Содержание

Термины и определения

Введение

1. Теоретическая часть
   1. Анализ предметной области и изучение объекта исследования, а также оценка изученности проблемы;
   2. Выбор облачной платформы, ее преимущества и недостатки;
   3. Формирование требований к программной системе и концепция программного продукта;
   4. Выбор платформы и языка программирования для разработки системы;
2. Практическая часть
   1. Разработка архитектурных решений и реализация собственной системы на языке программирования Python 3;
   2. Настройка облачной среды Yandex Cloud для разворачивания приложения;
   3. Разворачивание приложения в облачной среде Yandex Cloud;
   4. Представление результатов исследования, связанных с способностью программной системы масштабировать вычислительные ресурсы приложения.
3. Источники

# Введение

В наше время распределенные системы стали основой любой крупной компании, работающей с большим потоком входных данных. Основной задачей таких компаний является обработка нагрузки с максимальной скоростью и с минимальными затратами на ресурсы.

Первый из вариантов решения данной проблемы является статический расчет необходимых мощностей системы на основании максимального и минимального объема входных данных. Такой подход имеет место быть, но назвать его оптимальным нельзя, так как во время снижения нагрузки мы получаем простаивающие компоненты системы, а при ее увеличении снижение скорости работы системы.

Вторым вариантом является динамическое регулирование количества рабочих узлов системы. В этом случае мы получаем возможность обеспечить оптимальную мощность для текущего момента времени.

Все наиболее современные автоматизированные системы пришли к варианту динамического масштабирования. И далее возник вопрос удобства ее использования. Не каждая инфраструктура позволяет в режиме реального времени автоматизировано добавлять и удалять рабочие машины. И часть системы, отвечающая за балансировку нагрузки, может не предоставлять функционала динамического изменения конечных точек назначения потока данных.

# Теоретическая часть.

## 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 [1]. На рисунке 1 приведена статистика, как вели себя цены на топливо за последнее десятилетие. Продолжительный рост на графике очевиден.

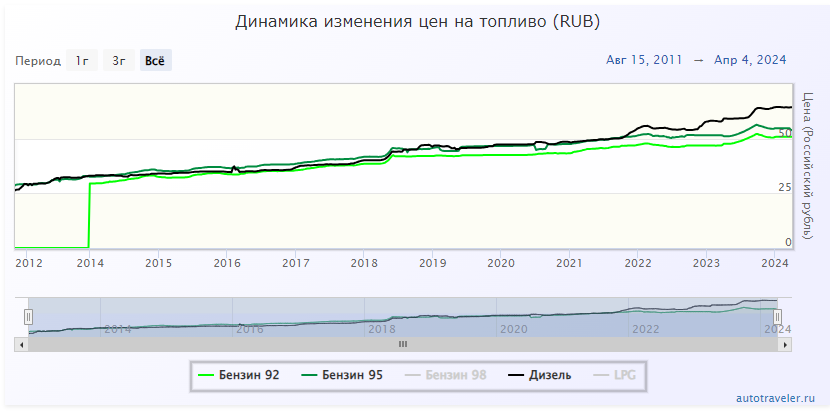


Рис. 1 – Динамика цен на 92, 95 бензин и дизель

Рост цены на топливо оказывает непосредственное влияние на автотранспортные предприятия и организации, чья деятельность осуществляется в основном автомобильными грузоперевозками, в статье Плотниковой О. В. [2] 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 [5, 6]. 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. [3] 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 [4, 8, 9], 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.

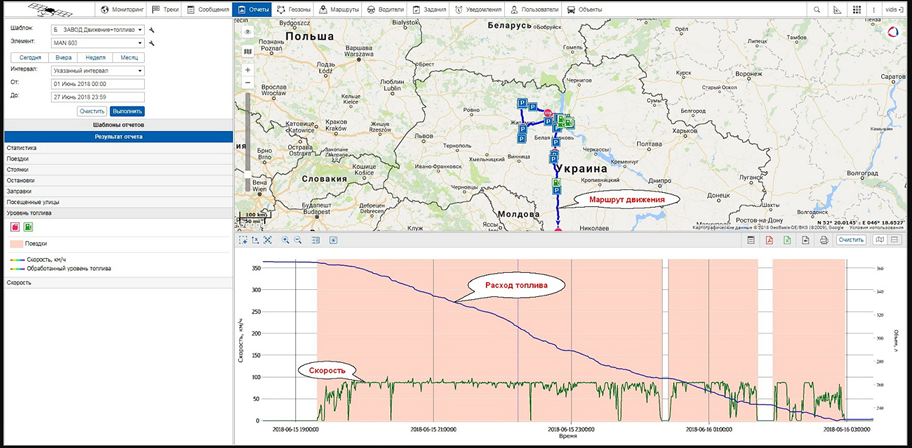


Рис. 2 – Пример внешнего вида программы

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.

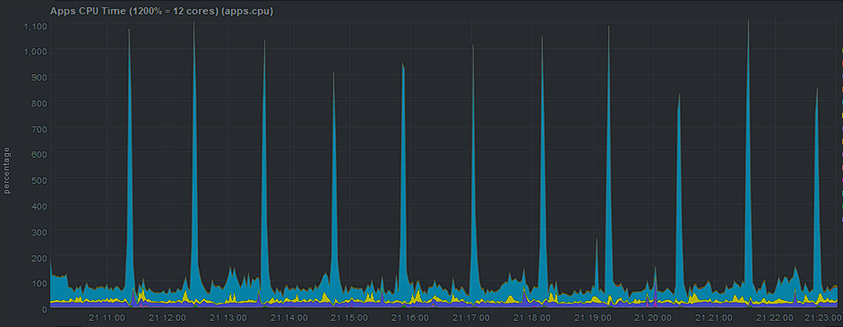


Рис. 3 – Периодическая нагрузка на сервер

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 [5, 7], 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.

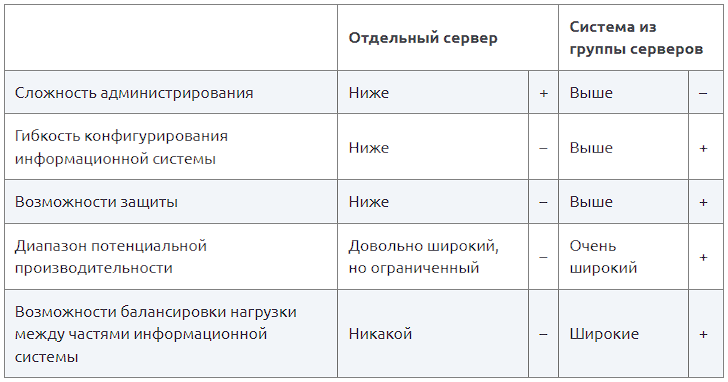


Рис. 4 - Сравнение способов облачного развёртывания

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.

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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.

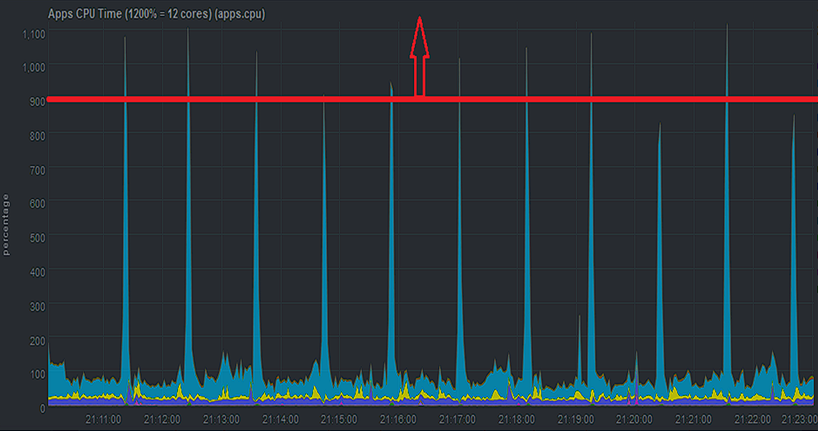


Рис. 5 – Вертикальное масштабирование в облаке

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).

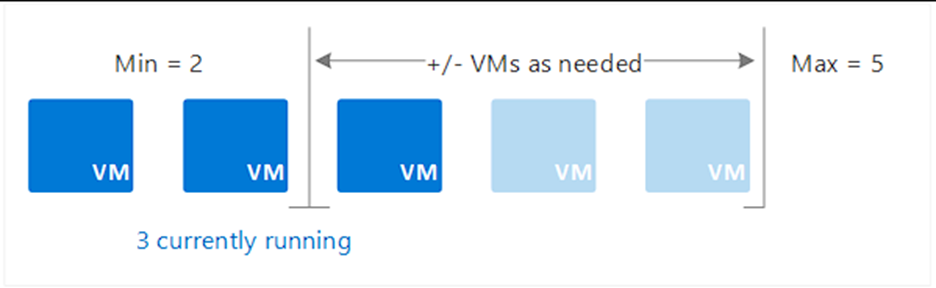


Рис. 6 - Диапазон количества выделяемых ВМ для автомасштабирования

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.

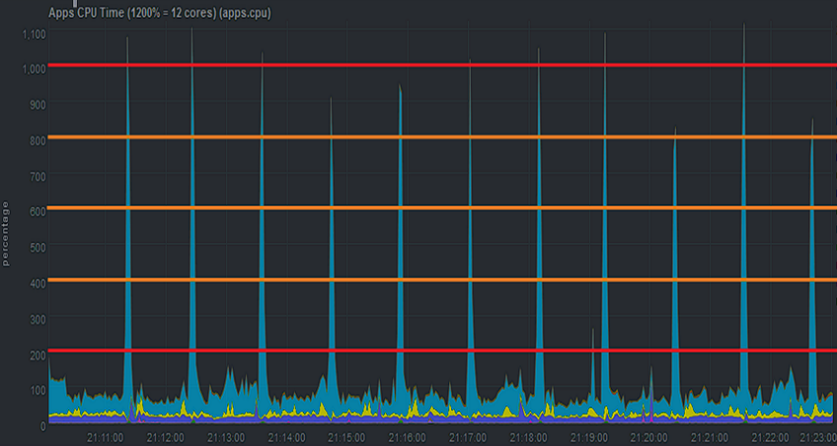


Рис. 7 – Горизонтальное масштабирование (автомасштабирование) в облаке

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.

## Формирование требований к программной системе и концепция программного продукта

Наша программная система должна отвечать следующим требованиям:

1. Сбор данных с неопределенного числа источников с минимальной задержкой 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 компонента, отвечающего за масштабирование системы, позволяющее динамически адаптировать мощность системы в зависимости от нагрузки;
7. Наличие удаленного доступа к контролирующему компоненту для удобного управления и мониторинга системы.

Концепция программного продукта заключается в следующем: система, развернутая в облачном решении в режиме реального времени, принимает запросы от источников данных. В качестве этих источников выступают установленные на автомобилях системы, отображающие уровень топлива, скорость, положение, 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 система будет осуществлять сбор и хранение информации о нагрузке развернутых мощностей. Она должна быть в состоянии некоторое время хранить эти данные и быть в состоянии 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, согласно конфигурационному файлу, состоящему из набора правил, определяющих количество развернутых мощностей в зависимости от текущей нагрузки.

Компонент web-консоли будет напрямую взаимодействовать с модулем масштабирования и предоставлять пользователю 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.

## Выбор платформы и языка программирования для разработки системы

Проанализировав существующие облачные платформы, была выбрана Yandex Cloud. Данное решение было принято на основании ключевых факторов при развертывании решения 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 как богатую клиентскую базу, так и отсутствие проблем, связанных с “молодыми” продуктами: непредсказуемые падения в процессе работы, периодические перебои связи с платформой и малочисленную базу знаний по типовым ошибкам пользователей.

Возможность автомасштабирования системы можно достигнуть двумя способами: создание множества виртуальных машин или контейнеризация. Второй способ более предпочтительный, развертывание целой виртуальной машины 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 его использования 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 поставленным 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 утилитой “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 пользователем. 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.

Так как frontend часть будет отображаться в браузере пользователя, необходимо создать страницы сайта, отображающие как необходимую для мониторинга информацию, так и текущую конфигурацию модуля масштабирования, с возможностью ее редактирования. 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 понятной.

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 для критических сервисов критерии, 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 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.

Как система 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.

# Практическая часть

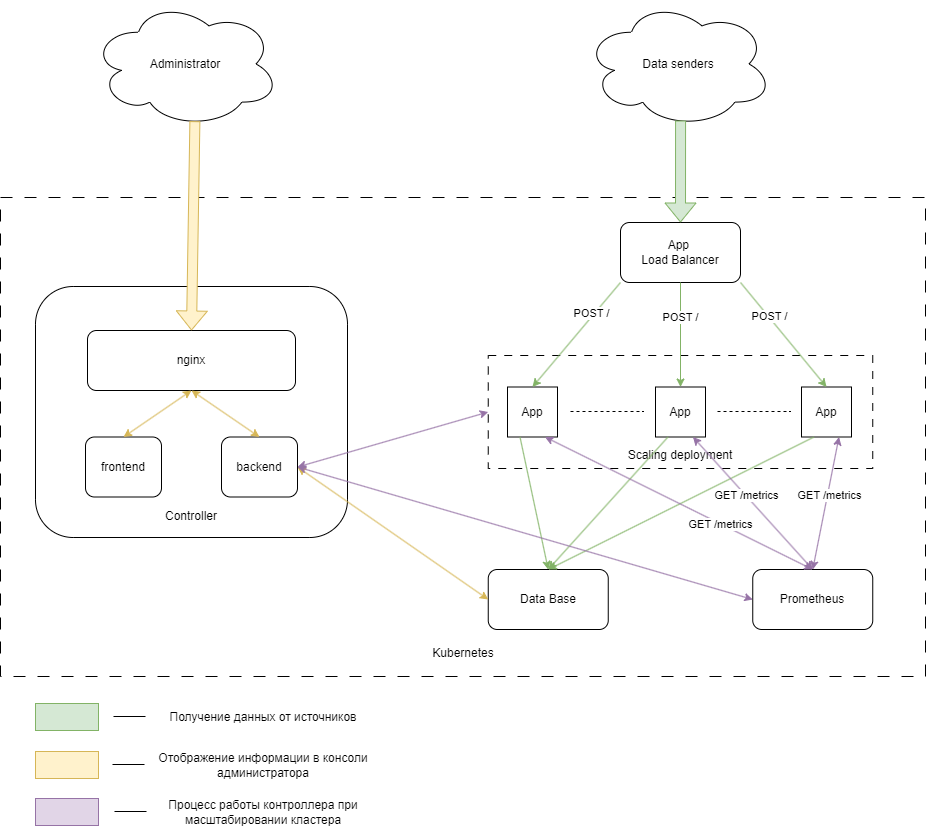
## Разработка архитектурных решений и реализация собственной системы на языке программирования Python 3 и TypeScript

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” [10] 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 [11]. Данное решение предназначено, преимущественно, для получения и ответа на http запросы, однако разработчиками предоставлена возможность параллельно с основной задачей выполнять дополнительные [12]. 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 HTTP-API интерфейс [13,14, 17] и можно обойтись без использования библиотек, но оперирование программными сущностями значительно упрощает разработку и избавляет от излишних ошибок.

Проанализировав схему работы разрабатываемой системы (см Рис. N)



(Рис. N – 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 [18], 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 HTTP-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

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 ниже (см Рис. N).



(Рис. N – 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 **MaterialUI** и **ReCharts** для таблиц и графиков соответственно.

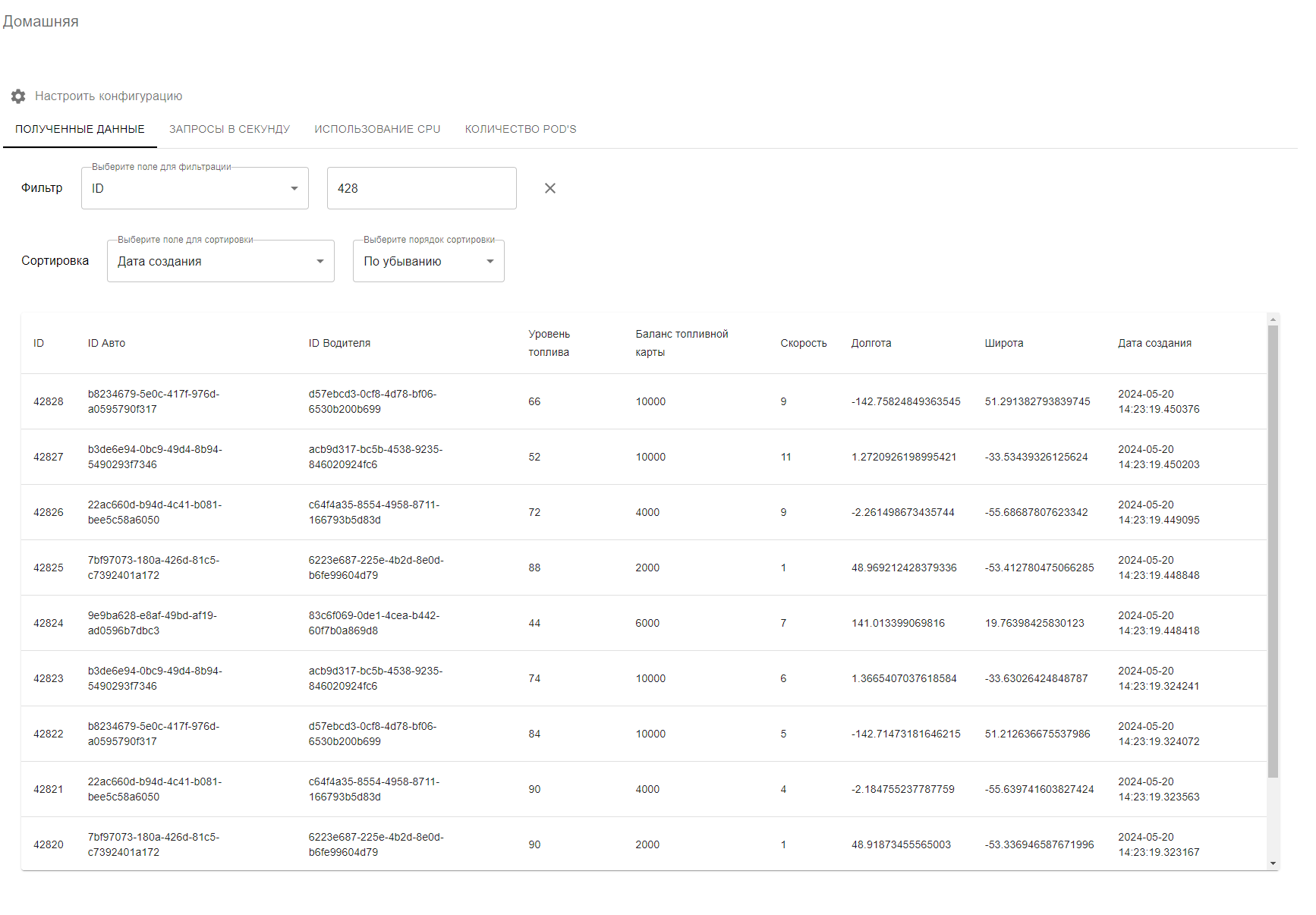


Рис. N – Главная страница



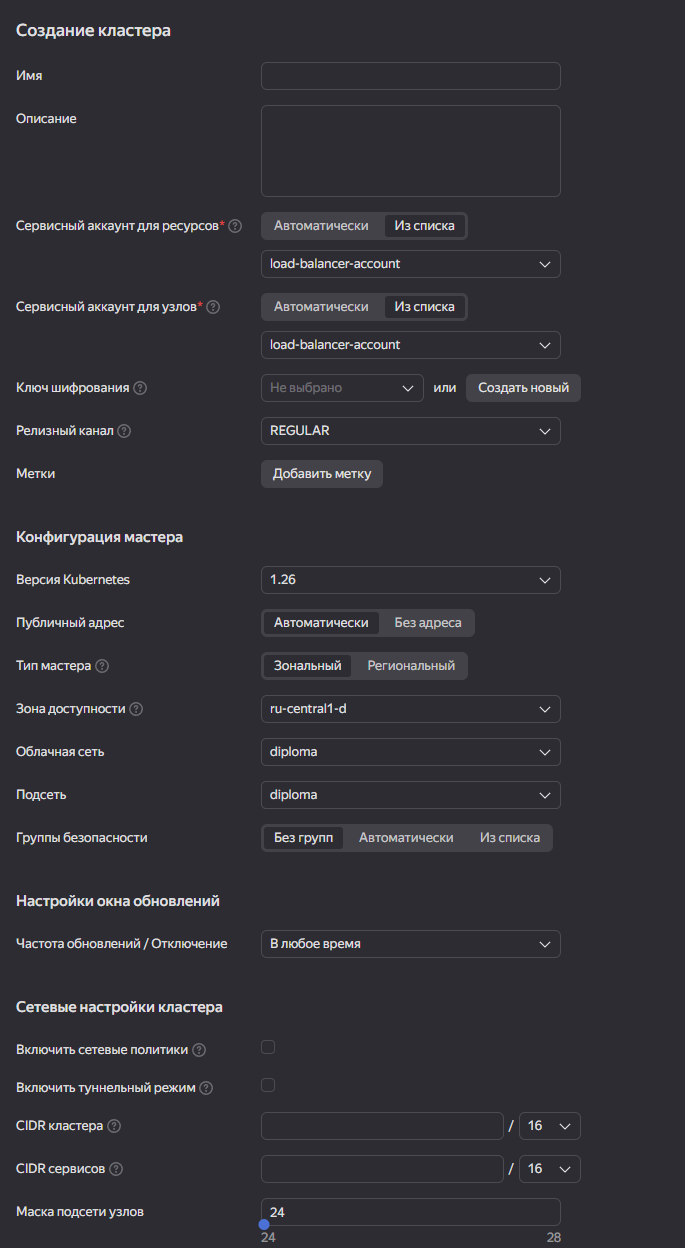
Рис. N – Страница конфигурации

На данном этапе последним шагом будет являться настройка **nginx** web-сервера и упаковка всех разработанных компонентов в **Docker-**контейнер.

## 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 текущей 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, необходимо 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.

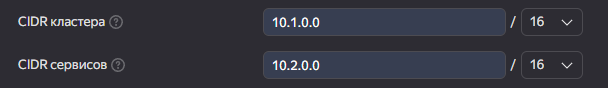


(Рис. Н – Форма создания кластера Kubernetes)

На странице создания кластера **Kubernetes** (см. Рис) 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 **CIDR Кластера/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. Н – Значение для публичного адреса)

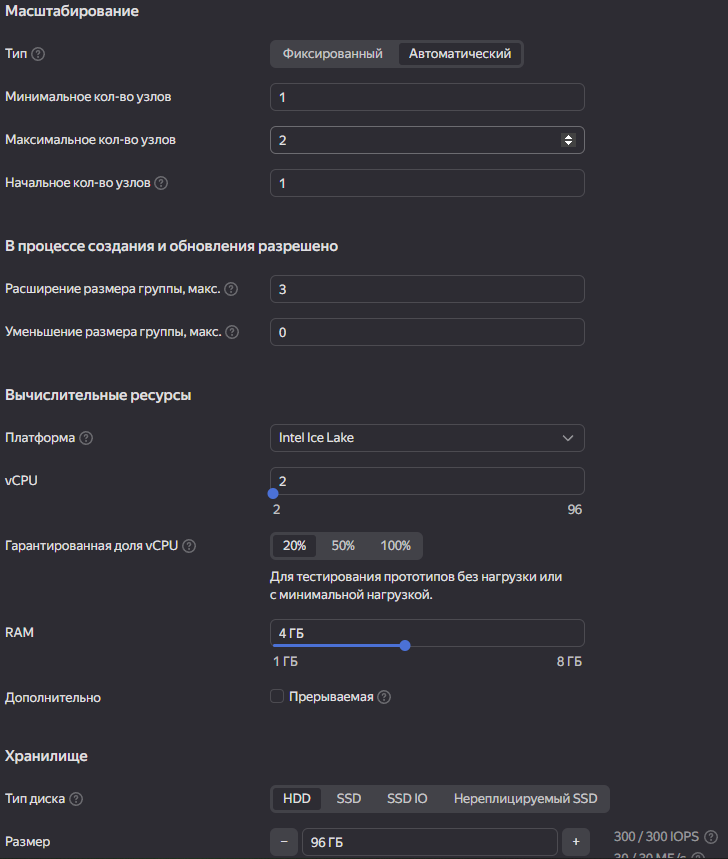


(Рис. Н – Значение для CIDR кластера/сервисов)

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. Н – Рекомендуемые параметры для группы узлов)

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

## 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 **Docker** контейнерах 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 **Prometheus** и **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 [15], 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”. 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. Н – развернутая PostgreSQL)

Следующим шагом будет размещение части системы, которая будет подвержена масштабированию в процессе работы. Обращаясь к параграфу 3.1, можем увидеть, что она представляет из себя 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 **Prometheus** происходит 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 **nginx** web-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 [N]).

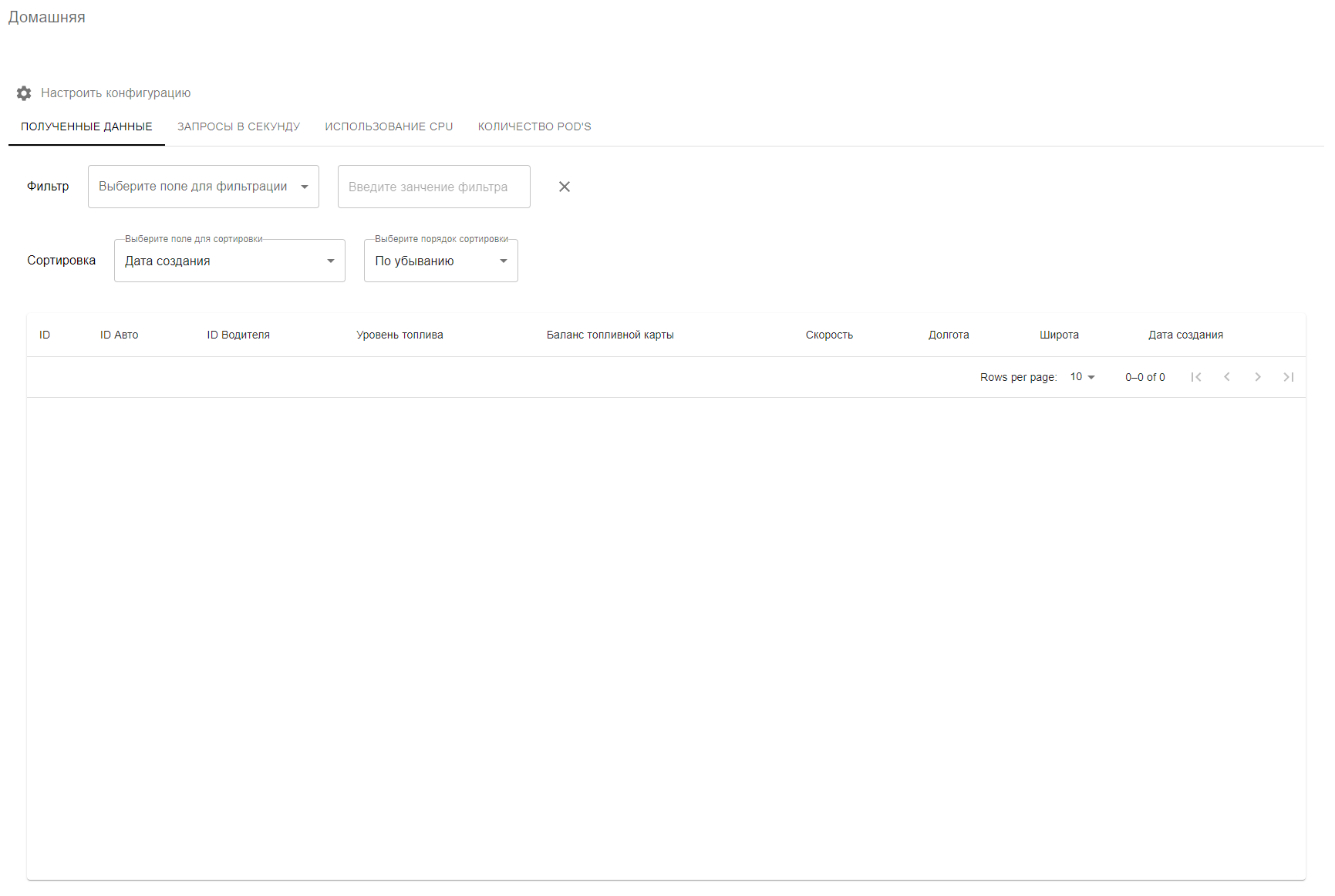


(Рис. N – Статус работы сервиса **nginx**)

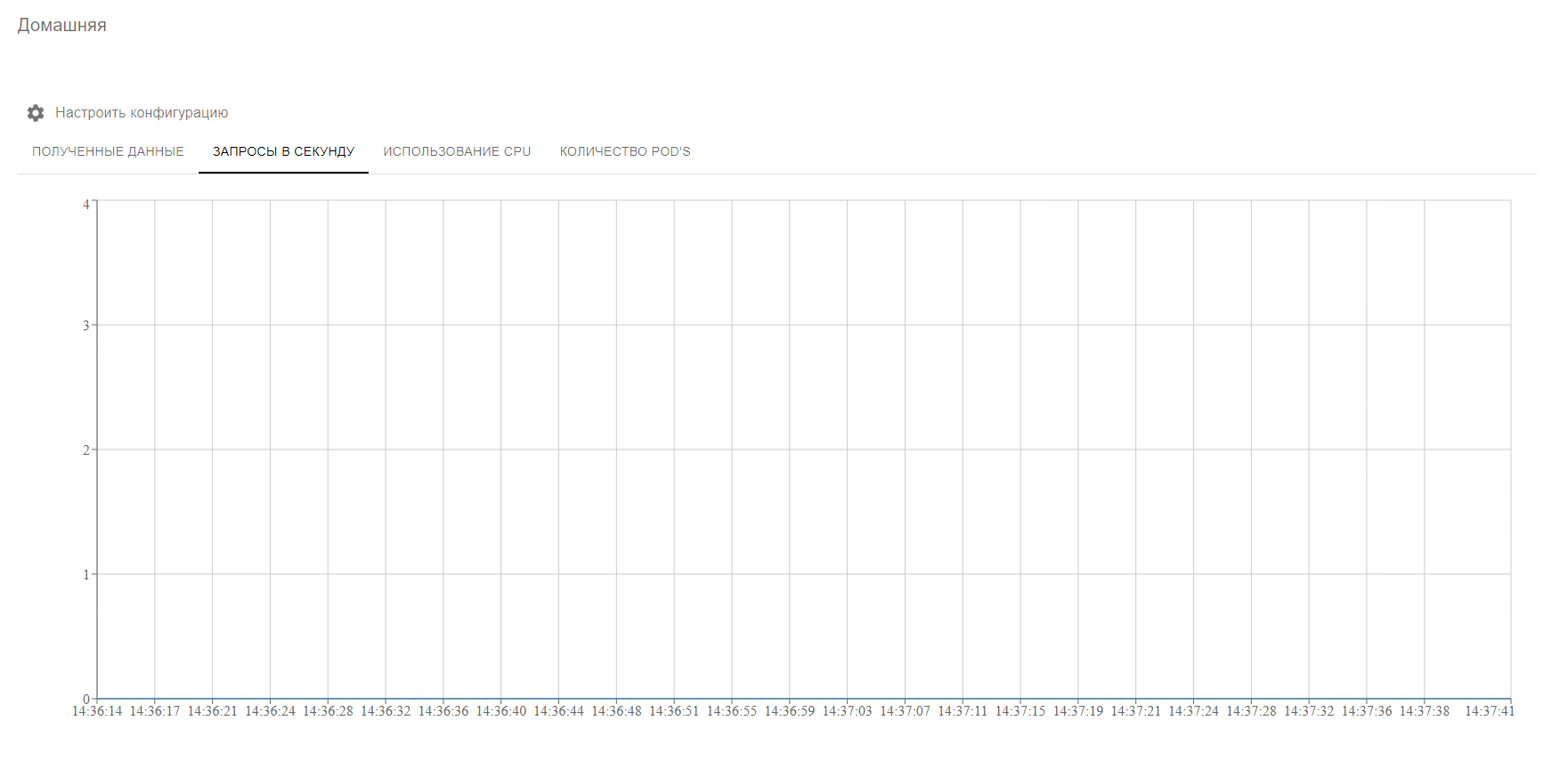
Учитывая то, что доступ к консоли администратора должен быть извне кластера, создадим сервис, который нам его предоставит. Используем LoadBalancer. Несмотря на то, что его основной целью является распределение нагрузки, он также имеет внешний IP адрес, что как раз, в нашем случае, является необходимы 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. N). На второй график с временной линией, находящейся в положении ноля (см. Рис. N). На третьей также должен присутствовать график (см Рис. N). Четвертая вкладка представляет собой временную линию такого же формата, как и на предыдущих двух (см Рис. N).



(Рис. N – Изначальный вид вкладки “Полученные данные”)

****

(Рис. N – Изначальный вид вкладки “Запросы в секунду”)



(Рис. N – Изначальный вид вкладки “Количество POD’s”)

Последним 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. N).



(Рис. N – Изначальный вид страницы конфигурации)

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

# Источники

1. Цены на бензин по данным Росстат // РосИнфоСтат URL: https://rosinfostat.ru/tseny-na-benzin (дата обращения: 05.04.2024).
2. Плотникова О. В., Стулова А. Н. Влияние изменения стоимости топлива на себестоимость грузоперевозок // Молодой ученый. - 2020. - №26 (316). - С. 117-120.
3. Кошевой Н. Д., Гусев С. С. Разработка и исследование системы контроля расхода топлива // Автомобильный транспорт. - 2013. - №32. - С. 41-46.
4. Контроль расхода топлива // GURTAM URL: https://gurtam.com/ru/wialon/fuel-control (дата обращения: 05.04.2024).
5. 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.
6. Петряков Д. С. Совершенствование и экономическое обоснование метода снижения затрат на ГСМ. / Петряков Д. С. // Вестник Курской государственной сельскохозяйственной академии. – 2012.
7. Как обеспечить масштабируемость облака // xelent URL: https://www.xelent.ru/blog/kak-obespechit-masshtabiruemost-oblaka/ (дата обращения: 05.04.2024).
8. МОНИТОРИНГ ТРАНСПОРТА // МОНИТОРИНГ АВТО URL: https://www.monitoring-auto.ru/ (дата обращения: 05.04.2024).
9. Система контроля расхода топлива на транспорте // ITOB URL: https://itob.ru/solutions/fuel-control/ (дата обращения: 05.04.2024).
10. https://docs.aiohttp.org/en/stable/web\_quickstart.html
11. <https://habr.com/ru/companies/otus/articles/549814/>
12. <https://docs.aiohttp.org/en/stable/web_advanced.html>
13. <https://prometheus.io/docs/prometheus/latest/querying/api/>
14. <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.30/>
15. <https://habr.com/ru/companies/T1Holding/articles/781368/>
16. <https://habr.com/ru/companies/vk/articles/493820/>
17. <https://wiki.postgresql.org/wiki/HTTP_API>
18. <https://habr.com/ru/companies/postgrespro/articles/574702/>
19. https://www.tiobe.com/tiobe-index/