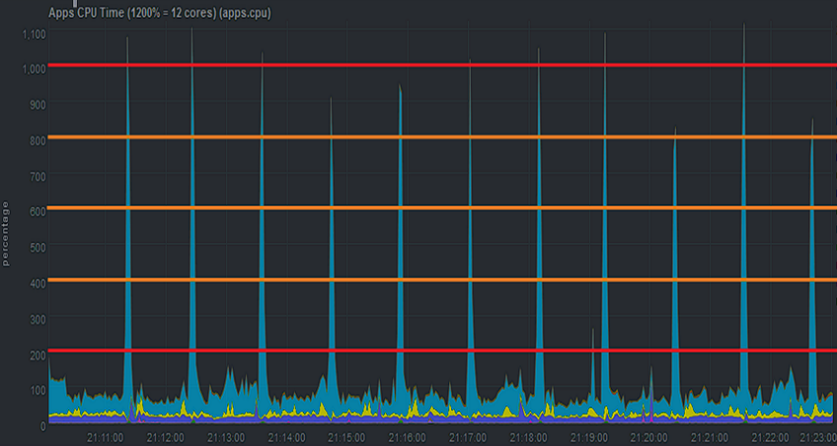


Рис. 7 – Горизонтальное масштабирование (автомасштабирование) в облаке

Итого выбранной облачной платформой в данной работе будет Yandex Cloud. Владелец, в зависимости от кривой нагрузки может сам выбрать как ему удобнее масштабировать свою автоматизированную систему. В дополнение нужно сказать про место для базы данных, его всегда можно докупить. Обычно, в выбранном плане мощности уже достаточно места на хранилище, но системы бывают разные и проблемы с хранилищем не возникнет. К тому же большим преимуществом этой облачной платформы является применение искусственного интеллекта для прогнозирования и рекомендаций для наиболее выгодного и эффективного использования платформы.

## **1.3 Формирование требований к программной системе и концепция программного продукта**

Наша программная система должна отвечать следующим требованиям:

1. Сбор данных с неопределенного числа источников с минимальными задержками и отсутствием потери информации;
2. Долговременное хранение полученной информации с возможностью ее последующего извлечения и фильтрации;
3. Удаленный доступ к данным с удаленных отправителей и их визуализация в целях анализа и формулирования выводов;
4. Возможность развертывания в облачной среде, обеспечивая гибкость и масштабируемость на основе облачных инфраструктурных решений;
5. Предоставление данных о загруженности компонентов в реальном времени для оперативного мониторинга и управления ресурсами;
6. Гибкое конфигурирование модуля, отвечающего за масштабирование системы, позволяющее динамически адаптировать мощность системы в зависимости от нагрузки;
7. Наличие удаленного доступа к контролирующему модулю для удобного управления и мониторинга системы.

Концепция программного продукта заключается в следующем: система, развернутая в облачной среде в режиме реального времени, принимает запросы от источников данных. В качестве этих источников выступают установленные на автомобилях у пользователей системы, отображающие уровень топлива, скорость, местоположение, баланс топливной карты и другие данные. В зависимости от интенсивности потока данных система должна динамически изменять свою мощность, то есть автомасштабироваться. В силу особенностей существующих на данный момент платформ, масштабирование будет производиться только в горизонтальном режиме, вертикальный вызывает неизбежную остановку обрабатывающих информацию мощностей и снижает общую производительность системы.

Отдельная подсистема будет осуществлять сбор и хранение информации о нагрузке развернутых мощностей. Она должна быть в состоянии некоторое время хранить эти данные и быть способна извлекать данные не со статически заданных конечных точек, а с изменяющегося во времени набора рабочих машин.

Так как представленные на рынке облачные решения имеют недостаточно гибкую систему автомасштабирования, эту функцию возьмет на себя дополнительный модуль. Данный модуль будет соединяться с системой мониторинга и в автоматическом режиме принимать решения о увеличении или уменьшении мощности распределенного решения. Регулирует мощность конфигурационный файл, состоящий из набора правил, определяющих необходимое количество развернутых мощностей в зависимости от текущей нагрузки.

Компонент web-консоли будет напрямую взаимодействовать с модулем масштабирования и предоставлять администратору данные, отображающие текущую нагрузку. Доступ к полученным данным должен отображаться в отдельном окне и фильтроваться по заданным полям. Также в него будет включена возможность просмотреть или изменить примененную на контролирующем модуле конфигурацию.

В свою очередь, система долговременного хранения данных будет предоставлять историческую информацию, которая поступила от источников. На основе этих данных можно будет создавать графики и проводить анализ, расхода топлива или иных требуемых метрик.

## **1.4 Выбор платформы и языка программирования для разработки системы**

На основе анализа существующих облачных платформ, была выбрана Yandex Cloud. Данное решение было принято на основании ключевых факторов при развертывании решения в облаке: полнота функционала для реализации автомастшабируемых систем, гибкое и интуитивное администрирование, возможность автоматизации управления платформой и стабильность работы компонентов системы.

Это решение полностью соответствует всем требуемым параметрам, так как уже давно вышло на рынок и имеет богатую клиентскую базу. Кроме того, отсутствуют проблемы, связанные с «молодыми» продуктами: непредсказуемые «падения» в процессе работы, периодические перебои связи с платформой и малочисленная база знаний по типовым ошибкам пользователей и операторов.

Возможность автомасштабирования системы достигается двумя способами: создание множества виртуальных машин или контейнеризация. Второй способ более предпочтительный поскольку, развертывание целой виртуальной машины — это длительный процесс, который, в то же время, будет увеличивать стоимость использования облачной платформы.

Контейнеризация означает упаковку исполнительной части системы в контейнер – легковесную ОС с размещенной внутри нее полезной нагрузкой. Преимущество использования такого подхода совместно с балансировщиком нагрузки заключается в бесшовном увеличении или уменьшении мощности распределенной системы.

В качестве ПО оркестровки контейнеризированных приложений будет выступать Kubernetes. Он обладает возможностью в режиме реального времени менять количество активных контейнеров, организовывать балансировку нагрузки между ними и их изолированное взаимодействие друг с другом. Все эти свойства позволят нам производить мониторинг нагрузки системы и автоматически регулировать ее мощность, передавая обязанности по развертыванию, контролю и маршрутизации компонентов системе оркестровки.

Главная часть системы – модуль регулирования мощности масштабируемой системы должен быть реализован с использованием наиболее современного языка программирования, имеющего широкое сообщество разработчиков. Данный выбор даст возможность в дальнейшем модернизировать его, не прибегая к длительному изучению спецификации и принципов работы языка.

Отвечает всем предъявленным требованиям язык программирования общего назначения Python версии 3.12. Он является одним из наиболее популярных на 2024 год [19], имеет богатое сообщество разработчиков и позволяет разрабатывать разноплановые программные продукты эффективно и в короткие сроки.

Взаимодействие с внутренними компонентами Kubernetes будет производиться с помощью поставляемой утилиты «kubectl». Именно она будет отвечать за передачу данных о необходимом в данный момент количестве используемых контейнеров.

Для обеспечения доступности управления разрабатываемым модулем необходимо реализовать его web составляющую. Она разделена на 2 части – frontend и backend.

Библиотека «aiohttp» позволяет backend-у веб-сервера обрабатывать запросы асинхронно. Это означает, что один поток может обрабатывать несколько запросов одновременно, не дожидаясь ответа от каждого из них.

Преимущества асинхронной обработки:

* Повышение производительности: за счет параллельной обработки запросов сервер может обрабатывать больше запросов в секунду, что приводит к снижению времени отклика и повышению производительности системы.
* Снижение нагрузки на сервер: асинхронная обработка запросов позволяет уменьшить нагрузку на сервер, что особенно важно при работе с большим количеством пользователей.
* Масштабируемость: асинхронные приложения легко масштабируются на многоядерные процессоры и серверы с большим количеством CPU, что позволяет увеличивать производительность системы по мере роста нагрузки.

Дополнительно «aiohttp» поддерживает шифрование SSL, что обеспечивает безопасность передачи данных между сервером и клиентом. SSL шифрует данные, делая их нечитаемыми для злоумышленников, даже если они имеют доступ к сетевому трафику.

SSL обеспечивает конфиденциальность данных, передаваемых между сервером и пользователем. Это особенно важно для передачи конфиденциальных данных, таких как пароли, номера кредитных карт и т.д. Также он гарантирует целостность данных, передаваемых между сервером и пользователем. Это означает, что данные не могут быть изменены или испорчены во время передачи. Дополнительно SSL может использоваться для аутентификации сервера, что позволяет убедиться, что пользователь подключается к легитимному серверу, а не к подделке, дублеру.

Что касается frontend части, которая будет отображаться в браузере администратора, необходимо создать страницы сайта, отображающие как необходимую для мониторинга информацию, так и текущую конфигурацию модуля масштабирования, с возможностью ее редактирования. Необходимо отметить, что доступ к сайту должен предоставляться только квалифицированному пользователю, то есть администратору, в целях исключения возможности некорректной конфигурации системы. Также реализация такой возможности должна быть удобной и для разработчика. Для этих целей целесообразнее всего использовать NextJS, современное решение для создания Web приложений. Его специфика позволит сделать отображение информации наиболее интуитивно понятным.

Как и в любой масштабной системе, для frontend-а нашего приложения нам потребуется надежный и производительный веб-сервер. Nginx идеально подходит для этой задачи по целому ряду причин.

Nginx использует асинхронную архитектуру, основанную на событиях, что обеспечивает его высокую производительность и масштабируемость. Это означает, что он может обрабатывать большое количество запросов одновременно, не блокируя другие процессы. В довесок данное решение сочетает в себе простоту, эффективность и стабильность – немаловажные критерии для критических сервисов, к которым относится система контроля расхода топлива транспортной компании.

Нельзя не отметить, что асинхронная природа Nginx отлично сочетается с асинхронным backend-ом, реализованным на базе библиотеки «aiohttp». Это позволяет обеспечить максимально отзывчивую работу системы для каждого пользователя. В купе с тем, что Nginx является бесплатным программным обеспечением, модуль становится доступным для использования и модификации сторонними разработчиками.

Проведя сравнение Nginx с Apache Web Server, который является его предшественником, можно выделить некоторые преимущества, в пользу первого. Во-первых, исправляя недостатки предка, Nginx имеет более понятную и простую конфигурацию, во-вторых, он использует асинхронную модель функционирования, обеспечивающую производительность и экономичность, по отношению к используемым ресурсам компьютера.

В качестве системы долговременного хранения данных выберем PostgreSQL. Она прекрасно подходит для большого потока информации и является наиболее широкопрофильной и надежной реляционной СУБД. Если сравнивать ее с иными решениями, к примеру MySQL, то на лицо будет преимущество в обеспечении атомарности запросов к СУБД и разнообразности поддерживаемых типов данных, что, в нашем случае, является немаловажным фактором. Также согласно данным в таблице 2 она имеет достаточно быстрое время отклика.

Таблица 2 – Тесты производительности PostgreSQL

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тест № | Количество клиентов | Время теста (с) | Количество транзакций | TPS (транзакции/с) | Среднее время задержки (мс) | Максимальное время задержки (мс) | Примечания |
| 1 | 10 | 60 | 12000 | 200.00 | 5.0 | 10.0 | Начальные настройки |
| 2 | 20 | 60 | 23000 | 383.33 | 5.2 | 12.5 | Увеличено количество клиентов |
| 3 | 50 | 60 | 55000 | 916.67 | 5.5 | 15.0 | Высокая нагрузка |
| 4 | 100 | 60 | 100000 | 1666.67 | 6.0 | 20.0 | Максимальная нагрузка |
| 5 | 10 | 120 | 24000 | 200.00 | 5.1 | 10.2 | Продленный тест |
| 6 | 20 | 120 | 46000 | 383.33 | 5.3 | 12.7 | Продленный тест с увеличенной нагрузкой |
| 7 | 50 | 120 | 110000 | 916.67 | 5.6 | 15.5 | Продленный тест с высокой нагрузкой |
| 8 | 100 | 120 | 200000 | 1666.67 | 6.1 | 21.0 | Продленный тест с максимальной нагрузкой |

# **2 ПРАКТИЧЕСКАЯ ЧАСТЬ**

## **2.1 Разработка архитектурных решений и реализация собственной системы на языке программирования Python 3 и TypeScript**

Разберём, как необходимо спроектировать систему, чтобы она реализовывала функционал, приведённый в анализе источников, касающихся современных систем мониторинга и контроля расхода топлива. К тому же, нужно, чтобы система была расширяема. В параграфе 1.3 были выведены требования к программной системе. Разработаем в соответствии с ними архитектуру нашей системы.

Для начала обратимся к документации разработчиков библиотеки aiohttp. На странице «быстрого старта» [11] мы можем найти пример развертывания самого примитивного web-сервера, принимающего HTTP-запросы. Так как библиотека aiohttp является асинхронной, мы «из коробки» получаем достаточную для высоконагруженного сервиса производительность, что в случае с многопоточным вариантом, было бы затруднительно [12]. Данное решение предназначено, преимущественно, для получения и ответа на http запросы, однако разработчиками предоставлена возможность параллельно с основной задачей выполнять дополнительные [13]. Используем это свойство на этапе реализации контролирующего модуля.

Система спроектирована так, что для ее работоспособности должны быть активны все остальные компоненты: Kubernetes, Prometheus и PostgreSQL. Для работы с ними предусмотрены соответствующие библиотеки на языке python. Как известно, вышеописанные системы предоставляют REST-API интерфейс [14,15, 18] и можно обойтись без использования библиотек, но оперирование программными сущностями значительно упрощает разработку и избавляет от излишних ошибок.

Проанализировав схему работы разрабатываемой системы, изображенной на рисунке 8,

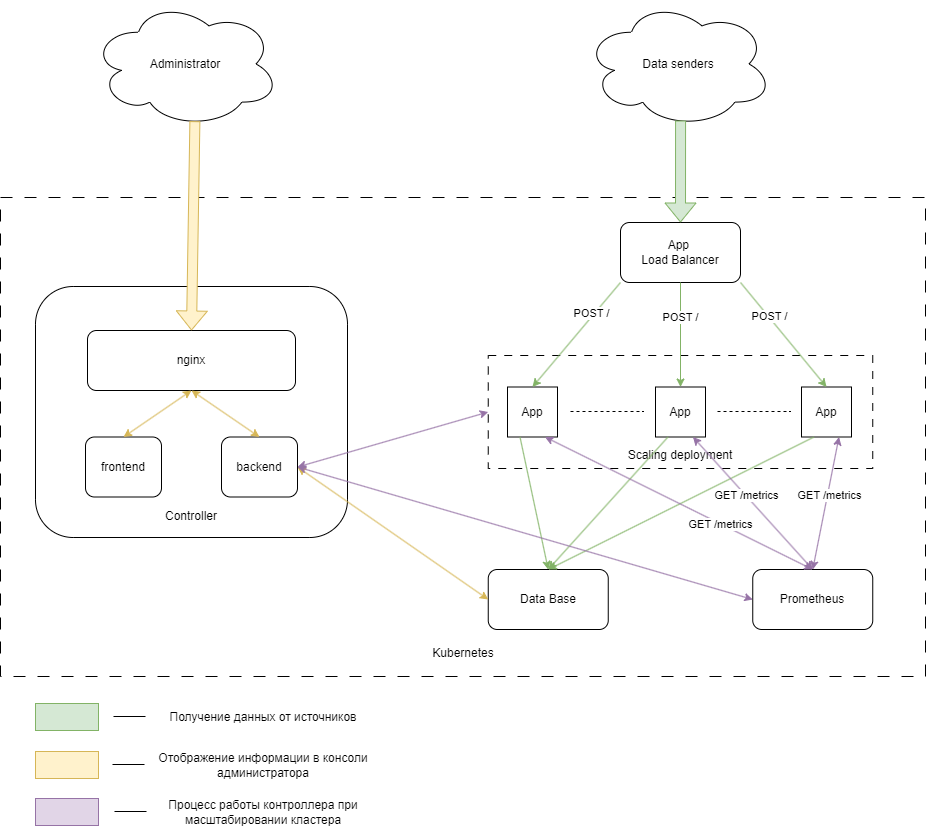


Рис. 8 – Схема работы разрабатываемой системы

начнем с компонента, отвечающего за полезную нагрузку.

В его обязанности входит обработка потока данных от удаленных источников и запись полученной информации в СУБД. В силу того, что мы придерживаемся концепции асинхронного приложения, взаимодействие с СУБД будет проводиться аналогичным образом, с помощью библиотеки asyncpg. Мы будем выделять пул соединений с базой данных и по мере необходимости запрашивать их из него. Такая необходимость возникает в силу того, как работает выбранная нами СУБД – PostgreSQL. Через одно соединение в момент времени может выполняться только одна операция [19], а мы утверждаем, что к нашему приложению может обращаться большое количество пользователей за раз.

Также, нельзя забывать о том, чтобудет производиться мониторинг именно его нагрузки. Prometheus сам по себе не ставится на каждый компонент, для которого необходимо собирать метрики, поэтому нам достаточно на каждом из требуемых элементов запустить web-сервер и настроить метод для получения необходимых данных. В нашем случае, web-сервер — это основная функция компонента, так что достаточно воспользоваться библиотекой aiohttp\_prometheus\_exporter, которая отлично подходит для отслеживания нагрузки на сервер, работающий с использованием aiohttp. Библиотека собирает данные о количестве http запросов и использовании процессорного времени – как раз те метрики, которые мы используем для регулирования количества развернутых мощностей.

Вторым шагом разработаем компонент контроллера. Его можно логически поделить на две части. Первая будет отвечать за отображение состояния контролируемой системы и возможность конфигурирования модуля, а вторая, в свою очередь, за непосредственное регулирование количества развернутых в кластере мощностей. Несмотря на то, что они логически разделены, между ними существует внутренние связи, в виде соединения с СУБД, Prometheus и управляющим узлом Kubernetes.

Реализация первой части схожа с компонентом полезной нагрузки - Web-сервер с набором RESP-API методов, но в данном случае их количество будет больше. Опираясь на описанные выше требования, составим список таких методов:

1. GET /config – получение текущей конфигурации модуля;
2. POST /config – изменения текущей модуля на предоставленную;
3. GET /requests\_stat – получение данных о количестве запросов в секунду на данный момент на каждом «поде»;
4. GET /cpu\_stat - получение данных об использовании процессорного времени на данный момент каждым «подом»;
5. GET - /pods\_count – отображение числа развернутых на данный момент времени «подов»;
6. GET - /auto\_info – извлечение из базы данных полученной от удаленных источников информации, в соответствии с заданными параметрами.

В силу того, что мы собираемся размещать backend на одной машине с frontend, для каждого из описанных выше методов необходимо добавить префикс «/api/v1».

Переходя ко второй части, необходимо отметить, что алгоритм ее работы опирается ровно на те же самые данные, что получает в web-консоли администратор системы. Это нам дает прозрачность ее функционирования и возможность настройки работы с той точностью, которая будет необходима.

Ранее была упомянута конфигурация контролирующего модуля. Данные, которые в ней находятся будут предопределять какие значения метрик, с какой частотой и приоритетом выбора будут влиять на масштабирование системы. Структура конфигурации включает в себя: шаг масштабирования, для компенсации скачкообразных изменений, политику выбора количества развернутых мощностей – меньшее, большее или среднее, время между опросами системы и три набора правил для количества запросов в секунду, используемого процессорного времени и текущей даты соответственно. Первые два работают по принципу таблицы: если значение метрики больше или равно определенной позиции, то выбираем заданное для нее значение рабочих узлов. Третьи же правила являются шаблонами для текущей даты. Допустим, если мы хотим, чтобы в 2024 году в июне у нас было определенное количество развернутых мощностей, то указываем в конфигурации значение года и месяца. В правиле дату можно конкретизировать вплоть до минуты. К сожалению, унифицированную, предопределенную конфигурацию предоставить невозможно, так как использование аппаратных мощностей и предполагаемое количество запросов к системе в секунду будет разниться. Пример конфигурации приведен на рисунке 9.



Рис. 9 – Пример конфигурации контролирующего модуля

Предпоследним шагом будет frontend составляющая. Используя выбранный ранее фреймворк NextJS, создадим две рабочие страницы нашей web-консоли. На главной странице в виде графиков будут располагаться все отображаемые контролирующим модулем метрики и в виде таблицы с настраиваемыми фильтрами полученные от удаленных источников данные. Все они будет разделены по соответствующим вкладкам. На второй странице разместим работу с конфигурацией. Можно будет как увидеть ее актуальное состояние, так и отредактировать его. Корректность передаваемых данных проверяется в режиме реального времени во встроенном редакторе. Для визуального оформления страниц будем использовать библиотеки MaterialUIи ReChartsдля таблиц и графиков соответственно. На рисунках 10 и 11 можем увидеть html страницы с реализованным функционалом.

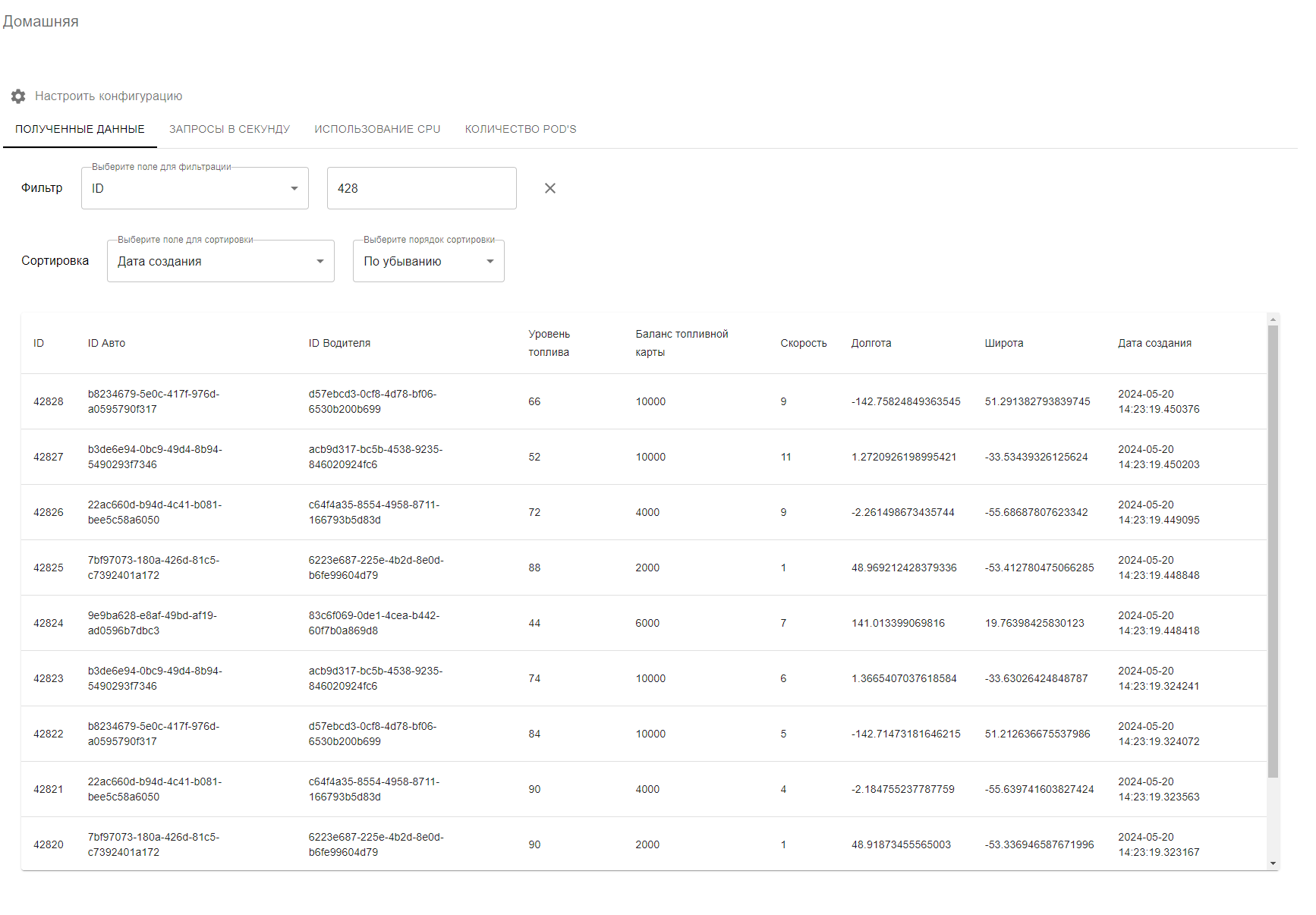


Рис. 10 – Главная страница



Рис. 11 – Страница конфигурации

На данном этапе последним шагом будет являться настройка nginxweb-сервера и упаковка всех разработанных компонентов в Docker**-**образ.

В случае nginx его настройка заключается в организации доступа к статическим html файлам и проксирование запросов к REST-API на реализованный backend. Оба этих пункта мы указываем в его файле конфигурации frontend.conf.

Упаковка компонентов в Docker-образы представляет собой создание минимально рабочей версии операционной системы и копирование в нее созданной полезной нагрузки. В нашем случае мы используем образ, включающий в себя python 3.12 и web-сервер nginx. Путем перемещения на него необходимых файлов и настроек получим готовые контейнеры, которые можно использовать в нашей системе.

## **2.2 Настройка облачной среды Yandex Cloud для разворачивания приложения**

Как и в каждой современной облачной среде, ее пользователю необходимо завести персональный аккаунт, и привязать к нему платежную систему. Стоит заметить, что новым пользователям выдается пробный денежный лимит, который можно потратить на изучение платформы. В рамках настоящей работы эти шаги подробно рассматриваться не будут в виду достаточно богатой документации самой платформы и тривиальности производимых действий. Далее нам необходимо найти те компоненты, которые нам будут необходимы.

Так как в качестве платформы для размещения контейнеров был выбран Kubernetes, следует найти соответствующий ему компонент облачной среды. Managed Service for Kubernetes – то, что нам необходимо. В процессе настройки среды будут настроены и включены дополнительные элементы платформы, но, преимущественно, это будет произведено в автоматическом режиме без нашего участия.

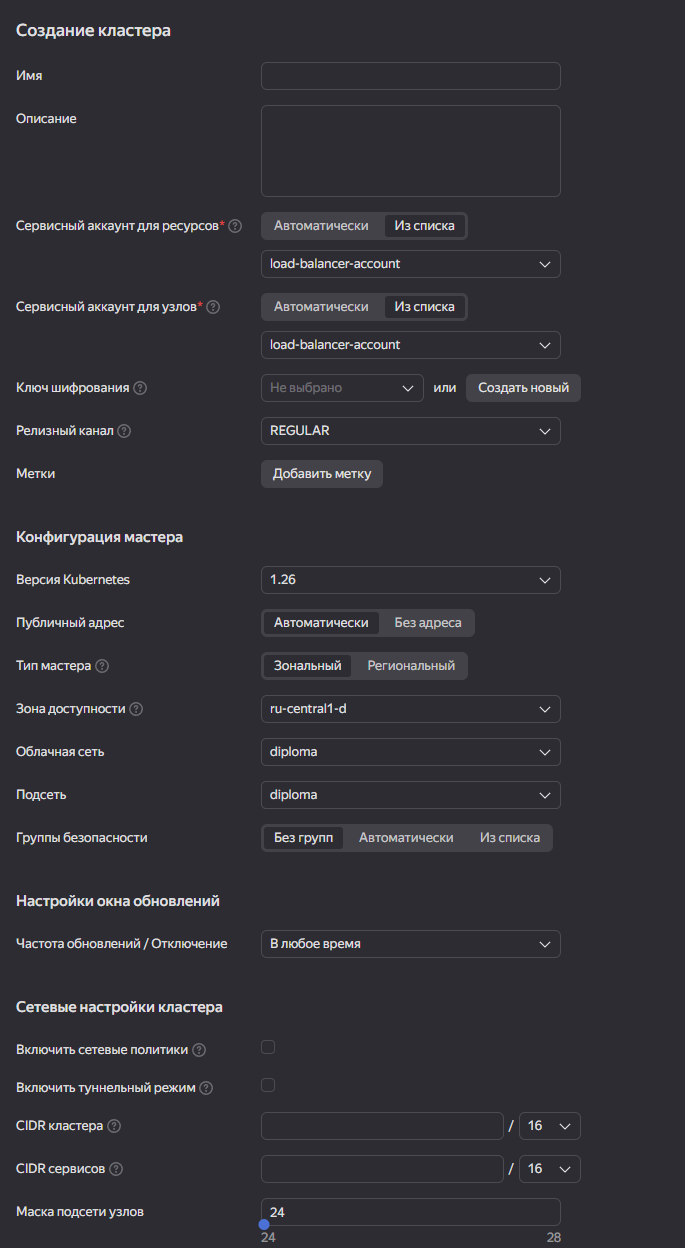


Рис. 12 – Форма создания кластера Kubernetes

На странице создания кластера Kubernetes, приведенной на рисунке 12, нас встречает множество параметров. В рамках нашей задачи большинство можно оставить в значении по умолчанию или «создать новый». Обратить внимание необходимо на пункты «Публичный адрес» и «CIDRКластера/Сервисов». Первый должен быть в положении – Автоматически, так как нам необходим доступ к кластеру извне. Второй и третий пункты отвечают за подсети разворачиваемых ресурсов, один из возможных вариантов значений – 10.1.0.0/16 и 10.2.0.0/16. Таким образом мы получим публичный адрес для нашего кластера и сбалансированные по количеству хостов пулы адресов, в которых будут размещаться наши рабочие мощности.



Рис. 13 – Значение для публичного адреса

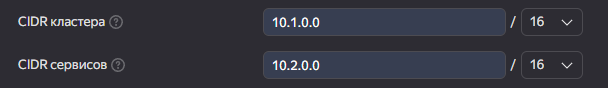


Рис. 14 – Значение для CIDR кластера/сервисов

После заполнения формы и подтверждения создания кластера, дожидаемся пока платформа закончит обработку нашего запроса. Индикатором этого будет являться статус – «Running» и состояние – «Healthy». Далее сразу же возникает следующая задача, заключающаяся в создании группы узлов. Эта сущность является отображением реально используемых физических мощностей для нашей системы.

Нельзя не отметить, что в варианте размещения кластера Kubernetes на своих серверах данный этап лег бы на плечи администратора системы, все создания, изменения и удаления физических машин производились бы вручную. Это является очередным плюсом в пользу использования облачных сред.

Опираясь на вышесказанное, переходим к форме создания группы узлов. На данном этапе необходимо тщательно проанализировать потребности будущей системы: минимальное и максимальное количество узлов, используемая мощность процессора, постоянная и оперативная память, все эти ресурсы, в зависимости от запрашиваемого количества, будут иметь разную цену. В рамках тестового проекта строго рекомендуется оставить все параметры в минимальном значении. Поле публичный адрес необходимо также перевести в состояние – «Автоматически».

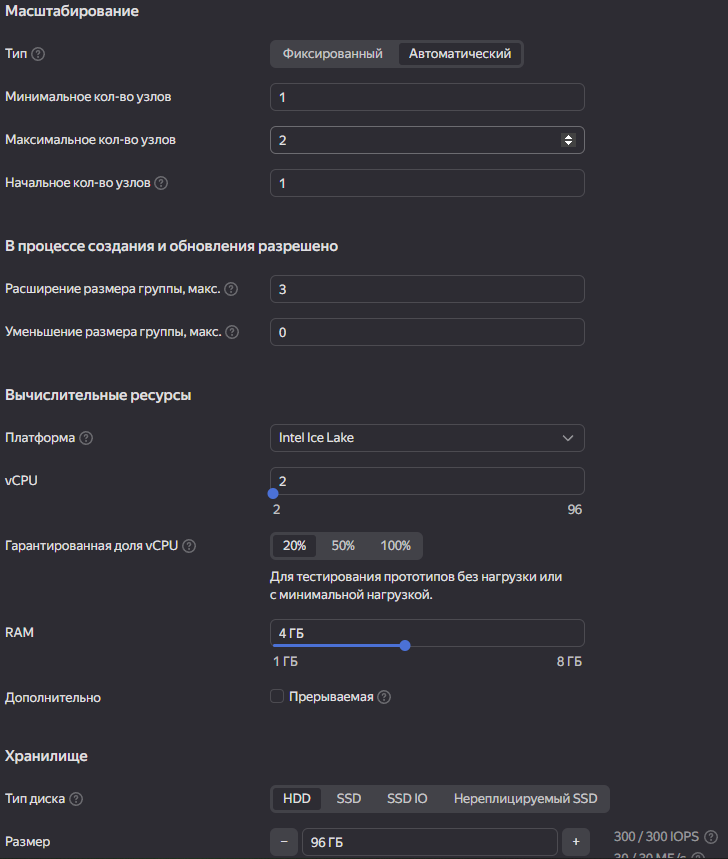


Рис. 15 – Рекомендуемые параметры для группы узлов

Дополнительно разработчики Yandex Cloud рекомендуют добавить приложение **«**ExternalDNS c плагином для Yandex Cloud DNS», через встроенный в web-консоль управления «Marketplace». Данное дополнение снизит поток трафика внутри кластера и будет как разгружать ресурсы, так и экономить деньги, которые взимаются за них.

## **2.3 Разворачивание приложения в облачной среде Yandex Cloud**

Убедившись, что создание кластера прошло успешно и он готов для выполнения своей целевой функции, мы можем приступить к непосредственному размещению системы в нем. Необходимо помнить, что только развертывание всех компонентов и корректная настройка всех элементов системы позволит ей правильно функционировать. Также, в силу выбранного метода работы системы, все компоненты, о которых пойдет речь ниже, находятся в Dockerконтейнерах и все манипуляции с ними производятся с помощью специальный yaml-файлов, описанных в коде проекта. Во всех дальнейших действиях этот факт подразумевается.

Для всех манипуляций, производимых ниже, на машине, с помощью которой будет производиться процедура, необходимо установить и настроить две программы: kubectl – консольная утилита для администрирования кластера Kubernetes и helm – его пакетный менеджер. С их помощью мы сможем выполнить все необходимые действия быстро без затруднений.

На данном этапе мы будем, как использовать элементы, созданные ранее, так и добавлять новые. Связано это с тем, что Prometheusи PostgreSQL, задействованные сторонние системы, разворачиваются из заготовленного шаблона или публичного Docker**-**образа, а после уже настраиваются под наши нужды.

Начнем с PostgreSQL. Так как это СУБД, то каких-либо программных зависимостей она иметь не может, ведь хранит в себе только информацию и является конечной точкой потока данных. В силу того, что мы работаем в рамках кластера, у развертывания СУБД есть одна особенность – необходимость в статическом хранилище данных. Если информация в базе данных будет находиться в самом «поде», то при его удалении они утратятся [16], что, очевидно, нас не устраивает. Данную проблему решает хранилище, по сути своей пространство, зарезервированное на одном из физических узлов кластера, которое, в независимости от жизненного цикла ‘пода’, оставит наши данные в сохранности.

На данном этапе мы создадим хранилище размером 4 гигабайта на одном из узлов кластера по пути «/devkube/postgresql». В рамках использования для системы мониторинга расхода топлива, такого хранилища при потоке а 10000 записей в день хватит на 10 лет. После того, как оно создано, занесем в кластер информацию об этом. Выполнив все эти пункты можем переходить к самому развертыванию PostgreSQL.

Для упрощения процесса развертывания воспользуемся пакетным менеджером **Helm**. Для выполнения требуемых действий введем команду:  
«helm install dev-pg bitnami/postgresql --set primary.persistence.existingClaim=pg-pvc,auth.postgresPassword=pgpass».  
В случае успеха, в списке «подов» появится наша СУБД.



Рис. 16 – развернутая PostgreSQL

Следующим шагом будет размещение части системы, которая будет подвержена масштабированию в процессе работы. Обращаясь к параграфу 1.3, можем увидеть, что она представляет собой web-сервер с одним API методом, обращающийся к базе данных. Так как для внешних приложений количество реплик, развернутых в момент времени неизвестно, их необходимо объединить, общей точкой. В качестве нее выступит балансировщик нагрузки – лучшее и единственное решение в данном случае. Он гарантирует достаточно равномерное распределение нагрузки [16], между узлами системы, а также предоставит внешний IP адрес для нее.

Далее перейдем к системе, осуществляющей мониторинг нагрузки масштабируемой части. Учитывая, что развертывание Prometheusпроисходит из готового образа, зададим расположение внутренней базы данных, время ее ротации и аппаратные ограничения использования процессора и оперативной памяти, как «/prometheus/», «12h», «1» и «1Gi» соответственно.

В силу принципа своей работы, Prometheus, должен фильтровать, с каких компонентов системы ему забирать информацию, а с каких нет. Существует множество различных конфигураций и вариантов настройки, даже уведомление о достижении порогового значения метрики, но нам необходимо делать выборку исключительно для нашей системы, поэтому создадим одно единственное правило на забор информации – собирать данные с «подов» кластера, у которых значение prometheus.io/scrape равняется «enable».

После всех вышеперечисленных действий, перейдем к самому главному компоненту – контролирующему модулю. Именно для его работы, необходимы все предыдущие компоненты системы. Сам по себе он представляет nginx web-сервер, который либо предоставляет доступ к web-консоли, либо к методам API. Так как он будет представлен в одном единственном экземпляре мы можем напрямую создать сущность в кластере и предоставить доступ к ней извне. При разворачивании данного компонента есть необходимость связать его со всеми предыдущими. Самым быстрым и оптимальным способом будет занесение в переменные среды адреса других компонентов:

1. PROMETHEUS\_HOST – адрес Prometheus сервера;
2. PG\_HOST – адрес СУБД PostgreSQL.

Получить их можно с помощью команды:  
«kubectl get pods -o wide»,  
запущенной на машине, с которой производится процедура развертывания.

Для того, чтобы убедиться, что процесс прошел успешно мы должны, для начала, проверить, что в списке «подов» появился созданный нами ранее контроллер. Получить эту информацию можно с помощью команды, описанной выше.

Вторым шагом проверим корректность работы nginxweb-сервера. Для этого напрямую подключимся к «поду» контроллера с помощью команды:  
«kubectl exec --tty --stdin controller-pod -- /bin/bash»,   
введенной в консоль. После получения доступа к удаленному терминалу, выполним команду:  
«service nginx status»,  
в случае успеха мы должны увидеть сообщение об успешной работе сервиса nginx(см. Рис. 17).



Рис. 17 – Статус работы сервиса nginx

Учитывая то, что доступ к консоли администратора должен быть извне кластера, создадим сервис, который нам его предоставит. Используем LoadBalancer. Несмотря на то, что его основной целью является распределение нагрузки, он также имеет внешний IP адрес, что как раз, в нашем случае, является необходимы требованием. Подключив созданный сервис напрямую к «поду» контроллера, введем в консоль на локальной машине следующую команду:  
«kubectl get services»,  
в ее выводе найдем созданные ранее сервис и его EXTERNAL-IP. Именно по этому IP адресу мы будем обращаться к нашей web-консоли.

Введя в любом web-браузере в строку поиска полученный адрес, мы попадем на главную страницу администрирования. Последним шагом проверки работоспособности контроллера будет просмотр всех страниц и вкладок сайта и обнаружение на них соответствующих данных.

Для начала рассмотрим главную страницу. На ней мы можем увидеть 4 вкладки: полученные данные, запросы в секунду, использование CPU и количество POD’s. На первой должна отображаться пустая таблица (см Рис. 18). На второй график с временной линией, находящейся в положении ноля (см. Рис. 19). На третьей также должен присутствовать график (см Рис. 20). Четвертая вкладка представляет собой временную линию такого же формата, как и на предыдущих двух (см Рис. 21).

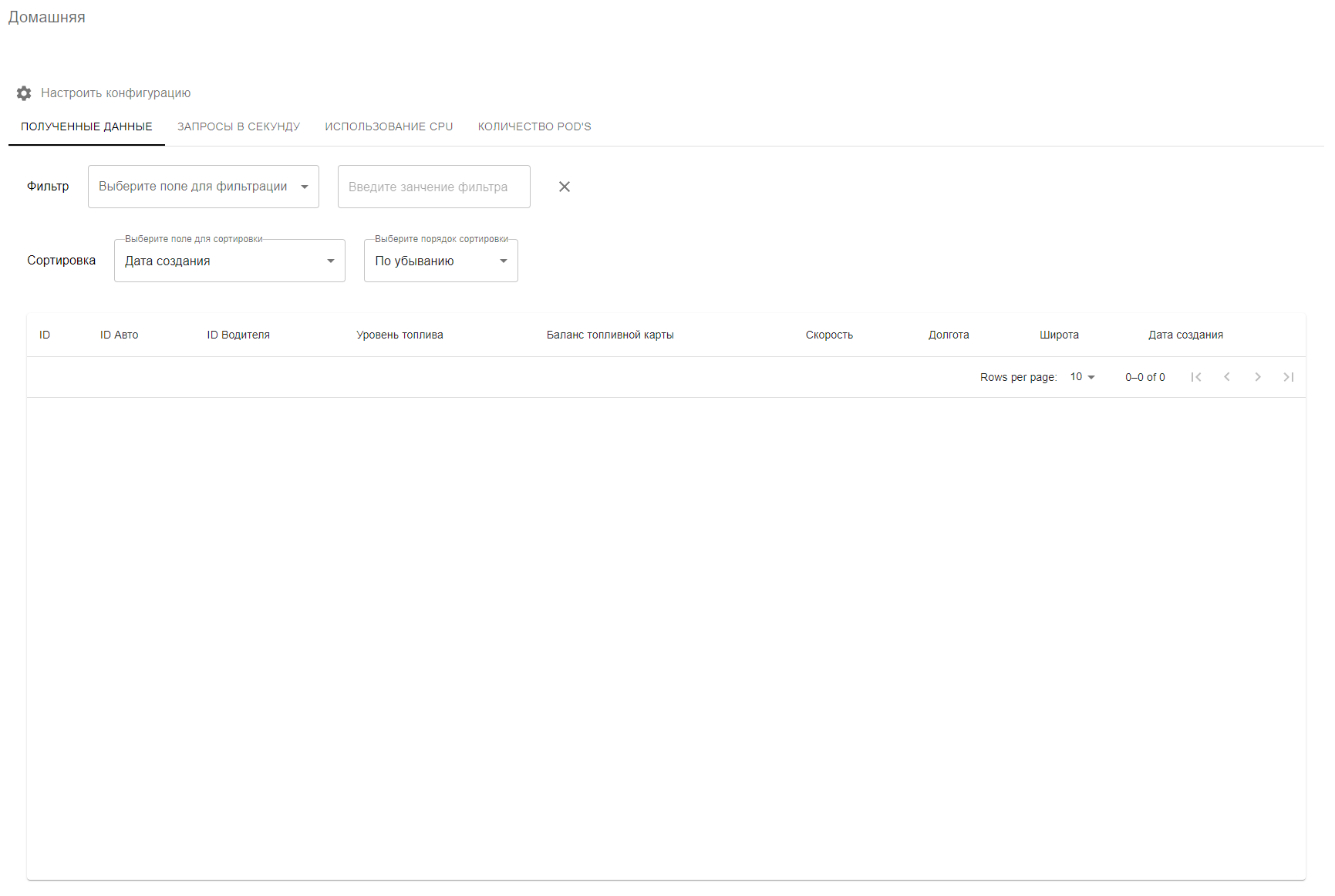


Рис. 18 – Изначальный вид вкладки «Полученные данные»

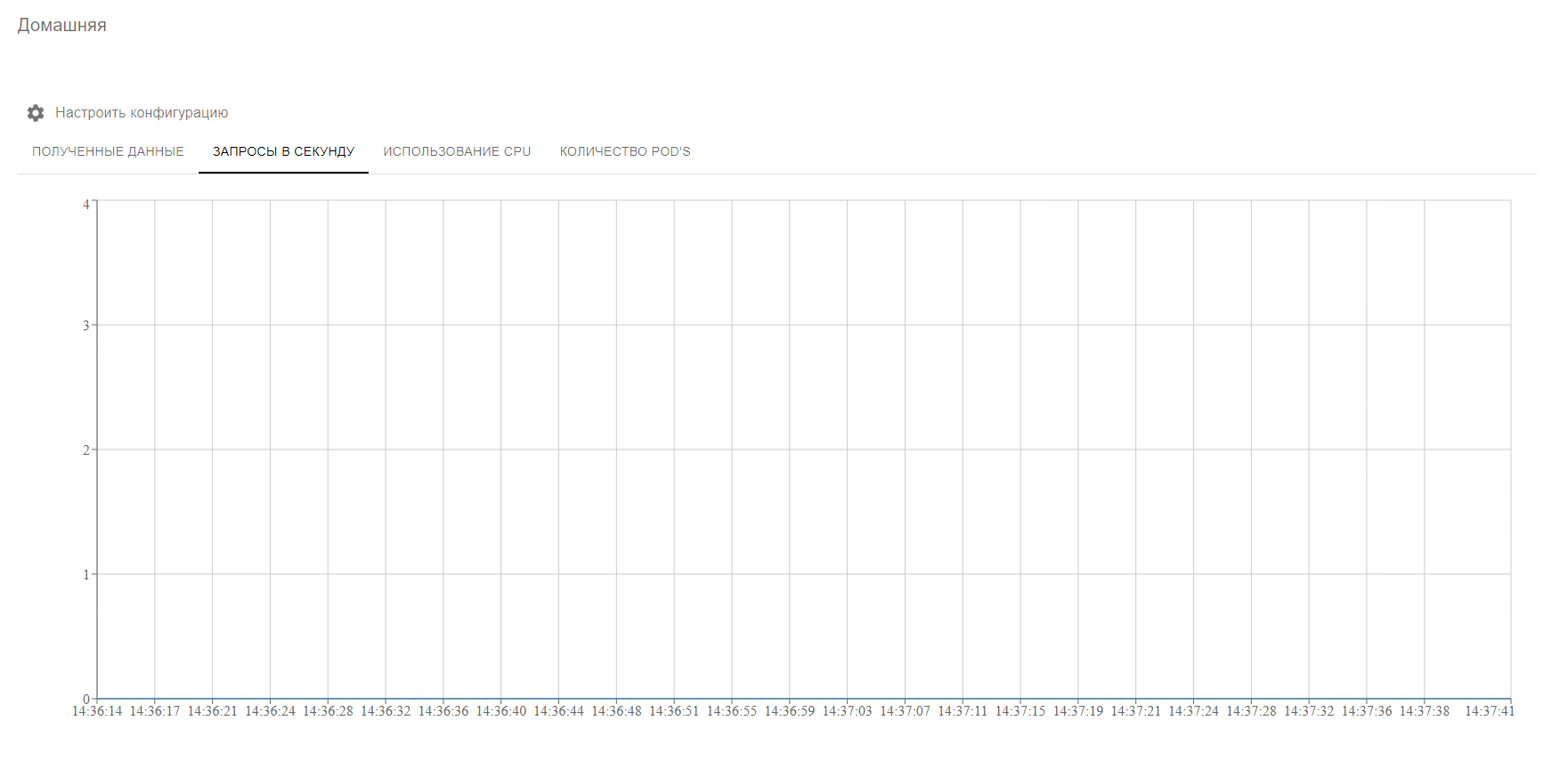
****

Рис. 19 – Изначальный вид вкладки «Запросы в секунду»

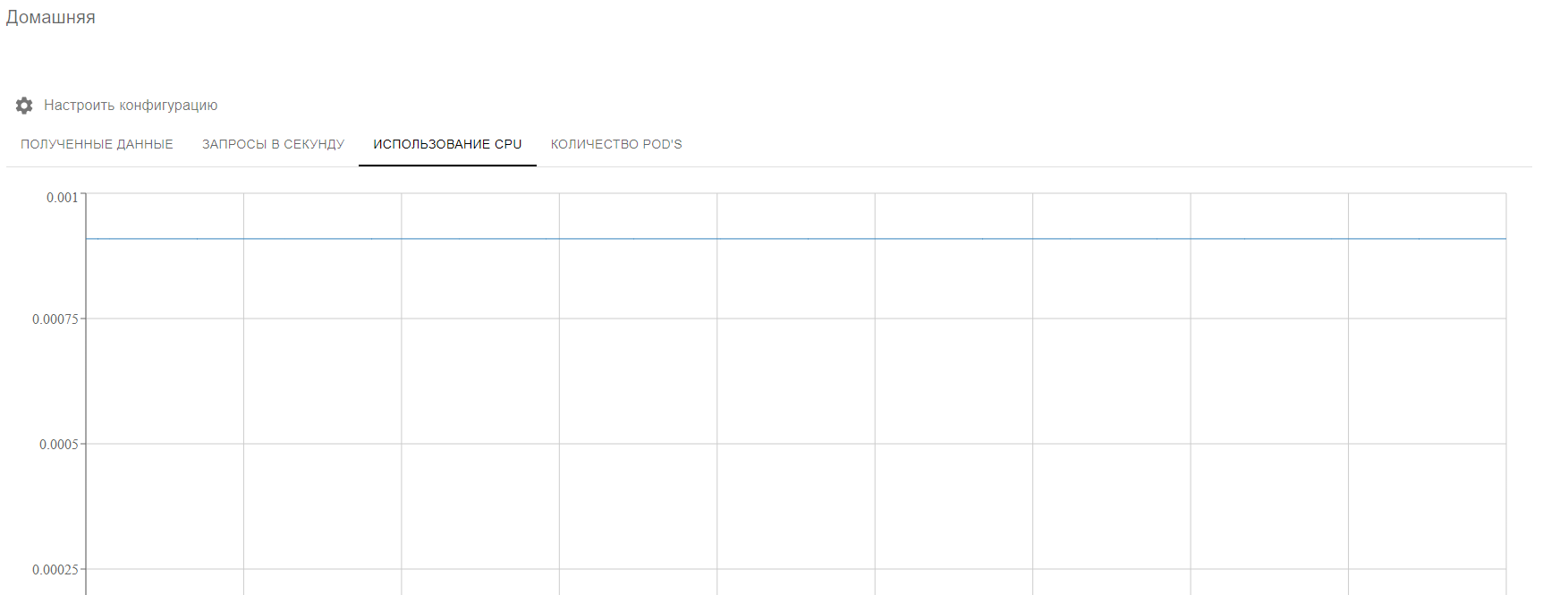
****

Рис. 20 – Изначальный вид вкладки «Использование CPU»



Рис. 21 – Изначальный вид вкладки «Количество POD’s»

Следующим шагом будет проверка работоспособности страницы настройки конфигурации. Попав на нее после нажатия на главной странице кнопки «Настроить конфигурацию», убедимся в наличие текста в json-формате (см Рис. 22).



Рис. 22 – Изначальный вид страницы конфигурации

Выполнив все вышеперечисленные действия, мы можем утверждать, что все компоненты нашей системы были успешно развернуты и система готова к работе.

Последним шагом будет расчет максимальной и минимальной нагрузки, которую наше приложение может обрабатывать, то есть какое именно количество автомобилей и с какой интенсивностью могут отправлять данные, чтобы наша система корректно их обрабатывала и использовала для этого минимально возможное количество ресурсов.

В случае минимальной нагрузки, она может составлять ноль запросов в минуту и ноль активных автомобилей. При таком сценарии система, скорее всего будет считать, что необходимо задействовать нулевое количество мощностей, но, так как, согласно принципу ее работы необходим хотя бы один рабочий узел, то именно в количестве одной штуки он будет присутствовать.

При прогнозировании максимальной допустимой нагрузки на систему, необходимо понять, какие части какую нагрузку могут выдержать. Часть, отвечающая за балансировку потока данных, может хранить в себе около 38 миллионов записей [22]. Это значит, что мы можем развернуть именно такое количество узлов для обработки входящих данных. Очевидно, что предел достижим только теоретически, такое количество автомобилей, посылающих данные одновременно достаточно маловероятно для топливо снабжающей компании.

Следующая часть, кластер web-серверов с полезной нагрузкой динамически масштабируется в зависимости от потока данных и ограничен выделенным на него пулом IP адресов. В случае заданной нами конфигурации облака, это 65534. Но, зная, что при настройке облачного решения было указано, что могут быть созданы только два рабочих узла — это количество снижается до 220 штук, согласно ограничениям платформы. Учитывая факт, что один сервер обрабатывает сразу несколько соединений за раз, здесь мы тоже не можем рассчитать допустимый предел.

Переходя к расчету допустимой нагрузки для базы данных, обратимся к таблице 2. Мы можем видеть линейный рост времени ее ответа при увеличении количества транзакций в секунду. Составим уравнение прямой по максимальной и минимальной задержкам:

(200 т/c; 5 м/с), (1666 т/с; 6.1 м/с),

по формуле канонического уравнения прямой получим . При допущении максимальной задержки в 100мс получим 126809 клиентов одновременно. Беря во внимание размер хранилища в 4 Гб, который мы выделили ранее, рассчитаем время, которое система выдержит при такой нагрузке.

Переведем гигабайты в байты 4 Гб это 4294967296 байт, зная, что одна запись, полученная от автомобиля, весит 104 байта и отправляется она раз в 100 мс, получим количество раз, сколько такое число клиентов сможет записать свои данные:

4294967296 / (126809 \* 104) / 100 = 3.25669,

Это означает, что система сможет корректно функционировать около трех минут.

Рассчитывая максимальную нагрузку, мы предполагали, что данные с автомобилей передаются моментально и нет дополнительных задержек, но беря во внимание возможные изменения поведения отдельных компонентов системы при получении пиковых нагрузок, будем ориентироваться на данное значение как достоверное.

## **2.4 Представление результатов исследования, связанных с способностью программной системы масштабировать вычислительные ресурсы приложения**

Для того, чтобы проверить возможности системы по автомасштабированию необходимо провести ее стресс-тест. В силу отсутствия реальных удаленных источников, а именно автомобилей, находящихся на данный момент в пути, данных сымитируем их, реализовав генератор данных.

Генератор будет содержать в себе объект автомобиля со следующими атрибутами:

1. ID автомобиля;
2. ID водителя;
3. Баланс топливной карты;
4. Текущая скорость;
5. Уровень топлива в баке;
6. Значение широты;
7. Значение долготы.

Первые два атрибута создаются один раз при старте генератора, а остальные случайно изменяются в пределах области допустимых для них значений. Баланс топливной карты, текущая скорость, уровень топлива в баке должны быть больше ноля, а значения широты и долготы от минус девяноста до плюс девяноста и минус ста восьмидесяти до плюс ста восьмидесяти соответственно.

Один экземпляр такого генератора создает шестьдесят запросов в минуту. Запустим сто таких реплик.

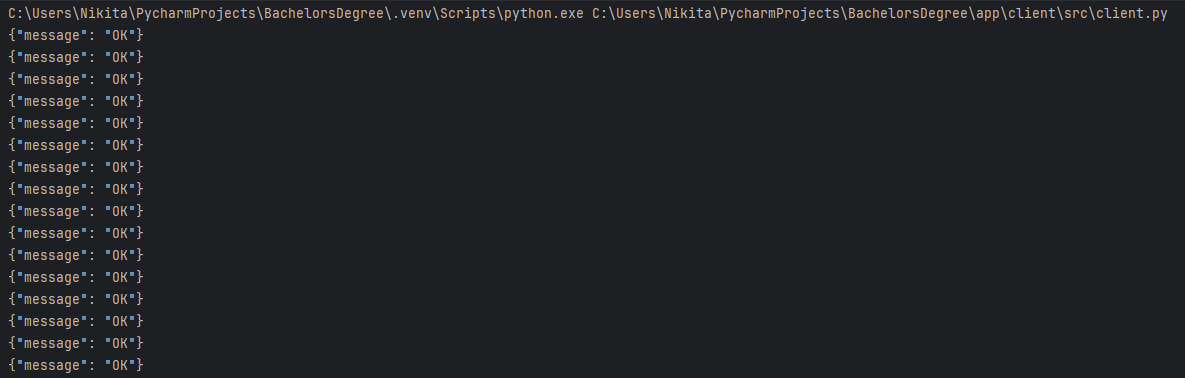


Рис. 23 – Работа одного генератора данных

В консольном выводе можем увидеть, что система корректно обрабатывает поток данных и сообщает об успешном завершении операций. Но для нас это не является главным показателем. Чтобы убедиться в заявленном функционале системы, рассмотрим предоставляемые ею метрики.

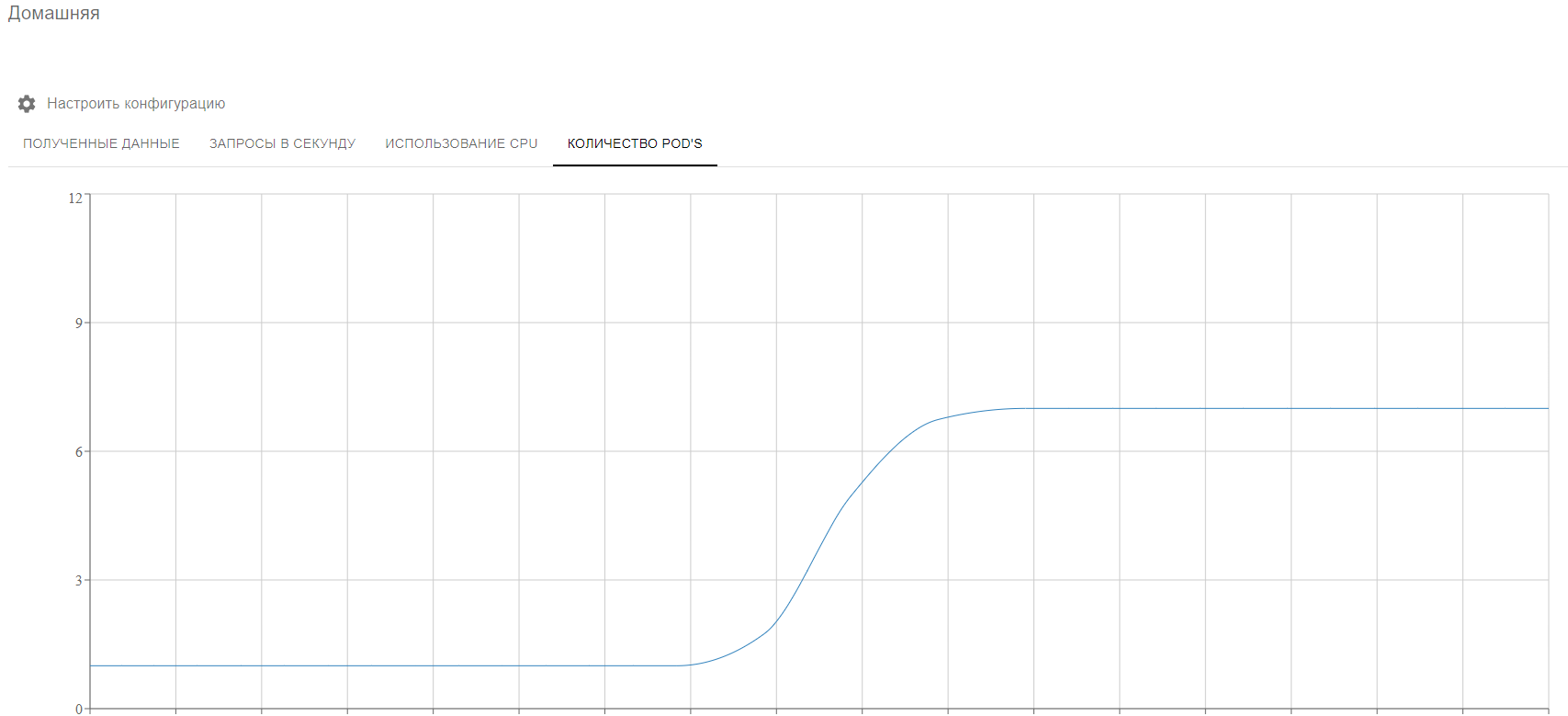


Рис. 24 – Увеличение количества «подов» при возросшей нагрузке



Рис. 25 – Нагрузка на систему при увеличении входящего потока данных

На первом графике мы видим, как система, заметив рост нагрузки, отреагировала на это увеличением количества развернутых «подов». Анализируя второй график, мы можем заметить, что дополнительно развернутые мощности забрали на себя нагрузку и распределили ее между собой. Неравномерность распределения объясняется принципом работы протокола HTTP: каждый клиент через балансировщик нагрузки устанавливает соединение с выбранным для него узлом и отправляет данные именно ему, в свою очередь, новые соединения распределяются на вновь созданные «поды», что мы и можем наблюдать на графике.

Отключим несколько экземпляров генератора данных для демонстрации возможностей системы по снижению количества задействованных мощностей.

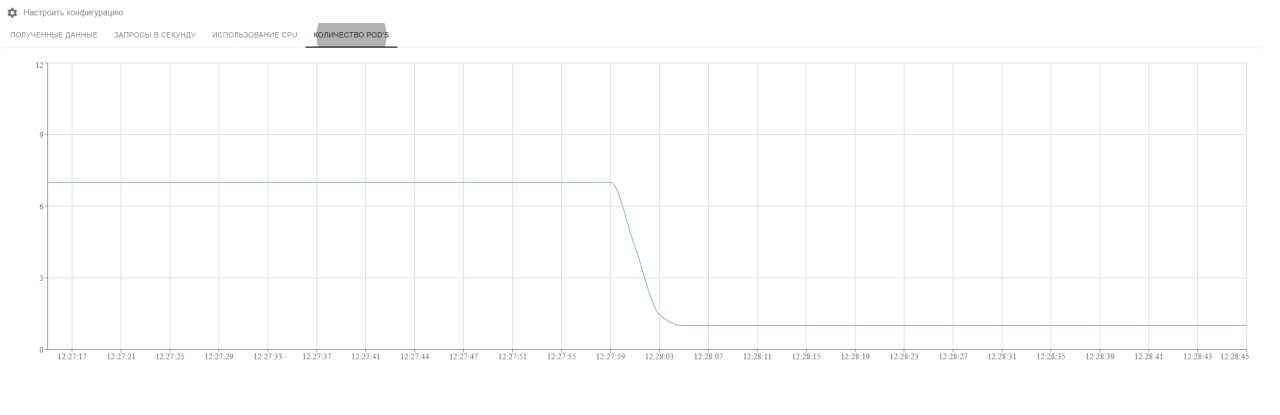


Рис. 26 – Уменьшение количества «подов» при спаде нагрузки

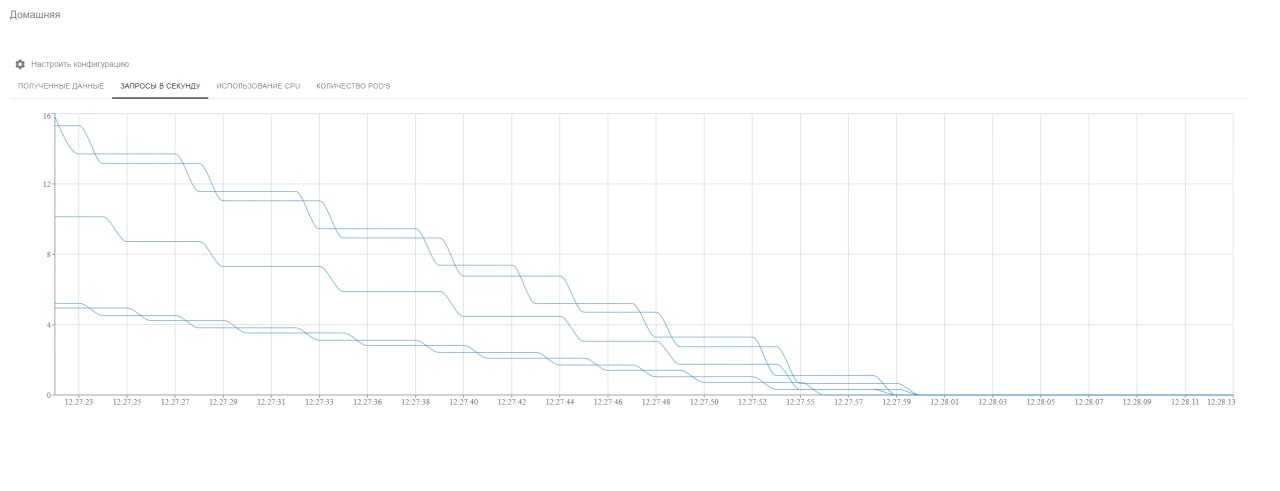


Рис. 27 – Снижение входящего потока данных

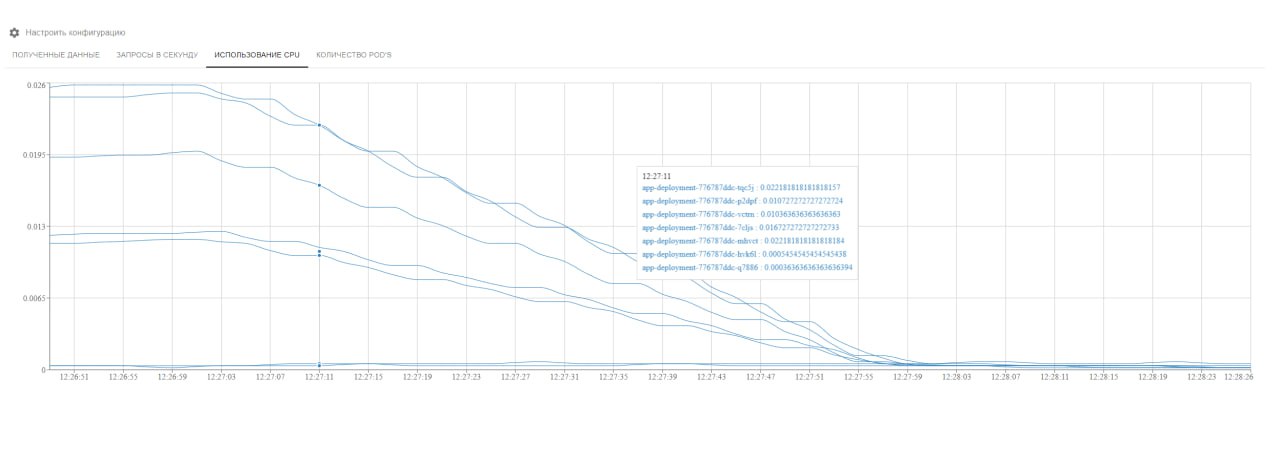


Рис. 28 – Снижение загруженности используемых мощностей

На двух последних графиках (см Рис. 27, 28) мы видим уменьшение значений загруженности, и система, вслед за ними, незамедлительно сокращает системные ресурсы, выделенные на обработку потока данных.

Проведя тест на способность системы к автомасштабированию в условиях переменной нагрузки мы увидели, что она соответствует всем заданным требованиям и выполняет свою целевую функцию – отказоустойчивость при росте интенсивности входящей информации и экономичность при ее снижении.

# **ЗАКЛЮЧЕНИЕ**

По результатам исследования и разработанному комплексу программ, можно сделать следующие выводы:

• Повышена надежность узлов распределенной информационно-вычислительной системы при минимизации объема аппаратных и вычислительных ресурсов этой системы в автоматическом режиме;

• Разработана математическая модель функционирования узлов высоконагруженной распределенной информационно-вычислительной системы;

• Разработан метод накопления консолидированной информации и прогнозирования состояния кластера серверов обработки данных;

• Разработана методика прогнозирования нагрузки на кластерную вычислительную систему в пределах краткосрочной перспективы;

• Разработан подход к оптимизации производительности вычислительного кластера, позволяющий регулировать количество процессинговых серверов в зависимости от текущей нагрузки в автоматическом режиме;

• Разработаны алгоритм и комплекс программ, предназначенные для нагрузочного тестирования самооптимизирующегося вычислительного кластера.

В добавок, можно выделить несколько ключевых моментов, на которых стоит заострить внимание. Во-первых, современные реалии требуют унифицированных и компактных решений, которые могли бы быть использованы где угодно с минимальными затратами на адаптацию, а лучше с их отсутствием, как таковых, во-вторых, только крупные многомиллионные компании могут позволить себе обширный парк вычислительных мощностей. Рассматривая примененные в работе концепции, принципы и технологии, мы можем заметить, что именно два этих пункта в ней были ключевыми. Контейнеризация решает проблему унифицированности – нам необходимо иметь Docker-образ и среду для его исполнения, никаких дополнительных затрат на адаптацию, размещение и интеграцию нет. В свою очередь, использование облачных платформ избавляет компанию от необходимости тратить денежные и человеческие ресурсы на поддержку работоспособности своих собственных серверов, что в нынешних условиях может составлять ощутимую часть бюджета.

В добавок, если внимательно изучить схему работы всей программной системы, то можно увидеть, что она логически разделена на несколько модулей и, что самое важное, они не вмешиваются в работу друг друга. Каждый из них отвечает за свою собственную задачу, поставленную конкретно ему, а взаимодействие реализуется с помощью вспомогательных систем, что распространено и практично в проектировании подобных решений.

Все поставленные цели и задачи были выполнены в полной мере и полном объеме, что иллюстрируется работающим и доступным сервисом.

Как было упомянуто выше, система крайне лояльна к переносу и интеграции со сторонними системами, единственное, что она требует это наличие указанной исполняемой среды.

В рамках сравнения технического уровня системы со схожими решениями можно выделить наличие ряда определенных преимуществ у нашего подхода. В случае со встроенными в Kubernetes системами горизонтального масштабирования мы получаем гораздо более глубокий уровень прозрачности производимых действий и возможностей тонкой настройки, необходимой в конкретно взятой ситуации.

Экономическая эффективность решения состоит в потенциальном уменьшении штата сотрудников, отвечающих за сервис, и перенаправление бюджета на его развитие или продвижение компании на рынке.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. С.Б. Беневоленский, Т.А. Чернова, А.В. Карпов. Модельный анализ функционирования коммуникационного сервера в распределенной системе непрерывного мониторинга состояния сложных технических объектов. "Фундаментальные исследования" №12, 2010 г. -с. 66-72
2. Цены на бензин по данным Росстат // РосИнфоСтат URL: https://rosinfostat.ru/tseny-na-benzin (дата обращения: 05.04.2024).
3. Плотникова О. В., Стулова А. Н. Влияние изменения стоимости топлива на себестоимость грузоперевозок // Молодой ученый. - 2020. - №26 (316). - С. 117-120.
4. Кошевой Н. Д., Гусев С. С. Разработка и исследование системы контроля расхода топлива // Автомобильный транспорт. - 2013. - №32. - С. 41-46.
5. Контроль расхода топлива // GURTAM URL: https://gurtam.com/ru/wialon/fuel-control (дата обращения: 05.04.2024).
6. Хлопцев А. Н. Оптимизация расходов предприятий на топливо и горюче-смазочные материалы при использовании систем диспетчерского управления автотранспортными средствами, использующими технологии спутниковой навигации. / Хлопцев А. Н. // Вестник Московского университета МВД России. – 2013. - №4 – С.206-210.
7. Петряков Д. С. Совершенствование и экономическое обоснование метода снижения затрат на ГСМ. / Петряков Д. С. // Вестник Курской государственной сельскохозяйственной академии. – 2012.
8. Как обеспечить масштабируемость облака // xelent URL: https://www.xelent.ru/blog/kak-obespechit-masshtabiruemost-oblaka/ (дата обращения: 05.04.2024).
9. МОНИТОРИНГ ТРАНСПОРТА // МОНИТОРИНГ АВТО URL: https://www.monitoring-auto.ru/ (дата обращения: 05.04.2024).
10. Система контроля расхода топлива на транспорте // ITOB URL: https://itob.ru/solutions/fuel-control/ (дата обращения: 05.04.2024).
11. Web Server Quickstart // aiohttp documentation URL: <https://docs.aiohttp.org/en/stable/web_quickstart.html> (дата обращения: 28.04.2024).
12. Multithreading // ХАБР URL: <https://habr.com/ru/companies/otus/articles/549814/> (дата обращения: 28.04.2024).
13. Web Server Advanced // aiohttp documentation URL: <https://docs.aiohttp.org/en/stable/web_advanced.html> (дата обращения: 28.04.2024).
14. HTTP API // Prometheus URL: <https://prometheus.io/docs/prometheus/latest/querying/api/> (дата обращения: 28.04.2024)
15. Kubernetes API Reference Docs // Kubernetes URL: <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.30/> (дата обращения 28.04.2024).
16. Работа с хранилищами в Kubernetes: руководство для инженеров // Хабр URL: <https://habr.com/ru/companies/T1Holding/articles/781368/> (дата обращения 28.04.2024).
17. Балансировка нагрузки и масштабирование долгоживущих соединений в Kubernetes // Хабр URL: <https://habr.com/ru/companies/vk/articles/493820/> (дата обращения 28.04.2024).
18. HTTP API – PostgreSQL Wiki // PostgreSQL URL: <https://wiki.postgresql.org/wiki/HTTP_API> (дата обращения: 28.04.2024).
19. Запросы в PostgreSQL: 1. Этапы выполнения // Хабр URL: <https://habr.com/ru/companies/postgrespro/articles/574702/> (дата обращения 28.04.2024).
20. TIOBE Index for May 2024 // TIOBE URL: <https://www.tiobe.com/tiobe-index/> (дата обращения 01.05.2024).
21. How many rules can iptables support? // serverfault URL: <https://serverfault.com/questions/479549/how-many-rules-can-iptables-support> (дата обращения 01.05.2024).
22. Э. Таненбаум, М. ванн Стеен Распределенные системы, принципы и парадигмы – издательский дом Питер, 2003. – 877 с.
23. John W. Rittinghouse, James F. Ransome Cloud Computing Implementation, Management, and Security - CRC Press, 2010. – 340 с.
24. M. Agarwal, V. Bhat, Z. Li, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, M. Parashar, B. Khargharia, and S. Hariri. AutoMate: Enabling Autonomic Applications on the Grid. In Proceedings of Autonomic Computing Workshop The Fifth Annual International Workshop on Active Middleware Services (AMS 2003) IEEE Computer Society Press, pages 48–57, Seattle, WA, June 25 2003.
25. IBM Corporation. An architectural blueprint for autonomic computing. April 2003.
26. S. Hariri and M. Parashar. Handbook of Bioinspired Algorithms and Applications, chapter The Foundations of Autonomic Computing. CRC Press LLC, 2005.
27. S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International, April 9-11 2003.
28. P. Horn. Autonomic Computing:IBM’s perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/, Oct 2001. IBM Corp.
29. G. Kaiser, P. Gross, G. Kc, J. Parekh, and G. Valetto. An Approach to Autonomizing Legacy Systems. In Workshop on Self-Healing, Adaptive and Self-MANaged Systems, SHAMAN, New York City, NY, June 23 2002.
30. J. Kubiatowicz. OceanStore: Global-Scale Persistent Storage. http://oceanstore.cs.berkeley.edu/publications/talks/StanfordOceanStore.pdf, Spring 2001. Stanford Seminar Series, Stanford University.
31. G. M. Lohman and S. S. Lightstone. SMART: Making DB2 (More) Autonomic. In VLDB 2002 28th International Conference on Very Large Data Bases, Kowloon Shangri-La Hotel, Hong Kong, China, August 20-23 2002.
32. J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank–A Heterogeneous Scalable SAN file system. IBM Systems Journal, 42(2):250–267, 2003.
33. A. Montresor. The Anthill Project Part II: The Anthill Framework. http://www.cs.unibo.it/projects/anthill/papers/anthill-4p.pdf, 2001. The Anthill Project Documentation.
34. V. Narasayya. AutoAdmin: Towards Self-Tuning Databases, November 13 2002. Guest Lecture at Stanford University.
35. University of Sussex. Adaptive system lectures. http://www.cogs.susx.ac.uk/users/ezequiel/AS/lectures/AdaptiveSystems3.ppt, 2003.
36. M. Parashar, Z. Li, H. Liu V. Matossian, and C. Schmidt. Self-Star Properties in Complex Information Systems, volume 3460 of Lecture Notes in Computer Science, chapter Enabling Autonomic Grid Applications: Requirements, Models and Infrastructures. Springer Verlag, 2005.
37. R.V. Renesse, K.P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed systems monitoring, management, and data mining. ACM Transaction on Computer Systems, 21(2):164–206, 2003.
38. Макурин Ю.Д., Сивохин А.В. Проектирование и реализация баз данных и клиентских приложений, М.: Издательство типография Тугушева, 2009.
39. Таненбаум Э. С., Ван Стеен М. Распределенные системы. Принципы и парадигмы СПб.: Питер, 2007.
40. C.Richardson POJOs in action: Developing enterprise applications with lightweight frameworks Manning Publications Co., 2006.
41. E.Newcomer, G.Lomow Understanding SOA with Web Services Addison Wesley Professional, 2004.
42. J.M. Crichlow The Essence of Distributed Systems Prentice Hall, 2000.
43. Дейт К.Дж. Введение в системы баз данных, 6-е издание. СПб., Издательский дом «Вильямс», 2000.
44. Фаулер М.Архитектура корпоративных приложений М.: Издательский дом «Вильямс», 2004.
45. G.Coulouris, J.Dollimore, T.Kindberg Distributed Systems: Concepts and Design, Third Edition Addison-Wesley, 2001.
46. R.Monson-Haefel J2EE™ Web Services Addison Wesley, 2003.

# **ПРИЛОЖЕНИЕ А**

URL для доступа к репозиторию с исходным кодом: <https://github.com/SkiStalker/BachelorsDegree>

QR-код для доступа к репозиторию с исходным кодом:  
