



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ

ПИ (филиал) ДГТУ в г. Таганроге

Факультет «Высшего образования»»

наименование факультета

Кафедра «Технический сервис и информационные технологии»»

наименование кафедры

ЛАБОРАТОРНАЯ РАБОТА

Дисциплина (модуль) Перспективные информационные технологии

наименование учебной дисциплины (модуля)

на тему: Лабораторная работа 3.1 «Схемы распределительных систем с IoT. Виды и типы»

Направление подготовки/специальность 09.03.02 Информационные системы и технологии

код

наименование направления подготовки/специальности

Направленность (профиль) Информационные системы и технологии

Номер зачетной книжки 2282149 Группа ВЗ ИСиТ – 41

Обучающийся

подпись, дата

А.Ю.Галетко

И.О. Фамилия

Контрольную работу проверил _____

подпись, дата

М.В. Орда-Жигулина

должность, И.О. Фамилия

Таганрог
2026г.

Содержание

Введение	3
Задание 1	4
Задание 2	8
Задание 3	12
Задание 4	17
Задание 5	19
Заключение	24

					490000.000			
Изм.	Лист	№ докум.	Подпись	Дат	Перспективные информационные технологии			
Разраб.		Галетко А.Ю.						
Провер.		Орда-Жигулина М.В						
Н. Контр.								
Утверд.					<div>Лит.</div> <div>Лист</div> <div>Листов</div> <div> <div></div> <div>2</div> <div>24</div> </div> <div>ПИ (филиал) ДГТУ в г. Таганроге</div>			

Введение

Соблюдение стандартов, таких как ГОСТ, обеспечивает единообразие, понятность и универсальность создаваемых схем. Это позволяет разным специалистам «говорить на одном языке» при проектировании, сборке и обслуживании сложных систем. В данной работе мы будем ориентироваться на стандарты ГОСТ 2.701 и 2.702 для классификации и правил выполнения схем.

Цель лабораторной работы: изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ, освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры и приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

Для достижения поставленной цели необходимо решить следующие задачи:

- изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ;
- освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры;
- приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

					490000.000	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Задание 1

Типы схем по ГОСТ 2.701

Структурная – основные функциональные части (Edge/Fog/Cloud) и связи между ними.

Функциональная – процессы/потоки данных и управления (события, задержки и т.д.).

Принципиальная – уровень элементов для электроники и логики (датчики, интерфейсы).

Соединений/подключения/расположения – топология портов, шин, кабелей, размещения узлов и т.д.

Диаграммы для распределённых систем IoT (UML/SysML ↔ ГОСТ)

– Deployment – узлы и сети: Edge-устройства, Fog-шлюз (Home_Gateway), Cloud-сервисы → структурная схема.

– Component – сервисы и драйверы (MQTT-брокер, Ingestor, Threshold/ML) → структурная/функциональная.

– Sequence/Activity – обмены и потоки (publish/subscribe, алерты) → функциональная.

– State – режимы устройств/узлов (online/offline, in_work) → функциональная (состояний).

– Package/Context – границы подсистем, ответственность и уровни.

Для описания распределенных систем IoT наиболее наглядными и практичными являются структурные и функциональные схемы. В отличие от детализированных принципиальных схем, они позволяют абстрагироваться от аппаратной реализации и сосредоточиться на ключевых компонентах, связях и потоках данных, что делает их понятными для более широкой аудитории, включая не-инженеров. В контексте программной инженерии этим целям лучше всего соответствуют диаграммы UML/SysML. В данной работе мы сосредоточимся на:

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

– Deployment – диаграмме (UML) как аналоге структурной схемы (ГОСТ): она показывает «из чего состоит система» — устройства, узлы и физические соединения.

– State – диаграмме (UML) как аналоге функциональной схемы состояний (ГОСТ): она показывает «как ведет себя компонент» — его реакцию на события и смену состояний.

Необходимые навыки:

Базовые навыки работы с компьютером, умение пользоваться веб-браузером. Опыт работы с графическими редакторами или программированием приветствуется, но не является обязательным.

1. Компоненты системы

Система состоит из пяти компонентов, распределенных по уровням вычислений:

Уровни архитектуры:

- Краевой уровень (Edge): Робот-датчик — сбор сырых данных
- Туманный уровень (Fog): Fog-обработчик — локальная обработка
- Облачный уровень (Cloud): Ноутбук-сервер — аналитика и хранение

Связующие компоненты:

- Робот-курьер — транспортировка данных
- Смартфон — буферизация и передача

На рисунке 1 представлена схема работы компонентов системы.

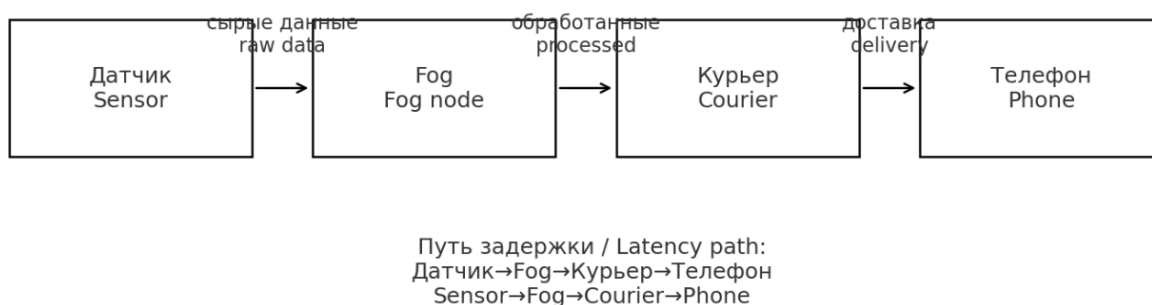


Рисунок 1 – схема работы компонентов системы

Следующая схема представляет собой логическую модель компонентов распределенной системы «Умный дом». Она показывает основные

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

функциональные блоки и направление потока данных между ними. Каждый компонент выполняет строго определенную роль в цепочке обработки информации: от сбора данных (робот-датчик) до их конечного анализа (ноутбук-сервер).

На рисунке 2 представлена схема архитектуры системы.

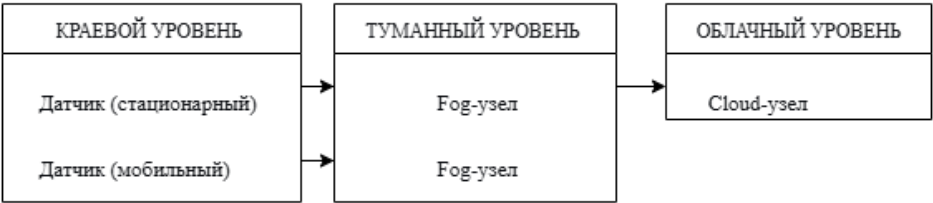


Рисунок 2 - Схема архитектуры системы

Схема на рисунке 3 является технической реализацией логической модели, показанной на Рисунке 2. Она детализирует как именно компоненты взаимодействуют на практике, показывает сетевые интерфейсы, протоколы передачи данных и физическую структуру системы. Здесь наглядно отображено разделение на сетевые сегменты (например, локальную сеть датчиков и облачный сервер).

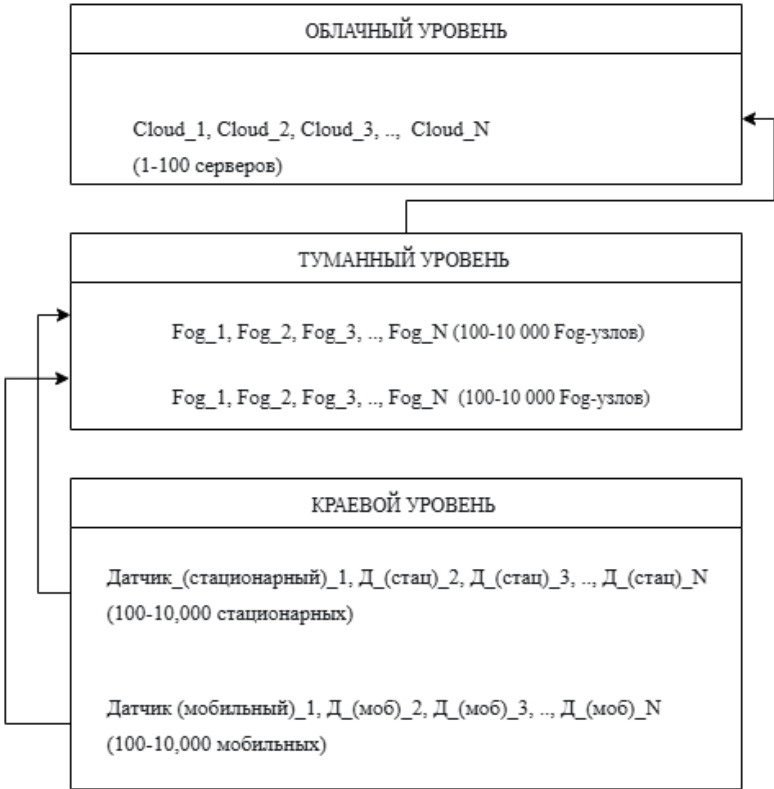


Рисунок 3 – Схема эталонной архитектуры системы

Взаимосвязь и взаимодействие между схемами

Обе схемы описывают одну и ту же систему, но на разных уровнях абстракции:

– Рисунок 2 отвечает на вопрос «ЧТО делает система?» — показывает основные роли и поток данных.

– Рисунок 3 отвечает на вопрос «КАК система это делает?» — показывает техническую реализацию и сетевые взаимодействия.

Краткое описание взаимодействия:

1. Робот-датчик собирает данные и передает их Fog-обработчику по локальному сетевому протоколу (например, Wi-Fi или Bluetooth)

2. Fog-обработчик выполняет первичную обработку и передает результат роботу-курьеру через локальную сеть

3. Робот-курьер доставляет данные смартфону через интернет (мобильная сеть 4G/5G)

4. Смартфон буферизует данные и передает их ноутбуку-серверу через облачное API или прямое сетевое соединение.

					490000.000	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Задание 2

GOOGLE COLAB

Google Colab – бесплатная облачная среда разработки, которая позволяет писать и выполнять код на Python прямо в браузере.

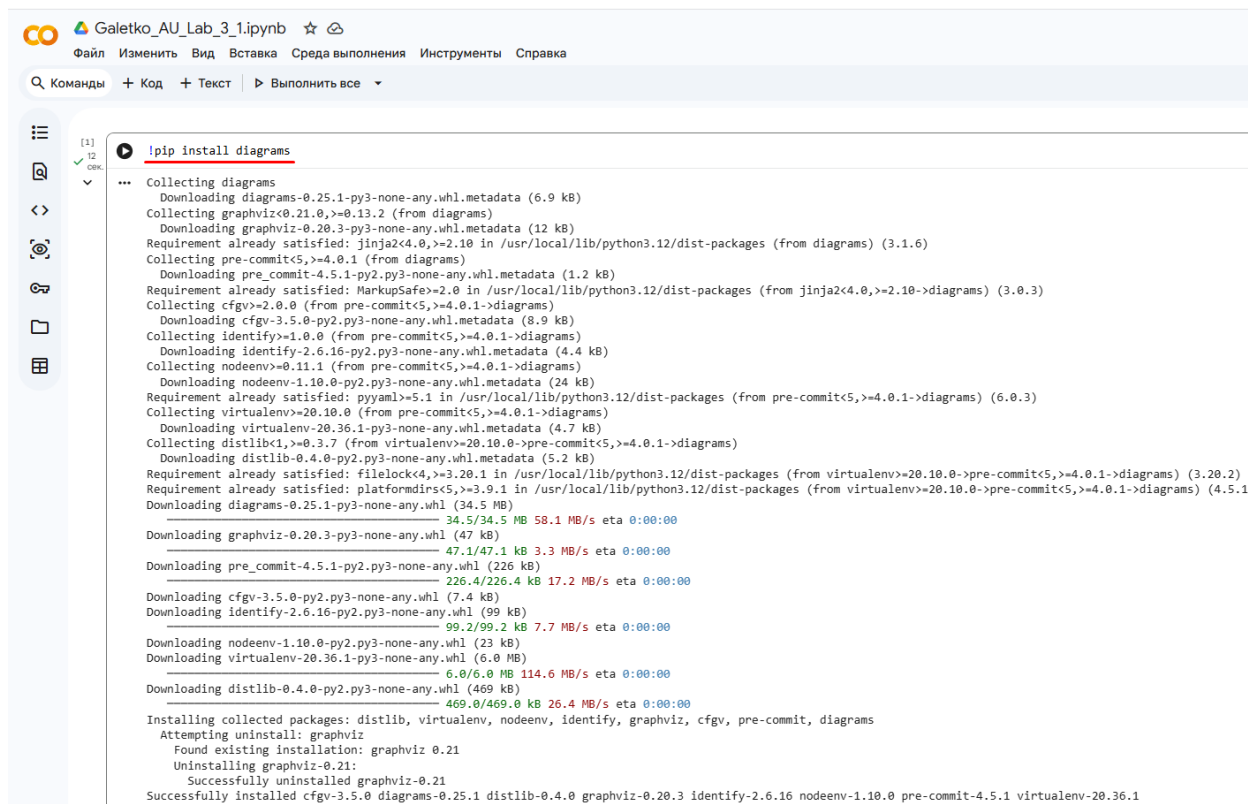
Diagrams – библиотека для Python, которая позволяет программно создавать диаграммы архитектуры облачных систем.

2.1 Пример работы в Google Colab

1. Открыли новый блокнот на <https://colab.research.google.com>.
2. В новой ячейке установили и импортировали библиотеку для построения графиков:

```
>>> !pip install diagrams
```

На рисунке 4 представлена реализация команды для установки и импорта библиотеки.



```
Galetko_AU_Lab_3_1.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка

Команды + Код + Текст | ▶ Выполнить все

[1] 12 OK
!pip install diagrams

... Collecting diagrams
  Downloading diagrams-0.25.1-py3-none-any.whl.metadata (6.9 kB)
Collecting graphviz<0.21.0,>=0.13.2 (from diagrams)
  Downloading graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: Jinja2<4.0,>=2.10 in /usr/local/lib/python3.12/dist-packages (from diagrams) (3.1.6)
Collecting pre-commit<5,>=4.0.1 (from diagrams)
  Downloading pre_commit-4.5.1-py2.py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: MarkupSafe<2.0 in /usr/local/lib/python3.12/dist-packages (from Jinja2<4.0,>=2.10->diagrams) (3.0.3)
Collecting cfgv<2.0.0 (from pre-commit<5,>=4.0.1->diagrams)
  Downloading cfgv-3.5.0-py2.py3-none-any.whl.metadata (8.9 kB)
Collecting identify<1.0.0 (from pre-commit<5,>=4.0.1->diagrams)
  Downloading identify-2.6.16-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting nodeenv<0.11.1 (from pre-commit<5,>=4.0.1->diagrams)
  Downloading nodeenv-1.10.0-py2.py3-none-any.whl.metadata (24 kB)
Requirement already satisfied: pyyaml<5.1 in /usr/local/lib/python3.12/dist-packages (from pre-commit<5,>=4.0.1->diagrams) (6.0.3)
Collecting virtualenv<20.36.1-py3-none-any.whl.metadata (4.7 kB)
Collecting distlib<1,>=0.3.7 (from virtualenv<20.36.1-py3-none-any.whl->pre-commit<5,>=4.0.1->diagrams)
  Downloading distlib-0.4.0-py2.py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: filelock<4,>=3.20.1 in /usr/local/lib/python3.12/dist-packages (from virtualenv<20.36.1-py3-none-any.whl->pre-commit<5,>=4.0.1->diagrams) (3.20.2)
Requirement already satisfied: platformdirs<5,>=3.9.1 in /usr/local/lib/python3.12/dist-packages (from virtualenv<20.36.1-py3-none-any.whl->pre-commit<5,>=4.0.1->diagrams) (4.5.1)
Downloading diagrams-0.25.1-py3-none-any.whl (34.5 MB)
34.5/34.5 MB 58.1 MB/s eta 0:00:00
Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
47.1/47.1 kB 3.3 MB/s eta 0:00:00
Downloading pre_commit-4.5.1-py2.py3-none-any.whl (226 kB)
226.4/226.4 kB 17.2 MB/s eta 0:00:00
Downloading cfgv-3.5.0-py2.py3-none-any.whl (7.4 kB)
99.2/99.2 kB 7.7 MB/s eta 0:00:00
Downloading identify-2.6.16-py2.py3-none-any.whl (23 kB)
6.0/6.0 MB 114.6 MB/s eta 0:00:00
Downloading nodeenv-1.10.0-py2.py3-none-any.whl (6.0 MB)
469.0/469.0 kB 26.4 MB/s eta 0:00:00
Installing collected packages: distlib, virtualenv, nodeenv, identify, graphviz, cfgv, pre-commit, diagrams
  Attempting uninstall: graphviz
    Found existing installation: graphviz 0.21
    Uninstalling graphviz-0.21:
      Successfully uninstalled graphviz-0.21
  Successfully installed cfgv-3.5.0 diagrams-0.25.1 distlib-0.4.0 graphviz-0.20.3 identify-2.6.16 nodeenv-1.10.0 pre-commit-4.5.1 virtualenv-20.36.1
```

Рисунок 4 – Реализация команды для установки и импорта библиотеки

3. Создали новую ячейку и скопировали в нее следующий код для визуализации архитектуры:

```
from diagrams import Diagram
from diagrams.aws.compute import EC2
from diagrams.aws.network import InternetGateway
from diagrams.onprem.client import User
from diagrams.onprem.compute import Server
from diagrams.aws.iot import IotSensor

with Diagram("Архитектура IoT системы 'Умный дом'", show=False,
filename="iot_architecture"):
    # Определяем ноды (компоненты) системы с более подходящими
    иконками
    sensor = IotSensor("Робот-датчик")
    fog = EC2("Fog-обработчик")
    courier = InternetGateway("Робот-курьер")
    smartphone = User("Смартфон\n(Буфер) ")
    server = Server("Ноутбук-сервер")

    # Строим конвейер обработки данных
    sensor >> fog >> courier >> smartphone >> server

print("Схема сохранена как 'iot_architecture.png'")
print("Файл будет доступен в левой панели управления файлами
Colab")
```

На рисунке 5 представлена реализация кода.

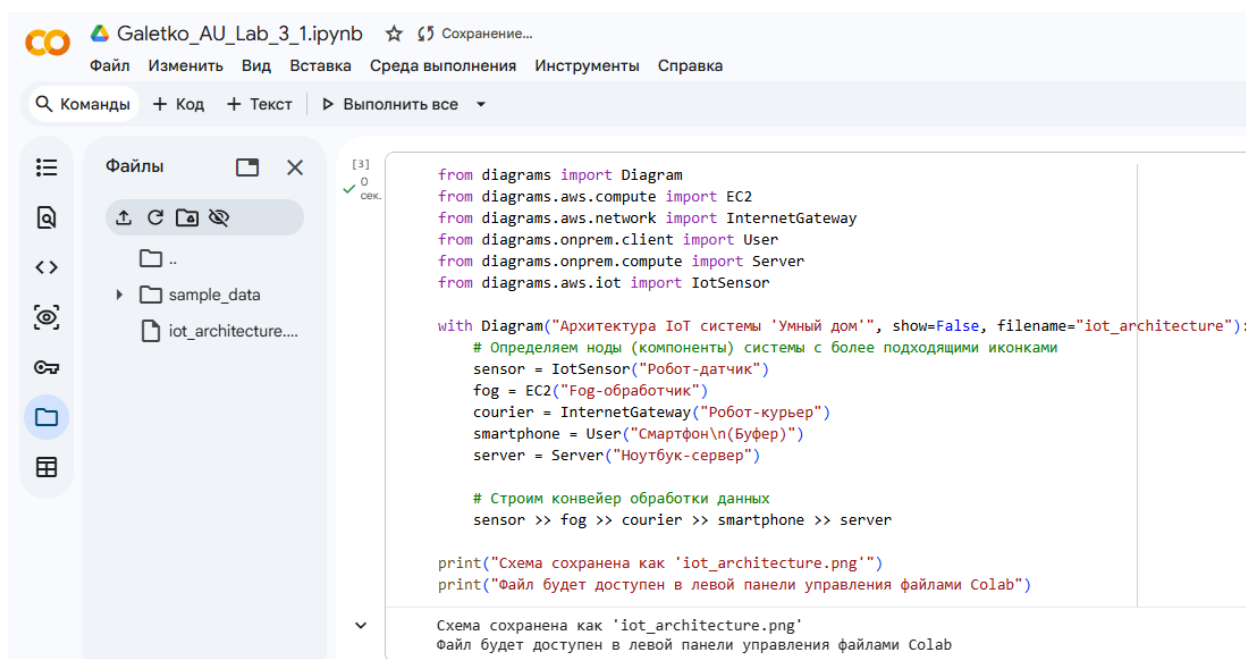


Рисунок 5 – Реализация кода

Запустили ячейку. После выполнения код создал файл iot_architecture.png в левой панели управления файлами. Открыли его, чтобы увидеть схему нашей системы.

На рисунке 6 представлена полученная диаграмма.

iot_architecture.png



Рисунок 6 – Полученная диаграмма

2.2 Задание для построения схемы через Google Colab

Используя пример из раздела 2.1, построили Deployment-диаграмму для системы «Умный дом», отображающую физические узлы и связи между ними.

Изменили код следующим образом:

- Добавили второго «Робота-датчика»
- Добавили второго «Fog-обработчика»
- Изменили направление данных: первый датчик → первый Fog, второй датчик → второй Fog
- Оба Fog-обработчика передают данные на «Ноутбук-сервер»
- Изменили иконки для объектов

Измененный код:

```
from diagrams import Diagram, Cluster
from diagrams.aws.iot import IotCamera, IotButton, IotMqtt
from diagrams.aws.compute import Lambda, Fargate
from diagrams.aws.general import MobileClient, TraditionalServer

with Diagram("Модифицированная архитектура IoT 'Умный дом' (все иконки изменены)", show=False, filename="modified_iot_architecture_all_icons", direction="LR"):
    with Cluster("Edge Layer"):
        sensor1 = IotCamera("Робот-датчик 1")
        sensor2 = IotButton("Робот-датчик 2")

    with Cluster("Fog Layer"):
        fog1 = Lambda("Fog-обработчик 1")
        fog2 = Fargate("Fog-обработчик 2")
```

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

```

courier = IotMqtt("Робот-курьер")
smartphone = MobileClient("Смартфон\n(Буфер)")
server = TraditionalServer("Ноутбук-сервер")

```

```

sensor1 >> fog1
sensor2 >> fog2
fog1 >> courier
fog2 >> courier
courier >> smartphone >> server

```

```

print("Модифицированная схема сохранена как
'modified_iot_architecture_all_icons.png')
print("Файл будет доступен в левой панели управления файлами
Colab")

```

На рисунке 7 представлена реализация измененного кода. На рисунке 8 представлена полученная измененная диаграмма.

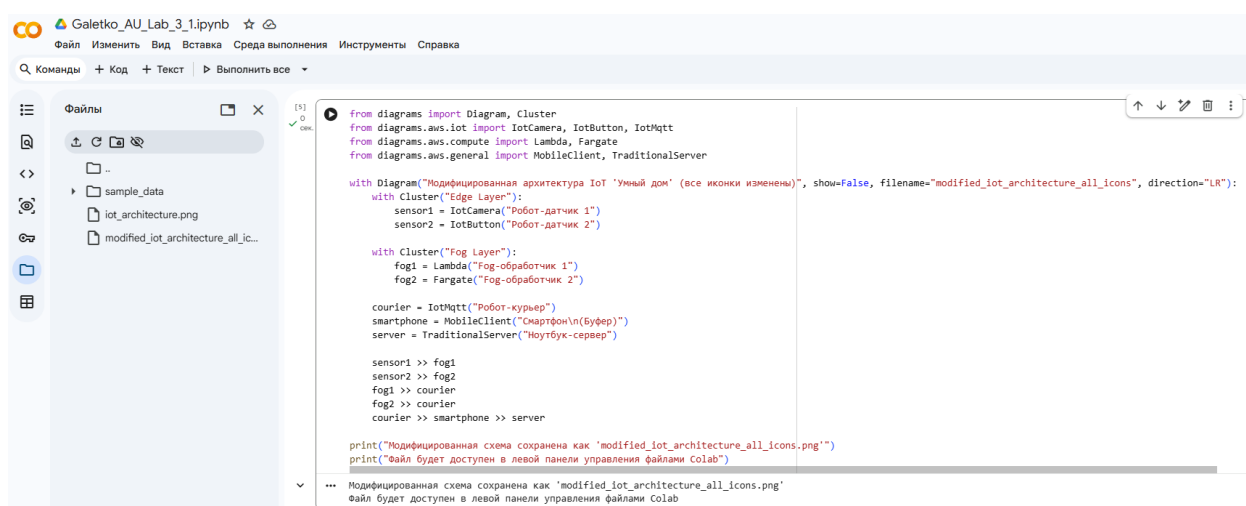


Рисунок 7 – Реализация измененного кода

iot_architecture.png modified_iot_architecture_all_icons.png ×

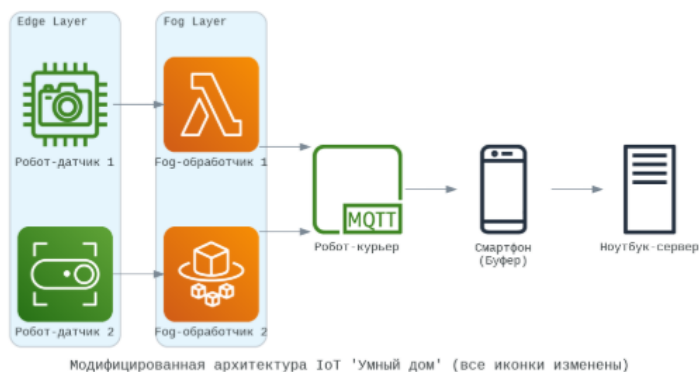


Рисунок 8 – Полученная измененная диаграмма

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

Задание 3

PLANTUML EDITOR

PlantUML Editor — это инструмент для создания, редактирования и визуализации диаграмм с помощью простого текстового языка.

3.1 Работа в PlantUML Editor

1. Открыли онлайн-редактор PlantUML:
<https://www.plantuml.com/plantuml/>

2. Скопировали и вставили следующий код для создания Deployment-диаграммы в панель редактора.

Листинг:

```
@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Робот-датчик 1]
    [Робот-датчик 2]
    [Робот-датчик N]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Фог-обработчик 1)
    (Фог-обработчик 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Ноутбук-сервер)
}

[Робот-датчик 1] --> (Фог-обработчик 1) : Wi-Fi
[Робот-датчик 2] --> (Фог-обработчик 1) : BLE
[Робот-датчик N] --> (Фог-обработчик 2) : LoRaWAN
(Фог-обработчик 1) --> (Ноутбук-сервер) : HTTPS
(Фог-обработчик 2) --> (Ноутбук-сервер) : MQTT
@enduml
```

На рисунке 9 представлена структурная схема, созданная с помощью PlantUML.

					490000.000	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

```

1 @startuml
2 skinparam shadowing false
3 skinparam defaultFontName Arial
4 skinparam nodesep 10
5 skinparam ranksep 20
6
7 title Deployment-диаграмма IoT системы "Умный дом"
8
9 node "Краевой уровень (Edge)" as edge_layer {
10     [Робот-датчик 1]
11     [Робот-датчик 2]
12     [Робот-датчик N]
13 }
14
15 node "Туманный уровень (Fog)" as fog_layer {
16     (Fog-обработчик 1)
17     (Fog-обработчик 2)
18 }
19
20 cloud "Облачный уровень (Cloud)" as cloud_layer {
21     (Ноутбук-сервер)
22 }
23
24 [Робот-датчик 1] --> (Fog-обработчик 1) : Wi-Fi
25 [Робот-датчик 2] --> (Fog-обработчик 1) : BLE
26 [Робот-датчик N] --> (Fog-обработчик 2) : LoRaWAN
27 (Fog-обработчик 1) --> (Ноутбук-сервер) : HTTPS
28 (Fog-обработчик 2) --> (Ноутбук-сервер) : MQTT
29 @enduml

```

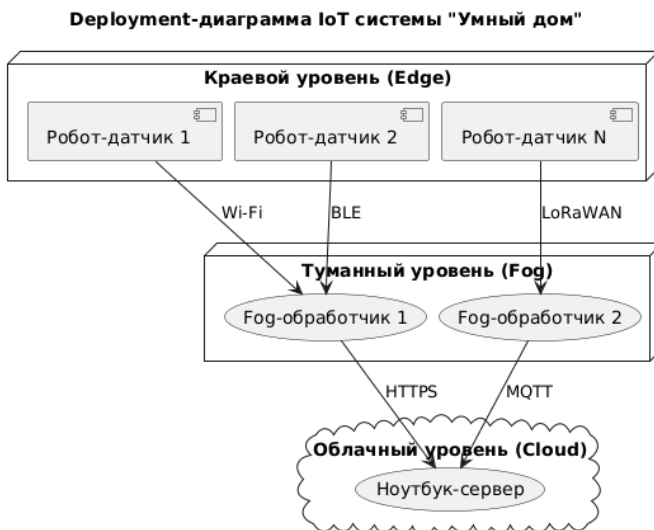


Рисунок 9 – Структурная схема, созданная с помощью PlantUML

3. Диаграмма появилась сама, мы увидели структурную схему, отображающую узлы системы и связи между ними.

4. Создали State-диаграмму. Очистили редактор и вставили следующий код:

```

@startuml
skinparam defaultFontName Arial

title State-диаграмма Fog-обработчика

[*] --> Ожидание

Ожидание --> Сбор_данных : получен запрос
Сбор_данных --> Обработка : данные считаны
Обработка --> Передача : результат готов
Передача --> Ожидание : отправлено подтверждение

Ожидание --> Перегрузка : очередь переполнена
Перегрузка --> [*] : аварийное отключение
@enduml

```

На рисунке 10 представлена State-диаграмма.

```

1 @startuml
2 skinparam defaultFontName Arial
3
4 title State-диаграмма Fog-обработчика
5
6 [*] --> Ожидание
7
8 Ожидание --> Сбор_данных : получен запрос
9 Сбор_данных --> Обработка : данные считаны
10 Обработка --> Передача : результат готов
11 Передача --> Ожидание : отправлено подтверждение
12
13 Ожидание --> Перегрузка : очередь переполнена
14 Перегрузка --> [*] : аварийное отключение
15 @enduml

```

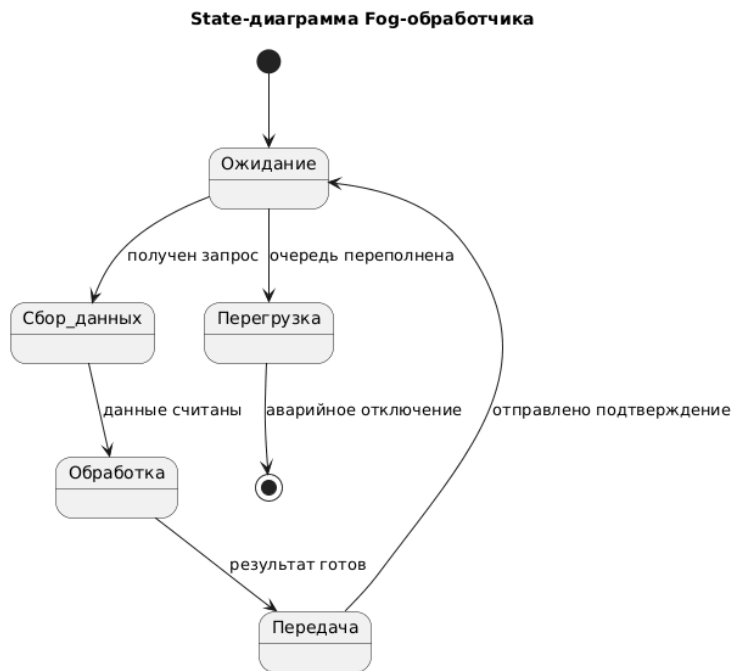


Рисунок 10 – State-диаграмма

3.2 Задание для построения схемы через PlantUML Editor

Используя примеры выше, создали в PlantUML Editor:

1. Deployment-диаграмму, отображающую не менее 3 краевых устройств, 2 Fog-узлов и 1 облачного сервера. Изменили названия компонентов и типы соединений на свои.

Листинг:

```

@startuml
skinparam shadowing false
skinparam defaultFontName Arial
skinparam nodesep 10
skinparam ranksep 20

title Deployment-диаграмма IoT системы "Умный дом"

node "Краевой уровень (Edge)" as edge_layer {
    [Датчик температуры 1]
    [Датчик влажности 2]
    [Датчик движения 3]
}

node "Туманный уровень (Fog)" as fog_layer {
    (Fog-шлюз 1)
    (Fog-шлюз 2)
}

cloud "Облачный уровень (Cloud)" as cloud_layer {
    (Сервер анализа)
}

```

```

[Датчик температуры 1] --> (Fog-шлюз 1) : ZigBee
[Датчик влажности 2] --> (Fog-шлюз 1) : Bluetooth
[Датчик движения 3] --> (Fog-шлюз 2) : Wi-Fi
(Fog-шлюз 1) --> (Сервер анализа) : Ethernet
(Fog-шлюз 2) --> (Сервер анализа) : MQTT
@enduml

```

На рисунке 11 представлена Deployment-диаграмма.

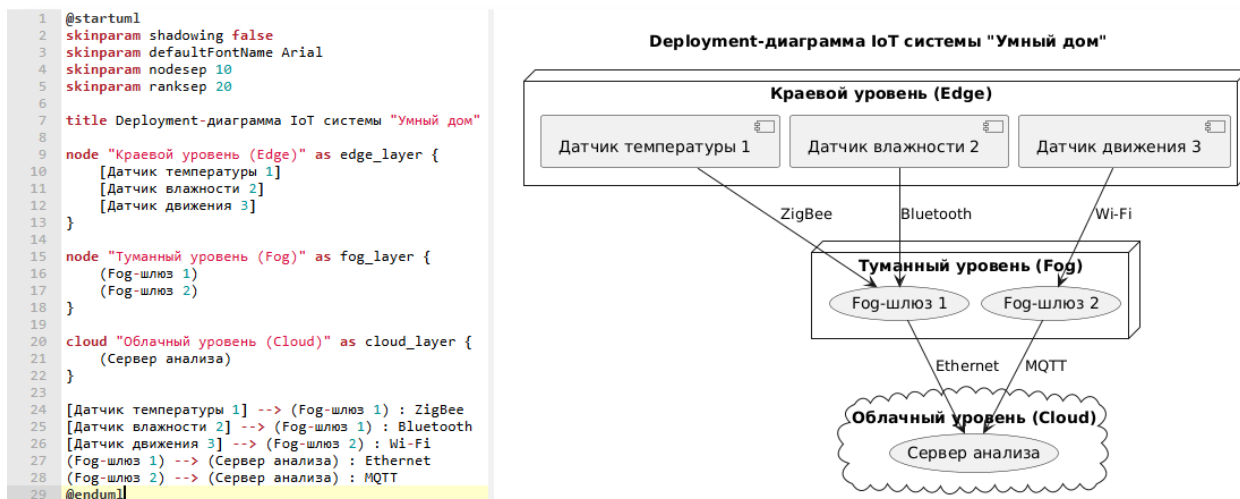


Рисунок 11 – Deployment-диаграмма

2. Создали State-диаграмму для «Робота-курьера». Добавили состояния. Сохранили обе диаграммы как PNG-файлы для отчета.

Листинг:

```

@startuml
skinparam defaultFontName Arial

title State-диаграмма Робота-курьера

[*] --> Зарядка
Зарядка --> Ожидание_задания : Заряжен
Ожидание_задания --> Навигация_к_смартфону : Получено задание
Навигация_к_смартфону --> Передача_данных : Прибыл
Передача_данных --> [*] : Данные переданы

Передача_данных --> Перегрузка : Буфер полон
Перегрузка --> Ожидание_задания : Проблема решена
@enduml

```

На рисунке 12 представлена State -диаграмма «Робота-курьера».

```

1 @startuml
2 skinparam defaultFontName Arial
3
4 title State-диаграмма Робота-курьера
5
6 [*] --> Зарядка
7 Зарядка --> Ожидание_задания : Заряжен
8 Ожидание_задания --> Навигация_к_смартфону : Получено задание
9 Навигация_к_смартфону --> Передача_данных : Прибыл
10 Передача_данных --> [*] : Данные переданы
11
12 Передача_данных --> Перегрузка : Буфер полон
13 Перегрузка --> Ожидание_задания : Проблема решена
14 @enduml

```

State-диаграмма Робота-курьера

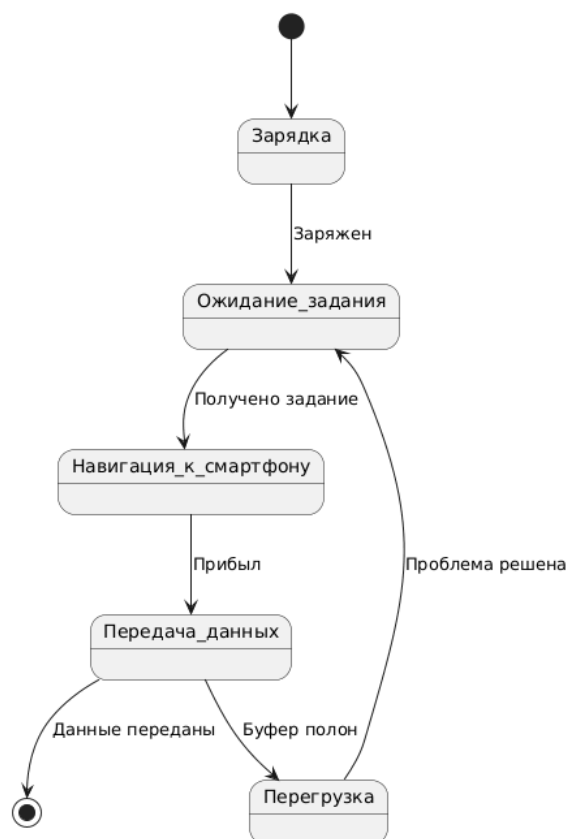


Рисунок 12 – State -диаграмма «Робота-курьера»

Задание 4

DRAW.IO

Draw.io (с 2020 года официально называется Diagrams.net) — бесплатный онлайн-сервис для создания диаграмм, блок-схем и визуальных схем любой сложности.

4.1 Работа в Draw.io

1. Перешли на сайт <https://app.diagrams.net/>.
2. Создали новую диаграмму ("File" -> "New").
3. В левой панели выбрали раздел "Network". Перетащили на холм иконки, соответствующие компонентам IoT-системы:

- Датчик (Sensor) для робота-датчика.
- Компьютер (Computer) или Сервер (Server) для Fog-обработчика и ноутбука-сервера.
- Мобильное устройство (Mobile) для смартфона.
- Из раздела "General" добавили стрелки и блоки для соединений и подписей.

4. Соединили компоненты линиями, имитируя поток данных от датчика к серверу. Подписали каждый компонент и линию связи (протокол).

На рисунке 13 представлена созданная диаграмма.

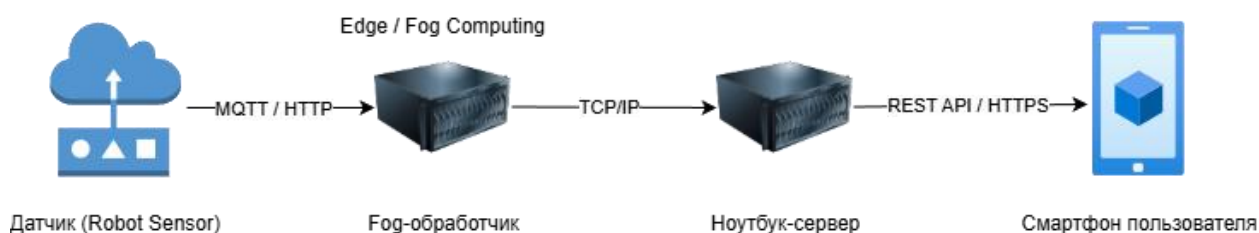


Рисунок 13 – созданная диаграмма

4.2 Задание для построения схемы через Draw.io

Повторили в Draw.io логическую модель системы (аналог Рисунка 2 из данного документа), используя базовые фигуры (прямоугольники, стрелки) из раздела "General". Схема наглядно показывает пять основных

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		17

компонентов системы и направление потока данных между ними. Сохранили диаграмму в формате PNG.

На рисунке 14 представлена логическая модель системы.

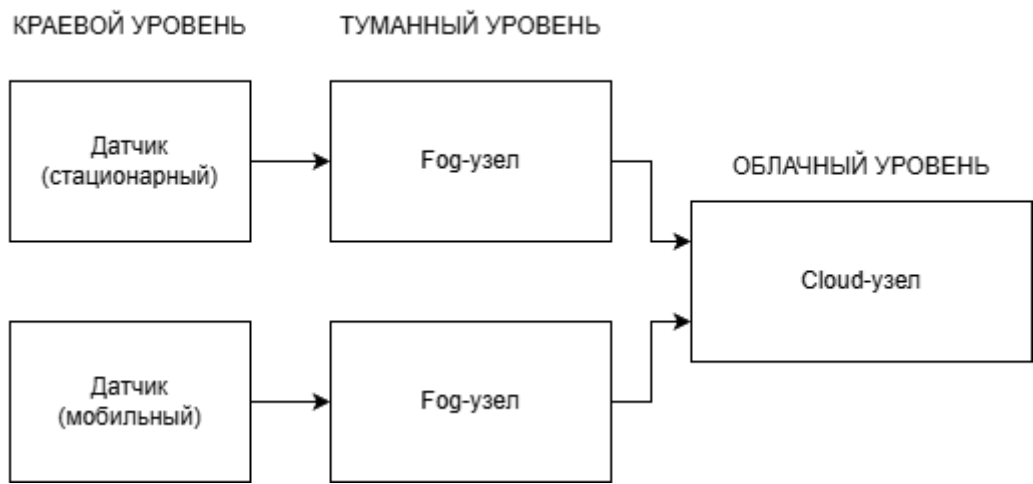


Рисунок 14 – Логическая модель системы

Задание 5

Индивидуальное задание.

Используя один из изученных инструментов – PlantUML, построила две диаграммы для распределенной IoT-системы, соответствующей моему варианту №9.

За основу взяла трехуровневую архитектуру «Край-Туман-Облако».

Вариант	Краевые устройства (Edge)	Fog-узлы	Облачные серверы (Cloud)	Характеристика системы
9	6	3	1	Умеренная нагрузка

1. Deployment-диаграмма (структурная схема), отображающая физические узлы системы (краевые устройства, Fog-узлы, облачные серверы) и связи между ними.

Листинг:

```
@startuml
skinparam backgroundColor #F8F8F8
skinparam shadowing false
skinparam nodesep 40
skinparam ranksep 60
skinparam defaultFontName Arial
skinparam stereotypeCBackgroundColor #ADD1B2
skinparam stereotypeIBBackgroundColor #ADD1B2

title Вариант 9: Deployment-диаграмма распределённой IoT-системы\n(6 Edge, 3 Fog, 1 Cloud, умеренная нагрузка)

package "Edge (Краевой уровень) - 6 устройств" #LightBlue {
    [Датчик 1 (камера)] as e1
    [Датчик 2 (кнопка)] as e2
    [Датчик 3 (термостат)] as e3
    [Датчик 4 (аналитика)] as e4
    [Датчик 5 (промышленный)] as e5
    [Датчик 6 (события)] as e6
}

package "Fog (Туманный уровень) - 3 узла" #LightGreen {
    [Fog-узел 1] as f1
    [Fog-узел 2] as f2
    [Fog-узел 3] as f3
}
```

					490000.000	Лист
Изм.	Лист	№ докум.	Подпись	Дата		19

```

package "Cloud (Облачный уровень)" #LightGray {
    [Ноутбук-сервер (аналитика и хранение)] as c
}

[Робот-курьер\n(транспортировка данных)] as courier
[Смартфон\n(буферизация)] as phone

e1 --> f1 : ZigBee
e2 --> f1 : Wi-Fi
e3 --> f2 : Bluetooth
e4 --> f2 : ZigBee
e5 --> f3 : Wi-Fi
e6 --> f3 : Ethernet

f1 --> courier : Ethernet
f2 --> courier : Ethernet
f3 --> courier : Ethernet

courier --> phone : Bluetooth
phone --> c : Интернет

note bottom of f1
    Локальная обработка
    и фильтрация данных
end note

note bottom of courier
    Транспортный робот
    (передача между уровнями)
end note

note bottom of c
    Глубокая аналитика
    и долговременное хранение
end note

@enduml

```

На рисунке 15 представлена реализация кода. На рисунке 16 изображена Deployment-диаграмма.

					490000.000	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

```

1 getstartuml
2 skinparam backgroundColor #F8F8F8
3 skinparam shadowing false
4 skinparam nodesep 60
5 skinparam ranksep 60
6 skinparam defaultFontName Arial
7 skinparam stereotypeBackgroundColor #ADD8E6
8 skinparam stereotypeBackgroundColor #ADD8E6
9
10 title Вариант 9: Deployment-диаграмма распределённой IoT-системы (6 Edge, 3 Fog, 1 Cloud, умеренная нагрузка)
11
12 package "Edge (Краевой уровень) - 6 устройств" #lightblue {
13     [Датчик 1 (камера)] as e1
14     [Датчик 2 (кнопка)] as e2
15     [Датчик 3 (термостат)] as e3
16     [Датчик 4 (аналитика)] as e4
17     [Датчик 5 (промышленный)] as e5
18     [Датчик 6 (события)] as e6
19 }
20
21 package "Fog (Туманный уровень) - 3 узла" #lightgreen {
22     [Fog-узел 1] as f1
23     [Fog-узел 2] as f2
24     [Fog-узел 3] as f3
25 }
26
27 package "Cloud (Облачный уровень)" #lightgray {
28     [Ноутбук-сервер (аналитика и хранение)] as c
29 }
30
31 [Робот-курьер (транспортировка данных)] as courier
32 [Смартфон (буферизация)] as phone
33
34 e1 --> f1 : ZigBee
35 e2 --> f2 : Wi-Fi
36 e3 --> f2 : Bluetooth
37 e4 --> f2 : ZigBee
38 e5 --> f3 : Wi-Fi
39 e6 --> f3 : Ethernet
40
41 f1 --> courier : Ethernet
42 f2 --> courier : Ethernet
43 f3 --> courier : Ethernet
44
45 courier --> phone : Bluetooth
46 phone --> c : Интернет
47
48 note bottom of f1
49     Локальная обработка
50     и фильтрация данных
51 end note
52
53 note bottom of courier
54     Транспортный робот
55     (передача между уровнями)
56 end note
57
58 note bottom of c
59     Глубокая аналитика
60     и долговременное хранение
61 end note
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

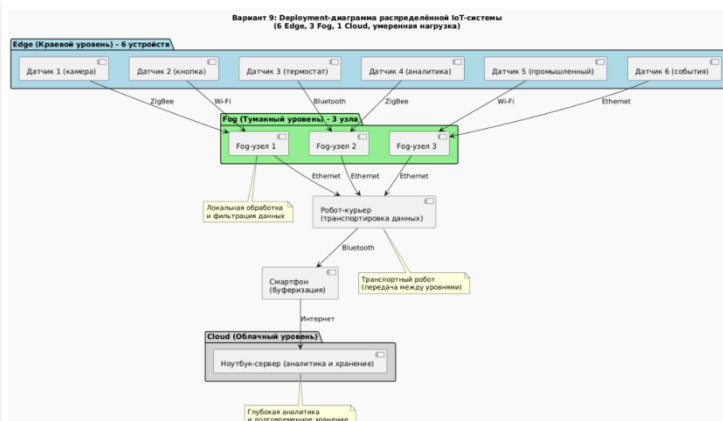


Рисунок 15 – Реализация кода

Вариант 9: Deployment-диаграмма распределённой IoT-системы (6 Edge, 3 Fog, 1 Cloud, умеренная нагрузка)

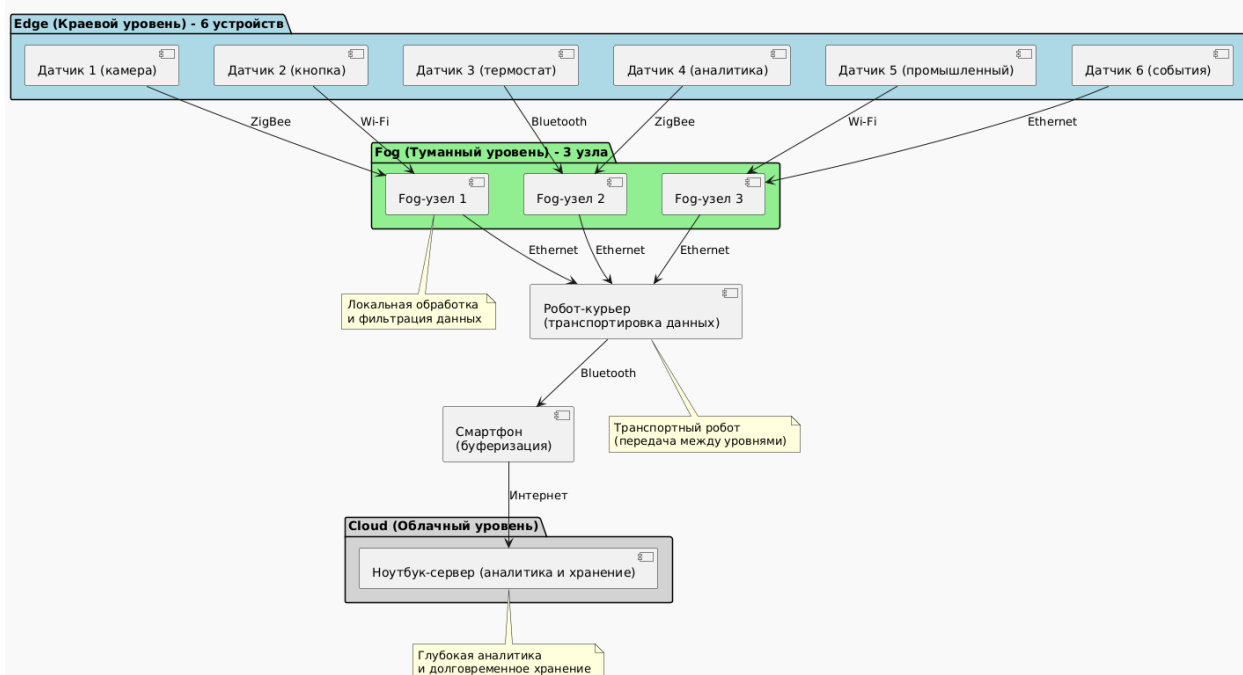


Рисунок 16 – Deployment-диаграмма

2. State-диаграмму (функциональная схема состояний), описывающую изменение состояния одного из Fog-узлов (например: Режим ожидания -> Получение задачи -> Обработка -> Передача результата -> Перегрузка/Ожидание).

Листинг:

```

@startuml
skinparam backgroundColor #F8F8F8
skinparam state {
    BackgroundColor<<idle>> #E0FFE0
    BorderColor<<idle>> #006400
    BackgroundColor<<active>> #E0F0FF

```


На рисунке 17 представлена реализация кода. На рисунке 18 изображена State-диаграмма.

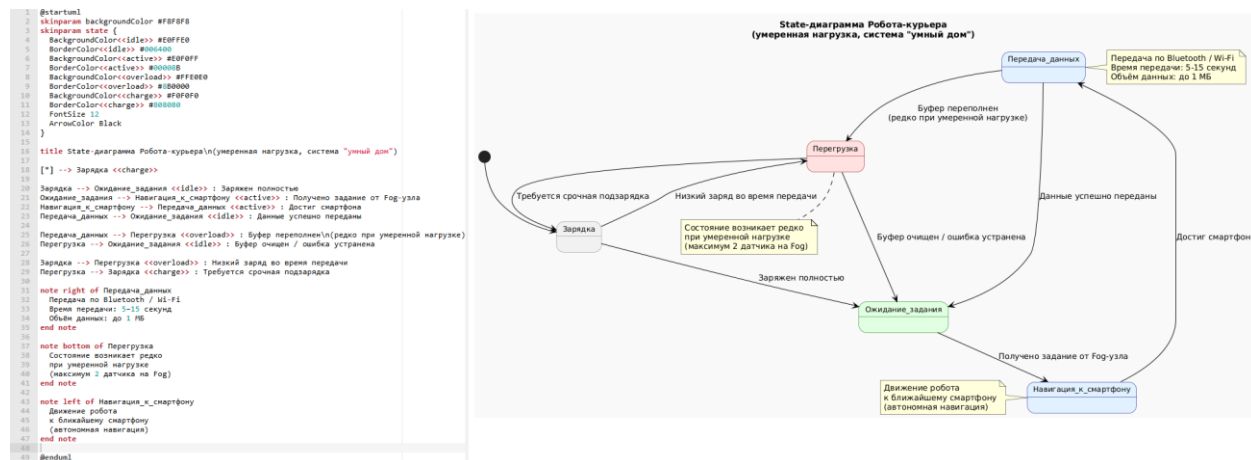


Рисунок 17 – Реализация кода

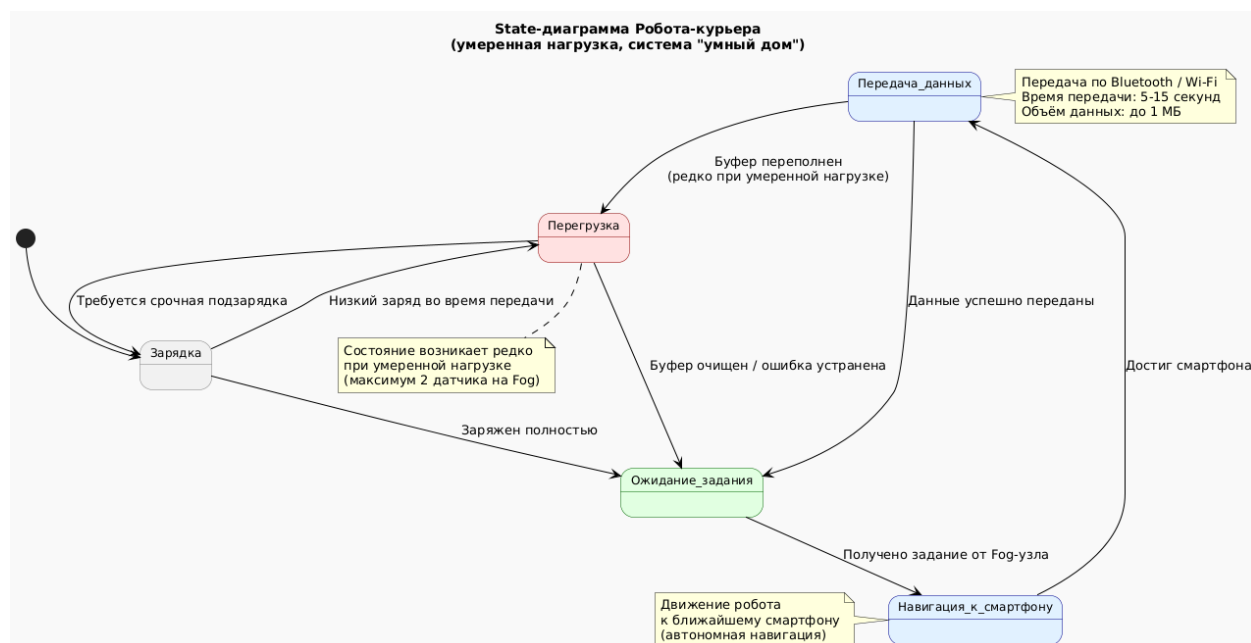


Рисунок 18 - State-диаграмма

За основу взяла трехуровневую архитектуру «Край-Туман-Облако».

Заключение

В ходе выполнения лабораторной работы были достигнуты все поставленные цели и задачи. Выполненная работа позволила изучить принципы построения схем распределительных систем IoT в соответствии со стандартами ГОСТ, освоить инструменты Google Colab, PlantUML Editor и draw.io для визуализации архитектуры и приобрести навык создания структурных и функциональных диаграмм (Deployment и State) для IoT-системы.

В процессе работы были разработаны:

- Deployment-диаграммы, отражающие физическую структуру системы (узлы Edge, Fog, Cloud и связи между ними).
- State-диаграммы, моделирующие поведение компонентов (например, работа-курьера и Fog-обработчика) в ответ на события и изменения состояний.

Особое внимание уделено соответствию диаграмм стандартам ГОСТ 2.701 и 2.702, а также их адаптации к современным нотациям UML/SysML. В индивидуальном задании была реализована трехуровневая архитектура «Край–Туман–Облако» для системы с умеренной нагрузкой, включающая 6 краевых устройств, 3 Fog-узла и 1 облачный сервер.

Таким образом, в ходе выполнения работы получены практические компетенции, необходимые для проектирования и документирования распределённых IoT-систем с использованием современных инструментов и стандартов.

					490000.000	Лист
						24
Изм.	Лист	№ докум.	Подпись	Дата		