## DELIVERABLE

### D2.2 - Data Infrastructure for ITS High-throughput Sensor based Setups – Initial Version

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of Document</td>
<td>Demonstrator</td>
</tr>
<tr>
<td>Contractual date of delivery</td>
<td>M15 - 30/07/2016</td>
</tr>
<tr>
<td>Actual Date of Delivery</td>
<td>M15 – 30/07/2016</td>
</tr>
<tr>
<td>Deliverable Number</td>
<td>D2.2</td>
</tr>
<tr>
<td>Deliverable Name</td>
<td>Data Infrastructure for ITS High-throughput Sensor based setups – initial version</td>
</tr>
<tr>
<td>Deliverable Leader</td>
<td>JSI</td>
</tr>
<tr>
<td>Work package(s)</td>
<td>WP2</td>
</tr>
<tr>
<td>Status &amp; version</td>
<td>V2.0 – Final (revised)</td>
</tr>
<tr>
<td>Number of pages</td>
<td>48</td>
</tr>
<tr>
<td>WP contributing to the deliverable</td>
<td>WP2</td>
</tr>
<tr>
<td>WP / Task responsible</td>
<td>T2.2</td>
</tr>
<tr>
<td>Coordinator (name / contact)</td>
<td>Mr. Konstantinos Thivaios (INTRASOFT International)</td>
</tr>
<tr>
<td>Other Contributors</td>
<td>JSI, UOW, NISSA, UNINOVA, KAPSCS, EP, ADRIA, LPP, BCC</td>
</tr>
<tr>
<td>EC Project Officer</td>
<td>Mr. Georgios Sarros (EC, INEA)</td>
</tr>
<tr>
<td>Keywords</td>
<td>Data Infrastructure, Data sources,</td>
</tr>
<tr>
<td>Abstract (few lines):</td>
<td>This deliverable is the report accompanying the first version of Optimum Data Infrastructure, with the focus on ITS High-Throughput Sensor based Setups. It presents and describes the architecture and current</td>
</tr>
</tbody>
</table>
implementation of the first version of the data infrastructure, how everything is connected together, serving the data to Optimum project and the tools that are used for specific set-ups.

### Deliverable resubmission

<table>
<thead>
<tr>
<th>EC Recommendation</th>
<th>Actions Performed</th>
<th>Section in the document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 2 gives an overview of the Toolset. It is however not clear who is using it and why the diverse toolset as it is still needs to be supported and how it fits in a (functional) architecture. There are many “can be used” but it is not transparent what “WILL” be used! This document provided the description of the first version of Optimum Data Infrastructure. Also, the effects of some of the approaches can be already noticeable. There is no information provided regarding the locations of platform.</td>
<td>Added a section where we exactly specify what is used and how. Similar section added for locations of the nodes.</td>
<td>Immediately after title in section 6 – Implementation.</td>
</tr>
</tbody>
</table>
Table of Contents
Deliverable resubmission........................................................................................................... 2
Executive summary.......................................................................................................................... 7
1 Introduction................................................................................................................................. 8
   1.1 Objectives of the deliverable................................................................................................. 8
   1.2 Structure of the deliverable................................................................................................. 8
2 Toolset ....................................................................................................................................... 9
   2.1 MongoDB ............................................................................................................................ 9
   2.2 PostgreSQL and PostGIS..................................................................................................... 9
   2.3 RDF Store .......................................................................................................................... 10
   2.4 RabbitMQ and Kafka .......................................................................................................... 10
   2.5 Spark .................................................................................................................................. 11
   2.6 Hadoop................................................................................................................................ 11
   2.7 QMiner ............................................................................................................................... 11
      2.7.1 Modules for sensor streams ......................................................................................... 11
      2.7.2 Example Applications ................................................................................................. 13
   2.8 API Management Console .................................................................................................. 14
   2.9 Data Retrievers .................................................................................................................. 14
   2.10 Mobile Collection Libraries ............................................................................................. 15
   2.11 Edge Processing (computing) ........................................................................................... 15
   2.12 Watchdog.......................................................................................................................... 15
   2.13 Pingdom ............................................................................................................................ 15
3 Hardware ................................................................................................................................... 16
   3.1 Traffic.ijs.si ......................................................................................................................... 16
   3.2 Optimum cloud infrastructure ............................................................................................ 16
      3.2.1 Spark on Mesos .............................................................................................................. 17
      3.2.2 Hadoop over YARN ...................................................................................................... 18
4 External Data hub (Running at LPP)......................................................................................... 20
   4.1 Technical details................................................................................................................... 21
Table of Figures

FIGURE 1. SPARK ON MESOS CONFIGURATION ................................................................. 17
FIGURE 2. SPARK ON MESOS INSTALLATION .................................................................. 18
FIGURE 3. HADOOP WITH YARN CONFIGURATION .......................................................... 18
FIGURE 4. HADOOP WITH YARN INSTALLATION ............................................................ 19
FIGURE 5: OPTIMUM DATA-HUB RUNNING INSIDE LPP .................................................. 21
FIGURE 6: API MANAGEMENT TOOL WEB INTERFACE – USER MANAGEMENT .................... 22
FIGURE 7: API MANAGEMENT TOOL WEB INTERFACE – API MANAGEMENT ....................... 23
FIGURE 8: ARCHITECTURE OF ONE OPTIMUM DATA-HUB NODE .................................... 27
FIGURE 9: EXAMPLE ORGANIZATION OF OPTIMUM DATA-NODES ................................. 29
FIGURE 10: CURRENT DATA INFRASTRUCTURE ORGANIZATION ..................................... 32
FIGURE 11: WATCHDOG’S WEB GUI .................................................................................. 36
FIGURE 12: CONNECTING NEW RETRIEVER TO WATCHDOG ......................................... 37
FIGURE 13: TRAFFIC.JS.SI PUBLIC OPTIMUM APIs FOR EASY DATA-ACCESS .................... 40

List of Tables

TABLE 1: SIZE OF THE CURRENTLY COLLECTED DATA-SETS .......................................... 34
TABLE 2: DATA THROUGHPUT OF SOME OF THE MEASURED SENSOR FEEDS THAT ARE BEING COLLECTED (THE RATES AND SIZES ARE APPROXIMATIONS BASED ON SHORT MEASUREMENT – THE PLATFORM DOESN’T HAVE AUTOMATIC MEASUREMENTS YET) .......................... 35
## Definitions, Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AIT</td>
<td>Austrian Institute of Technology</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AM</td>
<td>Adria Mobil</td>
</tr>
<tr>
<td>BCC</td>
<td>Birmingham city council</td>
</tr>
<tr>
<td>DATEX</td>
<td>Data Exchange</td>
</tr>
<tr>
<td>Dx</td>
<td>Deliverable (where x defines the deliverable identification number e.g. D2.1)</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDF</td>
<td>Inverse Document Frequency</td>
</tr>
<tr>
<td>IP</td>
<td>Infraestruturas de Portugal</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transportation Systems</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JSI</td>
<td>Jozef Stefan Institute</td>
</tr>
<tr>
<td>LPP</td>
<td>Ljubljana Public Transport</td>
</tr>
<tr>
<td>LS</td>
<td>Luis Simões</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MQTT</td>
<td>MQ Telemetry Transport</td>
</tr>
<tr>
<td>NER</td>
<td>Named Entity Recognition</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>ODI</td>
<td>Optimum Data Infrastructure</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>REST</td>
<td>REpresentational State Transfer</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>Tfidf</td>
<td>Term Frequency – Inverse Document Frequency</td>
</tr>
<tr>
<td>Tx</td>
<td>Task (where x defines the deliverable identification number e.g. T2.2)</td>
</tr>
<tr>
<td>UoW</td>
<td>University of Wolverhampton</td>
</tr>
<tr>
<td>WPx</td>
<td>Work package (where x defines work package identification number e.g. WP2)</td>
</tr>
</tbody>
</table>
Executive summary

This deliverable presents and describes the approach, organization, achievements, the working live software and hardware, behind the first version of Optimum data infrastructure. It is aligned with the developments in tasks T2.1, T2.2 and T2.3 and serves as a basic cornerstone for further developments in the project, especially the work packages that depend on the data (sensors feeds and other data) for machine learning, analytics and proactive decisions. It builds upon the work described in the D2.1 (Data Infrastructure Architecture and Implementation Plan).

Data availability (real time and historical) is of crucial importance in any ITS system, especially when relying on the state-of the art technologies from the field of AI and machine learning. Without the data, it’s not only near to impossible to guarantee useful and working solutions, but it’s also really hard to even plan and build them. For this reasons, the primary goal of this initial version of the infrastructure was to be able to collect and provide the real-time and historical data to Optimum project, with the secondary goal to do that as efficiently, cost effective and easy as possible.

Optimum project has a diverse set of partners, with diverse set of skills, organizational structures, existing technologies and also interests. For this reason, we tried to avoid solutions that would need consortium partners to vastly change the way they already handle their systems data and people, or impose a need for costly personnel training, or locking into risky technologies and solutions. This was achieved by also diversifying the data-infrastructure with a mix of standard and also novel data storage and manipulation approaches, making it distributed while at the same time imposing a necessary structure to seamlessly work and support the project as a whole.

This deliverable is also serving as a basis for D2.3 (Data Infrastructure for ITS Multi-modal Data and Social Media Streams – Initial version), which runs on the same physical and organizational infrastructure.
1 Introduction

1.1 Objectives of the deliverable

The goal of this deliverable is to present the first version of Optimum data infrastructure for sensor streams (the same infrastructure is used for other modalities – D2.3), which is a live system collecting, cleaning storing, enriching and serving traffic related sensory data.

Collecting good quality of the data and later maintaining the data and also the collection, is an expensive and time consuming process. There are various companies that live solely on the business of acquiring and selling the (traffic related) data. On top of that, after the data is collected, using the data can incur costs as well, especially if this needs special training, custom software and coping with the data outages and unreliability. This can be to some extent solved with a reliable, fast and easy to use infrastructure for data-collection manipulation. In this deliverable (demonstrator report), we explain how the data infrastructure is implemented and how it addresses these issues and make the data of a decent quality readily available to Optimum project with minimal overhead to the developers, data-providers and other Optimum consortium partners.

1.2 Structure of the deliverable

The deliverable is structured in the following way:

- Section 2 provides a list of software tools that are part of the data infrastructure and are incorporated in the running services.
- Section 3 provides the info about the hardware specifications the Optimum Data Infrastructure is running on.
- Section 4 shows the approach to incorporate external data-source into the Optimum Data infrastructure, while solving data legal-issues and not disrupting the company’s everyday business.
- Section 5 describes the general idea, approach and architecture of the current, first version of the Optimum Data Infrastructure
- Section 6 provides the details on the implementation of the initial version of the data infrastructure
- Section 7 wraps up the deliverable and states next improvements and work for the next version
2 Toolset

Optimum data infrastructure is composed from various tools and libraries (internal - developed specifically for Optimum and external - preexisting) that are grouped into the scalable and distributed data infrastructure. While distributed approach allows us scalability, robustness (when service on one server is disrupted, the rest of the infrastructure remains unaffected) and easier implementation (some partners want to have their own infrastructure, or have limited rights for data distribution), we still allow single point of entry for data-access, which makes it much easier to find, access and manipulate.

These tools are briefly described below, and then in more detail in sections 5 and 6, where we present how they are incorporated into the full data-infrastructure, how everything fits together and how it serves Optimum project and the consortium.

Because we want to emphasize the flexibility

2.1 MongoDB

MongoDB\(^1\) is a cross-platform open source database which was developed for scalability and use in web applications. In balanced applications using read-write-modify operations, it can achieve approximately 30.000 operations per second. However, performance is not guaranteed if operations exceed given memory. It uses a document-oriented data model (NoSQL), instead of relational tables. Data is stored in BSON (Binary JSON, a serialization of JSON format with support for embedding and nesting of documents and arrays.) format and organized in collections instead of tables and columns used by SQL databases.

MongoDB also supports a dynamic schema, so modifying a database can be done in an instant and does not need to affect previous records. It is very useful for storing things like text content with various metadata, messages, data about locations, etc.

This is a major advantage when storing data which does not have a rigidly defined structure, though normal organized entities can be presented through MongoDB as well. It was mainly designed for self-contained data without much interaction between different collections (unlike SQL).

2.2 PostgreSQL and PostGIS

PostgreSQL is a cross-platform open source relational database system. It uses tables and columns for organizing records (stored as rows) and relationships between them. It is fully ACID compliant which guarantees reliability of transactions and queries executed on its databases and implements ANSI-SQL: 2008 standard with full support for subqueries, regular expression

\(^1\) Webpage: [https://www.mongodb.com/](https://www.mongodb.com/)
matching and XML support. Database size is virtually unlimited, though there is a maximum of 250 – 1600 columns per table (depends on column type) and 32 TB size limit on a single table.

It’s generally faster for complex queries than NoSQL databases due to rigid structure. However, it’s still slower than some other SQL database systems for simple reads, though it scales better than most.

While many other relational database implementations are available (MySQL, Microsoft SQL, DB2...), we decided on using PostgreSQL due to its support for geospatial data through PostGIS plugin. PostGIS adds implementations for location queries, geographical coordinates, multiple geometry types, spatial operators for determining geospatial measurements such as area, distance, perimeter, etc.

2.3 RDF Store

Resource Description Framework store (RDF store, also called a triplestore) is a purpose-built NoSQL database for storage and retrieval of triples – data entities with three components: subject, predicate and object (example: Alice knows Bob, Alice is 28).

Query languages vary on implementation, though SPARQL 2013 is recognized by W3C as one of the key technologies of the semantic web and recommended solution when using RDF stores. A collections of triples (RDF statements or records) represents a labeled, directed multi-graph. Thus, RDF data-model is naturally suited to research into various patterns (such as road traffic) and online communities (social networks).

Some common implementations of RDF stores include Boca (Java), 4store (C), Dojo Data (JavaScript) and BrightstarDB (C#).

2.4 RabbitMQ and Kafka

Kafka\(^2\) and RabbitMQ\(^3\) are open source publish-subscribe messaging systems. Both use a message broker through which data consumers send and receive messages. There is no direct connection between individual consumers (subscribers).

RabbitMQ uses an implementation of MQTT standard publish-subscribe messaging protocol. It supports asynchronous processing, message persistency as well as routing and filtering functionality. Due to complexity and more features, RabbitMQ generally achieves lower performance thank Kafka (20.000 – 30.000 messages per second).

Kafka implements its own messaging protocol and is generally used when consumers need to transfer huge volume of messages (100.000 per second). However, it lacks certain advanced

\(^2\) Webpage: http://kafka.apache.org/
\(^3\) Webpage: https://www.rabbitmq.com/
features for message routing and broker rules. Those require additional software to be installed.

2.5 Spark
Apache Spark is an open source processing engine, used for machine learning, processing big data and executing complex SQL queries, etc. It provides a unified interface for programming fault-tolerant server clusters (though operation on single server node is possible).

Data is stored in resilient distributed datasets (RDDs), which are distributed over the whole machine cluster and generally kept in memory. RDDs function as a working set for usage by parallel programs which share a limited form of distributed memory.

Researchers have managed to sort 100TB of data in 23 minutes with Spark, three times faster than Hadoop system, 72 minutes. Throughput is dependent on network capability and general cluster performance.

2.6 Hadoop
Apache Hadoop\(^4\) is an open source framework and a collection of libraries that allow developers to implement distributed processing of large data sets across server clusters with high availability and fault tolerance. Data is stored in Hadoop Distributed Filesystem (HDFS). By default, it uses MapReduce system for parallel processing, which runs as a series of jobs – separate Java applications for processing chunks of data.

2.7 QMiner
The main purpose of the QMiner library is to enable building full stack data-mining applications with in a flexible but highly efficient manner. Most building blocks are implemented in C++ for performance and exposed as a Node.js native add-on which glues them together.

2.7.1 Modules for sensor streams
We first present the main architectural aspects and then some concrete applications.

2.7.1.1 Storage layer
Each data unit is referred to as a record (row) and can be stored in a store (table). Records in a given store must be consistent with its schema, which describes the field (columns) and their types. QMiner implements several variants of the storage model, the most notable is the implementation with the assumption that records have integral identifiers which represent contiguous arrays (no holes are allowed). This representation is suitable for time series processing, where one only adds new data and deletes the oldest data (as a queue) which can

be exploited by other components. The storage location of each field can be specified (memory or disk) and several indexes are available (geographic index, B-Tree index, inverse-index).

2.7.1.2 Feature extraction
Feature extractors are components which map from records (which are consistent with a store schema) to linear algebra vectors (sparse/dense) or matrices (sparse/dense). The main interface between the storage layer and the analytics components is through linear algebra structures, enabled by feature extractors. That is, the machine learning interfaces all work with linear algebra inputs. In the context of the current deliverable, the most important are numeric feature extractors that support several types of data normalization (online unit variance and box constraints).

2.7.1.3 Analytic components
The components include offline (batch) and online (incremental) versions of the most popular datamining approaches (for example: support vector classifiers and regressors, neural networks, k-means clustering, nearest neighbor, dimensionality reduction techniques such as PCA, SVD, MDS, etc.) The majority of analytics components are implement in C++ and may be compiled using the OpenBLAS library for highly efficient linear algebra computations.

2.7.1.4 Stream aggregates
Stream analytics is enabled by stream aggregates, which are computational units that react to events from the storage layer. The most typical use case is to trigger some computation every time a new record is added to the data store (similar to database triggers, but with more structured). Stream aggregates implement interfaces and may be chained to form more complex pipelines. For example, an Exponential Moving Average (EMA) stream aggregate computes a smoothed version of an input time series and may be fed into another stream aggregate (for example another EMA aggregate which results in the computation of Double Exponential Moving Average). Several important time series processing functions are implemented as stream aggregates in QMiner. Relevant to building and modifying data mining verticals is the ability to implement stream aggregates purely in Javascript. These aggregates interact seamlessly with the built in C++ stream aggregates.

2.7.1.5 Stream aggregate pipelines
Chained groups of stream aggregates form encapsulated complex computational units. The logic flow of whole processing system can be controlled by using filter stream aggregates which check for certain conditions for each incoming data record and trigger whole chains (which correspond to pipelines) when conditions are met. For example, a single store can hold records with: sensor ID, timestamp and sensor value. Then for each sensor ID X we create a processing pipeline (for example, a linear regressor) and connect it to a filter aggregate that relays records
from sensor X to the corresponding pipeline. This enables one to create a very "flat" family of pipelines whose updating logic is controlled by the filters.

2.7.2 Example Applications

2.7.2.1 Road traffic travel time prediction based on loop sensors
QMiner can be used to simply compose highly efficient traffic prediction verticals (see example\(^5\)). Using efficient NoSQL like storage layer\(^6\) (optimized for time series data) sensor streams are fed into windowed data structures that are integrated with stream processing modules, feature extractors\(^7\) and data mining components. Components especially relevant to sensor prediction are the following:

1. Time series merger maps two time series sampled at different times to two time series sampled at the same time points. This is achieved by using time series interpolation (zero-hold or linear).
2. Time series resampler estimates the time series values at user specified points, most typically at equally spaced intervals (which is needed when data is not equally sampled).
3. Online recursive linear regression involves an iterative updating (learning) procedure integrated with a forgetting strategy that is useful when the dynamics of the time series change over time (for example, daily traffic patterns during the winter are substantially different from the daily patterns during the summer, due to shorter days).
4. Stream aggregate\(^8\) filters are needed when data is fed into a single store with multiple processing pipelines attached.

For example, each sensor may be modelled for several prediction horizons and fed into multiple processing pipelines.

2.7.2.2 Road traffic anomaly detection
QMiner can be used not only for prediction of traffic conditions but also as an anomaly detector, where one continuously observes and models the distribution of sensor measurements and looks for anomalous conditions. The analytics component\(^9\) models the high-dimensional empirical measure based on the data and computes the distance to measure on a sliding window (the distribution can drift in time). The points with high distance to measure are detected as anomalies. By using feature extractors (map records to vectors)\(^10\), coupled with

---

\(^5\) Webpage: [https://github.com/qminer/qminer/blob/master/examples/timeseries/timeseries.js](https://github.com/qminer/qminer/blob/master/examples/timeseries/timeseries.js)


\(^10\) Webpage: [https://rawgit.com/rupnikj/qminer/master/nodedoc/module-qm.html#~FeatureExtractorNumeric](https://rawgit.com/rupnikj/qminer/master/nodedoc/module-qm.html#~FeatureExtractorNumeric)
window buffer stream aggregates\textsuperscript{11}, one can model complex types of anomalies: from point anomalies (a single value is unlikely), sequence anomalies (a series of values is unlikely, even though each value can appear to be normal. This is achieved by using time-delayed embeddings of time series) and long term distribution shifts (for example, detecting that the road conditions on a particular segment are slowly degrading). Efficient modelling of distribution changes is based on sketching algorithms\textsuperscript{12} implemented as stream aggregates, which track distribution statistics with low computational and memory requirements.

2.7.2.3 \textit{QMiner for mobility patterns}

The stream aggregate framework is suitable for efficient implementations of (SPD), frequent location extraction (FLE) and state sequence modelling (SSM). SPD can be implemented as a filtering stream aggregate that processes GPS coordinates and outputs a stream of labels (stay point candidate, finalized stay point, path point). This is fed into the frequent location component, which performs online clustering and outputs a stream of labels describing the status (new location added, location updated, locations merged). Modelling frequent location sequences (paths) involves tracking and updating statistics of state traversal, suitable to the stream aggregate pipelines in QMiner.

2.8 \textbf{API Management Console}

API Management Console is a tool for managing access to API endpoints for end users, which has been developed internally at JSI. The tool supports integrated APIs as well as adding external API endpoints and controlling access to them. It presents a graphical web interface for managing users and APIs. User authentication and access control are provided by security tokens, passed in headers of API requests. It also supports authentication through HTTP GET API call through which external APIs can authenticate their own users.

2.9 \textbf{Data Retrievers}

In order to provide expected results Optimum relies heavily on data sources. For that reason, data collection represents an important activity of our project. The data sources that we identified are numerous and correspond to different countries, data types and low- to highly-frequent data. Each data source has its own data retriever, which collects, parses and stores the data in file or database to be used by Optimum platform. The existing retrievers are described in section 6.1 and 6.2. For some of the working retrievers, the connection with the watchdog is not done yet (the primary focus was to get the data), but is work in progress and will be finished in a weeks following this report deliverable. The same is with some of the possible new data-sets and retrievers for the second version of the data-infrastructure.

\textsuperscript{11} Webpage: https://rawgit.com/rupnikj/qminer/master/nodedoc/module-qm.html#StreamAggregateTimeSeriesWindowVector
\textsuperscript{12} Webpage: https://github.com/tdunning/t-digest/blob/master/docs/t-digest-paper histo.pdf
2.10 **Mobile Collection Libraries**

Besides the data available from other sources and retrieved with the tool described above (section 2.9), Optimum project will have its own mobile application, which can also be used to retrieve useful and needed information about the users and also their surroundings. For this, we are developing mobile software libraries that are optimized to efficiently collect GPS, accelerometer and other sensor data, while not consuming a lot of the battery. These are Android and iOS libraries that can be readily incorporated into any other mobile application, wanting to use some of the Optimum functionality. Besides just collecting the data, the libraries are ready for the edge processing, which means, that some of the data-analysis such us personal mobility patterns can be already done on locally on the phone, and thus releasing the burden from the servers.

2.11 **Edge Processing (computing)**

While edge processing is not strictly a software toolset, but more a paradigm, we still describe it here. Basically it means that the processing and analysis of the data is moved to the sources of the data, and thus reduces the network traffic and central server loads. It serves in Optimum project as an important approach which allows us to get some otherwise unobtainable information (for example from inside the motorhome), interact with users and helps to save the battery and data usage in our collection apps, which already do some processing of the collected data and thus send less requests to the servers (section 2.10).

2.12 **Watchdog**

Watchdog is a server developed internally at JSI with the purpose of having an overview of the numerous data retrievers. When necessary, it (re)starts a retriever and keeps basic log of the retrievers’ activities in one place; significantly lowering maintenance time and complexity of retrieving data from multiple sources.

2.13 **Pingdom**

Pingdom\(^{13}\) is a tool for monitoring webpages and web applications by tracking uptime, downtime and other specific performance metrics, such as page load time and performance over various browser. It works as an external service, meaning it monitors a webpage or a server from outside of the customers’ networks and using multiple probe servers for confirming service status.

It also provides status logs and rudimentary cause analysis based on gathered data and alerts user by email, SMS, phone notification or through various business chat software solutions.

\(^{13}\) Webpage: [https://www.pingdom.com/](https://www.pingdom.com/)
3 Hardware

Stable data collection, storage and retrieval is a crucial component of Optimum platform and traffic related research. Besides the software support, the hardware infrastructure must be powerful enough to be able to serve the data-processing needs of the project. For this reasons, we have two bigger hardware set-ups, one for development and deployment and another for deployment of cloud big data solutions. Both hardware set-ups are already online, being used for collecting and serving traffic related data to Optimum.

3.1 Traffic.ijs.si

For the purpose of development, testing the distributed approach of the infrastructure, collecting and serving the data to Optimum, two physical servers was set-up locally at JSI. They are exposed to Optimum partners and public under traffic.ijs.si and promet.ijs.si domains.

It consists of two machines, one serving a reverse proxy, and another high performance machine as the main data-server.

The reverse proxy set-up allows us to bring together various services that could in theory be scattered around the world, or running on many internal machines, under one unified domain. This simplifies the access for Optimum platform users, since all the services are available through one point of entry (traffic.ijs.si).

The other server is dedicated to running all the required tools and is the main entry point for Optimum users’ data needs. As this is a high performance machine, we are planning to use it for big data processing tools which will be coming into consideration in the later stages of the project (alongside Optimum Cloud infrastructure which consists of many smaller virtual machines and is thus better prepared for distributed computing).

Hardware being used:

- CPU: Intel® Xeon® Processor E5-2667 v2(25M Cache, 3.30 GHz) 8 cores
- RAM: 256GB DDR3 ECC
- HDD: 1TB SSD + 1TB(SYS) + 2x4TB(RAID1) = 10 TB

3.2 Optimum cloud infrastructure

In the context of OPTIMUM project, all data infrastructure and architecture plans are prepared to include big-data tools (Hadoop\textsuperscript{14}, Spark\textsuperscript{15}). The OPTIMUM cloud infrastructure comprises the two following installations:

\textsuperscript{14} http://hadoop.apache.org/
\textsuperscript{15} http://spark.apache.org/
- Spark on Mesos\textsuperscript{16} and
- Hadoop over YARN\textsuperscript{17}.

The access to these big-data tools is restricted to the OPTIMUM partners for security reasons. In order to gain access please consult the consortium partners.

### 3.2.1 Spark on Mesos

The first available installation is Spark on Mesos, in which Mesos is used for cluster management (see Figure 1).

![Spark on Mesos configuration](image)

Figure 1. Spark on Mesos configuration

There are four available nodes, specifically:

- 1 master/slave node and
- 3 slave nodes.

This installation is accessible at: [http://mesosoptimum.euprojects.net](http://mesosoptimum.euprojects.net) (see Figure 2).

\footnotesize
\textsuperscript{16} [http://mesos.apache.org/](http://mesos.apache.org/)

\textsuperscript{17} [http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html](http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html)
3.2.2 Hadoop over YARN
The second available installation is Hadoop with YARN, in which YARN is used for cluster management (see Figure 3).

- There are four available nodes, specifically:
- 1 master node and
- 3 slave nodes.

This installation is accessible at: http://bigdataoptimum.euprojects.net (see Figure 4).

Figure 4. Hadoop with YARN installation
4 External Data hub (Running at LPP)

One of the more important chunks of the work, which also influenced the decisions around the architecture for the data infrastructure services, is the Data and API Management tool that was initially built to be able to access internal LPP Data and then extended to be generally applicable.

LPP is a use-case partner handling public transport in the city of Ljubljana. They are operating with approx. 250 busses, 897 bus stations, 32 bus lines (460km), covering all Ljubljana and extending to other nearby cities. They operate with around 40 million travelers yearly. In order to be able to effectively get the data about the bus transport, without interrupting their normal activities, additional supporting infrastructure was needed.

The problem they faced was a lack of control over data access – it could be either open to the public or not. From the previous experiences, open access exposed them to server overloads resulting in the business interruption and potential personal information leaks. This is the problem which applies to many companies operating with useful traffic data, so the solution we implemented is applicable and repeatable for various possible future data-sets where real-time acquisition would be desirable or needed.

The solution was to install one of the Optimum data-hub virtual machines inside their data headquarters. This data-hub replicates their internal database in real time, preventing malicious behaviour and system overloads from affecting their main system and database. This replicated database is then used for Optimum and other public data needs through additional layer of security and management (API Management Console), which allows LPP company to fine control the access to their data (see Figure 5).
In this API Management tool installed on LPP server we added an Optimum user, which has access to all relevant data sources needed for the project. The Optimum user is then used to access the LPP data through other Optimum Data hubs (Figure 9).

### 4.1 Technical details

The API management tool is composed from multiple components. There is an internal MongoDB database for storing user and API endpoint data, graphical interface for managing users and APIs as well as external API handling with two distinct modes of authentication.

#### 4.1.1 Graphical Web Interface

The service uses an online interface, accessible through login screen, which allows the administrator to manage users and APIs.

The user management section (See Figure 6) provides a form for adding new users, as well as a list of users, their access rights and other information. Through this interface, we can also delete users, check their authentication tokens and select which APIs they have access to.
Figure 6: API Management tool web interface – User management

API management section (See Figure 7) provides a form for adding APIs from external services to master access control list. User must specify full API URL and API service hostname, as well as enter a project name which is an internal designation in our API management tool, used for generating dynamic API endpoint URLs. Optionally, API notes may be added, such as description of parameters or explanation of returned results.
As seen in the bottom panel of Figure 7, users can view the list of APIs entered into the system. Here, it is also possible to delete APIs, check notes or edit API properties, such as URLs and project names on a case by case basis. Each API status can be set to Private, meaning end users need authentication token and appropriate access rights to call this API, or Public, allowing anonymous users without tokens access to this endpoint as well. APIs can also be fully disabled, for example when underlying service is in maintenance mode or unexpectedly unavailable.

### 4.1.2 User Authentication

Upon being entered into the database in web interface, each user is issued a uniquely generated authentication token (This token can be re-generated through web interface by administrator).

Two possible means of authenticating are possible. Primary method requires users to pass their authentication token to server in HTTP request header, while making a request to our service, using an API URL generated by concatenating hostname of our service and URL of the actual API. Upon receiving the request, server checks passed security token against database entries and then forwards the request along with appropriate parameters to requested external API.
service. If API endpoint is public, no token is needed. In that case, server will automatically forward request to external API service. This method allows for actual API services to live isolated on internal networks, so users never actually come into direct contact with them.

Secondary method does not actually route requests through our service, but allows external APIs to set up their own systems and upon users’ HTTP requests call a special /authenticate API, passing authentication token and API name as parameters. In this case, our tool will query the database for the corresponding API and check whether this token can actually access it. The result of this check is sent back to caller as a Boolean value specifying whether user has been authenticated for this API or not.

4.1.3 API Handling
Current implementation supports handling of internal and external APIs. Internal APIs are part of the service deployment and are authenticated in Javascript middleware function upon being requested.

External APIs are added through aforementioned web interface. Each database record stores API URL without hostname, API’s service hostname, arbitrary project name and various status values. Upon receiving a request for an external API endpoint, server first checks for a corresponding API in the database.

Example of dynamic external API URL construction:

Example call: [http://traffic.ijs.si/API/opendata/promet/events/](http://traffic.ijs.si/API/opendata/promet/events/)

Hostname of our server: [http://traffic.ijs.si/API/](http://traffic.ijs.si/API/)

Project name: opendata

API URL: /promet/events/

The matching service first removes our hostname string and separates remaining URL string into project name, API URL and optional URL parameters. Then, a MongoDB collection if queried for matching APIs with corresponding project name and URL. If such API is found, user’s authentication token is checked against another database collection. If the user has appropriate rights, server constructs a new HTTP request for forwarding to the actual API service. In this process, authentication token is stripped from HTTP header, while other URL parameters and request headers are bundled into the new request. Result of this forwarded request is then returned to the original caller.
5 Architecture

The main purpose of Optimum Data Infrastructure (ODI) is to reliably and in a simple way provide relevant and up-to-date traffic-related data feeds to relevant services and work packages inside Optimum project, while making the architecture general enough to be able to address also the specific requirements of individual partners. Optimum project is quite big, consisting of 19 partners, 5 of them use-case partners with internal traffic-related data and requirements. On top of that, the project requires additional external data sources.

We identified two sets of challenges that need to be addressed by the data infrastructure. First group are the challenges that needed to be immediately addressed and are hard requirements for even the first infrastructure to work (Current Requirements/Challenges). The second group are the challenges that we will run into at later stages of the project, but we still need to plan ahead, to be able to address them without the huge re-workings of the system (Later Stage Requirements/Challenges).

Current Challenges:

- ODI needs to provide easy access to the data for all WPs and partners: We are addressing this by introducing a central and unified way of accessing the data, without actually centralizing anything. This is achieved by API forwards (API Management console, section 6.5.1)
- ODI needs to address specific data-provider requirements, such us fine control over what is shared and over security and server loads. This proved to be crucial point for the data-acquisition which is allowing us to obtain additional data-sources.
- ODI needs to take into the consideration the complexity of project structure and adversity of consortium partners. Some partners have pre-existing technologies, services, tools, which need to be easily incorporated and cannot be changed, otherwise lots of the project money would be spent only on adapting the existing systems and procedures. Also partners have diverse human capital, knowledge and internal organizational procedures and systems. This all need to be taken into the consideration. We believe this is the most important challenge, which we tried to address in a way, that we allow the infrastructure to be de-centralized and can run separate parts inside specific partners existing machines, while it is still available through the central API. Like in the case of LPP. This challenge is again solved through de-centralized system, with one simple point of entry, while not closing other ways of using it. Figure 9).
- ODI data provision needs to be stable and robust, with as little of downtime as possible. This is addressed by introducing external ‘watchdog’ service – Pingdom (section 15), which monitors activities of individual APIs and data-feeds and immediately reports outages or errors. Also, if the main API access goes down, the de-centralized nature of
the systems still allows the direct access. Additionally, to the external watchdog service, we also implemented the internal watchdog, which is monitoring all the data-retrievers, restarting them when necessary or reporting to the external watchdog when the problem is unsolvable and needs human attention.

- ODI needs to provide some historical windows of the data, for successful analysis and training of the statistical and machine learning methods. This is solved by introducing data-retrievers, which are accepting (in the case of edge processing) and also periodically retrieving all relevant data-sources, keeping a raw history of the data and passing the retrieved data to the cleaner and indexing modules (often implemented already inside retriever).

- ODI needs to be able to cope with the various modalities of the data. Various types of sensor streams, text and also video and pictures (see the details in D1.3, D2.1 and D2.3). This is done by allowing and combining various storage technologies (SQL, NoSQL, Machine Learning platforms).

- ODI needs to provide a standardized data-access for specific data types/modalities. For this we agreed to comply with the DATEX II standard (See D2.1, D1.3.), and extending it with well documented extensions when necessary.

Later Stage Requirements:

- In the later stages of the project, when the data-loads and users will most likely increase, the current hardware and software infrastructure might not be able to cope with the data-load and processing complexity.
  - For this reasons the Optimum Cloud Infrastructure is set-up for big-data tools, which can be readily employed by the processing algorithms. These set-ups include Hadoop, Spark, on top of Mongo DB.
  - Additionally, the de-centralized architecture reduces the data loads on one machine – the load tends to be more evenly distributed across all the nodes.
  - We are reducing the load with edge-processing
  - Additionally, once all the use-cases and data pathways (pipelines) will be known, we will be able to optimize these and also provide publish subscribe model for crucial real-time services (See D2.1 for details).

- During the course of the project, it will become clearer, which data sources and data nodes are more crucial. Also, there will be a high chance that some of the nodes will not function properly. This will be handled by moving the affected nodes more towards the central infrastructure (JSI, Intrasoft servers), where it will be managed directly by WP2 or Optimum consortium as opposed to unreliable external partner.

- Reliable backup – when the infrastructure setup will become too complicated to quickly replicate, or bring back after the hardware failure. From the current measures we add
approximately 550 GB of new data/month (a lot of this is from multi-modal data such as tweets, weather and traffic cameras – D3.2), which will only grow through bringing in new data-sets, acquiring users, etc. This is projected to be at least 15TB by the end of the project.

- In later stages of the project, it is often crucial to be very quick and easy to deploy a new data-set or functionality. For this reason, we plan to provide the default Optimum Data-hub node wrapped as a VM instance.

5.1 **Optimum data hub**

Even with the Data Infrastructure being decentralized, all of the nodes needs to be compatible (at least the inputs and outputs) and are following the same internal structure and logical components (Optimum data-hub) as depicted on Figure 8. On the figure we can also see a depicted data-flow, which goes from left (inputs) to the right (OPTIMUM API outputs).

![Figure 8: Architecture of one Optimum data-hub node](image)

Optimum data-hub node (as depicted on Figure 8 above), needs at least API Management and Internal Watchdog tools, to be incorporated into the general Optimum Infrastructure. Additionally to the required tools, it can consists of optional components from the toolset.
(Chapter 2), depending on the functionality and implementation of the node. While we are encouraging the toolsets (especially the tools implemented specifically for Optimum), prepared for use in Optimum project, we are not limiting the nodes if they need any of other software not available in the standard set of tools or a custom implementation of existing ones.

**Data Retrievers**

A standard Optimum data-hub node usually consist of one or more data-retrievers, which are able to pull or accept data from the external data-sources and are constantly updating internal watchdog about their status and the number of records obtained.

**RAW Storage**

Data Retrievers that are currently available in the Optimum repository also dump all the retrieved data into the raw files, which can be used as a backup or later analysis if we decide that the indexed storage needs some additional data not used before.

**Cleaning and harmonization module**

This is a logical module. Sometimes it is implemented inside retrievers and sometime as a separate component, using some of the Optimum Machine Learning tools such as QMiner. The task of this module is to filter out the obvious mistakes in the data, and converts the data into proper format, so it is aligned with the rest of the Optimum structures and can be indexed.

**Indexed Storage**

This module is also logical, since it can be implemented using any of the available data storage tools (MongoDB, PostgreSQL, QMiner ...). It serves as the main storage for the later clean data retrieval through the API.

**Enrichment**

This logical module is also completely dependent on the end goal, and the type of the data. This module can be connected directly to the Cleaning and Harmonization module, or ca be triggered by the insert into the Indexed storage, where it pulls out the data enriches it with additional information and then inserts the data back. The enrichment tools are mostly done on top of QMiner (or Rapid Miner) machine learning tool.

**API, API Management**

This tools allows an easy API management and a standardized way to open the Optimum data-hub to the external world through a web service API. This tools is described in more details in chapters 2.8, 4.1.3 and 6.5.1. Although this is a requirement for any of the data-hub nodes, it can be replaced by custom solution if necessary.
Watchdog, Pingdom

This is also one of the required modules, which makes sure that the data-hub node works well, has updated data and is responding to the external requests. Once any of the internal modules are registered to this module, they need to constantly provide updates, otherwise the watchdog tries to restart them, or reports to Pingdom, which is external watchdog. Similarly, as internal modules need to report to the Watchdog, Watchdog itself is constantly reporting status of the node to the external Pingdom service. This way developers and Optimum data infrastructure managers are immediately notified when something goes wrong in any of the nodes.

5.2 Connecting it all together

As we see in the chapter 5.1, data-hub nodes can be really simple, consisting of just the API module serving some data, or single retriever, data-base and the API. Or they can be complex and big machines, having hundreds of retrievers with complex enrichment and cleaning pipelines. The important thing is that all this is combined through the combinations of API Management consoles, so a completely remote node is able to serve the data of another node, without almost any additional work and inducing minimal latency (Figure 9).

![Figure 9: Example organization of Optimum data-nodes](image-url)
access point if needed. As we see on the example Figure 9, a partner can quickly find all available information and API’s through main API node, but if it needs a faster, no latency access for something that’s available through node 0, nothing blocks the partner to connect directly to node 0 (if he has the access permission given by the node owner).
6 Implementation

To be able to show the flexibility and interoperability of the infrastructure set-up, we presented (section 2) the most common tools (but not limited to only these) that are available and supported inside the infrastructure set-up, and showed how they can be used and linked together. While this gives a general overview and shows the rich set of options, at this time we already know what and where will be used for most of the infrastructure:

- **File based storage** is used for backup and re-play in case of the need
- **MongoDB** is used for fast data storage and historical retrieval on all data-feeds except floating car data
- **PostgreSQL** is used for geo-spatial indexing (floating car data, personal patterns)
- **QMiner (C++ with Node.js interfaces)** is used for almost all real-time modelling, predictions and patterns detection, except for proprietary algorithms of some partners. These are properly integrated as well
- **Retriever and collector** modules are **Java (90%), Node.js (5%) and Python (5%).**
- **Watchdogs** are Java Tomcat Servlets
- **Mobile Libraries** are native (Java on Android, Objective C on iOS)
- **Edge processing** is Java, C and C++ for Motorhomes, native for phones. Motorhomes have custom edge processing hardware.
- **API Management console** is a node.js applications backed-up with Mongo and pub-sub mechanism
- **Publish-subscribe** mechanism is RabbitMQ for the main infrastructure and MQTT for edge processing.
- **Hadoop/spark** cluster is running on the main (Intrasoft) servers. At the moment of writing the deliverable it is being used for testing, since the overhead of Hadoop is still bigger than the benefits.

At the time of the writing the main parts and nodes of the infrastructure are already running and being able to serve the data. We structured the nodes in a way that best serve the specific data-providers and developers maintaining the nodes and data-retrievers. It made most sense that each technical partner takes care for the data from the use-case provider from the same country.

The infrastructure is distributed in the following way:

- Main server cluster and one point of entry is at Intrasoft premises
- One big server (8 CPU, 256 GB of RAM) is at JSI, where it collects Slovenian real-time data. It is set up to forward requests to Intrasoft cluster when needed.
- External node with an API Management instance is running on a machine inside LPP, shielding and replicating their main database.
- Other nodes in Uninova and UoW are collecting and serving tweets and UK, AT and PT data.
- Mobile Collection libraries are integrated in Optimum mobile app.
- Edge processing modules are running inside Adria motorhomes and sending data to JSI node.

This way there is less latency, and easier communication overhead while setting up the system.

Figure 10: Current Data Infrastructure organization

Each of these country-dependent nodes is then connected through the API Management Console to the main API access (one point of entry), which makes it much easier for the developers to use and find the data. Optimum users who want to reduce the latency and have access rights, can still access the data directly from the source nodes, or even the source, if there is a need. But in this case the data is not cleaned, standardized and aligned with the rest.

6.1 Collecting Sensor Data Feeds

The work of the infrastructure started at the beginning of the project. Because the project is highly focused on the research and novel approaches to the traffic related analysis through the
use of big-data technologies artificial intelligence and machine learning, data is the most important thing. This is especially critical for the machine learning algorithms, which can ever only work on data of a decent quality. For this reasons our primary task from the beginning was to start collecting as many data as possible and as soon as possible. This was achieved, since we have a historical data set of almost all the crucial data-sets since January 2016, and for some of the data-sets (Slovenian open-data), even from the beginning of the project.

Within the consortium we came to an agreement that technical partners will provide technical support to corresponding pilots from the same country. Meaning, that Vienna, West Midlands, Portugal and Slovenian pilots are supported by AIT, UoW, UNINOVA and JSI, respectively. Data collection is one of the basic tasks performed by technical partners in that perspective. Since the data sources were already thoroughly described in D1.3 and D2.1, we only provide a short overview for each pilot below.

**West Midlands**

Birmingham city council is providing sensor readings of flows, average speeds, congestions, occupancies, travel times, incidents, parking, and road works through REST calls. This data is stored in central mongo DB repository.

Traffic event data from National Traffic Information Service is available in DATEX II standard form. This data is also stored in mongo DB repository.

**Vienna**

The data available for this pilot covers rentable bicycles locations data, shared cars locations, travel time data from taxi trips, car parks, public transport routes and public transport stops and also weather. Raw data from taxi trips is currently being stored locally in Vienna for privacy reasons; aggregated data (real-time and 10-15 years of historic data), however, is available to whole Optimum consortium.

**Slovenia**

There are 2 pilot cases in Slovenia. Some of the data collected is specific to each of pilots (e.g. bus schedules, motorhome coordinates), whether some data sources relate to both pilots (e.g. traffic events).

LPP provides bus data which is already described in Section 0.

AdriaMobil also provided GPS data of 10 motorhomes with custom built hardware, detailed sensor data of one motorhome and OBD data (CO2 emission, average consumption, average speed per trip etc.) of 10 motorhomes. We are collecting the data and sending GPS locations directly to mobility patterns service (traffic.ijs.si – Enrichment), whereas the rest is stored either
in PostgreSQL or raw format. Additionally, one motorhome is running custom software that is able to run analytics on the internal sensor values and thus do a lot of processing there (edge processing), before the already processed data comes into the Optimum Data Infrastructure.

There is also a lot of traffic-related open source data available for both Slovenian pilots that we collect: loops sensors, traffic events, wind measurements, parking spaces, weather data and shared bicycle data. This data is being collected since the beginning of the project, so we also have data for training of the models for at least a year.

**Portugal**

IP holds available highway counter which contain information about vehicle passages, road occupancy and average speeds. This data is delivered in two main ways: by Secure FTP, in CSV or XLSX formats and via SQL dumps. A data collection and harmonization adaptor was already implemented, in R language for testing purposes. The data is stored in mongo DB repository. Similarly, toll counters’ data by the different concession holders count vehicle passages for classes 1 to 5. Historical and present data is available.

Traffic event data is available for IP’s infrastructure from 2011. Historic data can be retrieved as SQL dumps, whereas daily updates are available via SOAP WEB service. After harmonization, the data is stored in mongo DB.

LS provided truck events and consumption historical data of their fleet. The data is delivered in CSV files and stored in mongo DB.

Data harmonization that is performed will be described in detail in D2.4 Data Harmonization.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portugal Highway counters</td>
<td>10.572</td>
</tr>
<tr>
<td>Slovenian general traffic (loop sensors, parking, wind, bicycle, events, …)</td>
<td>290GB</td>
</tr>
<tr>
<td>Adria Motorhome sensors data</td>
<td>36.49GB</td>
</tr>
<tr>
<td>Birmingham Traffic measurements</td>
<td>55.9GB</td>
</tr>
<tr>
<td>Optimum general DB (the rest of the data-sets)</td>
<td>527GB</td>
</tr>
<tr>
<td>Personal Mobility Patterns (raw GPS from users, trucks, motorhomes)</td>
<td>13.2GB</td>
</tr>
</tbody>
</table>

*Table 1: Size of the currently collected data-sets*
### Table 2: Data throughput of some of the measured sensor feeds that are being collected (the rates and sizes are approximations based on short measurement – the platform doesn’t have automatic measurements yet).

<table>
<thead>
<tr>
<th>Name of the data-set</th>
<th>Records/h</th>
<th>size: MB/h</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parking sensors</td>
<td>1440</td>
<td>0.28</td>
</tr>
<tr>
<td>Loop counter sensors (LJ)</td>
<td>44640</td>
<td>6.4</td>
</tr>
<tr>
<td>Loop and toll sensors (PT)</td>
<td>3240</td>
<td>0.47</td>
</tr>
<tr>
<td>Wind sensors</td>
<td>418</td>
<td>0.018</td>
</tr>
<tr>
<td>Smart Motorhome sensors</td>
<td>86,160</td>
<td>1.72</td>
</tr>
<tr>
<td>Motorhome coordinates</td>
<td>540</td>
<td>0.756</td>
</tr>
<tr>
<td>Collection libraries (per user)</td>
<td>120</td>
<td>0.168</td>
</tr>
<tr>
<td>Turners Trucks coordinates</td>
<td>9600</td>
<td>3.63</td>
</tr>
</tbody>
</table>

6.2 Watchdog and maintenance

Data retrievers can fail for different reasons - sometimes host machines would shut down because of power failures or restart to install some updates and consequently shut down the retriever; there could be errors when retriever is trying to connect to the data providers, the data could be corrupted etc. In some cases, these failures are noticed immediately and appropriate measures are taken, whereas in some situations it takes a while before anyone notices that the data is not being retrieved anymore.

Since we’re dealing with a large number of data retriever in this project, the maintenance time and complexity have risen to a level, where we recognized the need for a watchdog service. The idea of it watchdog is to have an overview of all the data retrievers, as well as have it respond to retriever failures by trying to fix it or alerting the user if the problem persists.

The watchdog that we developed is implemented in Java and runs on Apache Tomcat server. Each retriever has to ping the watchdog to signal that it’s alive. If for whatever reason the ping is not received for a while the retriever will be restarted. Each retriever also sends data to the watchdog about current status: exceptions, last data update and the number of objects received. This information can be viewed in a web GUI (Figure 11). The settings of the retrievers written in Java can also be changed through the web interface. In addition, the watchdog is connected to Pingdom service which sends alerts when watchdog signals that one the retrievers is down. This initial version of the proposed watchdog was already described in D2.1.
Since then we made some improvements in the web GUI. Now it is possible to add retrievers to
the watchdog through the GUI (bottom left corner in Figure 11): only a runnable file (the
retriever) and two batch files (one that starts the retriever and one that shuts down the
retriever) are needed to connect a retriever to the watchdog. The upload web interface is
presented in Figure 12. We also added a colored circle to each of the retrievers in the web GUI.
Green circle means that the retriever is up and collecting the data; yellow circle means that the
retriever is up but there is a problem with data retrieval and a red circle signals that the
retriever is down. We also added buttons that allow us to restart or stop a retriever from the
browser.

The watchdog neither cares in which programming language the retriever is written not it cares
about the data type collected – the data is not necessarily a sensor stream; it can also be a
social media stream (e.g. tweets), photo or other data type.

Currently the internal watchdog for traffic.ijs.si overviews 8 retrievers, which cover GPS
coordinates, weather, parking spaces, traffic events, wind, motorhomes and bike sharing
system data.

<table>
<thead>
<tr>
<th>Retriever</th>
<th>Status</th>
<th>Connection</th>
<th>Objects received</th>
<th>Last data update</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordsRetriever</td>
<td></td>
<td></td>
<td>561</td>
<td>07/25/2016 19:28:56</td>
</tr>
<tr>
<td>CamerasRetriever</td>
<td></td>
<td></td>
<td>277</td>
<td>07/25/2016 06:05:16</td>
</tr>
<tr>
<td>WeatherRetriever</td>
<td></td>
<td></td>
<td>1</td>
<td>07/25/2016 19:46:47</td>
</tr>
<tr>
<td>ParkingsRetriever</td>
<td></td>
<td></td>
<td>24</td>
<td>07/25/2016 19:44:32</td>
</tr>
<tr>
<td>CountersRetriever</td>
<td></td>
<td></td>
<td>744</td>
<td>07/25/2016 17:36:17</td>
</tr>
<tr>
<td>TrafficEventsRetriever</td>
<td></td>
<td></td>
<td>0</td>
<td>07/25/2016 19:45:17</td>
</tr>
<tr>
<td>BurjaRetriever</td>
<td></td>
<td></td>
<td>8</td>
<td>07/25/2016 17:44:51</td>
</tr>
<tr>
<td>MotorhomeStatusRetriever</td>
<td></td>
<td></td>
<td>9</td>
<td>07/25/2016 19:49:16</td>
</tr>
<tr>
<td>AdriaRetriever</td>
<td></td>
<td></td>
<td></td>
<td>01/01/1970 01:00:00</td>
</tr>
<tr>
<td>4ThOfficeMongoDb</td>
<td></td>
<td></td>
<td>0</td>
<td>01/01/1970 01:00:00</td>
</tr>
<tr>
<td>BicikeljRetriever</td>
<td></td>
<td></td>
<td>35</td>
<td>01/18/1970 01:11:01</td>
</tr>
</tbody>
</table>

**Figure 11: Watchdog’s web GUI.**
Source code is available in Optimum project’s central repository on GitHub\(^\text{18}\). Description and documentation are also available there.

6.3 Storage

As mentioned at the beginning of the chapter 0, storage is the second most important thing after the collection that is done when introducing a new data-set or data-hub node.

6.3.1 First stage storage

To be able to start collecting the history immediately, even before the data-structures for indexing the data are known, the retriever saves each response it gets into a raw file. This simple rule, allow us to start collecting the historical data almost immediately, when the rest of the pipeline is not finished yet. Additionally, even after all the workflow inside a specific data-hub node is done, we sometime need to re-import the data or add additional, previously unused field.

6.3.2 Second stage storage and Indexing

After the data is retrieved and cleaned, it’s usually (in all the data-hub nodes) indexed and stored in to a database, from where can be easily retrieved and explored. All the nodes are using at least one (if not all) of the data storage tools described in Section 2 (QMiner, MongoDB, PostgreSQL).

This indeed access is also used by the API Manager, to be able to serve the data outside of the node.

6.4 Pre-Processing and Enrichment

For some of the data-sources, the raw data cannot be immediately useful by the Optimum users, or it can be improved with merging with another data, or calculation of trends, clusters or some other analysis. For example, RAW GPS coordinates can be clustered on the fly, merged with the underlying road network and POI data, to detect user’s staypoints (name, address),

---

which roads he took to come there and how long he stayed there (this will be described in more detail in Online Stream Analytics Functions D3.5). Or, raw GPS data can be merged with the phone accelerometer sensors and bus Schedule, to detect when user is driving, and whether he is driving on the public transport, or not.

While this is more the domain of the WP3 and deliverable D2.5, we still mention it here, since a lot of this enrichment and analysis can happen (and often must) in real time as the data is coming in. For this reason, the data infrastructure must support this kind of real-time plug-ins or enrichment modules. The architecture for that is depicted as a green Enrichment module on Figure 8.

Besides cleaning of some data-sets and enrichment of raw GPS coordinates coming into the infrastructure, which is already working inside the data-hub nodes, there are many machine learning and raw data-processing methods inside QMiner platform (described in Section 2.7).

6.5 **Easy Access for Optimum Consortium**

As already mentioned, Optimum Data Infrastructure allows a ‘one point of entry’ access, even though it is a highly distributed system. This vastly improves the learning time developers need to start using the data. On the other hand, the system is distributed, not blocking any Optimum or even external partner to implement part of it inside their own system and on attach it to the system over the web service and through the API Management tool. This increases the reach of the platform and data availability and reduces the burden on the companies, which do not need to learn new tools, move the data to central location or do big changes in their existing systems and workflows. Additionally, it is extremely easy to add a new hub to the system and thus expand the infrastructure to gain access to new data, new domain, or gain additional processing capabilities.

6.5.1 **API Access and API Management Console**

The main tool for ‘one point of access, while allowing the system to be distributed and flexible, is the API Access and Management tool described already in D2.1, D6.1 and here in Sections 2.8 and 4.1).

Besides serving the internal data-hub node web service APIs, it is really easy to add external APIs. The requests coming to these external APIs, which can also be external in a sense of another Optimum data-hub node (or completely external web service), are then just forwarded to the proper destination. In this sense the API Console serves just as a forwarding mechanism, while adding an additional layer of control and security.

Currently we have a complete set of APIs open on traffic.ijs.si machine and are working on publishing the extended list (covering all the APIs) inside the Optimum Cloud Infrastructure cluster by the end of the month.
In Figure 13 is a screenshot of API management Console panel which lists all available APIs and provides API handling: there, each endpoint has options to be set as public and/or active, it can be deleted, it has information about the project it originally belongs to and has additional field for notes.

Example call to http://traffic.ijs.si/API/info/getApiInfo which provides a list of all available APIs is presented in Annex 1.
### API Access Controls

<table>
<thead>
<tr>
<th>Edit</th>
<th>API Route</th>
<th>Public</th>
<th>Active</th>
<th>Project</th>
<th>Notes</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/info/getApiInfo</td>
<td>Yes</td>
<td>Enabled</td>
<td>Internal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/routes/getRouteDetails</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/routes/getRouteGroups</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/routes/getRoutes</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/routes/getStationsOnRoute</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/stations/getRoutesOnStation</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/stations/getStationById</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/timetables/getArrivalsOnStation</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LPP/timetables/getRouteDepartures</td>
<td>Yes</td>
<td>Enabled</td>
<td>LPP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/bikeifel/list/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/burja/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/burjkazik/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/cameras/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/countors/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/events/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/promet/parkirocka/pk/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>opendata/vreme/report/</td>
<td>Yes</td>
<td>Enabled</td>
<td>opendata</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/backup</td>
<td>No</td>
<td>Disabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/evaluations</td>
<td>Yes</td>
<td>Enabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/evaluations/id</td>
<td>No</td>
<td>Disabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/get-stores-list</td>
<td>Yes</td>
<td>Enabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/get-stores-recs/store</td>
<td>No</td>
<td>Disabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/routes</td>
<td>Yes</td>
<td>Enabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/traffic-predictions</td>
<td>Yes</td>
<td>Enabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/traffic-predictions/id</td>
<td>No</td>
<td>Disabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/traffic-predictions/add</td>
<td>No</td>
<td>Disabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>traffic/traffic-predictions/get-sensors</td>
<td>Yes</td>
<td>Enabled</td>
<td>traffic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 13:** traffic.ijs.si public Optimum APIs for easy data-access
6.5.2 Security
Currently the data infrastructure implements security through API Management Console, where we can assign users and control which data can specific user access. Users are identified through security tokens, which need to be attached in the header in each request. In the case of the external (LPP) data-server, the console is protected with the IP filter (only specific machine allowed to log-in), while the data-access is controlled by the management console. In the next release we will provide an advanced security approach as described in D6.1
7 Conclusions

This document provided the description of the first version of Optimum Data Infrastructure which is readily online at the time of the writing and is already collecting storing and serving the described data-sets. Also, the effects of some of the approaches can be already noticeable with the reduced overhead on the developers maintaining the data-retrievers and also with additional data-sets which would be otherwise inaccessible.

The infrastructure described here, can be used for any data-modality, and is architecturally exactly the same as described in the deliverable D2.3, except some of the specific tools that are more accustomed for natural language, video and text processing (as opposed the raw sensor measurements).

Additionally, the document provided rationale behind some decisions for the infrastructure like making it a distributed system and one point of entry for the data access, which solved many issues regarding the data and use-cases variety, maintenance and development simplicity.

For the initial version the first priority was to get and start collecting data as soon as possible, then second – make the data available to the project and only the third to harmonize all the data into the similar structure. For this reason in the initial version, some of the data-sets are only being identified collected and are not yet in the public API, or not yet aligned with the data of the same type. For example, each loop sensor data-source can still have some of the custom fields not existing in another API. These are the next steps for the data infrastructure, with the support of task T2.4 Data Harmonization. Additionally, we are in the process of adding API access counters and also improving the measurements on the side of the data-retrievers and watch-dog, which will serve as a quantitative information for the data infrastructure resource planning and a quality of service. Additionally, we are working on the initial publish-subscribe implementation that will be able to connect the most demanding APIs and users according to our measures.
Annex

Annex 1

Example call to http://traffic.ijs.si/API/info/getApiInfo, which provides a list of all available APIs.

```json
{
  success: true,
  data: {
    opendata: [
      {
        url: "/promet/events/",
        access: true,
        public: true,
        active: true,
        project: "opendata",
        notes: ""
      },
      {
        url: "/promet/burja/",
        access: true,
        public: true,
        active: true,
        project: "opendata",
        notes: ""
      },
      {
        url: "/promet/burjaznaki/",
        access: true,
        public: true,
        active: true,
        project: "opendata",
        notes: ""
      },
      {
        url: "/promet/counters/",
        access: true,
        public: true,
        active: true,
        project: "opendata",
        notes: ""
      }
    
```
notes: "",
},
{
url: "/promet/parkirisca/lpt/",
access: true,
public: true,
active: true,
project: "opendata",
notes: ""
}
},
{
url: "/promet/cameras/",
access: true,
public: true,
active: true,
project: "opendata",
notes: ""
}
},
{
url: "/promet/bicikelj/list/",
access: true,
public: true,
active: true,
project: "opendata",
notes: ""
}
},
{
url: "/vreme/report/",
access: true,
public: true,
active: true,
project: "opendata",
notes: "lat float, ki predstavlja zemljepisno širino, npr. 46.051418. Veljavne vrednosti so med 45.21 in 47.05. lon float, ki predstavlja zemljepisno dolžino, npr. 14.505971. Veljavne vrednosti so med 12.92 in 16.71."
}
],
LPP:
[
{
url: "/routes/getRouteGroups",
}
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
url: "/routes/getRouteDetails",
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
url: "/routes/getRoutes",
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
url: "/routes/getStationsOnRoute",
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
url: "/stations/getRoutesOnStation",
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
url: "/stations/getStationById",
access: true,
public: true,
active: true,
project: "LPP",
notes: ""
},
{
  url: "/timetables/getArrivalsOnStation",
  access: true,
  public: true,
  active: true,
  project: "LPP",
  notes: ""
}
],
Internal:
[
  {
    url: "/info/getApiInfo",
    access: true,
    public: true,
    active: true,
    project: "",
    notes: ""
  }
],
traffic:
[
  {
    url: "/routes",
    access: true,
    public: true,
    active: true,
    project: "traffic",
    notes: "List of available routes for Blaz's traffic predictions."
}
]
```json
{
  url: "/backup",
  access: false,
  public: false,
  active: false,
  project: "traffic",
  notes: "? Needs username and password, so this is disabled now."
},
{
  url: "/get-store-list",
  access: true,
  public: true,
  active: true,
  project: "traffic",
  notes: ""
},
{
  url: "/get-store-recs/:store",
  access: false,
  public: false,
  active: false,
  project: "traffic",
  notes: "Disabled due to using URL param."
},
{
  url: "/traffic-predictions/get-sensors",
  access: true,
  public: true,
  active: true,
  project: "traffic",
  notes: "Returns predictions for sensors"
},
{
  url: "/traffic-predictions",
  access: true,
  public: true,
  active: true,
  project: "traffic",
  notes: "Gets traffic predictions"
},
{
  url: "/traffic-predictions/:id",
```
access: false,
public: false,
active: false,
project: "traffic",
notes: "Disabled due to using URL param."
},
{
url: "/evaluations",
access: true,
public: true,
active: true,
project: "traffic",
notes: ""
},
{
url: "/evaluations/:id",
access: false,
public: false,
active: false,
project: "traffic",
notes: "Disabled due to using URL param."
},
{
url: "/traffic-predictions/add",
access: false,
public: false,
active: false,
project: "traffic",
notes: "Disabled due to using POST. Not implemented yet."
}