Московский государственный университет имени М. В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Кафедра Автоматизации Систем Вычислительных Комплексов

Бодров Антон Олегович

**Построение масштабируемой серверной платформы для сбора и хранения медицинской телеметрии на основе технологии виртуализации сетевых функций**

Курсовая работа

Научный руководитель: к.ф.-м.н., в.н.с. Бахмуров A.Г.

Консультант: Любицкий А.А.

Москва, 2021

Аннотация

Данная работа посвящена масштабированию серверной части платформы для сбора и обработки физиологических данных о человеке.

Проведён обзор способов виртуализации, на основании которого было выбран способ и ПО для виртуализации. Была изучена темы масштабирования приложений и балансировки нагрузки, после чего было выбрано ПО для их обеспечения.

В рамках работы реализуется масштабируемость ПО, предоставляющего возможность хранения и обработки данных мониторинга, выполняется балансировка нагрузки между копиями серверной части платформы.

Содержание:

1. Введение

2. Цели и задачи

2.1. Цели

2.2. Задачи

3. Обзор технологий виртуализации

3.1. Цель обзора

3.2. Способы виртуализации

3.3. Обоснование выбора способа виртуализации

3.4. Результаты обзора

3.5. Выводы из обзора

4. Описание реализации

4.1. Исходные данные

4.1.1. MQTT

4.2. Контейнеризация и масштабирование приложения

4.2.1 Основные понятия

4.2.2 Балансировка нагрузки при помощи HAproxy

4.2.3. Обоснование схемы балансировки и масштабирования приложения

4.2.4 Построение платформы при помощи docker-compose

4.2.5. Аутентификация

4.2.6. Dockerfile

5. Требования к эксперименту

6. Заключение

7. Список литературы

1 Введение

В современном мире IT с каждым годом всё большее распространение получает такая сфера, как IoT или Internet of Things (Интернет вещей). Данное понятие представляет собой компьютерную сеть физических предметов («вещей»), которые могут взаимодействовать друг с другом и внешней средой. Широкое распространение IoT получил в медицине, где он получил известность, как Интернет медицинских вещей (IoMT, Internet of Medical Things). IoMT используется для мониторинга состояния человеческого организма в реальном времени, позволяет улучшить процессы лечения, восстановления пациента и их мониторинг, а также усовершенствовать процесс физических тренировок.

Проблема мониторинга состояния человека актуальна не только в медицинским учреждениях, но и вне них. Сбор и обработка информации о текущем функционировании организма человека в реальном времени позволяют быстро анализировать медицинские показатели и выявлять патологии, что может быть определяющим для здоровья конкретного человека при критическом состоянии пациента. Помимо выгоды для самого пациента, мониторинг может быть чрезвычайно актуальным для специалистов в области медицины, которым данная информация будет полезна, как дополнительный источник для исследования поведения организма при различных заболеваниях. При тренировках мониторинг состояния необходим для грамотного распределения нагрузки на организм.

Одной из актуальных задач является создание платформы сбора и обработки физиологических данных с открытым исходным кодом, так как несмотря на обширность рынка портативных носимых устройств большинство из них являются проприетарными проектами. В качестве носимого устройства в данной работе взята умная одежда Hexoskin компании Carre Technologies Inc[1], способная считывать и обрабатывать показатели состояния человеческого организма. Hexoskin имеет открытый интерфейс для получения данных по Bluetooth, поэтому именно она выбрана для работы.

Ограничение на число клиентов платформы заключит её возможности в жёсткие рамки, поэтому для функционирования платформы с различным числом клиентов следует обеспечить её масштабируемость, то есть способность увеличивать производительность системы пропорционально дополнительным ресурсам (оборудованию), что может обеспечить применение современных технологий виртуализации. Таким образом мы сможем избавиться от ограничения на число клиентов платформы мониторинга, что позволит осуществлять мониторинг состояния значительного количества пациентов одновременно.

Данная работа посвящена регулированию производительности платформы для сбора и обработки медицинской телеметрии путём масштабирования, которое будет осуществлено с помощью подхода виртуализации сетевых функций. Данная стратегия подразумевает разделение функциональности сети и оборудования, которое её реализует, с помощью замены аппаратных ресурсов виртуальными.

2 Цели и задачи

2.1 Цели:

1. Разработка средств масштабирования серверной части платформы для сбора, хранения и обработки медицинской телеметрии
2. Получение опыта использования широко распространённых средства разработки и современных облачных технологий

2.2 Задачи:

1. Изучить исходные данные по задаче сбора медицинской телеметрии в ЛВК
2. Изучить способы виртуализации для серверной части платформы.
3. Выделить критерии сравнения способов виртуализации. Провести сравнение, на его основе отобрать ПО для проектирования системы на базе выбранного ПО.
4. Спроектировать масштабируемую серверную часть приложения по сбору телеметрии
5. Развернуть масштабируемую серверную часть приложения
6. Исследовать вопрос аутентификации пользователей

3. Обзор технологий виртуализации

3.1. Цель обзора

Целью обзора способов виртуализации является выбор способа виртуализации для дальнейшей реализации масштабирования платформы на его основании.

3.2. Способы виртуализации

В настоящее время осуществлять виртуализацию можно несколькими способами. Для этого можно использовать:

1. Виртуальные машины (VM)

2. Контейнеризацию

**Виртуальные машины (VM)**

Виртуальная машина – это программная и/или аппаратная система, [эмулирующая](https://ru.wikipedia.org/wiki/%D0%AD%D0%BC%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) [аппаратное обеспечение](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) некоторой [платформы](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%B0) (называемой гостевой платформой) и исполняющая программы для гостевой платформы на платформе-хозяине, изолирующая от платформы-хозяина приложения, бинарные файлы и операционные системы. Виртуальная машина обладает всеми виртуальными устройствами и виртуальным жёстким диском, на который устанавливается гостевая операционная система, драйверы устройств и прочие компоненты. Виртуальное оборудование функционирует, как реальное, при этом виртуальная машина изолирована от реального компьютера, однако может иметь общие каталоги файловой системы c ним.

Использование VM вызывает дополнительные расходы на виртуальное оборудование, запуск гостевой ОС и поддержку необходимого окружения для работы приложений.

**Контейнеры**

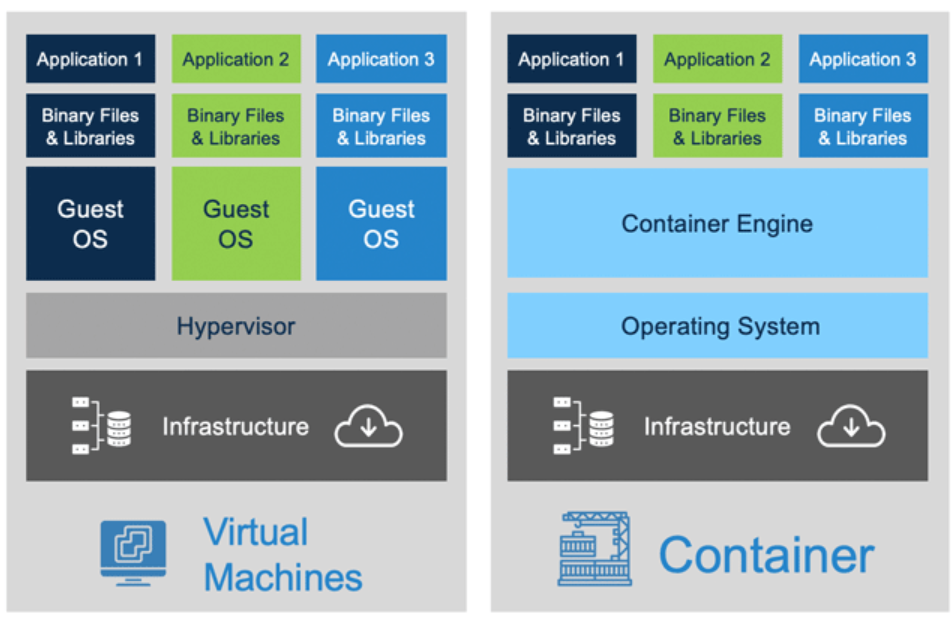
Контейнеризация – это метод легковесной виртуализации, при котором вместе упаковываются и изолируются от домашней ОС приложение, бинарные файлы и библиотеки. В отличие от виртуальных машин контейнеры обеспечивают виртуализацию на уровне операционной системы, а не аппаратного обеспечения [Рис.1]. Благодаря виртуализации на уровне ОС возможно быстрое развёртывание, простое масштабирование. Также уменьшается объём занимаемого места на жёстком диске по сравнению с виртуализацией с помощью виртуальных машин. 

Рис.1: Сравнение технологий виртуализации виртуальных машин и контейнеров

**Бессерверные вычисления**

Бессерверные вычисления – стратегия организации платформенных облачных услуг, при котором реализуется шаблон «функция как услуга», при котором для выполнения каждого вызова функции создаётся контейнер или виртуальная машина, уничтожающиеся после выполнения. Бессерверные вычисления также обеспечивают хорошую масштабируемость.

Стратегию бессерверных вычислений наиболее выгодно использовать, когда речь идёт о приложении, не требующем непрерывного выполнения. В данной работе рассматривается серверная часть приложения, непрерывно оперирующего с большим количеством данных в реальном времени.

* 1. Обоснование выбора способа виртуализации

Ниже [табл.1] приведено сравнение технологий виртуальных машин и контейнеризации[2]. В качестве критериев сравнения способа виртуализации отобраны:

* Уровень виртуализации
* Вид операционной системы
* Время начальной загрузки
* Использование ресурсов
* Предварительно построенные образы (в данной работе будут использованы предварительно построенные образы, поэтому этот критерий особенно важен)
* Пользовательские предварительно настроенные образы (для настоящей работы должно быть удобно настраивать свои образы с целью упаковки в них приложения)
* Размер
* Мобильность
* Время создания (образ не должен создаваться слишком долго)

|  |  |  |
| --- | --- | --- |
|  | Виртуальные машины | Контейнеры |
| Уровень виртуализации | Аппаратная | Операционная система |
| Операционная система | Отдельная у каждой виртуальной машины | Общая |
| Время загрузки | Больше | Маленькое |
| Использование ресурсов | Больше | Меньше |
| Предварительные построенные образы | Тяжело найти и управлять | Доступны на домашнем сервере |
| Пользовательские предварительно настроенные образы | Трудно настроить | Легко настроить |
| Размер | Больше, потому что виртуализацию происходит с операционной системой полностью | Меньше |
| Миграция | Легко перемещается с одного сервера на другой | Уничтожается и пересоздаётся вместо перемещения |
| Время создания | Дольше (в среднем минуты) | Меньше (в среднем секунды) |
| Производительность | Накладные расходы при передаче инструкций из гостевых ОС в хостовую ОС | Производительность, близкая к ОС хоста |

Табл.1 Сравнение технологий виртуальных машин и контейнеризации

Далее для сравнения выбраны различные ПО для виртуализации [табл.2]. В первую очередь для простоты работы с программным средством в данной работе от ПО требуется подробная документация, крупное community и свободная лицензия. Как было сказано выше, безусловным преимуществом технологии контейнеризации являются маленькие накладные расходы, поэтому этот критерий так же будет важным.

Итого, мы сравниваем следующие ПО для виртуализации:

* OpenVZ[3]
* LXC[4]
* Docker[5]
* Virtual Box[6]

В качестве критериев сравнения отобраны:

* Размер сообщества пользователей (с ПО с большим сообществом легче работать при разработке)
* Лицензия
* Год основания технологии (более старая технология более испытана пользователями)
* Размер накладных расходов
* Поддерживаемые ОС (ПО должно запускаться на любой из распространённых ОС)
* Подробность документации (недостаточно подробная документация может не позволить применить технологию для данных задач)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Open VZ | LXC | Docker | Virtual Box |
| Размер сообщества | маленькое | среднее | большое | большое |
| лицензия | [GNU GPLv2](https://en.wikipedia.org/wiki/GNU_General_Public_License) (принадлежит проприетарной Virtuozzo) | [GNU GPLv2](https://en.wikipedia.org/wiki/GNU_General_Public_License) | [Apache License 2.0](https://en.wikipedia.org/wiki/Apache_License) | [GNU GPLv2](https://en.wikipedia.org/wiki/GNU_General_Public_License) |
| Год основания технологии | 2005 | 2008 | 2013 | 2007 |
| Маленькие накладные расходы | + | + | + | - |
| Поддерживаемые ОС | Linux | Linux | Linux, FreeBSD, Windows, macOS, | Linux, Windows, acOS, Solaris, Genode OS Framework |
| Документация | подробная | подробная | Очень  подробная | подробная |

Табл.2 Сравнение ПО для виртуализации

3.4. Результаты обзора

Основными преимуществами контейнеров перед виртуальными машинами для данной задачи является доступность предварительно построенных образов и беспроблемное создание пользовательских предварительно настроенных образов. Также преимуществами контейнеров над виртуальными машинами являются незначительность накладных расходов при загрузке и построении. В качестве ПО для виртуализации выбрана технология Docker в связи с простотой в работе и доступностью документации в сравнении с другими технологиями контейнеризации. Также Docker дополнительно уменьшает занимаемый объём на жёстком диске за счёт переиспользования образов.

3.5. Выводы из обзора

В данном разделе был проведён обзор 8 программных средств для виртуализации, в результате которого для дальнейшего рассмотрения и реализации был выбран Docker.

4. Описание реализации

4.1. Исходные данные

На рис.2 показана упрощенная схема работы платформы, для серверной части которой необходимо реализовать масштабируемость[7].

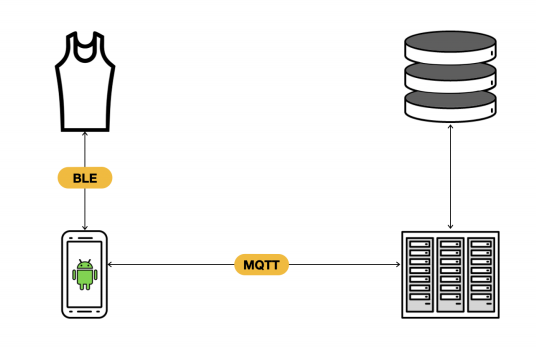


Рис.2 Схема работы платформы по сбору медицинской телеметрии

В работе сервиса можно выделить следующие этапы:

1. Показатели физиологического состояния пользователя считываются с помощью датчиков умной одежды Hexoskin, затем отправляются на мобильный телефон на операционной системе Android по протоколу Bluetooth Low Energy (BLE).
2. Считанные показатели принимаются мобильным телефоном и отправляются на сервер по протоколу MQTT
3. Сервер записывает полученные данные в базу данных, откуда их можно выгружать для медицинской обработки.

4.1.1. MQTT

Протоколом передачи между клиентом и сервером служит протокол MQTT[8].

MQTT – сетевой протокол, работающий поверх TCP/IP. MQTT ориентирован на обмен сообщениями между устройствами по принципу издатель-подписчик. Клиент может быть, как издателем или подписчиком, так и иметь обе роли одновременно. Обмен происходит через центральным сервер, который называется брокером[Рис.3]. Клиенты устанавливают TCP-соединение, и на его основании взаимодействуют с брокером. Клиенты публикуют топики по TCP-соединению. Каждое публикуемое сообщение содержит поле для имени топика, которое идентифицируют публикуемые данные. Аналогичным образом, клиент имеет возможность подписаться на определенные топики, если заинтересован в получении конкретных данных. Топики имеют иерархическую структуру в формате “дерева”, что упрощает их организацию и доступ к данным.

Для протокола MQTT выделяется 1883 порт. При использовании TLS (криптографического протокола, обеспечивающего защищённую передачу данных) для защиты соединения между клиентом и сервером вместо 1883 используется 8883 порт.

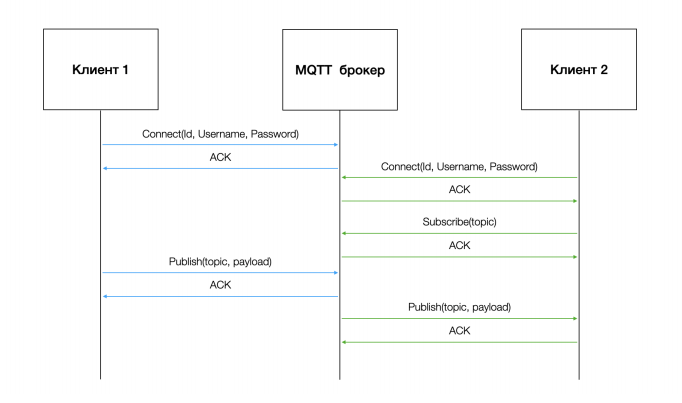


Рис.3 Схема работы протокола MQTT

Исходными данными для данной работы и являются сервис записи данных в БД и MQTT-брокер, вместе составляющие серверную часть платформы. Стоит задача масштабирования этой платформы.

4.2. Контейнеризация и масштабирование приложения

4.2.1. Основные понятия

Под масштабируемостью следует понимать способность системы справляться с увеличением рабочей нагрузки путём добавления ресурсов.

Масштабируемость можно разделить на горизонтальную и вертикальную. Под вертикальной масштабируемостью имеется в виду увеличение производительности каждого компонента системы с целью повышения общей производительности. Масштабируемость в этом контексте означает возможность заменять в существующей вычислительной системе компоненты более мощными и быстрыми по мере роста требований и развития технологий.

Горизонтальной масштабируемостью называют разбиение системы на более мелкие компоненты и разнесение их по отдельным машинам или увеличение количества серверов, параллельно выполняющих одну и ту же функцию. Масштабируемость в этом контексте означает возможность добавлять к системе новые узлы, серверы для увеличения общей производительности. Этот способ масштабирования может требовать внесения изменений в программы, чтобы последние могли в полной мере пользоваться возросшим количеством ресурсов. В данной работе рассмотрена именно горизонтальная масштабируемость.

Для оптимизации использования ресурсов, горизонтального масштабирования и обеспечения отказоустойчивости используется балансировка нагрузки – процесс распределения задач между несколькими серверами.

4.2.2. Балансировка нагрузки при помощи HAproxy

В качестве балансировщика нагрузки был выбран HAProxy[9].

HAProxy — серверное программное обеспечение для обеспечения [высокой доступности](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%B0%D1%8F_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и [балансировки нагрузки](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BD%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B8) посредством распределения входящих запросов на несколько обслуживающих серверов.

HAProxy зарекомендовал себя, как надёжное программное обеспечение. Программа используется в ряде высоконагруженных сайтов, включая Twitter, Instagram, GitHub, StackOverflow. HAProxy является программой с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и распространяется в соответствии с [GNU General Public License](https://ru.wikipedia.org/wiki/GNU_General_Public_License). Поэтому она выбрана в качестве решения для нашей задачи.

HAProxy может быть запущено в двух различных режимах – TCP и HTTP. Режимы различаются доступной функциональности. В режиме TCP каждая публикация клиента и каждый приход топика в рамках одного TCP-соединения происходит к одной конкретной копией брокера. В режиме HTTP каждый новый топик приходит на новый брокер. В настоящей работе стоит задача балансировки самого TCP-соединения, поэтому представлено использование HAProxy в режиме TCP.

Клиент отправляет несколько запросов HAProxy, выступающему в качестве балансировщика, и получает ответ. HAProxy распространяет запросы по узлам, взаимодействуя в каждом случае с одним MQTT-брокером[Рис.4][10]. Узел может либо принять запрос, если он не перегружен, либо вернуть null или ошибку, тогда HAproxy передаст запрос следующему узлу. Если ни один из узлов не сможет предоставить услугу, то HAproxy вернёт null-значение или ошибку клиенту[11].

4.2.3. Обоснование схемы балансировки и масштабирования приложения

Каждую копию приложения можно подключить к одному брокеру или ко всем сразу. В настоящей работе при распределении нагрузки необходимо избежать дублирования данных, иначе в базе данных Clickhouse пришедшие данные будут повторяться, а избавляться от дублей непосредственно в базе данных сложнее. При этом нам также важно избежать потери данных.

При подключении одного брокера ко всем копиям серверной части платформы возникает проблема «узкого места» при взаимодействии клиентов с брокером. Также возникает дублирование попадающих с базу данных топиков. Если избежать этого и распределить клиентов по копиям сервисов, то возникает проблема синхронизации. При выходе из строя одной из копий серверной части приложения необходимо сообщить об этом остальным (иначе произойдёт потеря данных), а это само по себе является нетривиальной задачей.

В случае разъединения MQTT-брокера и сервиса возникает проблема установки соединения между ними.

Если каждое приложение подписывается на каждый брокер, то также возникает проблема дублирования.

Поэтому каждая копия приложения соединена с одним брокером, и каждая такая пара упакована в Docker-контейнер[Рис.4]. Клиент устанавливает TCP-соединение через балансировщик с одним из брокеров, которые не перегружены. Балансировщик при этом так же упакован в контейнер. При выходе из строя одного из контейнеров, приложений или брокеров, TCP-соединение для соответствующим MQTT-брокером рвётся, и переустанавливается на другой работающий и свободный брокер. Docker-контейнеризация обеспечит изоляцию, благодаря которой получится избежать конфликтов библиотек.

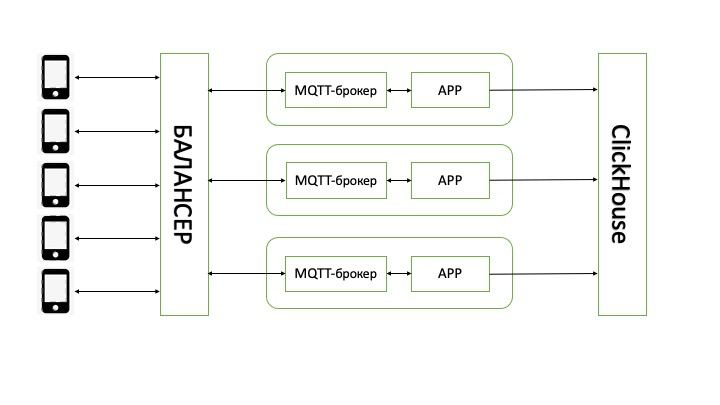


Рис.4 Балансировка нагрузки между копиями серверной части приложения

4.2.4. Аутентификация

Аутентификация осуществляется так же через один MQTT-брокер, который свободен в этот момент.

Протокол MQTT предоставляет поля для имени и пароля пользователя при запросе соединения с брокером. Эти поля используются для аутентификации, как показано на рис. 5

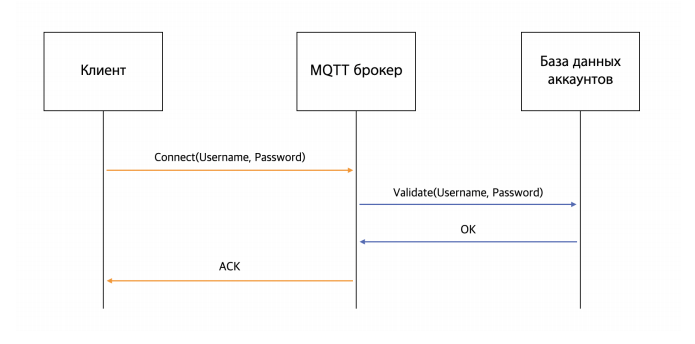


Рис.5 Процесс аутентификации

При подключении к брокеру клиент предоставляет введенные пользователем логин и пароль в полях сообщения Connect, брокер сверяет полученные данные с учётными записями, находящимися в базе данных акканутов. В настоящей реализации используется брокер Eclipse Mosquitto[12]. Eclipse Mosquitto — брокер с открытым исходным кодом (лицензии EPL/EDL), который реализует протоколы MQTT различных версий.

4.2.5. Построение платформы при помощи Docker Compose

На рис.4 балансировщик (он же балансер), а также каждая пара MQTT-брокера и приложения упакованы в контейнеры. Для управления большим количеством контейнеризированных процессов, входящих в один проект, удобно использовать Docker Compose, инструментальное средство Docker, позволяющее с помощью одной команды запускать множество сервисов. Файл docker-compose.yml содержит инструкции для запуска и настройки этих сервисов. В настоящей работе настраиваются и запускаются два сервиса – HAproxy, отвевающий за горизонтальное масштабирование приложения, и супервизор, приложение, запускающее MQTT-брокер и серверную часть платформы[Рис.6].

Запуск приложения происходит с помощью команды:

* docker-compose up – scale iomt01=3

Здесь число-параметр показывает количество копий супервизора, под контролем которого запускается брокер MQTT и серверная часть платформы.

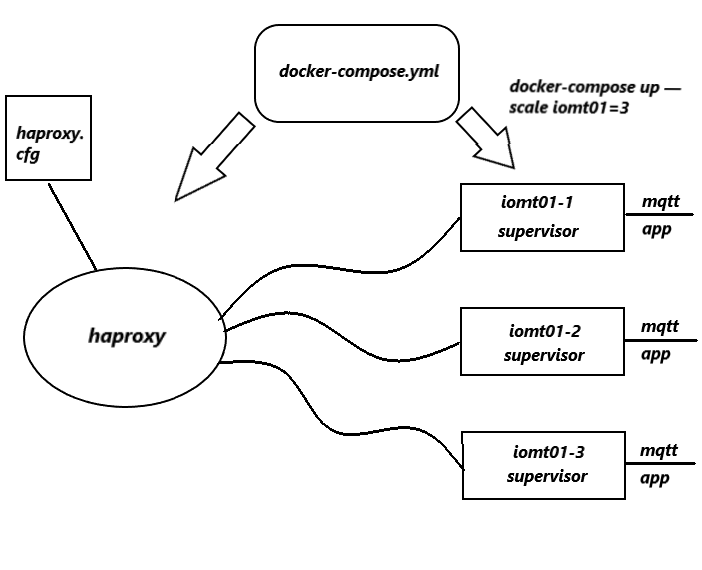


Рис.6 Запуск приложения с помощью Docker Compose

4.2.6. Dockerfile

Помимо основной задачи данной работы – масштабирования, контейнеры использованы для упрощения запуска приложения с помощью возможности упаковывать не только само приложение, но и его среду запуска.

Dockerfile –текстовый файл, в котором содержится список команд Docker-клиента. Ниже приведён пример структуры Dockerfile для серверной части платформы:

FROM ubuntu:18.04

RUN apt-get update && apt-get install -y build-essential libssl-dev python3-dev rustc libffi-dev

RUN apt-get install -y python3-pip

COPY requirements.txt /tmp/requirements.txt

RUN pip3 install --no-cache-dir -r /tmp/requirements.txt

RUN apt-get install -y supervisor

RUN mkdir -p /var/log/supervisor

COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf

COPY mqtt-daemon /iomt-project/mqtt-daemon

EXPOSE 5000 1883

CMD ["/usr/bin/supervisord"]

5. Требования к эксперименту

В предыдущих главах была продемонстрирована схема работы масштабируемой серверной платформы, однако было бы нагляднее показать все преимущества контейнеризации с помощью эксперимента. Будет сравниваться производительность масшатбируемой серверной части платформы и немасштабируемой.

Запустим определённое количество клиентов сначала с немасштабируемой серверной частью, будем увеличивать число клиентов, пока через определённый момент времени не будут начинаться проблемы с производительностью, такие как:

* потеря данных
* медленная работа
* накопление топиков

Затем проведём аналогичный запуск уже масштабируемой серверной части платформы с тем же количеством клиентов, что и в последнем эксперименте с немасштабируемой, отметим отсутствие вышеуказанных проблем.

Данное нагрузочное тестирование будет производиться с помощью программы ApacheBench.

6. Заключение

В рамках курсовой работы был проведён обзор существующих способов виртуализации, в результате которого было выбрано ПО для практической части курсовой работы.

В рамках реализации был упрощён запуск приложения путём его контейнеризации. Был разработан набор виртуальных сетевых функций и политик их масштабирования для реализации сбора, хранения и обработки медицинской телеметрии. Выполнено горизонтальное масштабирование серверной части платформы.

Завершающим этапом данной курсовой работы было описание реализации масштабируемой серверной части разрабатываемой открытой платформы для сбора и обработки физиологических данных о человеке.

В качестве возможных направлений для дальнейших исследований можно рассмотреть масштабирование веб-сайта платформы. автоматизацию масштабирования контейнеризированного приложения, а также проведение экспериментальных сравнений производительности платформы с реализацией масштабируемости и без неё.

1. Список литературы:
2. Hexoskin [Электронный ресурс]. – режим доступа:

<https://www.hexoskin.com/>

1. A.M.Potdar, D.G.Narayan, S.Kengond, and M.M.Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” Procedia Comput. Sci., vol. 171, no. 2019, pp. 1419–1428, 2020, doi: 10.1016/j.procs.2020.04.152.
2. OpenVZ Documentation. [Электронный ресурс]. - режим доступа:

<https://docs.virtuozzo.com/master/>

1. LXC Documentation. [Электронный ресурс]. - режим доступа:

<https://linuxcontainers.org/ru/lxc/documentation/>

1. Docker Documentation. [Электронный ресурс]. - режим доступа:

<https://docs.docker.com/>

1. Virtual Box Documentation. [Электронный ресурс]. - режим доступа:

<https://www.virtualbox.org/wiki/Documentation>

1. Ю.В. Аникевич “Разработка и реализация серверной части платформы для сбора и обработки физиологических данных о человеке”, 2020
2. QTT Version 3.1.1 documentation. [Электронный ресурс]. - режим доступа

[MQTT Version 3.1.1 (oasis-open.org)](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf)

1. HAproxy Documentation. [Электронный ресурс]. - режим доступа:
2. “How to build an high availability MQTT cluster for the Internet of Things” [Электронный ресурс] – режим доступа

[How to Build an High Availability MQTT Cluster for the Internet of Things | by Lelylan Blog | Medium](https://medium.com/@lelylan/how-to-build-an-high-availability-mqtt-cluster-for-the-internet-of-things-8011a06bd000)

1. Marischa Elveny, Ari Winata, Baihaqi Siregar and Rahmad Syah “A tutorial: Load balancers in a container technology system using docker swarms on a single board computer cluster”, EEO, vol. 19, issue 4, no. 2020, pp.744-751, doi: [10.17051/ilkonline.2020.04.178](http://dx.doi.org/10.17051/ilkonline.2020.04.178)
2. Eclipse Mosquitto [Электронный ресурс] – режим доступа

[Eclipse Mosquitto](https://mosquitto.org/)