# Nexthink

## Full-stack Software Engineer Assignment

### (Device Performance Index)

Armando J. C. Marques

Date: 20/11/2020

## Introduction

First of all I would like to give thanks to Nexthink for the opportunity to show my capabilities in this assignment and for taking the time to go through my work. This assignment required some good hours of hard work and research but nevertheless it was a very interesting challenge that resembles a real world problem, which I was very pleased to do.

For any question or clarification please feel free to contact: ajcmarques@icloud.com

## Requirements

To know the exact requirements to any project is half of the work of any software developer, making sure that everybody is in the same page and the right solution is found. In a real life scenario no assumptions would be made in other to get a real picture of the constraints and expectations from businesses owners.

The requirements for this assignment being as generic as they can be in the present form, would require a few good weeks of analysis, meetings and mail exchanges in order to pin down some of details from what is expected from this application.

# Assignment 1

Developments implemented for the assignment were made in a way as to get a proof-of-concept application and not a finished product, therefore parts of the applications were greatly simplified and performance and security issues were not addressed as they would be in a real scenario.

The backend is implemented with Spring Boot using JPA and a Rest API, it could be easily integrated as an microservice, the ports and request are open to everyone and require some sort of authentication (CORS, JWT) even in early development stages.

Unit tests were made as an example as time was scarce and Integrations tests would be more efficient and reliable.

There are some small issues in the frontend that would require some changes, but in other not to extend the work for more than one week it is delivered as it is.

Some points in the assignment were not too clear, there were doubts regarding on how the min and max values are calculated and if dpi values needed to be saved for each calculation, for simplification dpi was calculated min and max values from all the values in the samples.

## Backend

All the data processing was implemented in JPA as a mocked big data interface, since JPA is also used to persist Business data this was the easiest and simpler way to do it.

All the code in package 'bigdata' should be seen as a dummy implementation to be used only as proof of concept.

There are two Rest Controllers one for Samples and DPI related activities (SamplesController) and the other to handle business domain objects (BusinessController).

The code follows the common structure of Controller → Service → Repository for simplification the creating of interfaces for services was skipped.

There are two main entities Samples and DeviceDpi, samples are uploaded via a CVS file or read from a post request and then DPI values are calculated for each device and saved as DeviceDpi entities.

Spring boot is running a task at startup (AppStartupRunner) that reads entries from a CVS file and process the entries to populate the tables.

### Frontend

One improvement in the frontend would be to replace useContext with Redux, although the first is simpler and easier to use there were unexpected issues with the use of context hooks, the performance is not good and it causes flickering in the components, also there is an out-of-focus event that conflicts with elements as a TextField or Slider.

The application also would benefit with a caching clients and offices data, so that the names could be used in the tables without performance degradation.

User feedback and confirmation could be improved in the Samples page.

Filter controls for DPI are slow and would require a tweak, the range could be implemented with a Slider.

Finally, if this frontend was released to the client it would require some sort of integrations tests, but such requirement would demand a more extensive research to find the best solution.

## Big Data

This assignment revolves around the issue of processing and storage large amounts of data and real-time event processing, a subject somewhat beyond my expertise, as my main experience comes from developing applications in Java environments which is not at all the best solution for this field of engineering.

A common 3 layer architecture for a Java has limitations regarding the load and number of objects that can be processed in a graceful period of time.

### Browser

In the browser a React program can handle like a 1K array of objects like a search result before it gets to slow and required proper optimization.

### Application Server/Microservice

A Java Rest API using pagination can query large portions of data but a task processing elements from a ORM framework as JPA would have trouble with more than 100K, simple JBDC queries would increase this threshold up to a million.

The use of microservices and containers can increase the throughput of an application, but it wouldn't not make it more efficient or necessarily reduce the overall operation time.

### Database

SQL Databases are rarely the problem when it comes to performance however SQL are not designed for the type of work related with big data.

## Makeshift Big Data

Without prior knowledge having to design and implement a system to process large amount of data I would use regular SQL database and write all DPI calculation logic in the database using store procedures and temporary tables alike.

If data structures and calculations were simple enough this could be an efficient solution, in worst case scenario run simple JDBC Java processes in isolation to process the data.

## Big Data Architecture

A proper Big data solution would have to use proper frameworks and tools as Apache Kafka and Apache Hadoop.

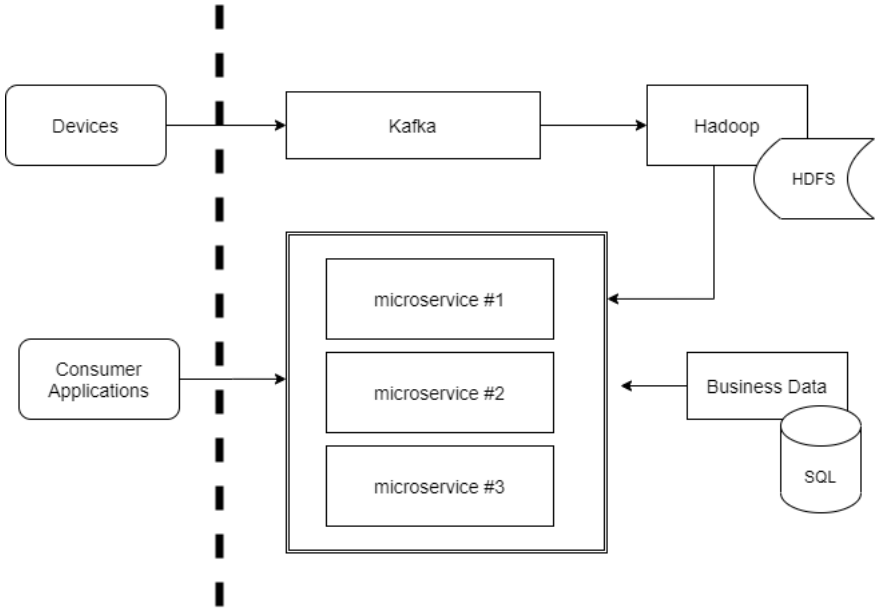



*Figure 1: Proposed architecture*

Apache Kafka is a low latency event stream platform and Hadoop is a set of utilities to store and process data, this being the core functionality the overall system would be designed around this elements.

As in a regular business application there would be a persistence layer using a SQL database and business logic could be implemented as microservices to improve performance and application scaling.

Consumer applications would use these services and there are some choices regarding the protocol:

- Http/Rest – The standard de facto in the last years, simple and reliable but requires custom or third party solutions for common problems as authentication and data mapping.
- GraqhQL – A more recent approach designed complex structures that support features like event subscription, authentication, however support is still in early phase of adoption in most platforms and environments.
- Reactive programming – Increasingly popular, namely in the Javascript world, is an event driven approach with focus on data streams.

# Alert Design

A monitoring and alert system would be easily integrated in the previous design, requirements demand a real-time solution where data has to be analysed in the stream and compared with historical data on-the-fly, most probable using a multithread architecture and other tools from the big data framework.

Any alert corresponding to a substantial decrease in performance would be sent to a message queue to be processed.

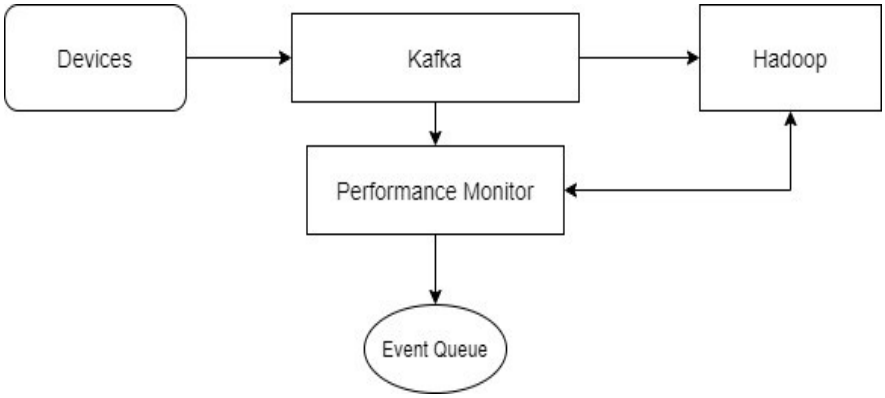
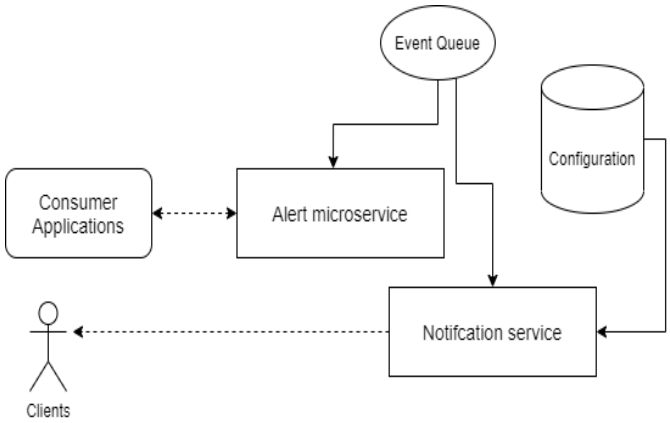



*Figure 2: Monitorization diagram*

## Event handling

There are two workflows that can be initiated from a performance degradation event:

alert, a message to consumer applications that can use the current API for retrieving data, some frameworks like GraqhQL or Reactive allow for event subscription.




notification, another service that would send notifications to entities (users, managers, devices,..) using any sort of mean (SMS, email, device notification,..) relying in configuration kept in a database.

There are many event queues available in the market: RabbitMQ, AWS SQS,  for a notifications server there already some services in the market like AWS SNS.

## Tests

It's difficult to think about tests in such an early phase as workflows and processes are not completely defined.

Unit tests can be important but the priority should be integration tests, distributed systems requires a test framework that joins all elements in a systems and validates its interaction.

There would be necessary to define a set of tasks to create test benches that recreate the environment in isolation for development and tests. Containerization and CI would play a very important role in other to allow integration of services and easy procedures to build and deploy such environment.

Due to performance, latency and throughput requirements it would be critical to have the ability of create and inject a high load of events in to measure and analyse the performance of the system.