Informatik 10 - Lehrstuhl für Rechnertechnik und Rechnerorganisation
Fakultät für Informatik
Technische Universität München

IoT Platform for data processing and access.

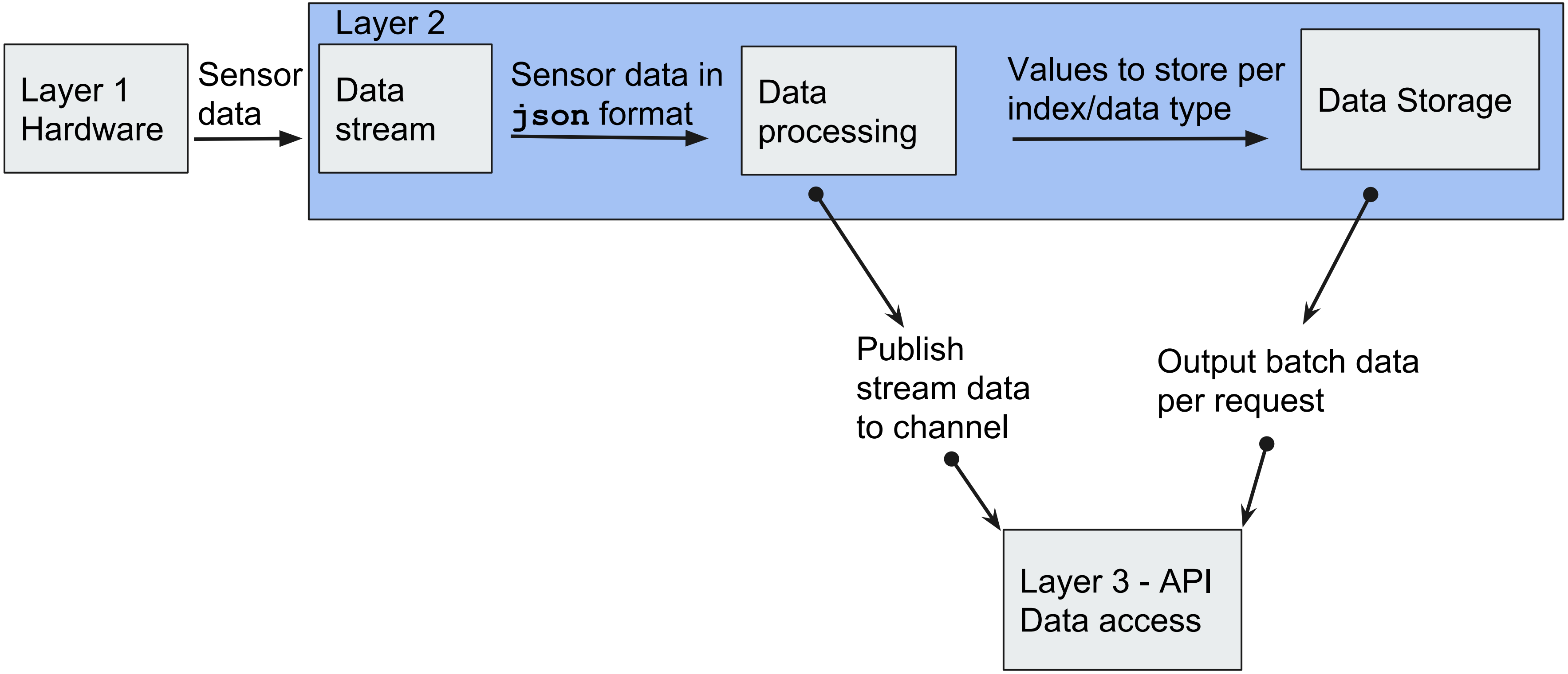
Master Practical Course WS 2017

Contents

Goals to Achieve

1. **Out-of-the-Box** Highly-Available & Fault-Tolerant system that can collect, store and provision data from sensors in a unified format.
2. Guaranteed **single** message delivery.
3. Possibility to **permanently** store the data.
4. High-speed data **compression**.
5. **Secure** data access using User Authentication.
6. Collect **performance data** and visualize it.

Information flow through the system



Sensor Layer

- Erkin Kirdan
- Mikayil Murad
- Hakan Uyumaz
- Ali Naci Uysal

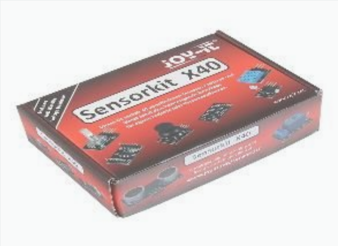
Overview

Team I:

Responsible for the development of the on-board part of the platform.

Goals:

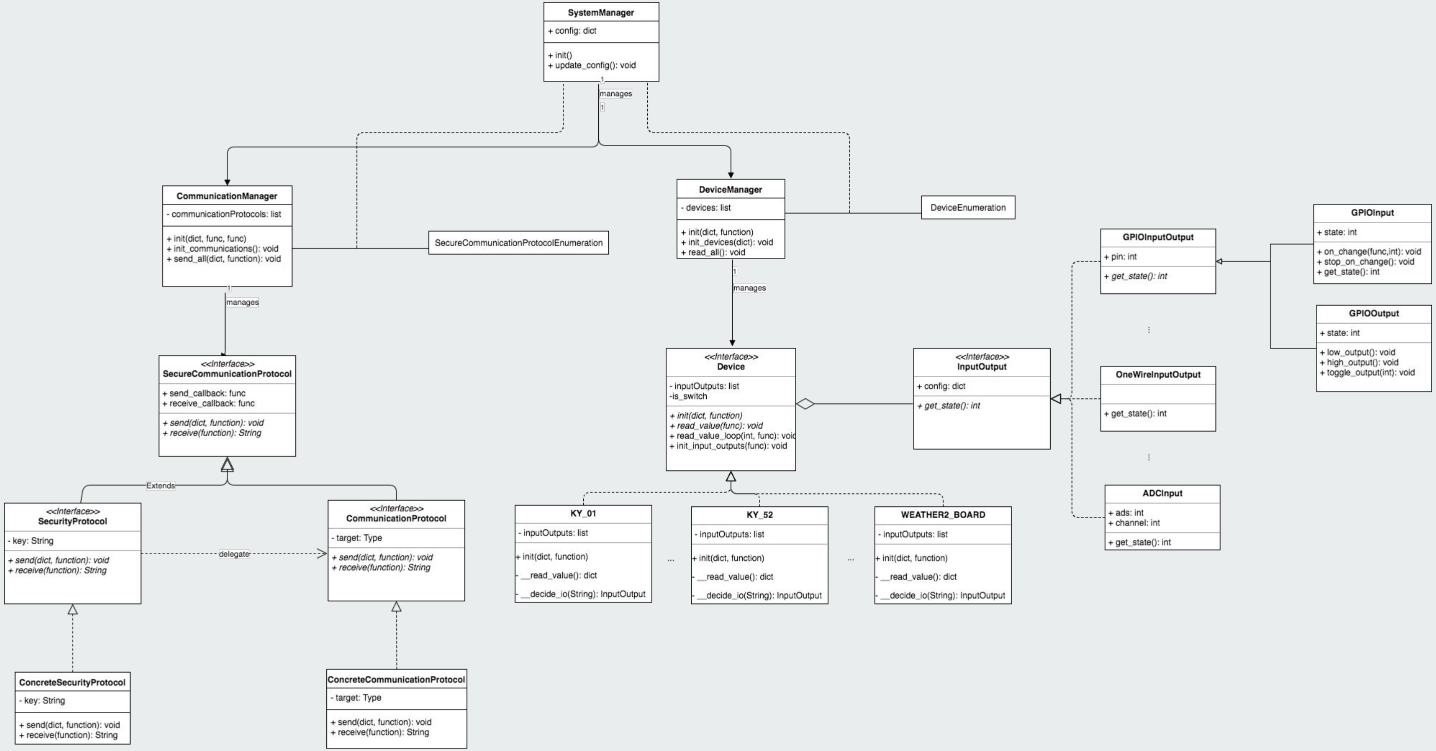
- Support data transmission for as many different boards and sensors out-of-the-box as possible
- Support easy deployment and setup



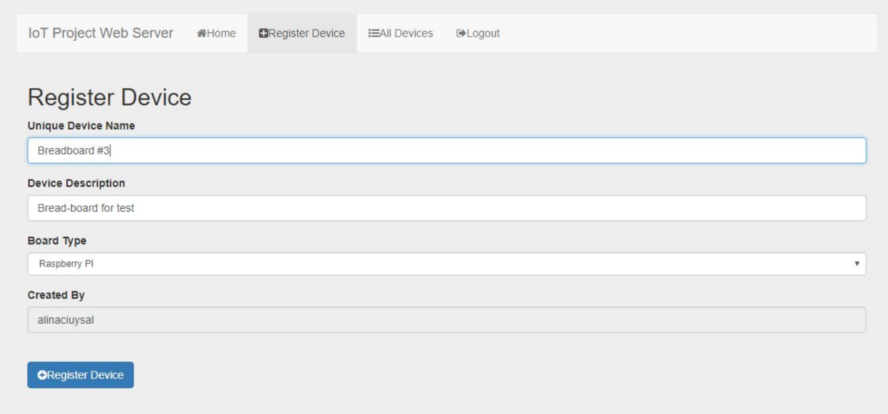
Raspberry PI 3

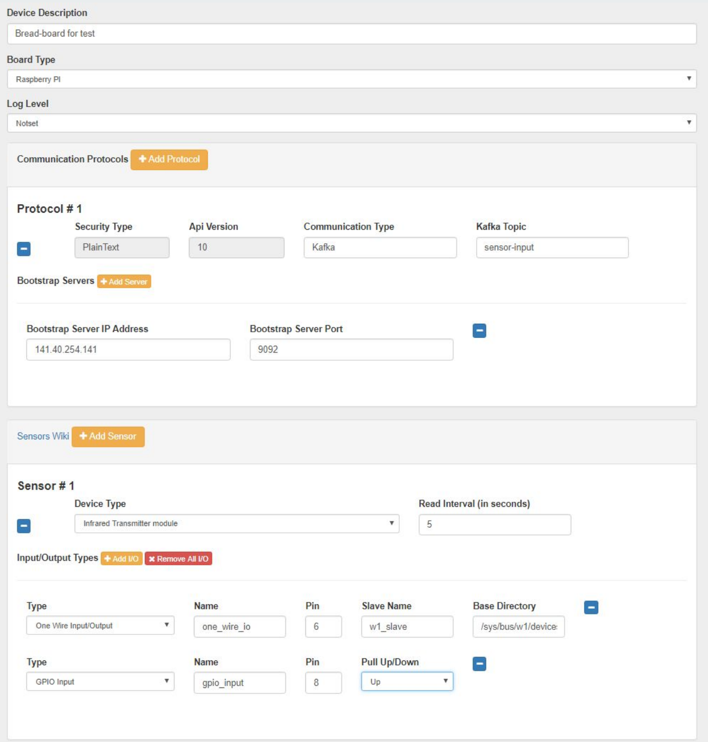
Odroid-XU 40

Architecture



MEAN Stack Web Application





Results

1. Provision of the data messages from sensors in the unified format including timestamps and values of the corresponding metrics
2. Basic out-of-the-box-deployed drivers for different types of single-board computers with different types of sensors connected
3. Adjustments of the layer configuration via settings files
4. Additionally built web application for easy configuration
5. Basic data pre-processing according to settings (e.g. use every n-th measure)
6. Easy to extend architecture for new types of boards and sensors

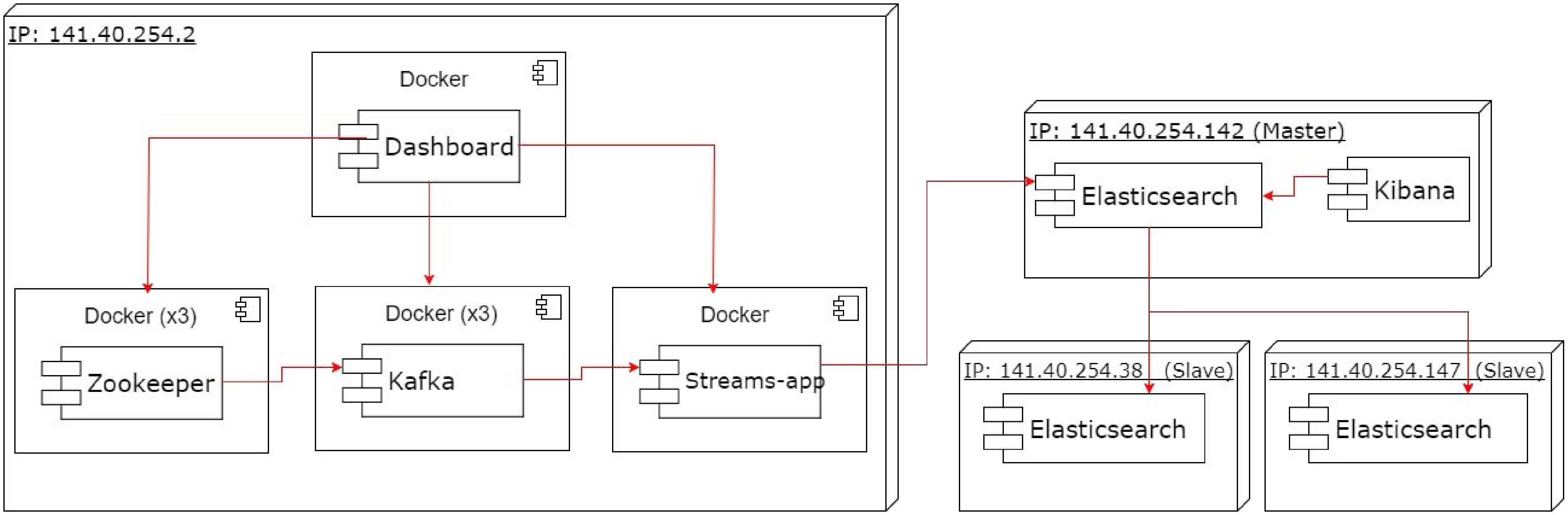
Data storage & processing layer

- Yesika Ramirez
- Moawiah Assali
- Atakan Yenel
- Sergey Nasonov

Contents

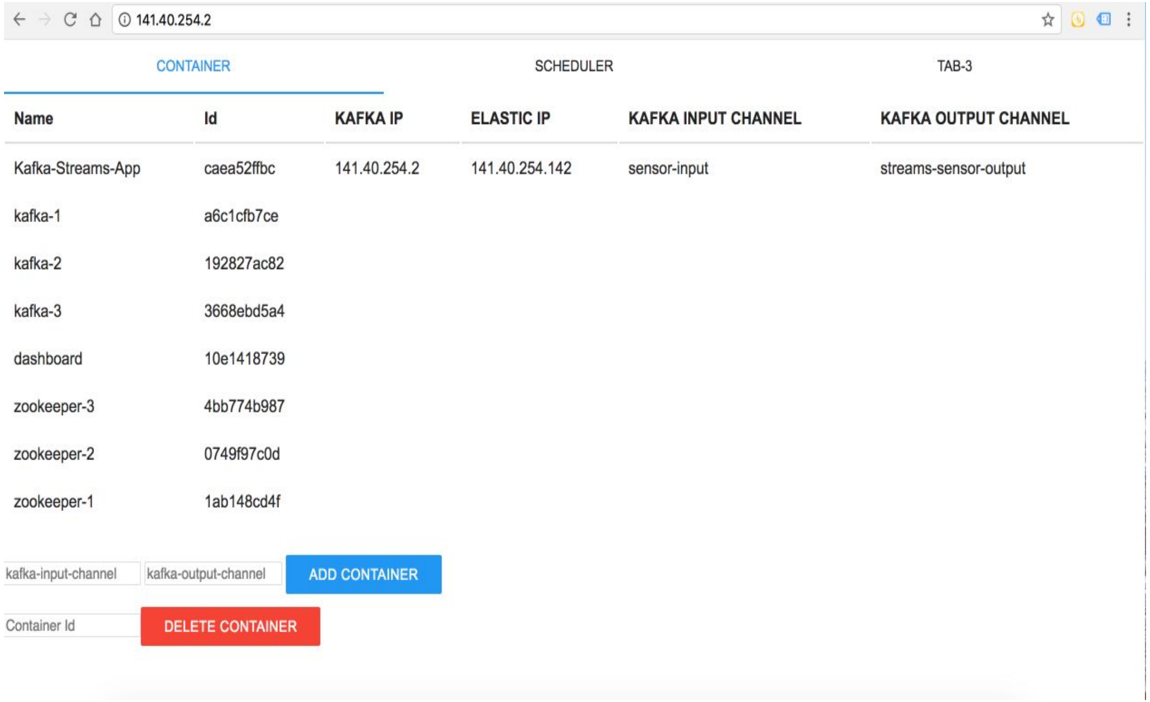
- Deployment Diagram
- Configurations
- Streaming & Processing
- ElasticSearch
- Advantages and Challenges

Deployment Diagram

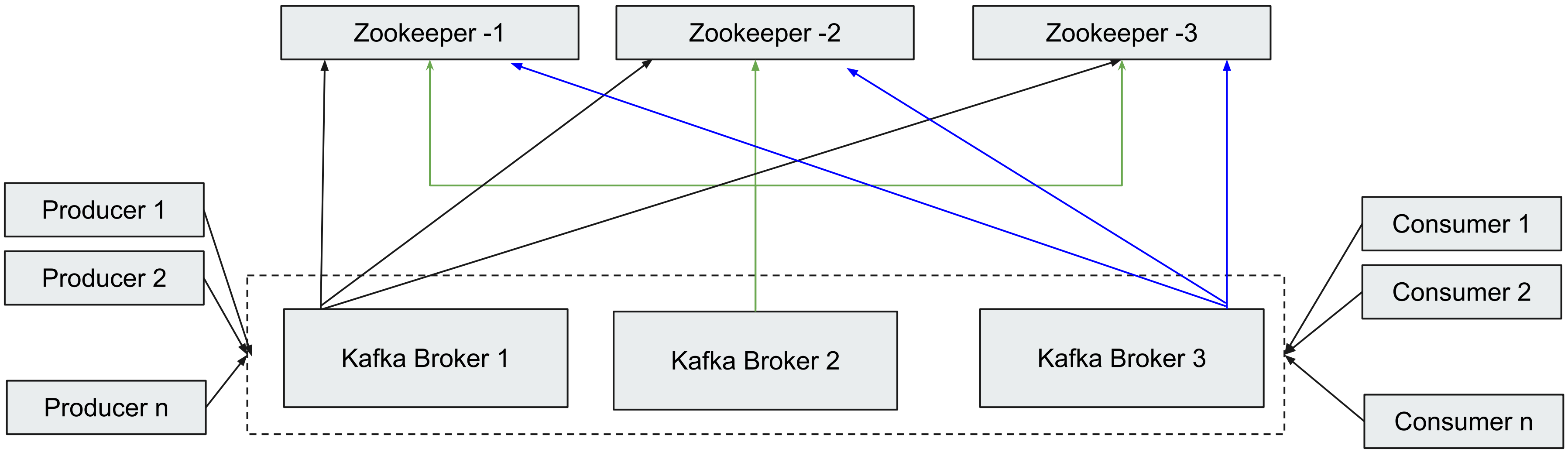


Configuration

- Input Channel
- Output Channel
- Database Index
- Database Document type
- The running process
- Kafka broker IP
- Elasticsearch IP



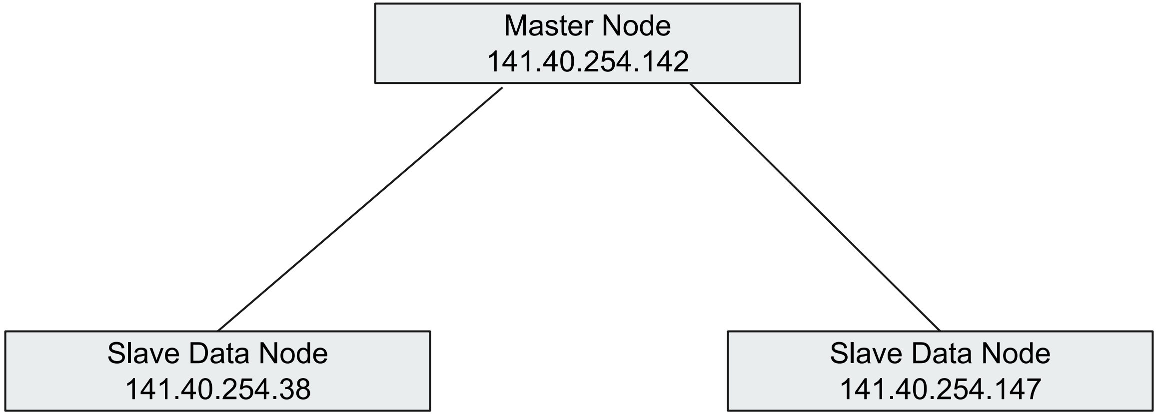
Streaming & Processing

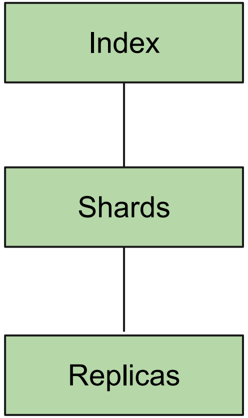


*Producer API *Consumer API *Stream API *Connector API

ElasticSearch

Ports: 9200/9300





*Master Eligible Node *Zen Discovery *Election Process

TECHNISCHE UNIVERSITÄT MÜNCHEN

Advantages

- Fault Tolerance & High Availability
- Scalability of processing engines
- Easy to deploy
- Low cost
- Flexibility via configuration files

Challenges

- Complexity:
  - Different vendors/versions of the used Software
  - Connectors between streams and ElasticSearch
- Technology Choice
- Security

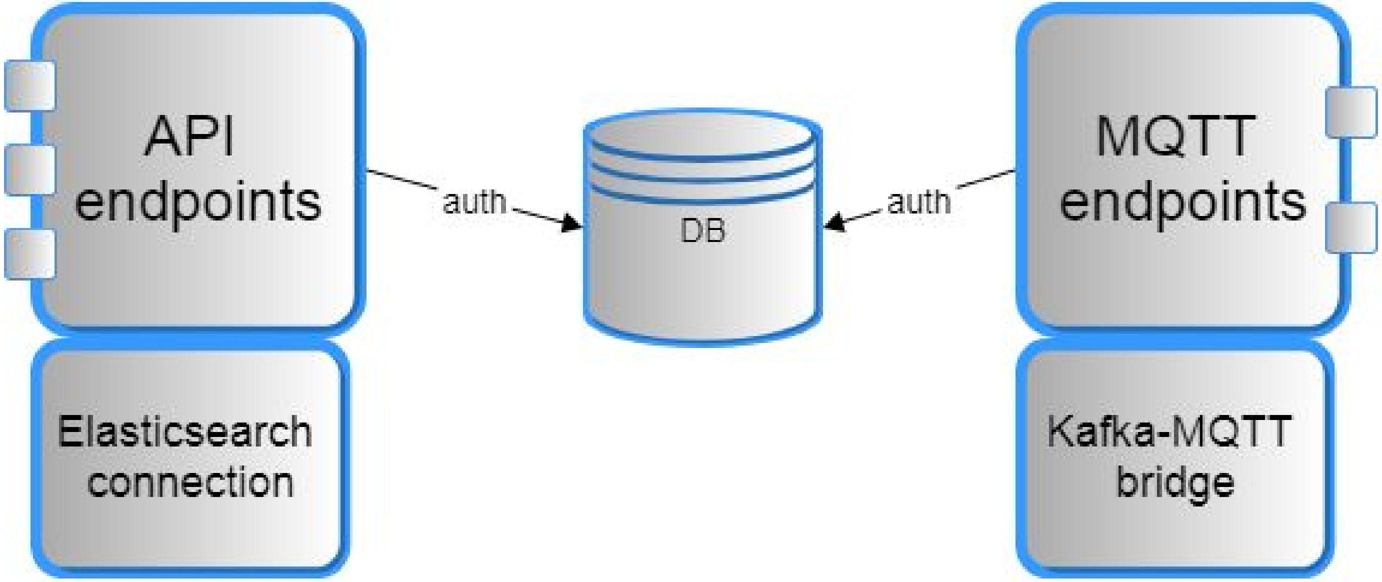
API Layer

- Sergei Drugalev
- Faisal Hafeez

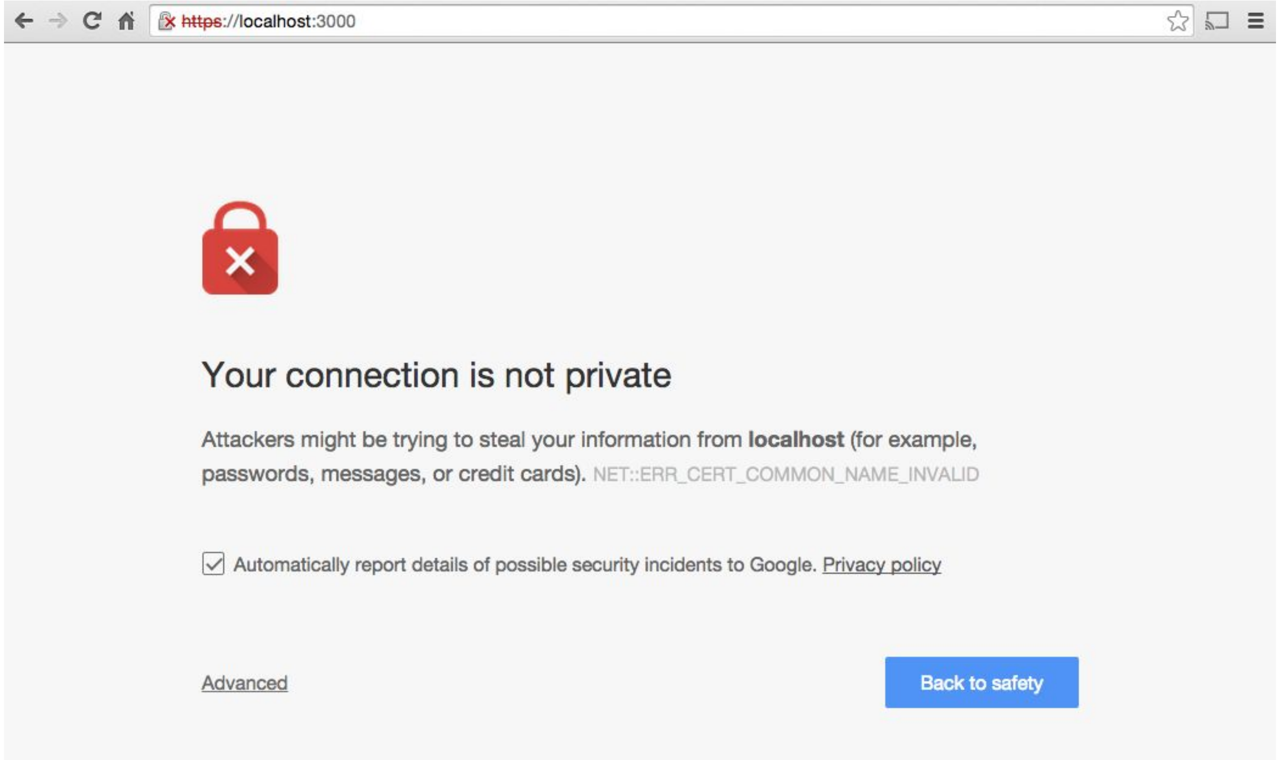
Contents

- HTTPS
- User authentication
- User Authorization
- API
- MQTT

Basic architecture



HTTPS


https://localhost:3000
Your connection is not private
Attackers might be trying to steal your information from localhost (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID
Automatically report details of possible security incidents to Google. Privacy policy
Advanced
Back to safety

User Authentication

- RQlite relational database
- Young project (~1 year old)
- SQLite storage engine
    - Extremely compact and lightweight
    - Doesn't support clustering
    - RQLite does!

User Authentication

- Basic HTTP authentication (login:password)
- Why not Facebook or Google?
  - Not everybody want/like using their private accounts
  - No web client
  - API server is not a website
  - Still easily doable
- Credentials -> ask your system administrator

User authorization

- Wait… Is it different?
- Users have configurable access rights
- Ordinary users cannot alter the data

MQTT

- RABBITMQ broker
- Same authentication AND authorization scheme
- Advantages
  - Lightweight
  - Easy to parse
  - Very little resources on client-side
  - Reliable (QoS 1 and 2)
  - Secure (user authentication + TLS, authorization)
  - Topic wildcards!!

MQTT - topic structure

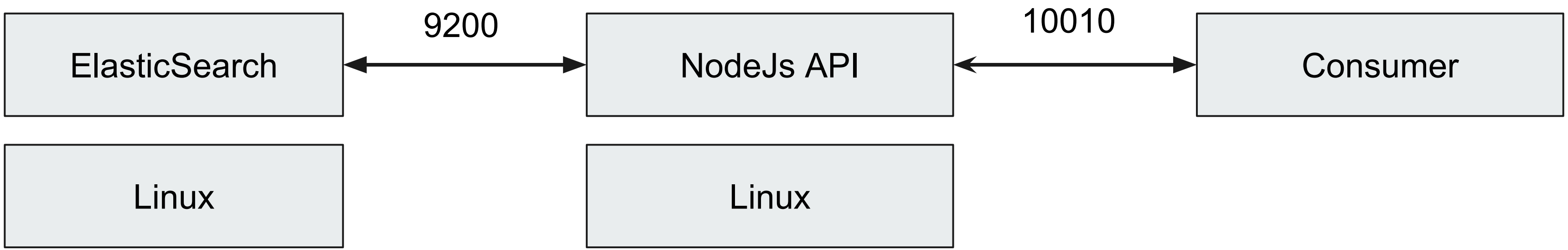
- TOPIC
  - / {board ID} / {sensor ID} / {Data Type}

- PAYLOAD
  - {

        value: {},
        unit: {}

    }

API



API Endpoints

- /get_all_boards
- /get_sensor_data/:board_id
- /get_sensor_data/:board_id/:sensor_id
- /get_sensor_data/:board_id/:sensor_id/:from
- /get_sensor_data/:board_id/:sensor_id/:from/:to

Challenges

- Dynamic IPs
- Backup machines
- Elasticsearch server down
- Disk Storage

Performance Testing

- Shumail Mohyuddin
- Muhammad Razzaque Bhatti
- Zaryab Khan

Goals

- Monitor the platform and its performance
- Alerts based on threshold values
- Visualize the collected data
- Highlight bottlenecks and recommend possible improvements

Technology stack

- **K6**

  Generate load via virtual users - Behavior test cases written in JavaScript

  Test platform performance under stress

- **InfluxDB, Prometheus**

  Time based database and metrics storage from multiple sources

- **Grafana**

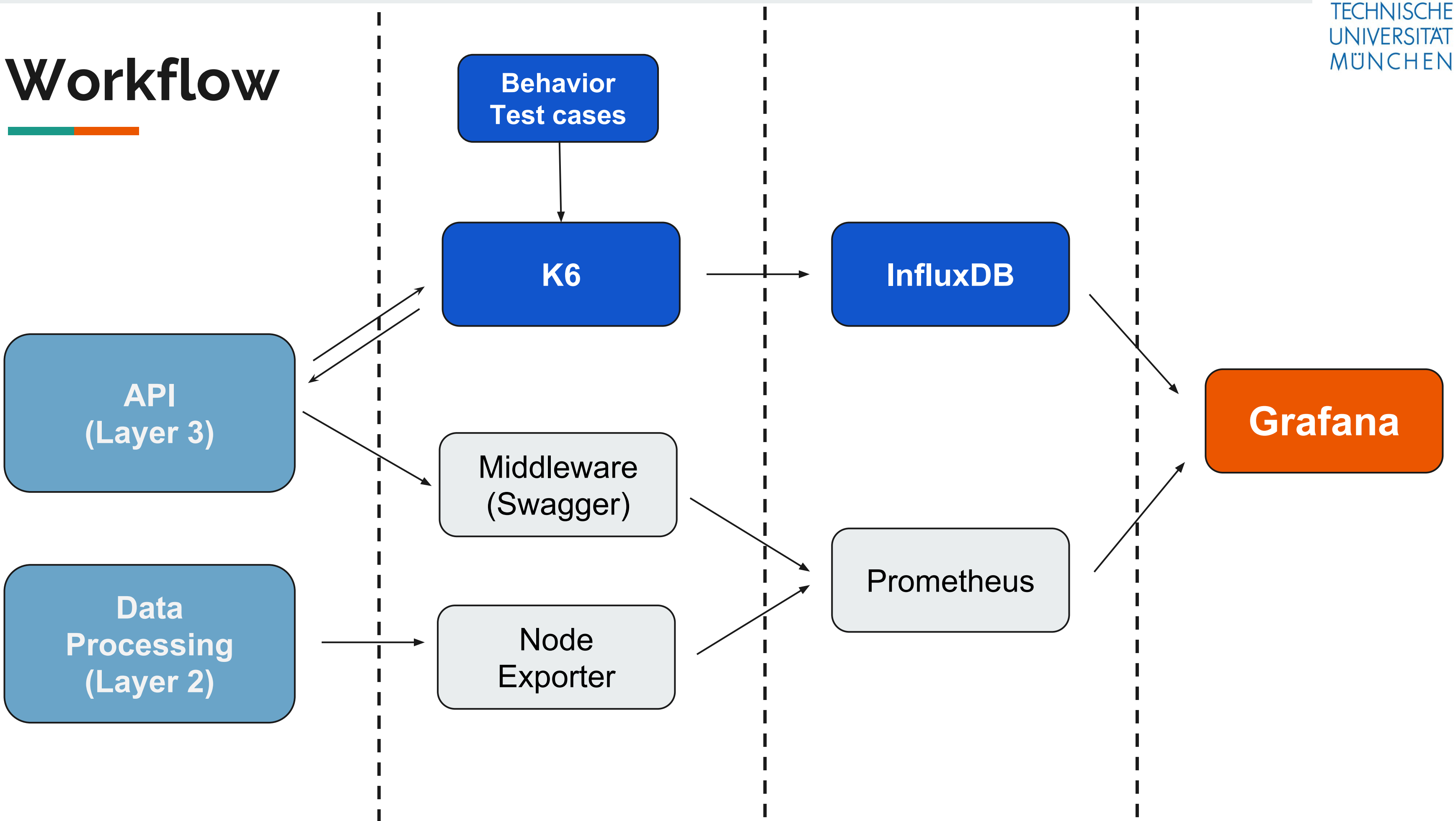
  Data visualization across multiple sources

- **Swagger API Framework middlewares with ExpressJS**

  Intercepting the API requests for event logging

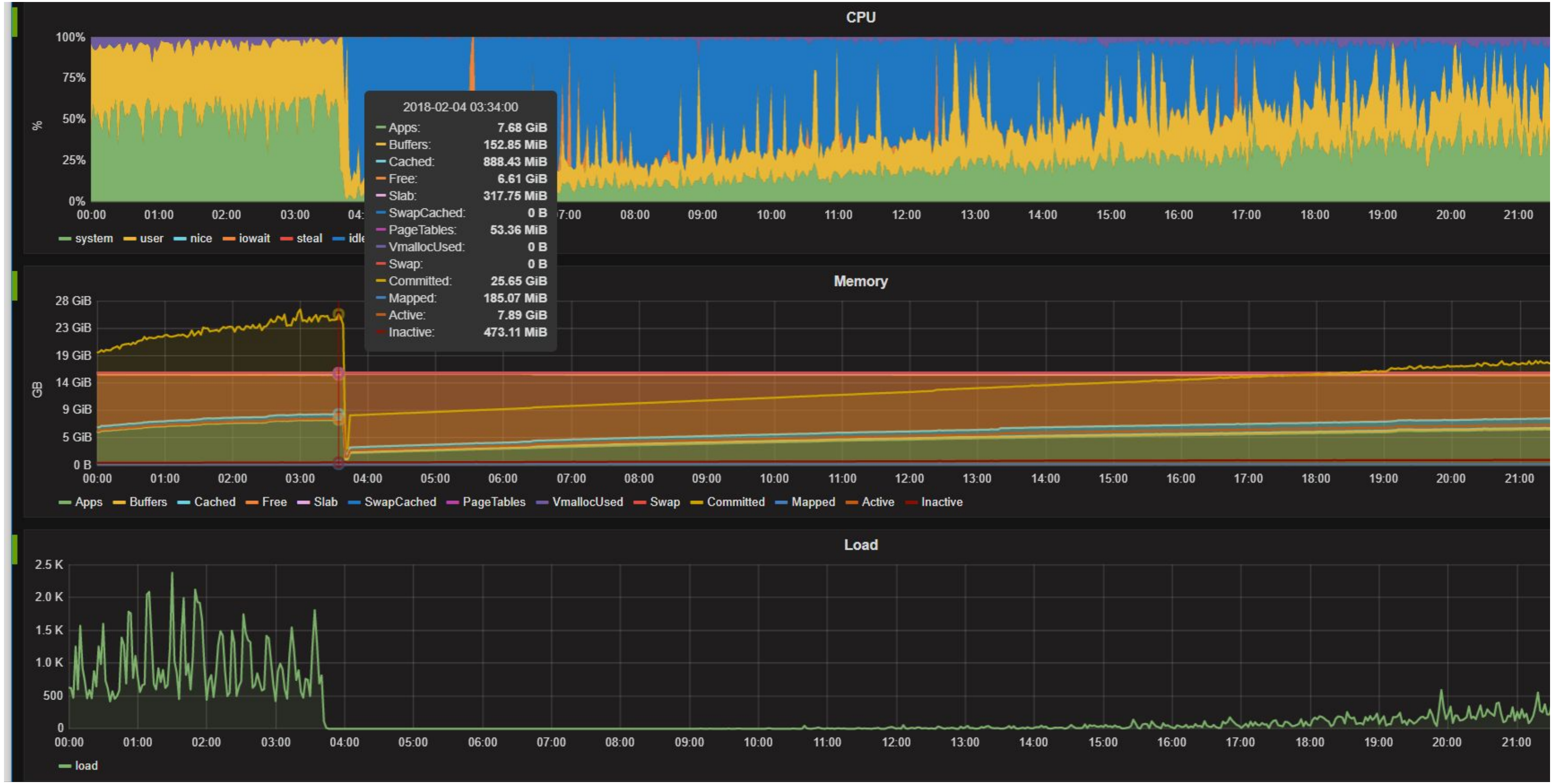
- **Node Exporter -** Infrastructure monitoring and alerts

Workflow



Behavior Test cases
K6
InfluxDB
API (Layer 3)
Middleware (Swagger)
Grafana
Data Processing (Layer 2)
Node Exporter
Prometheus

Data storage and processing monitoring

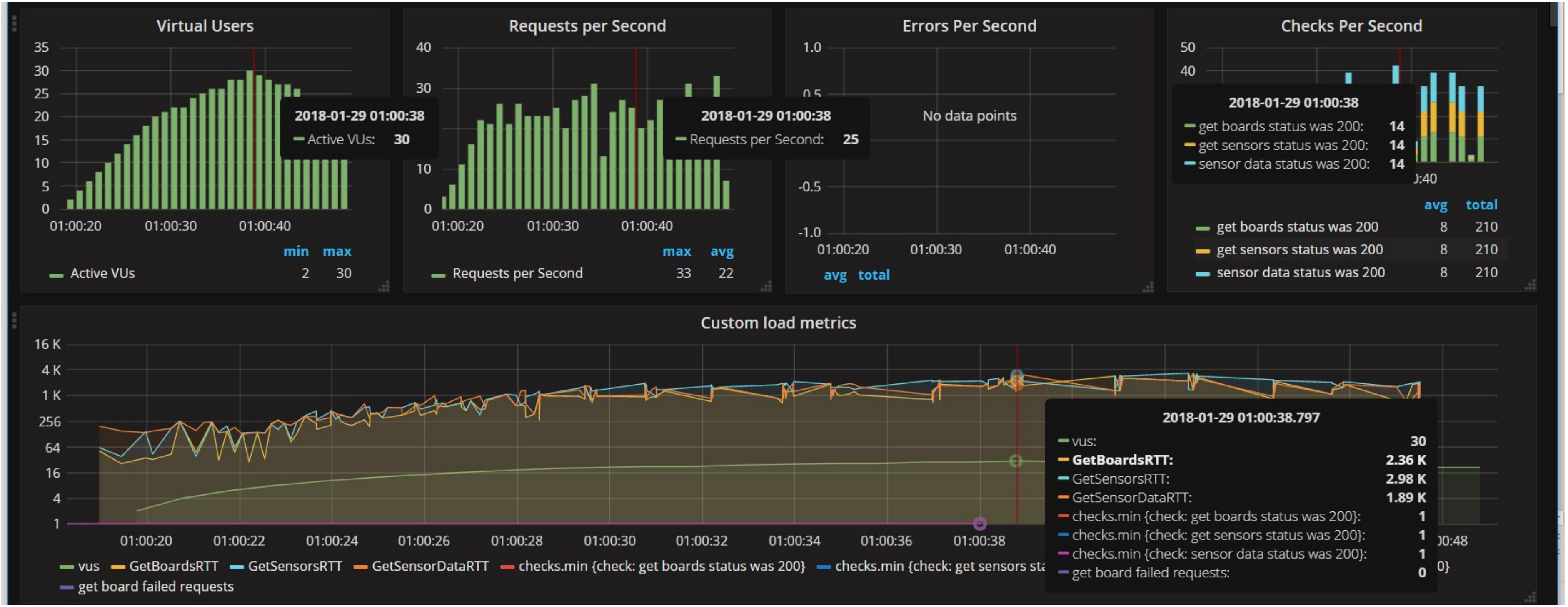


Stress Testing

- Virtual Users - 35
  - Increased Dynamically over time
  - Behavior:
    - First request on to get list of boards
    - Select a board and send second request for getting sensors
    - 3rd Request for fetching values from sensors
    - 2 more requests for fetching sensors data
  - One request sent by each VU for this behavior in every second.
  - ~8000 API calls in 30 seconds

- Results:
  - Alert generated when avg RTT over test > 3s
  - Threshold reached at VU= 30
  - RTT averages from 20 milliseconds to ~3 seconds under max stress.

Stress Testing



Monitoring Statistics Interfaces

- **REST API**

  GET on `<api-url>/swagger-stats/stats`

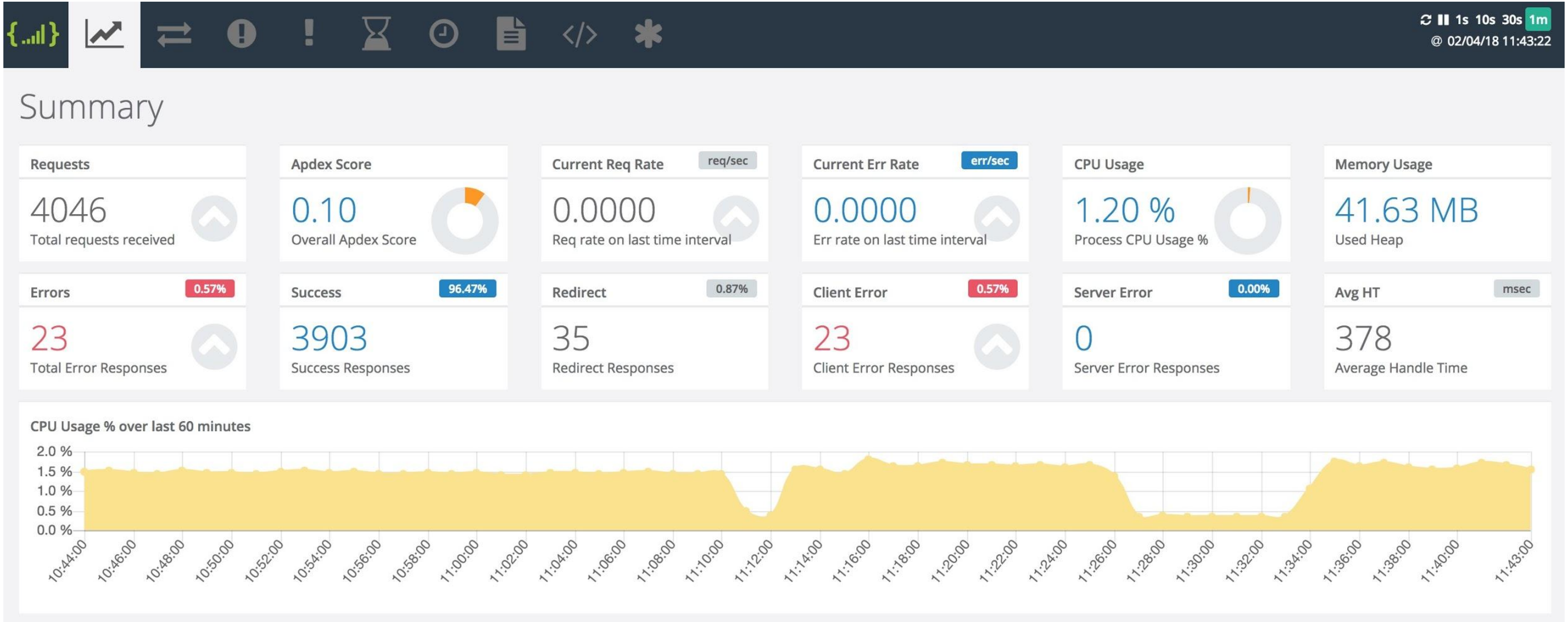
- **Prometheus Metrics**

  Available on `<api-url>/swagger-stats/metrics`

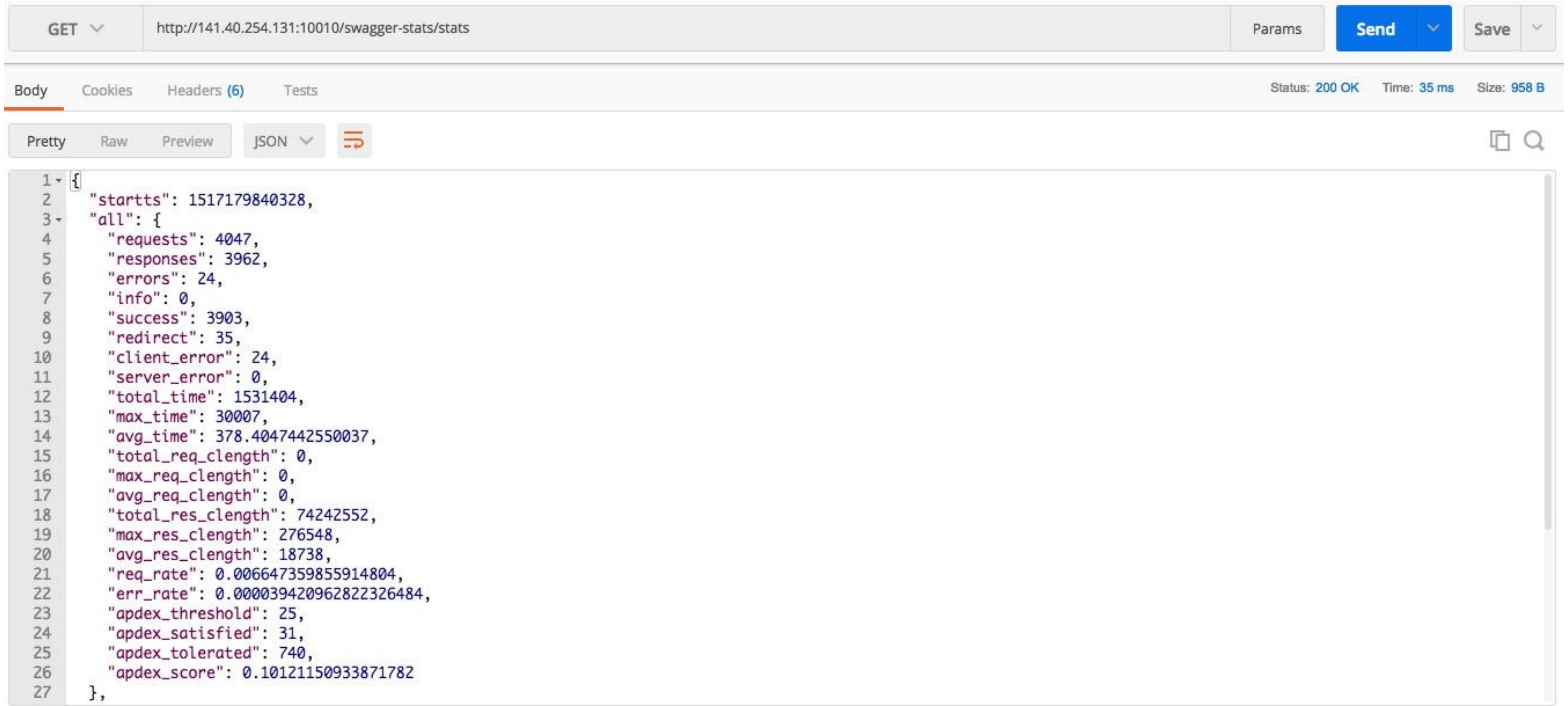
- **Visualisation Dashboard**

  Available on `<api-url>/swagger-stats/ui`

API Realtime Monitoring



API Statistics via API



Performance Recommendations

- Though current platform “survived” and didn’t crash...But.

- More resources (especially compute power) overall
    - CPU usage reached 100% under heavy stress on Layer-2
    - Similar recommendation for Layer-3

- Load-balancer behind Layer-3 (API) Entry point to have more worker nodes for fault-tolerance and scalability.
- Massive Scalability? Microservices architecture

Live Previews