

Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №3
по дисциплине
«Облачные и туманные вычисления»

Выполнил студент группы Р3415
Барсуков Максим Андреевич

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2025 г.

Содержание

Задание.....	2
Программная реализация.....	3
Сенсор.....	3
Облачная панель.....	4
Инфраструктура.....	5
Конфигурация Mosquitto.....	5
Docker Compose.....	5
Настройка Node-RED.....	7
Общий вид потока.....	7
Настройка узлов.....	8
Заключение.....	9

Задание

Цель: Смоделировать среду Edge/Fog, где данные обрабатываются локально (на "туманном" уровне) перед отправкой в Облако. Это критически важно для вашего курса, так как демонстрирует работу шлюзов (Gateways).

Сценарий: На заводе установлены датчики температуры. Нужен “Туманный узел” (Шлюз), который фильтрует нормальные показатели, но немедленно оповещает Облако, если температура превышает норму.

Архитектура (все компоненты запускаются в Docker):

- **Edge Device (Датчик):** Простой скрипт (контейнер Python), генерирующий случайную температуру и публикующий её в топик MQTT каждую секунду.
- **Fog Node (Шлюз):** Контейнер с Node-RED, подписывающийся на данные датчика и реализующий фильтрацию. *Логика:* Если Темп < 80, ничего не делать (или писать в локальный лог). Если Темп > 80, переслать в Облако.
- **Cloud (Центр):** Контейнер **Mosquitto** (MQTT брокер) или простой дашборд, принимающий только критические алерты.

Ход работы:

1. Создать специализированную сеть в Docker (factory-net).
2. Открыть визуальный редактор Node-RED (localhost:1880) и собрать логику: вход MQTT → Switch Node (если > 80) → выход MQTT (топик Облака).
3. Запустить контейнер “Датчика”.
4. Наблюдать, что контейнер “Облако” получает лишь критический трафик, демонстрируя экономию пропускной способности.

Ключевые знания: MQTT, фильтрация на краю (Edge Filtering), оптимизация трафика, логика работы шлюзов.

Программная реализация

Сенсор

Скрипт эмулирует работу датчика, отправляя случайные значения температуры в формате JSON в топик `factory/temp` локального брокера.

```
import time
import random
import json
import paho.mqtt.client as mqtt
import os

BROKER = os.getenv("MQTT_BROKER", "edge-broker")
PORT = int(os.getenv("MQTT_PORT", 1883))
TOPIC = os.getenv("MQTT_TOPIC", "factory/temp")
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)

def connect():
    while True:
        try:
            client.connect(BROKER, PORT, 60)
            print(f"Connected to {BROKER}:{PORT}")
            return
        except Exception as e:
            print(f"Connection failed: {e}. Retrying in 5s...")
            time.sleep(5)

connect()

while True:
    temp = random.randint(20, 100)
    payload = json.dumps({"temp": temp, "ts": time.time()})
    client.publish(TOPIC, payload)
    print(f"Sent: {payload} to {TOPIC}")
    time.sleep(1)
```

Облачная панель

Скрипт подписывается на топик cloud/alerts облачного брокера и выводит полученные сообщения (алерты) в консоль.

```
import paho.mqtt.client as mqtt
import os
import json

BROKER = os.getenv("MQTT_BROKER", "cloud-broker")
PORT = int(os.getenv("MQTT_PORT", 1883))
TOPIC = os.getenv("MQTT_TOPIC", "cloud/alerts")

def on_connect(client, userdata, flags, reason_code, properties):
    print(f"Connected to Cloud Broker with result code {reason_code}")
    client.subscribe(TOPIC)

def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())
        print(f"ALERT RECEIVED: Temp={data.get('temp')}°C at {data.get('ts')}")
    except:
        print(f"Received raw message: {msg.payload.decode()}")

client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.on_connect = on_connect
client.on_message = on_message

print(f"Connecting to {BROKER}...")
client.connect(BROKER, PORT, 60)
client.loop_forever()
```

Инфраструктура

Конфигурация Mosquitto

Для работы брокеров используется конфигурационный файл, разрешающий анонимный доступ и прослушивание порта 1883.

```
listener 1883
allow_anonymous true
```

Docker Compose

Файл `docker-compose.yml` описывает 5 сервисов:

1. sensor: Генератор данных.
2. edge-broker: Локальный MQTT брокер.
3. fog-node: Node-RED для обработки.
4. cloud-broker: Облачный MQTT брокер.
5. cloud-dashboard: Панель мониторинга.

Используются 3 сети для изоляции уровней: `sensor-net`, `fog-net`, `cloud-net`.

```
services:
  edge-broker:
    image: eclipse-mosquitto:2
    container_name: edge-broker
    ports:
      - "1883:1883"
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
    networks:
      - factory-net

  sensor:
    build: ./sensor
    container_name: sensor
    environment:
      - MQTT_BROKER=edge-broker
      - MQTT_TOPIC=factory/temp
    depends_on:
      - edge-broker
    networks:
      - factory-net

  fog-node:
    image: nodered/node-red
    container_name: fog-node
    ports:
      - "1880:1880"
    environment:
      - TZ=Europe/Moscow
    volumes:
      - node-red-data:/data
```

```
networks:
  - factory-net

cloud-broker:
  image: eclipse-mosquitto:2
  container_name: cloud-broker
  ports:
    - "1884:1883"
  volumes:
    - ./mosquitto.conf:/mosquitto/config/mosquitto.conf
  networks:
    - factory-net

cloud-dashboard:
  build: ./cloud_dashboard
  container_name: cloud-dashboard
  environment:
    - MQTT_BROKER=cloud-broker
    - MQTT_TOPIC=cloud/alerts
  depends_on:
    - cloud-broker
  networks:
    - factory-net

networks:
  factory-net:
    driver: bridge

volumes:
  node-red-data:
```

Настройка Node-RED

Общий вид потока

Поток состоит из следующих узлов:

1. mqtt in: Подписка на топик factory/temp от Edge Broker.
2. json: Парсинг входящего сообщения.
3. switch: Фильтрация значений температуры (> 80).
4. mqtt out: Публикация критических значений в топик cloud/alerts Cloud Broker.

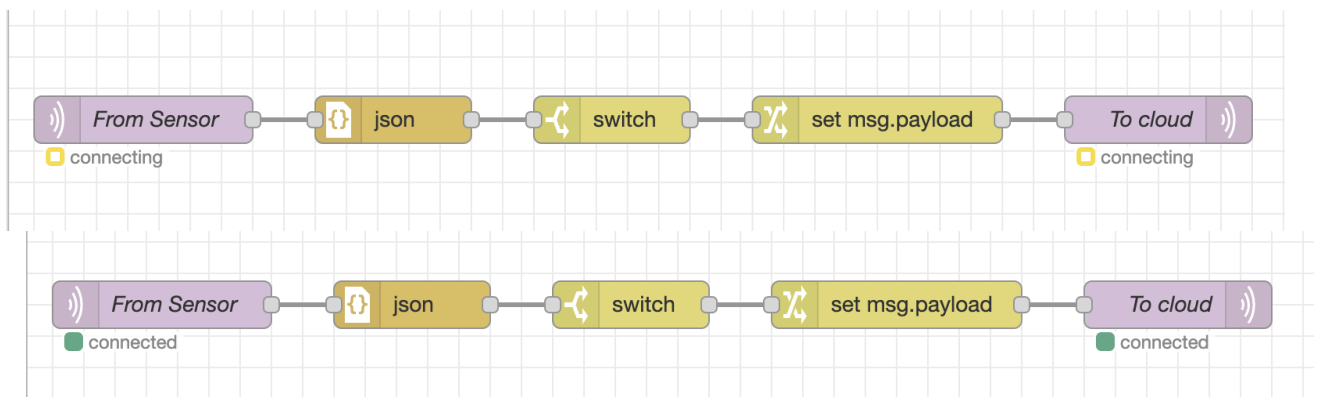


Рисунок 1 – Общий вид потока в Node-RED

Настройка узлов

Настройка подключения к локальному брокеру edge-broker. Условие фильтрации: пропускать сообщения, где `payload.temp > 80`.

The image displays two side-by-side screenshots of the Node-RED configuration interface. The left screenshot, titled 'Edit mqtt in node', shows the configuration for an MQTT In node. It includes fields for 'Server' (edge-broker), 'Action' (Subscribe to single topic), 'Topic' (factory/temp), 'QoS' (2), 'Output' (a String), and 'Name' (From Sensor). The right screenshot, titled 'Edit switch node', shows the configuration for a Switch node. It includes a 'Name' field, a 'Property' dropdown set to 'msg.payload.temp', and a comparison rule set to '>' with a value of '80'.

Рисунок 2 – Настройка узла MQTT In и Switch

Настройка подключения к облачному брокеру cloud-broker и топика назначения.

The image shows a screenshot of the Node-RED configuration interface for an MQTT Out node, titled 'Edit mqtt out node'. It includes fields for 'Server' (cloud-broker), 'Topic' (cloud/alerts), 'QoS' (blank), 'Retain' (blank), and 'Name' (To cloud). A tip box at the bottom states: 'Tip: Leave topic, qos or retain blank if you want to set them via msg properties.'

Рисунок 3 – Настройка узла MQTT Out

Заключение

В рамках лабораторной работы была развернута и протестирована архитектура Fog Computing (туманных вычислений). Основное внимание уделялось организации передачи данных с помощью протокола MQTT и реализации логики предварительной обработки на промежуточном узле (Fog Node) в среде Node-RED. Данное решение позволило осуществлять фильтрацию данных на границе сети, передавая в облачную платформу исключительно значимые события, что в итоге снизило объём передаваемых данных и вычислительную нагрузку на центральный облачный сегмент.