Бахмуров А.Г., Бодров А.О.

Построение масштабируемой платформы сбора медицинской телеметрии

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, в современных публикациях упоминаемый 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.

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.

1. Обзор технологий виртуализации

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

1. Виртуальные машины (VM)
2. Контейнеризацию [2]

1.1 Виртуальные машины (VM)

Виртуальная машина – это программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы (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 с ним.

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

1.2 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 системых [3], а не аппаратного обеспечения [Рис.1]. Благодаря виртуализации на уровне ОС, то есть изоляции только приложения, бинарных файлов и библиотек, возможно быстрое развёртывание, простое масштабирование. Также уменьшается объём занимаемого места на жёстком диске по сравнению с виртуализацией с помощью виртуальных машин.

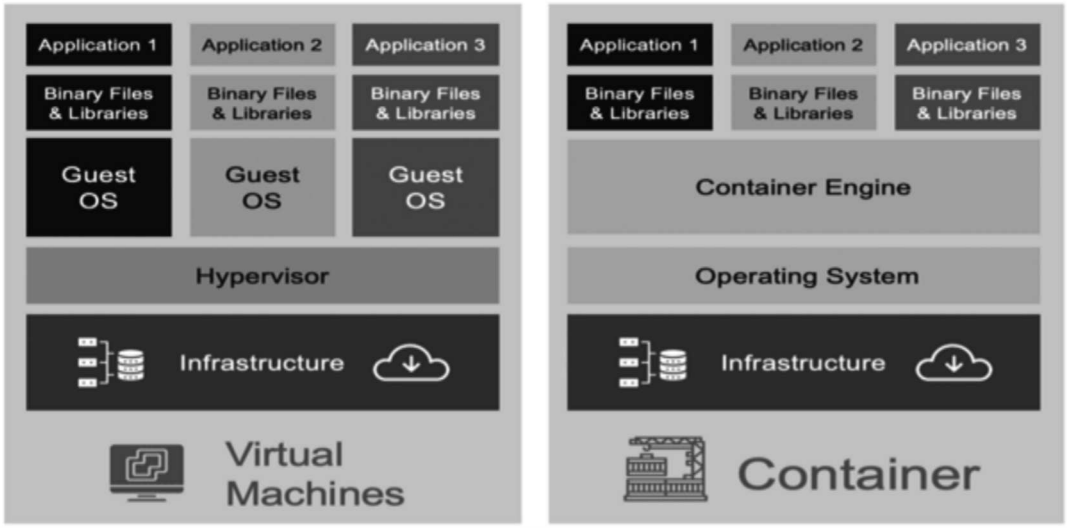

Рис. 1: Сравнение технологий виртуализации виртуальных машин и контейнеров

2. Описание реализации

2.1 Исходные данные

Схема работы приложения

На рис.2 показана упрощенная схема работы платформы, для серверной части которой необходимо реализовать масштабируемость [4].

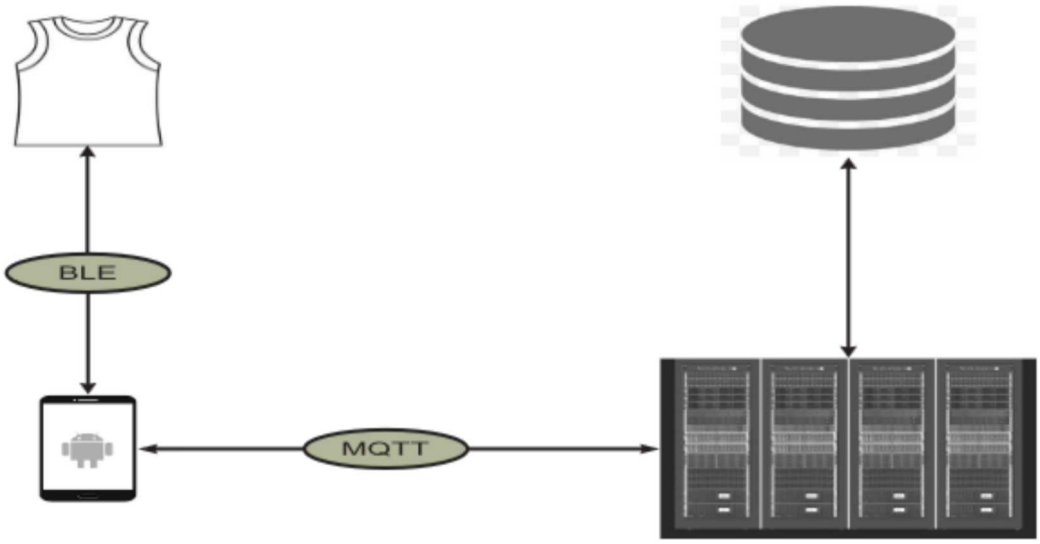

Рис. 2: Схема работы платформы по сбору медицинской телеметрии

В работе сервиса можно выделить следующие этапы:

1. Показатели физиологического состояния пользователя считываются с помощью датчиков умной одежды Hexoskin [1], 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.

MQTT

Передача данных между клиентом и сервером выполняется по протоколу MQTT [5].

MQTT (message queuing telemetry transport) – 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 центральный сервер, который называется брокером (Рис. 5). Клиенты взаимодействуют с брокером через TCP-соединение. Клиенты публикуют топики по TCP-соединению. Топиком назовём механизм, позволяющий давать имена сущностям. Каждое публикуемое сообщение содержит поле для имени топика, которое идентифицирует 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.

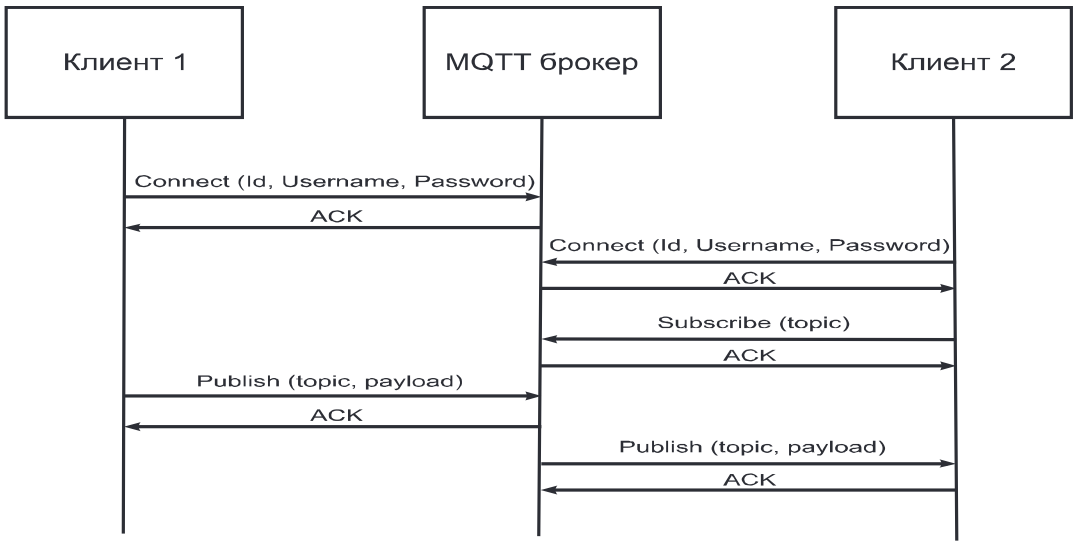

Рис. 3: Схема работы протокола MQTT

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

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.

2.2 Контейнеризация и масштабирование приложения

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.

Балансировка нагрузки при помощи HAproxy

В качестве балансировщика нагрузки был выбран HAProxy [6].

HAProxy — серверное программное обеспечение для обеспечения высокой доступности и балансировки нагрузки посредством распределения входящих запросов на несколько обслуживающих серверов.

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, а протокол MQTT работает поверх TCP, поэтому представлено использование HAProxy в режиме TCP.

Клиент отправляет запросы HAProxy, выступающему в качестве балансировщика, и получает ответ на каждый. HAProxy распространяет запросы по узлам, взаимодействуя в каждом случае с одним MQTT-брокером (Рис. 4) [7]. 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 [8].

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.

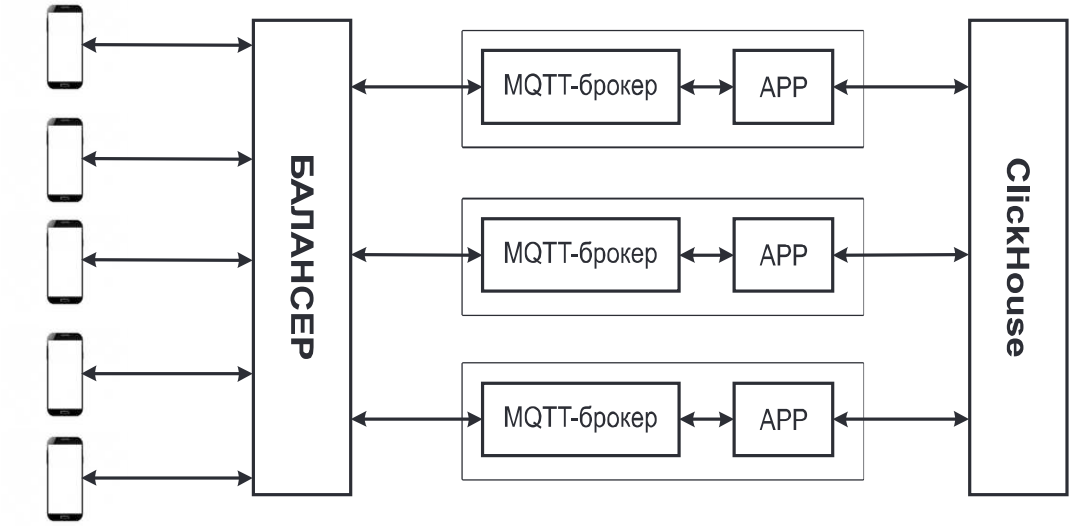

Рис. 4: Балансировка нагрузки между копиями серверной части приложения

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.

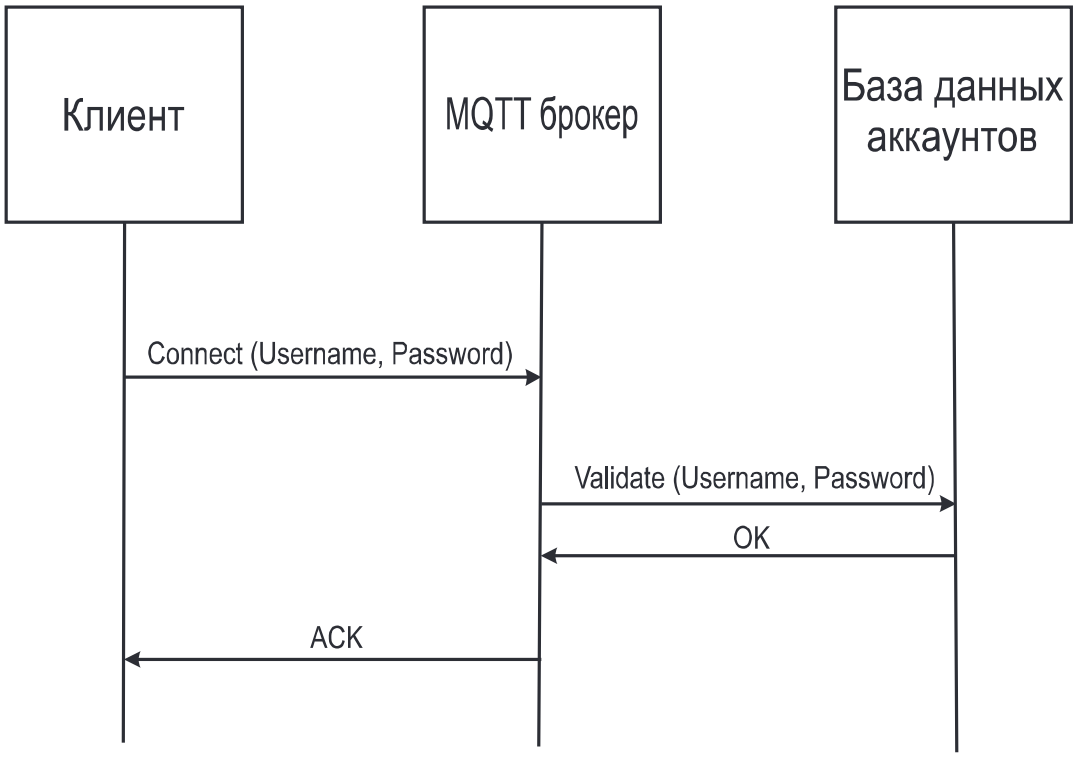

Рис. 5: Процесс аутентификации

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 [9]. Eclipse Mosquitto — брокер с открытым исходным кодом (лицензии EPL/EDL), который реализует протоколы MQTT различных версий.

3. Благодарности

Авторы благодарят сотрудника компании "Яндекс"А.А.Любицкого за техническую помощь и консультации по используемым в проекте программным средствам

Заключение

Разработанная в рамках данной работы система позволяет увеличить нагрузку на серверную часть приложения за счёт горизонтального масштабирования платформы с помощью технологий балансировки нагрузки и контейнеризации. За счёт проделанных исследований повышена отказоустойчивость приложения.

Литература

1. Hexoskin [Электронный ресурс]. – режим доступа: https://www.hexoskin.com/

2. 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.

3. Akf partners. [Электронный ресурс]. – режим доступа: https://akfpartners.com/growth-blog/vms-vs-containers

4. Аникевич Ю. В., Бахмуров А. Г., Чайчиц Д. А. Разработка и реализация платформы для сбора и обработки физиологических данных о человеке // Программные системы и инструменты. Тематический сборник № 20 / Под ред. В. А. Антоненко, В. В. Балашов, В. Г. Баула и др. — Т. 20. — Москва: 2020. — С. 36–48. https://drive.google.com/file/d/1L4YhNNTOrhu27o775aA2RqR_1dbgoB6v/view

5. MQTT Version 3.1.1 documentation. [Электронный ресурс]. - режим доступа: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

6. HAproxy Documentation. [Электронный ресурс]. - режим доступа: https://www.haproxy.com/

7. "How to build an high availability MQTT cluster for the Internet of Things" [Электронный ресурс] – режим доступа: https://medium.com/@lelylan/how-to-build-an-high-avai

lability-mqtt-cluster-for-the-internet-of-things-8011a06bd000

8. 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

9. Eclipse Mosquitto [Электронный ресурс] – режим доступа: https://mosquitto.org/