Internet stvari i servisa

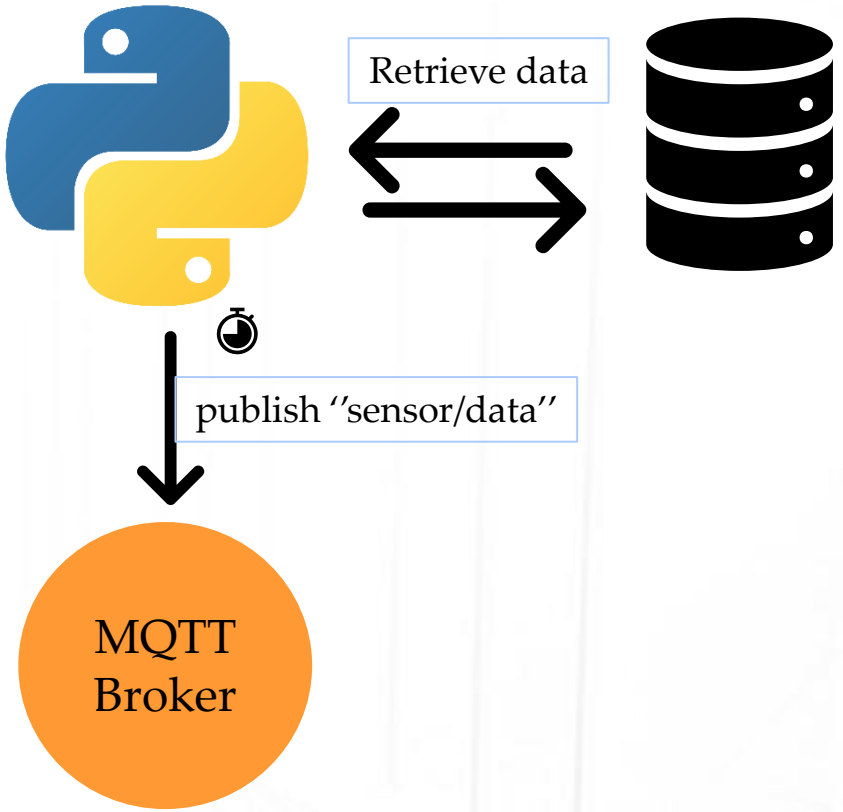
Filip Dojčinović
18135

Sadržaj

- Sensor
- Analytics
- EventInfo
- Docker
- Primer

Sensor microservice

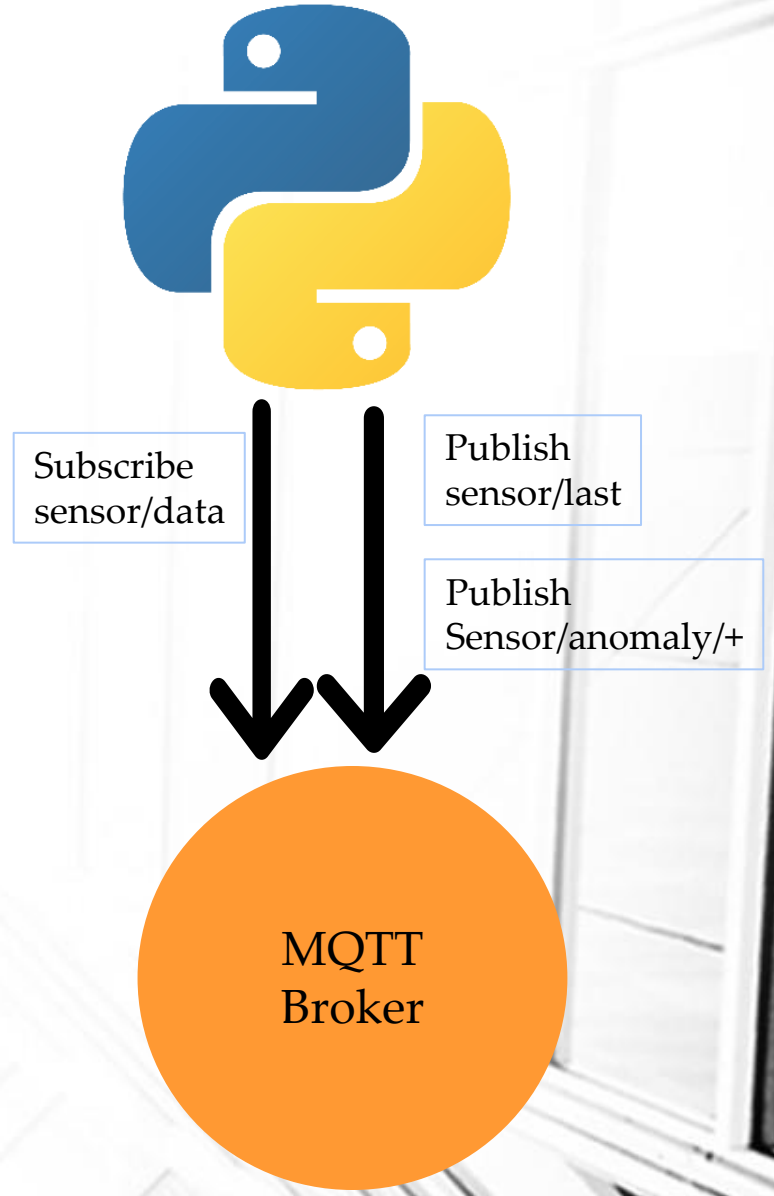
```python
def publish_sensor_data(config):
    client = connect_to_mongodb(config['MONGO_URI'])
    db = client[config['DB_NAME']]
    collection = db[config['COLLECTION_NAME']]
    mqtt_client = connect_to_mqtt_broker(config['BROKER_ADDRESS'], config['BROKER_PORT'])

    last_id = None

    while True:
        try:
            if last_id:
                query = {'_id': {'$gt': last_id}}
            else:
                query = {}

            item = collection.find_one(query, sort=[('_id', 1)])
            if item:
                last_id = item['_id']
                item['_id'] = str(item['_id'])
                json_data = json.dumps(item)
                mqtt_client.publish(config['TOPIC'], json_data)
                logging.info(f"Published sensor data to topic: {config['TOPIC']}")
            else:
                logging.info("No new data found in the collection.")
            time.sleep(config['INTERVAL'])
        except (PyMongoError, ConnectionRefusedError, OSError) as e:
            logging.error(f"Error during data retrieval or publishing: {e}")
            time.sleep(config['INTERVAL'])
```



Analytics Microservice

```python
def detect_anomaly(data):
    for field in ["AC_POWER", "DC_POWER", "DAILY_YIELD", "TOTAL_YIELD"]:
        value = data.get(field, 0)
        if value < 0:
            return f"{field} measurement_error: {value}"
        elif value > THRESHOLDS[field]:
            return f"{field} situational_anomaly: {value}"

    return False
```

```python
def on_message(client, msg):
    try:
        data = json.loads(msg.payload.decode())
        logging.info(f"Received data: {data}")

        anomaly_detected = detect_anomaly(data)

        if anomaly_detected:
            anomaly_type, anomaly_details = anomaly_detected.split(": ", 1)
            anomaly_data = {
                "type": "anomaly",
                "sensor_data": data,
                "details": anomaly_details.strip()
            }
            location = f"sensor/anomaly/{anomaly_type.strip()}"
            client.publish(location, json.dumps(anomaly_data))
            logging.info("Anomaly detected and published.")
        else:
            client.publish("sensor/last", json.dumps(data))
            logging.info("Published data to the sensor/last topic.")
```



EventInfo

```
constructor() {
  const brokerAddress = process.env.MQTT_BROKER_HOST || 'localhost';
  this.client = mqtt.connect(`my-mqtt-broker:${process.env.MQTT_BROKER_PORT}`);
  this.data = {
    AC_POWER: 0,
    DC_POWER: 0,
    TOTAL_YIELD: 0,
    DAILY_YIELD: 0
  };

  this.client.on('connect', () => {
    console.log('Connected to MQTT broker');
    this.subscribeToTopics();
});

  this.client.on('message', (topic, message) => {
    const data = JSON.parse(message.toString());
    if (topic === 'sensor/last') {
      this.handleSensorData(data);
    } else if (topic.startsWith('sensor/anomaly/')) {
      this.handleAnomalyData(data);
    }
  });

  this.client.on('error', (err) => {
    console.error('MQTT client error:', err);
  });

  this.client.on('offline', () => {
    console.warn('MQTT client is offline');
  });

  this.client.on('reconnect', () => {
    console.log('MQTT client is reconnecting');
  });
}
```

```
private subscribeToTopics(): void {
  const topics = ['sensor/anomaly/+', 'sensor/last'];
  topics.forEach((topic) => {
      this.client.subscribe(topic, (err) => {
          if (err) {
              console.error(`Failed to subscribe to topic ${topic}:`, err);
          } else {
              console.log(`Subscribed to topic ${topic}`);
          }
      });
  });
}

handleSensorData(data: DataModel) {
  this.store.addData(data);
}

handleAnomalyData(data: DataModel) {
  this.store.addData(data);
  this.alertUser(data);
}

alertUser(data: any) {
  console.log('Alerting user about anomaly:', data);
}
```

EventInfo

```
const router = Router();

router.get('/data', getData);

router.get('/data/:plantId', getDataByPlantId);

router.get('/last/:plantId', getLastDataByPlantId);


export default router;
```

```
const store = Store.getInstance();

export const getData = async (req: Request, res: Response, next: NextFunction): Promise<DataModel[] | undefined> => {
    try {
        const data = await store.getAllData();
        res.json(data);
        return data;
    } catch (error) {
        next(error);
    }
};



export const getLastData = async (req: Request, res: Response, next: NextFunction): Promise<DataModel | null> => {
    try {
        const lastData = await store.getLastData();
        if (lastData) {
            res.json(lastData);
            return lastData;
        } else {
            res.status(404).json({ message: 'No data found' });
            return null;
        }
    } catch (error) {
        next(error);
        throw error;
    }
};
```

EventInfo

- Singleton obrazac
- Struktura slična rečniku
- PlantId kao ključ
- Koristi DataModel Interfejs

```
export class Store {
    private static instance: Store;
    private data: DataModel[];

    private constructor() {
        this.data = [];
    }

    public static getInstance(): Store {
        if (!Store.instance) {
            Store.instance = new Store();
        }
        return Store.instance;
    }

    public addData(data: DataModel): void {
        this.data.push(data);
    }

    public getLastData(): DataModel | null {
        if (this.data.length === 0) {
            return null;
        }
        return this.data[this.data.length - 1];
    }

    public getAllData(): DataModel[] {
        return [...this.data];
    }
}
```

Docker
docker images

Python-sensor

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "main.py"]
```

Analytics

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "main.py"]
```

EventInfo

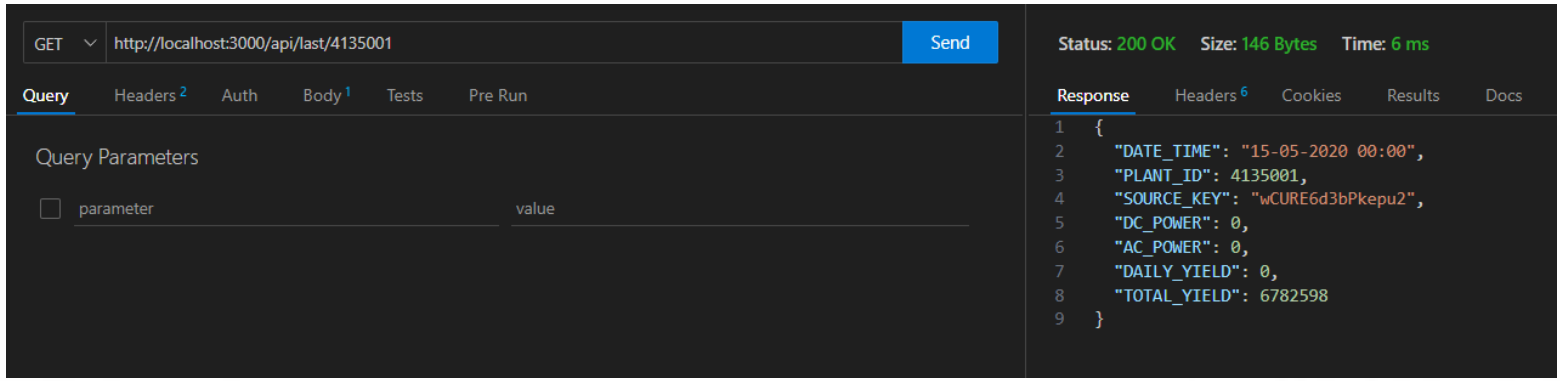
```
FROM node:14
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD [ "npm", "start" ]
EXPOSE 8080
```

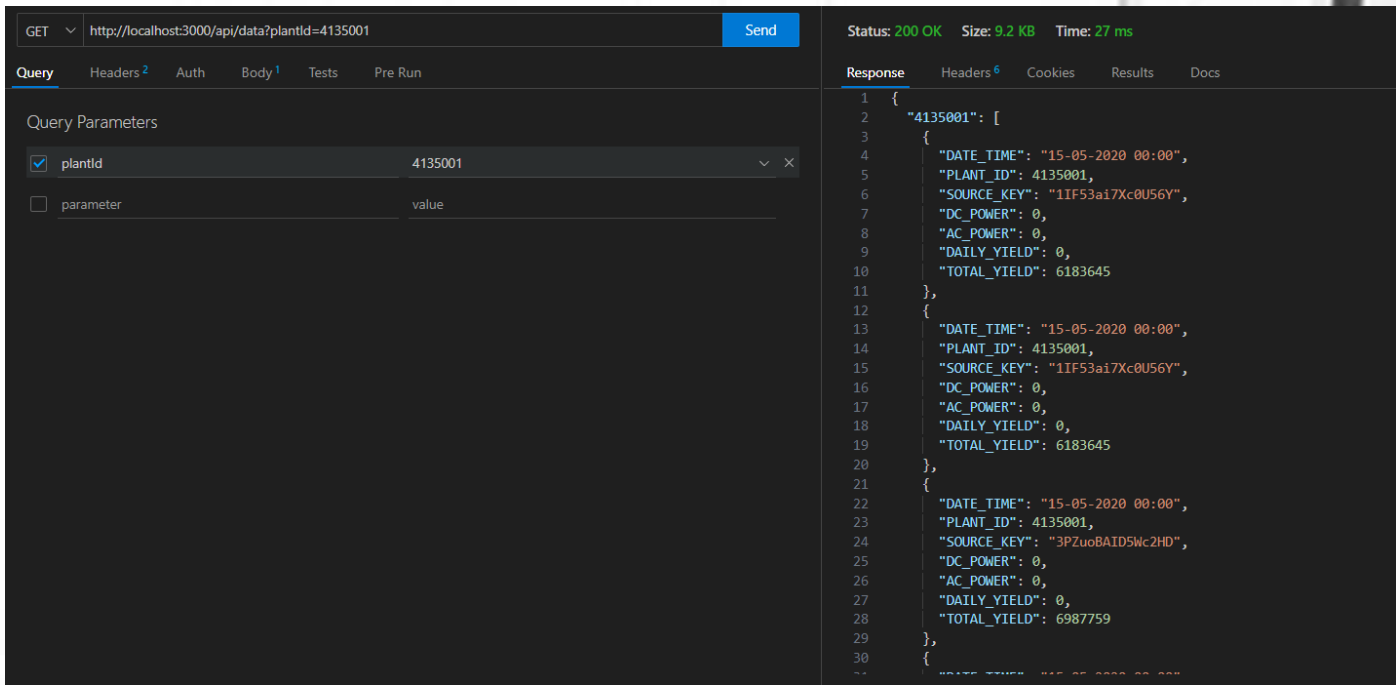
Docker
docker compose

```yaml
version: '3.8'

networks:
  my-network:

volumes:
  mosquitto-data:
  mosquitto-log:
  mosquitto.conf:
    external: true
    name: mosquitto.conf


services:
  python-sensor:
    image: python-sensor-image:1
    container_name: python-sensor-container
    #restart: always
    depends_on:
      - mqtt
    networks:
      - my-network
    environment:
      BROKER_ADDRESS: my-mqtt-broker
      BROKER_PORT: 1883

  analytics:
    image: analytics-image:1
    container_name: analytics-container
    #restart: always
    depends_on:
      - mqtt
    networks:
      - my-network
    environment:
      BROKER_ADDRESS: my-mqtt-broker
      BROKER_PORT: 1883
```

```yaml
eventinfo:
  image: eventinfo-image:1
  container_name: eventinfo-container
  #restart: always
  depends_on:
    - analytics
    - mqtt
  ports:
    - "3000:3000"
  networks:
    - my-network
  environment:
    BROKER_ADDRESS: my-mqtt-broker
    BROKER_PORT: 1883

mqtt:
  image: eclipse-mosquitto:2
  container_name: my-mqtt-broker
  volumes:
    - mosquitto.conf:/mosquitto/config
    - mosquitto-data:/mosquitto/data
    - mosquitto-log:/mosquitto/log
  networks:
    - my-network
  ports:
    - "1883:1883"
```

Rest API testing






The End