УДК 004.415 ***10.5281/zenodo.3594785***

Д. Авельцов,
Институт машиноведения и автоматики НАН КР, Бишкек

РАЗРАБОТКА ПРОГРАММНЫХ ЭЛЕМЕНТОВ ВИРТУАЛЬНЫХ ДАТЧИКОВ С ПОМОЩЬЮ КОНТЕЙНЕРОВ

В статье описаны методы виртуализации датчиков с применением docker контейнеров для предоставления датчиков как услуги клиентам облака, сети docker и примеры их использования. Приведены примеры создания docker образов и контейнеров с установкой виртуального датчика.

Ключевые слова: Виртуализация датчиков, docker, docker-контейнер, Infrastructure-as-a-Service, IaaS, сети docker, docker-compose, docker-swarm, reverse nginx proxy

Введение

Виртуализация датчиков – это метод, позволяющий нескольким клиентам работать в виртуальной среде и изолировать приложения(клиенты) от оборудования. Изоляция осуществляется промежуточным программным обеспечением, создающим несколько логических экземпляров физического сенсорного узла. Виртуализация датчиков используется в программно-аппаратных системах, основанных на архитектуре sensor-cloud [1].

Общая архитектура системы. Разрабатываемая система основывается на микросервисной архитектуре [2,3]. Микросервисная архитектура — вариант сервис-ориентированной архитектуры программного обеспечения, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей – микросервисов [4].

Свойства, характерные для микросервисной архитектуры:

- модули можно легко заменить в любое время: акцент на простоту, независимость развертывания и обновления каждого из микросервисов;
- модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;
- модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;
- архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.

Виртуализация датчиков (рис. 2) – это метод, позволяющий нескольким клиентам работать в виртуальной среде и изолировать приложения (клиенты) от оборудования. Изоляция осуществляется промежуточным программным обеспечением, создающим несколько логических экземпляров физического датчики.

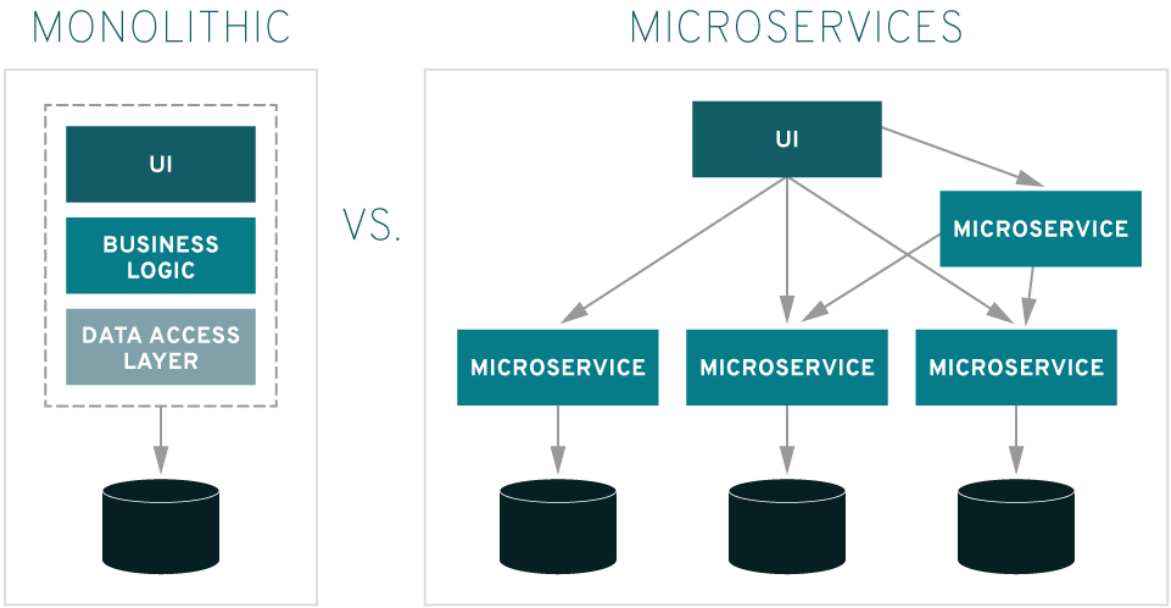

Рисунок 1 – Отличия микросервисного и монолитного архитектурного подхода в проектировании

Физические датчики проводят различные измерения (например, температуры, давления, освещенности, уровня вибрации, местоположения и т. п.), они оснащены миниатюрными вычислительными устройствами и средствами связи, работающими в заданном радиодиапазоне. Это позволяет устройству проводить измерения, самостоятельно проводить начальную обработку данных и поддерживать связь с внешней информационной системой. Такие датчики являются сенсорными узлами беспроводных сенсорных сетей (БСС) [5].

Виртуальный датчик является эмуляцией физического датчика, или программным представлением датчика, скрывающим аппаратный датчик. Виртуальные датчики обеспечивают представление сенсорных данных пользователю путем объединения данных от различных гетерогенных физических датчиков. На рисунке 3 показаны уровень физических датчиков и уровень виртуальных датчиков. Один физический датчик может быть представлен в нескольких виртуальных датчиках. Слой виртуальных датчиков лежит поверх узлов физических датчиков. Кроме того, он отделяет пользовательские приложения от физических датчиков.

Рассмотрим облако датчиков, которое предполагает их использование в качестве интерфейса между физической и цифровой средой. Облако датчиков позволяет использовать их в инфраструктуре путем виртуализации физического устройства в облаке, другими словами, предоставлять виртуальные датчики как сервис (SensIaaS, Sensing as a Service) [1]. Виртуальный датчик создается на серверах облака и предоставляется пользователю автоматически, также, как и другие облачные ресурсы: память, дисковое пространство, процессорное время, сетевая инфраструктура. Для этого в облаке должен быть подготовлен шаблон сервиса. Шаблоны предоставляются поставщиком датчиков и могут добавляться и удаляться из системы. Пользователь не имеет доступа к физическому датчику, вместо этого он может контролировать виртуальный датчик. Интеграция датчиков с облаком позволяет создать расширяемую и масштабируемую сеть. Метод виртуализации датчиков позволяет пользователю получать предпочтительную и точную информацию более эффективным способом из ограниченного числа датчиков. Кроме того, это помогает снизить энергопотребление и

стоимость развертывания и обслуживания всей сети. Виртуальные датчики имеют ряд преимуществ в таких случаях, как замена или обслуживание датчика. Виртуальный датчик непрерывно предоставляет данные пользователю, в то время как механизм сообщения с физическим датчиком является дискретным, что позволяет снизить энергозатраты батареи физического устройства. Кроме того, в случае сбоя физического датчика, либо потери связи с физическим датчиком, виртуальный датчик позволит продолжить обмен данными.

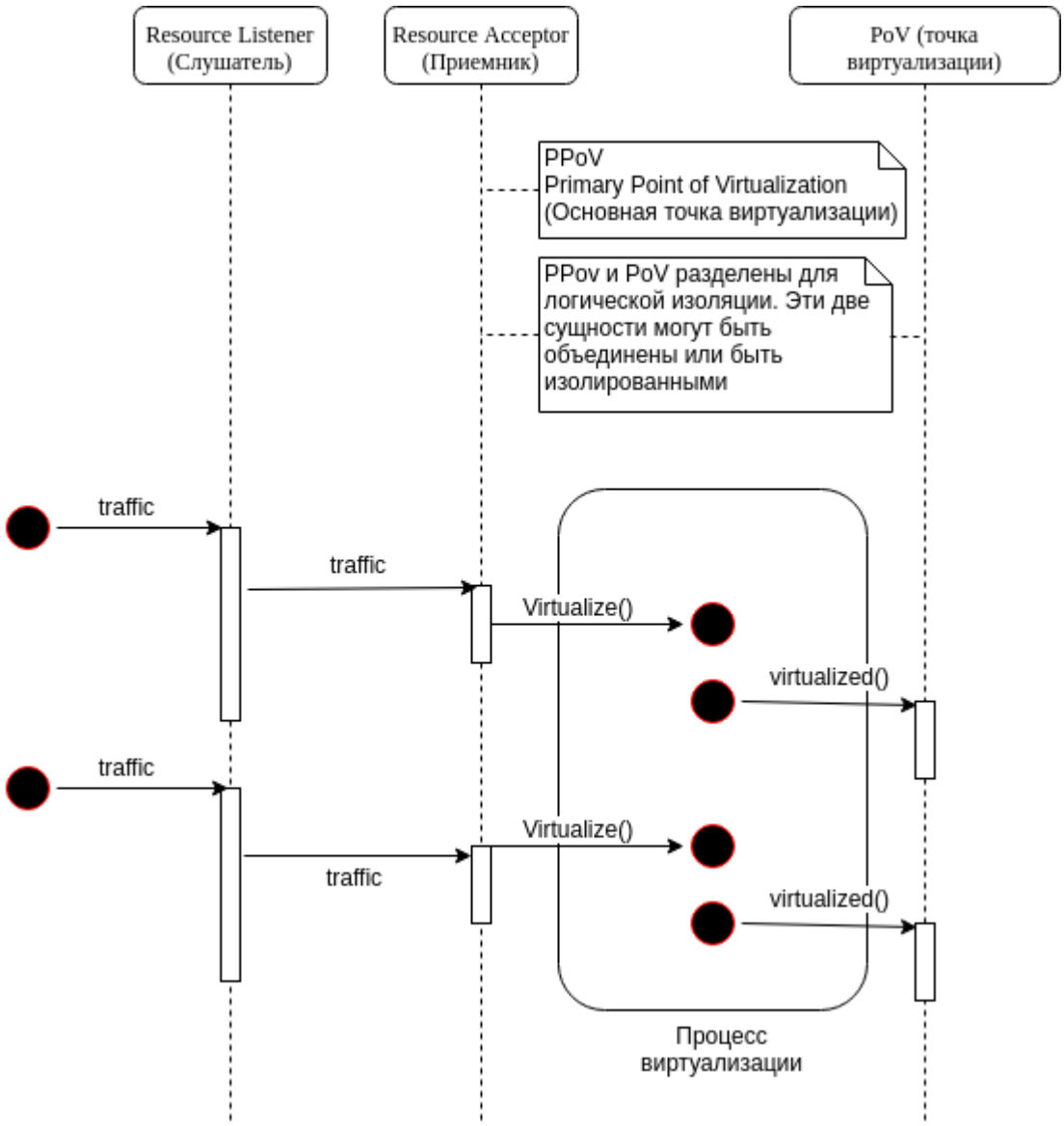

Рисунок 2 – Диаграмма последовательностей виртуализации датчиков

Инфраструктура как услуга (англ. Infrastructure-as-a-Service; IaaS) [1,6] – одна из моделей обслуживания в облачных вычислениях, по которой потребителям предоставляются по подписке фундаментальные информационно-технологические ресурсы – виртуальные серверы с заданной вычислительной мощностью, операционной системой (чаще всего предустановленной провайдером из шаблона) и доступом к сети. Platform as a Service (PaaS, «платформа как услуга») [1,6] – модель

предоставления облачных вычислений, при которой потребитель получает доступ к использованию информационно технологических платформ: операционных систем, систем управления базами данных, связующему программному обеспечению, средствам разработки и тестирования, размещенным у облачного провайдера. В этой модели вся информационно-технологическая инфраструктура, включая вычислительные сети, серверы, системы хранения, целиком управляется провайдером, провайдером же определяется набор доступных для потребителей видов платформ и набор управляемых параметров платформ, а потребителю предоставляется возможность использовать платформы, создавать их виртуальные экземпляры, устанавливать, разрабатывать, тестировать, эксплуатировать на них прикладное программное обеспечение, при этом динамически изменяя количество потребляемых вычислительных ресурсов.

Виртуальный датчик. Дано множество R физических датчиков {ri |∀ri∈R} , каждый имеет тип {ωi |∀ωi∈Ω} , подмножество(узел) B⊂R , имеющее один или несколько физических датчиков, может быть размещено за интерфейсом виртуального сенсора υi , на уровнях IaaS и PaaS следующим образом:

IaaS уровень

$$\upsilon_i \rightarrow \begin{cases} r_i & \textit{singular ...(i)} \\ \{r_i | r_i \in B\} & \textit{selector ...(ii)} \\ B & \textit{accumulator ...(iii)} \\ h(\mathrm{B}) & \textit{aggregator ...} \ (i\upsilon) \\ \{r_i : \forall r_i \in B \rightarrow f(r_i)\} & \textit{qualifier ...} \ (\upsilon) \\ \{r_i : \forall r_i \in B \rightarrow f(r_i, B)\} & \textit{context qualifier ...} \ (\upsilon i) \end{cases}$$

PaaS уровень

$$\upsilon_i \rightarrow \begin{cases} p(r_i^{\tau} : \tau \in T) & \textit{predicted ...} \ (\upsilon ii) \\ c(B) & \textit{computed ...} \ (\upsilon iii) \end{cases}$$

Описание для всех типов виртуальных датчиков, указанных выше: (i) Singular(Сингулярные): Соотношение между физическим датчиком и виртуальным – один-к-одному. Для каждого датчика ri∈B , создается виртуальный υi датчик.

(ii) Selector (Селективные): Набор одинаковых физических датчиков представлен как один виртуальный датчик υi. Виртуальный датчик получает данные от любого из физических датчиков группы, на основе алгоритма выбора. (соотношение многие-к-одному).

(iii) Accumulator (Аккумулирующие): Узел B, который содержит однородные или гетерогенные датчики и может быть представлен как один составной интерфейс предоставляемый виртуальным датчиком υi . (соотношение многие-к-одному).

(iv) Aggregator (Агрегирующие): Агрегатная функция h применяется к считываемым данным, которые приходят со всех физических датчиков ri: ∀ri∈B, и возвращаемый результат агрегатной функции представлен, как виртуальный датчик υi. (соотношение многие-к-одному).

(v) Qualifier (Квалифицирующие): Виртуальный датчик υi создается и выделяется соответствующему физическому датчику ri, но данные с физического датчика доставляются, только если они отобраны с помощью квалификационной функции f.

(vi) Context Qualifier (Квалифицирующие по контексту): Как и (v) Qualifier (квалифицирующие), но квалификационная функция f получает данные с одного или более физических датчиков {rj |∀rj∈B∧rj≠ri } находящихся в узле B.

(vii) Predictor (Прогнозирующие): Виртуальный датчик υi может содержать алгоритмом прогнозирования, который на основе анализа данных временных рядов на раннее считанные данные, может спрогнозировать возможные данные, которые будут получены от физического датчика, если этот датчик выйдет из строя.

(viii) Compute (вычисляющие): Виртуальный датчик υi может содержать некоторые вычислительные алгоритмы, которые анализируют входящие данные от сенсорных узлов и преобразуют их.

Виртуализация сети основана на виртуальных датчиках и виртуальных каналах, которые позволяют нескольким сетям сосуществовать на одной физической платформе. Виртуальные сети могут быть сформированы подмножеством физической сети. Виртуализация сенсорной сети может являться взаимосвязью различных БСС. Она формируется подмножеством универсальных узлов сенсорной сети, при этом подмножество или группа сенсоров предназначены для выполнения определенных задач или способствуют выполнению определенного приложения в данный момент времени [7].

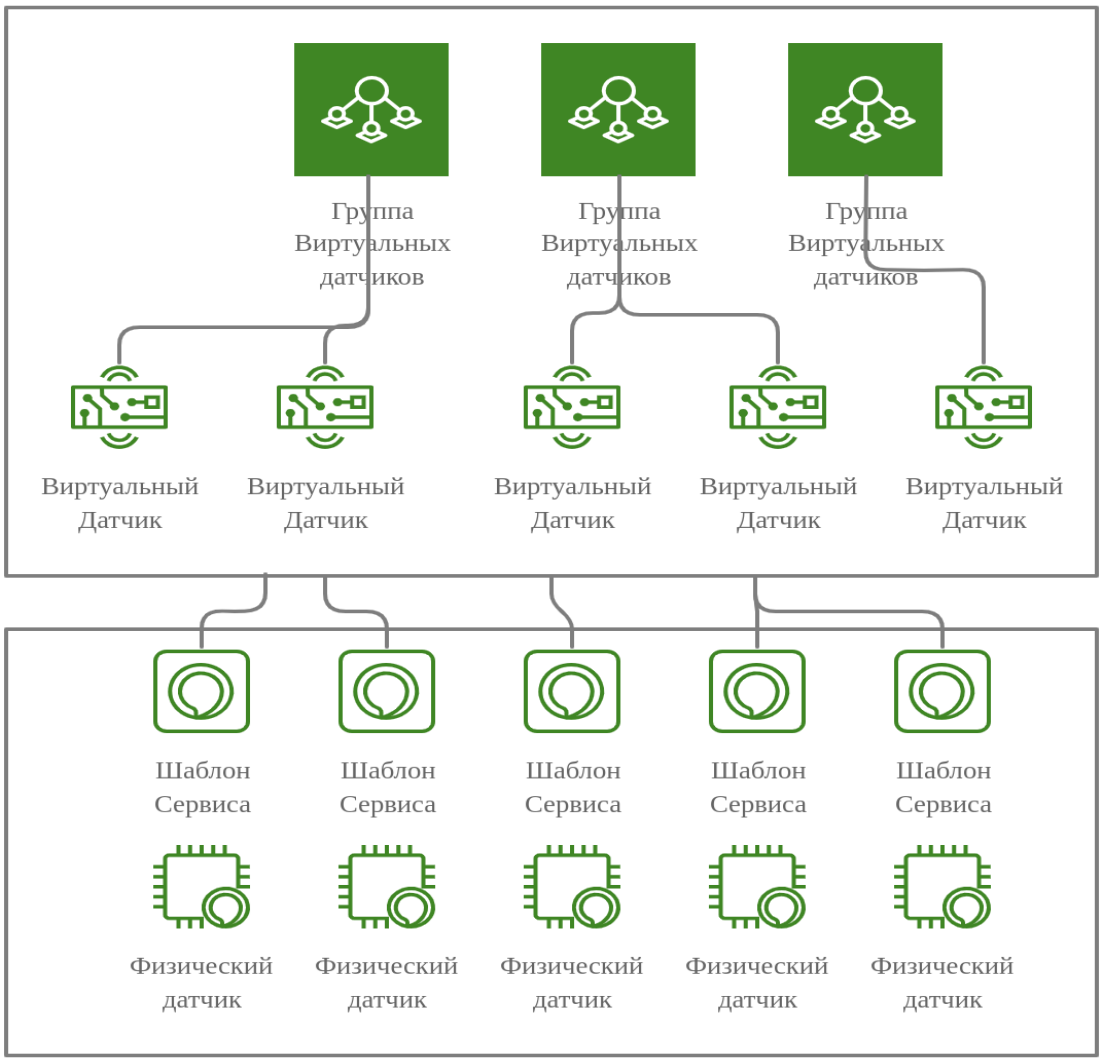

Рисунок 3 – Уровень физических датчиков и уровень виртуальных датчиков

Подключение датчика к системе и создание шаблона. Владелец физических датчиков подключает, физические датчики и регистрирует их в облачной системе, создает шаблоны датчиков (Рис. 4). Для создания шаблона датчика пользователю предоставляется веб-формы для указания формата передаваемых данных устройства [8,9] (Рис. 7).

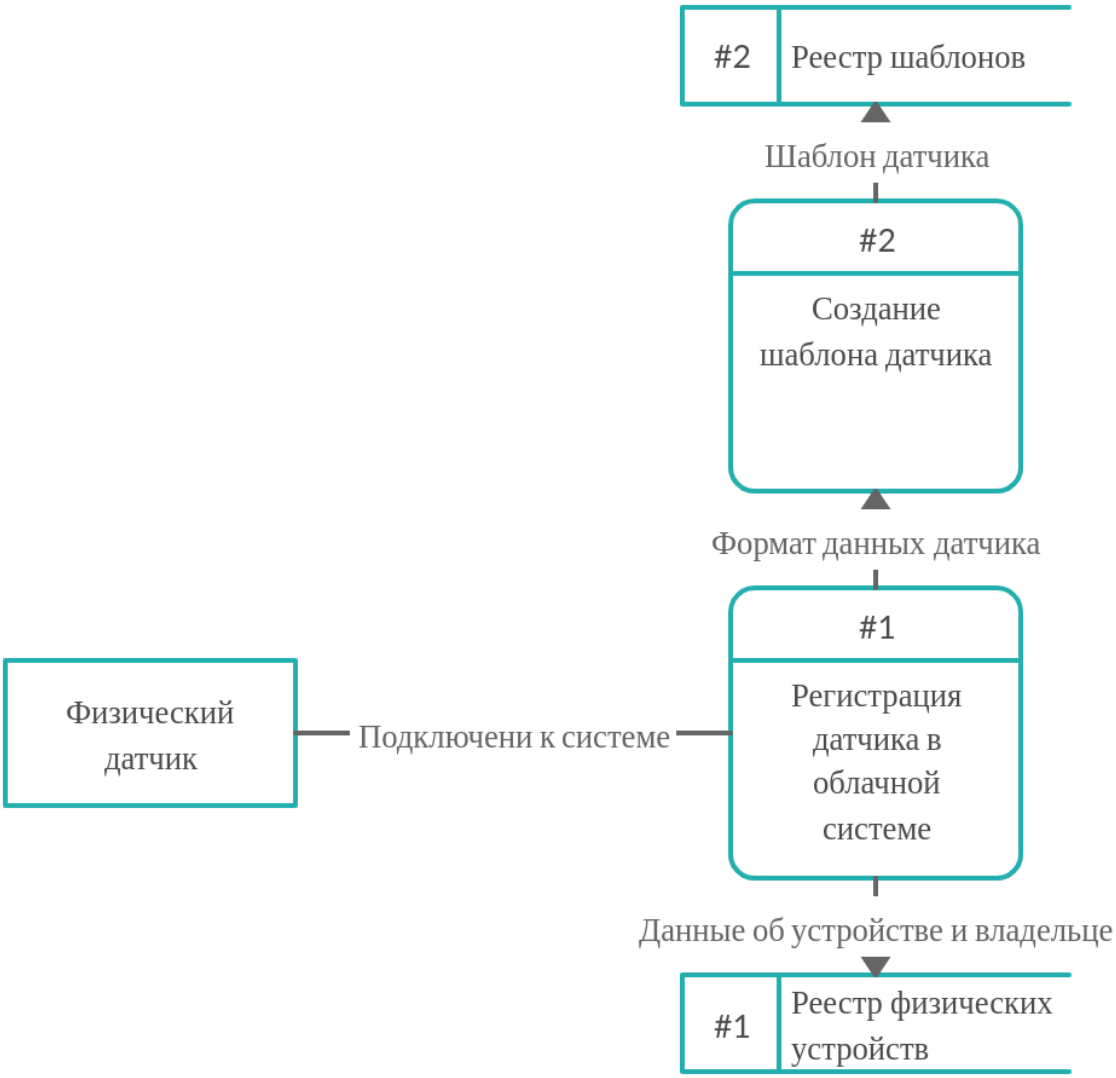

Рисунок 4 – Подключение датчика к системе и создание шаблона.

Главными действующими сущностями являются:

• Владелец физических датчиков (человек, частная или правительственная организация) – подключает, отключает физические датчики в своих БСС и регистрирует их в облачной системе, создает шаблоны датчиков, таким образом делая доступными для широкого использования данные своих датчиков (Рис. 5).

• Администратор/владелец инфраструктуры – управляет инфраструктурой облака, подключает и регистрирует другие ресурсы и создает шаблоны сервисов и датчиков. Шаблоны сервисов и датчиков хранятся в хранилище (каталоге) шаблонов (Рис. 5).

• Пользователи (люди, организации и приложения) – создают запросы на сервисы через Web-интерфейс и получают данные с физических датчиков [1] (Рис. 6).

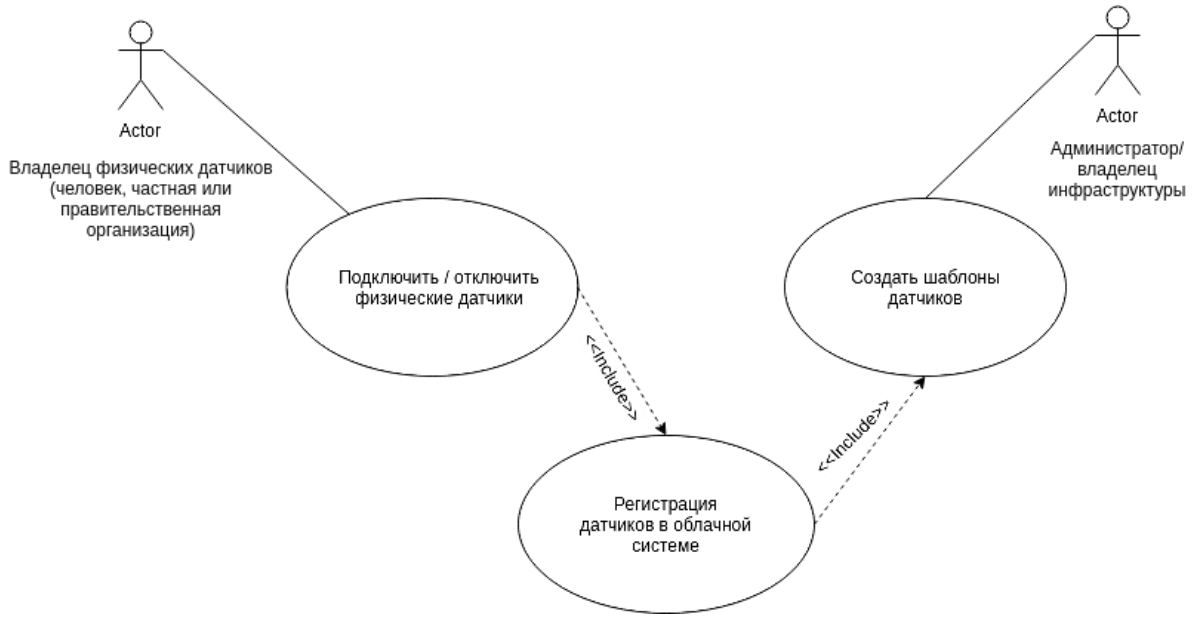

Рисунок 5 – Диаграмма вариантов использования для владельца физических датчиков и администратора инфраструктуры

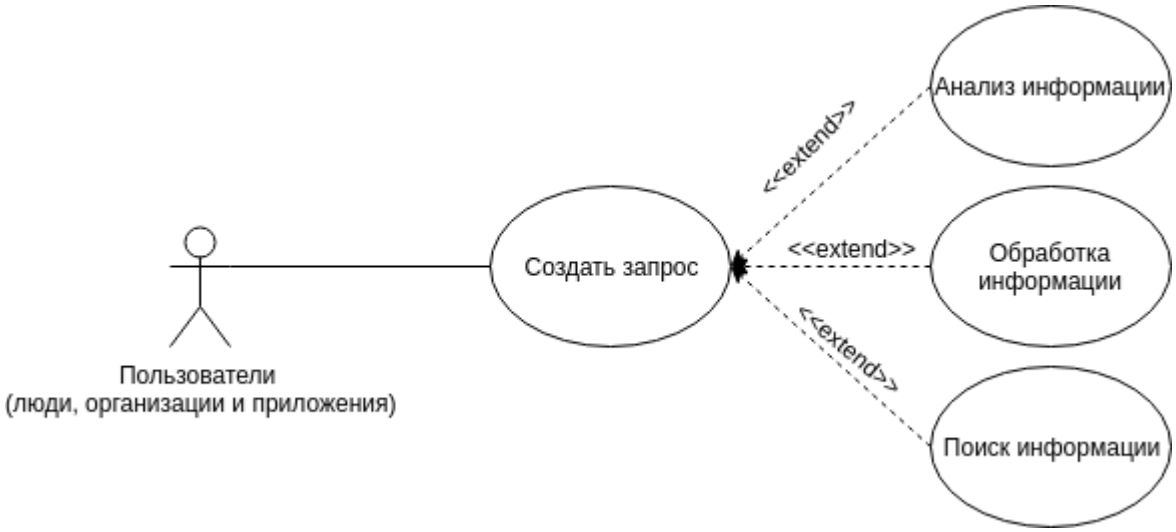

Рисунок 6 – Диаграмма вариантов использования для пользователей (люди, организации и приложения).

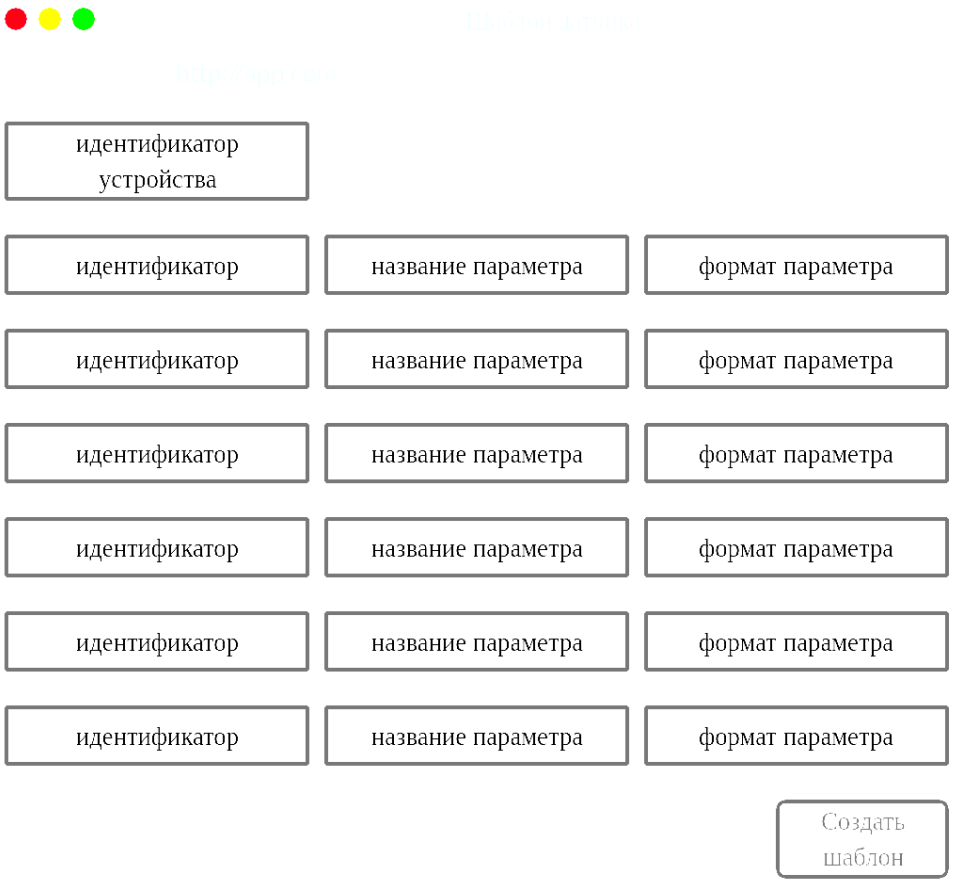

Рисунок 7 – Веб-форма создания шаблона датчика

Сети Docker. Для взаимодействия с внешним миром Docker применяет стек IP, используя TCP или UDP. Он поддерживает инфраструктуры адресации IPv4 и IPv6.

Сеть Docker построена на Container Network Model (CNM), которая позволяет создать свой сетевой драйвер. Таким образом, у контейнеров есть доступ к разным типам сетей и они могут подключаться к нескольким сетям одновременно[10]. Помимо различных сторонних сетевых драйверов, есть также встроенные драйверы:

- Bridge: в этой сети контейнеры запускаются по умолчанию. Связь устанавливается через bridge-интерфейс на хосте. У контейнеров, которые используют одинаковую сеть, есть своя собственная подсеть, и они могут передавать данные друг другу по умолчанию.
- Host: этот драйвер дает контейнеру доступ к собственному пространству хоста (контейнер будет видеть и использовать тот же интерфейс, что и хост).
- macvlan: Маквлан,- это сетевой драйвер, который позволяют назначать MAC-адрес контейнеру, делая его отображаемым как физическое устройство в вашей сети. Docker демон направляет трафик на контейнеры по их MAC-адресам. Использование macvlan драйвера иногда является лучшим выбором при работе с устаревшими приложениями, которые ожидают, что они будут напрямую подключены к физической сети.
- overlay/overlay2: Оверлей (Наложенная сеть), – это сетевой драйвер для соединения несколько демонов Docker между собой и которые позволяют docker-swarm

службам взаимодействовать друг с другом. Также возможно использовать оверлейные сети для облегчения связи между docker-swarm и автономным контейнером или между двумя отдельными контейнерами на разных Docker демонах. Эта стратегия устраняет необходимость выполнения маршрутизации на уровне ОС между этими контейнерами.

- none: означает отсутствие сетевого устройства − это сетевой драйвер, который позволяет отключать всю сеть для контейнеров. Обычно используется в сочетании с пользовательским сетевым драйвером.
- Network plugins: Дает возможность установить и использовать сторонние сетевые плагины с Docker контейнерами. Эти плагины доступны в Docker Store или у сторонних поставщиков услуг.
- Применение
- Мост (bridge) лучше всего использовать для связи нескольких контейнеров на одном и том же Docker хосте.
- Хост (host) сети лучше всего использовать, когда сетевой стек не должен быть изолирован от хоста Docker, но есть необходимость в том, чтобы другие аспекты контейнера были изолированы.
- Маквлан (macvlan) сети лучше всего использовать, когда требуется переход с выделенной VM на контейнеры или, требуется чтобы контейнеры выглядели как физические хосты в сети, каждый с уникальным MAC-адресом.

Пример создания overlay сети

```
$ docker network create -d overlay network-name
c8668b3fc7f6ac9d8694a34d225327f9abb1d195c9757966919ed4adf9b35cea
```

Подключение контейнера к сети

```
$ docker network connect <network_name> <container_name>
```

Overlay-сети используются в контексте кластеров (Docker Swarm), где виртуальная сеть, которую используют контейнеры, связывает несколько физических хостов, на которых запущен Docker. Когда запускается контейнер на swarm-кластере (как часть сервиса), множество сетей присоединяется по умолчанию, и каждая из них соответствует разным требованиям связи.

Создание контейнера с виртуальным датчиком в виде программного обеспечения. Для адресации в качестве временного решения используется reverse nginx proxy сервер jwilder/nginx-proxy[11]. Пример настройки nginx-proxy-compose.yaml services:

```
nginx-proxy:
  restart: always
  image: jwilder/nginx-proxy
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - "/etc/nginx/vhost.d"
    - "/usr/share/nginx/html"
    - "/var/run/docker.sock:/tmp/docker.sock:ro"
    - "/etc/nginx/certs"

letsencrypt-nginx-proxy-companion:
```

```
    restart: always
    image: jrcs/letsencrypt-nginx-proxy-companion
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
    volumes_from:
      - "nginx-proxy"
```

Создание контейнеров docker-compose -f nginx-proxy-compose.yaml up -d

Пример вывода в консоль

```
Creating network "go-docker_default" with the default driver
Pulling nginx-proxy (jwilder/nginx-proxy:)...
latest: Pulling from jwilder/nginx-proxy
a5a6f2f73cd8: Pull complete
2343eb083a4e: Pull complete
...
Digest:
sha256:619f390f49c62ece1f21dfa162fa5748e6ada15742e034fb86127e6f443b40bd
Status: Downloaded newer image for jwilder/nginx-proxy:latest
Pulling letsencrypt-nginx-proxy-companion (jrcs/letsencrypt-nginx-proxy-
companion:)...
latest: Pulling from jrcs/letsencrypt-nginx-proxy-companion
...
Creating go-docker_nginx-proxy_1 ... done
Creating go-docker_letsencrypt-nginx-proxy-companion_1 ... done
```

Пример образа контейнера с виртуальным сенсором внутри в качестве программного обеспечения

```
FROM golang:alpine AS build
RUN apk --no-cache add gcc g++ make git
WORKDIR /go/src/app
COPY . .
RUN go get ./...
RUN GOOS=linux go build -ldflags="-s -w" -o ./bin/sensor-app ./main.go

FROM alpine:3.10
RUN apk --no-cache add ca-certificates
WORKDIR /usr/bin
COPY --from=build /go/src/app/bin /go/bin
EXPOSE 80
ENTRYPOINT /go/bin/sensor-app --port 80
```

Пример настройки docker-compose файла [12] go-sensor-compose.yaml для виртуальных сенсоров

```
version: '2'
services:
  go-web-app:
    restart: always
    build:
      dockerfile: Dockerfile
      context: .
```

Создание контейнера [13] docker-compose -f go-sensor-compose.yaml up -d

```
Вывод на экран
Creating network "go-docker_default" with the default driver
Building go-sensor-app
Step 1/12 : FROM golang:alpine AS build
 ---> b97a72b8e97d
...
Successfully tagged go-docker_go-sensor-app:latest
WARNING: Image for service go-sensor-app was built because it did not already exist.
To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating go-docker_go-sensor-app_1 ... done
```

Организация структуры данных реестра датчиков. Структура базы данных разработана таким образом, чтобы избежать дублирования данных, обеспечить максимальную скорость доступа к ним, а также максимально облегчить программирование операций обращения к базе данных (Рис. 8).

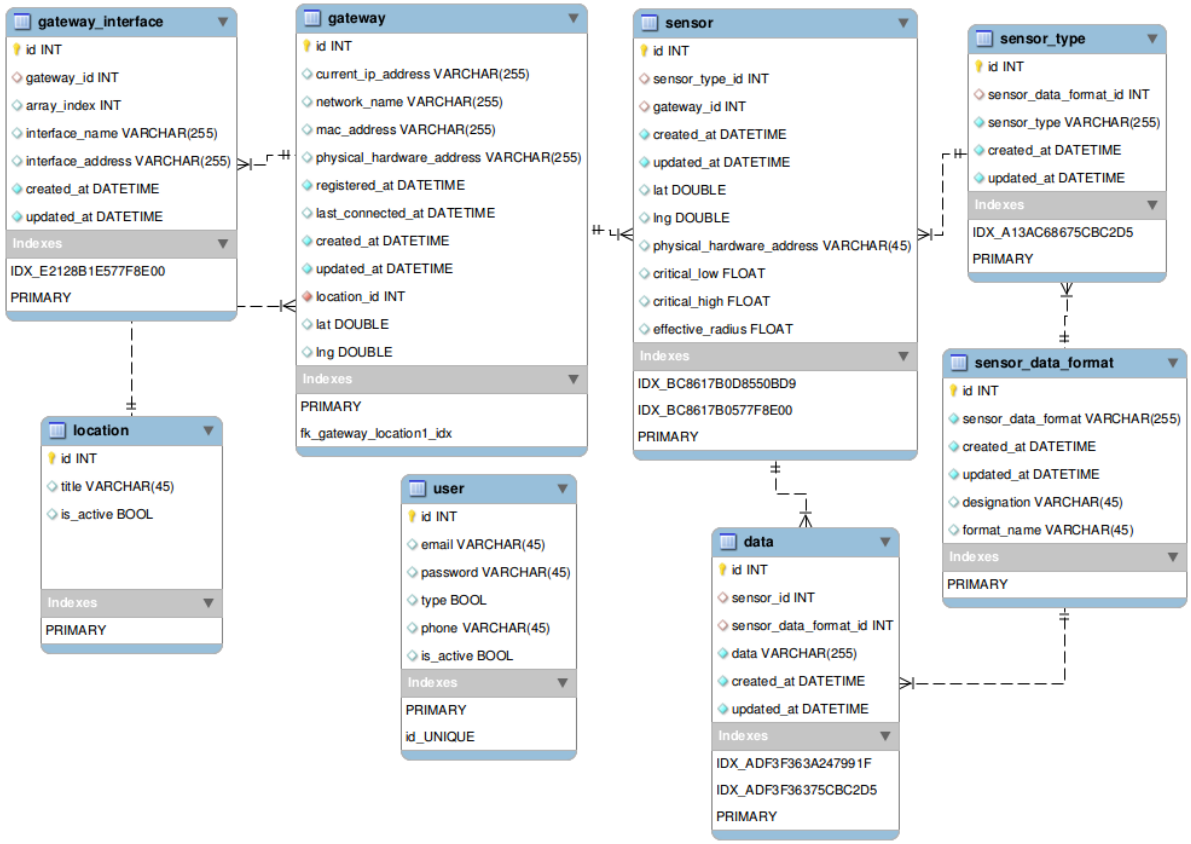

Рисунок 8 – Схема базы данных (бета версия)

Заключение

Данная статья является первой из серии статей о разработке программной системы виртуализации датчиков. Виртуализация измерительных устройств решает несколько проблем – снижает энергопотребление измерительных устройств путем дискретизации соединения между виртуальной копией устройства и его физическим представлением; масштабируемость устройства на необходимое количество пользователей; администрирование устройств и групп устройств; описание формата датчика, и его данных (шаблон датчика). Виртуальные датчики являются главным сервисом, предоставляемым облачными информационно-измерительными системами.

Литература

1. Гайдамако В.В. Инфраструктура SENSOR-CLOUD – облачные информационно-измерительные системы // Проблемы автоматики и управления. – 2018. – № 2 (35). С. 109–118
2. Richardson, C. (2019). “What are microservices?” Retrieved from https://microservices.io/. (дата обращения 03.12.2019).
3. Chris Richardson of Eventuate, Inc. May 19, 2015. “Introduction to Microservices”. https://www.nginx.com/blog/introduction-to-microservices/ (дата обращения 03.12.2019).
4. Fowler, M. (2014, March 25). “Microservices: a definition of this new architectural term.” Retrieved from https://martinfowler.com/articles/microservices.html. (дата обращения 03.12.2019).
5. Imran Khan, Student Member, IEEE, Fatna Belqasmi, Member, IEEE, Roch Glitho, Senior Member, IEEE, Noel Crespi, Senior Member, IEEE, Monique Morrow and Paul Polakos - Wireless Sensor Network Virtualization: A Survey.
6. Khan, I.; Errounda, F.Z.; Yangui, S.; Glitho, R.; Crespi, N., – Getting Virtualized Wireless Sensor Networks’ IaaS Ready for PaaS / Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference.
7. Zubair Khalid, Norsheila Fisal and Mohd. Rozaini – A Survey of Middleware for Sensor and Network Virtualization / 2014 UTM-MIMOS Centre of Excellence in Telecommunication Technology.
8. Sanem Kabadayi, Adam Pridgen, and Christine Julien The Center for Excellence in Distributed Global Environments The Department of Electrical and Computer Engineering The University of Texas at Austin – Virtual Sensors: Abstracting Data from Physical Sensors
9. Sanem Kabadayı and Christine JulienThe Center for Excellence in Distributed Global Environments The Department of Electrical and Computer Engineering The University of Texas at Austin – Remotely Deployed Virtual Sensors
10. https://docs.docker.com/v17.09/engine/userguide/networking/ (дата обращения 05.12.2019).
11. https://github.com/jwilder/nginx-proxy (дата обращения 05.12.2019).
12. https://docs.docker.com/compose/ (дата обращения 05.12.2019).
13. https://docs.docker.com/engine/reference/commandline/container/ (дата обращения 05.12.2019).