A Generic Microservice Architecture for Environmental Data Management

Eric Braun[1], Thorsten Schlachter[1], Clemens Düpmeier[1], Karl-Uwe Stucky[1], and Wolfgang Suess[1]

[1]Karlsruhe Institute of Technology, Karlsruhe, Germany
eric.braun2@kit.edu | thorsten.schlachter@kit.edu |
clemens.duepmeier@kit.edu | karl-uwe.stucky@kit.edu |
wolfgang.suess@kit.edu

Abstract. The growing popularity of Web applications and the Internet of Things cause an urgent need for modern scalable data management to cope with large amounts of data. In the environmental domain these problems also need a solution because of big data coming from a large amount of sensors or users (e.g. crowdsourcing applications). This paper presents an architecture that uses a microservice approach to create a data management backend for the mentioned applications. The main concept shows that microservices can be used to define separate services for different data types and management tasks. This separation leads to many advantages such as better scalability and low coupling between different features. Two prototypes, which are already implemented, are evaluated in this paper.

Keywords: Microservices, Data management, Environmental Information Systems (EIS), Databases, Time Series, REST, Semantic Web, Linked Data, Big Data

1 Introduction

Nowadays the Internet of Things and modern Internet infrastructures lead to a massive amount of data stored in data centers all around the globe. In the field of environment, the sensorization of the environment and new crowdsourcing applications will also produce large amounts of data which have to be stored, managed and analyzed timely in order to provide early input for decision makers and the general public. Often, the acquired data in such applications consists of a mixture of measurement data, more general time series data, structured master data or unstructured text or binary assets. The most useful tools to get an insight into this data are visualizations and data analysis. However, classical information system architectures and desktop data analysis and visualization tools have severe problems in handling large amounts of data and new techniques are needed to manage these amounts of data in a scalable way. Therefore, the Web based Information Systems (WebIS) and Data Life Cycle Lab Energy (DLLE) groups at KIT/IAI work on a new data management runtime environment based on microservices [1], which can be easily integrated with Big Data infrastructures, and

adfa, p. 1, 2011.

scalable data analysis and web based visualization tools. Based on the microservice reference architecture introduced by Gartner[1] a new modular set of distributed services instrumenting a polyglot distributed data management model on top of an underlying Big Data environment was implemented. These services can be used with additional infrastructure services to provide a very generic and flexible scalable data management infrastructure with all the features needed by modern large scale web based information system applications. Based on these services, existing environmental backend applications for data management, such as the environmental information system applications within the LUBW[2], can be brought to a state-of-the-art level of technology to provide a future-proof way of managing the massive amounts of data which will be gathered and analyzed by future Web and mobile environmental information system applications. This paper describes the chosen basic architectural approach and then focuses on the mentioned collection of basic microservices that were conceptualized and implemented to create a generic solution for an efficient large-scale (environmental) data management. Further papers will describe other aspects of the architecture and specific microservices.

2 Basic Concepts and Main Goals

Data management is needed in many application domains other than the environmental domain. For example, the research groups at the KIT are also involved in several smart grid projects in which it is critical to gather and analyze large amounts of data for future energy system solutions. Therefore, the KIT was looking for a data management solution which not only applies to the environmental but also to the smart grid domain. The architecture described in this paper fulfills this requirement and is conceptualized based on very generic data type notions that enable data management for nearly any domain. Current information system applications in the environmental area are often still implemented as big monoliths using a standard multitier architecture, in which the data tier is concentrated in one relational database model and access layer. The resulting data model has a very strong structure but ties the whole data management to one application or application domain and does not scale horizontally to meet performance needs in a world of big data. A microservice based architecture can replace this monolithic data management concept by providing a set of modular and low coupled data services whereas each service defines its own data management tier based on a more generic and reusable data model and the most adequate database technology underneath which is not necessarily a relational database system. Microservices in general can be defined as follows:

> “In short, the microservice architectural style is an approach to developing a single application as **a suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often

[1] http://blogs.gartner.com/gary-olliffe/2015/01/30/microservices-guts-on-the-outside

[2] Baden-Wuerttemberg State Institute for Environment, Measurements, and Nature Conservation, Karlsruhe, Germany

> an HTTP resource API. These services are **built around business capabilities and independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.”
> James Lewis and Martin Fowler, 2014 [emphasis added]

Microservices are typically designed to scale horizontally to meet the currently needed performance. Since microservices separate functionalities into distinct services, each functionality can be scaled independently from each other according to the need. This advantage directly applies to data management because often scaling is not needed for the whole data model, e.g. for some rather static or seldomly used parts, but only for providing access to highly frequented parts of the data.

In a polyglot data model different data services use different database technologies to provide a more generic data handling and data access. For example, a document oriented database can be used to implement a very generic model of structured master data storage without tying the implementation too much to a specific master data schema. Nonetheless, a separate schema service which describes specific types of master data more precisely can provide all benefits of having a strong schema without hardcoding the data type schemas into the implementation of the master data service.

Another advantage of microservices is the more modular development approach allowing projects to be performed by smaller independent development groups, each of them having to manage just one or a few microservices. The communication between microservices is often realized using REST over HTTP(S). This communication interface is the only component of a microservice that is visible from the outside. Everything else, like application and data logic, is hidden within the service. Therefore, the REST API is the only interface that has to be standardized across development teams. This has the benefit that all technical interfaces between the otherwise separate projects are externalized. However, this also makes it crucial to define a proper API with an appropriate documentation and manage these APIs application or enterprise wide (called API management) in order to guarantee long term stable interaction between microservices and clients.

This paper will demonstrate how dedicated microservices can be conceptualized which allows the management of large amounts of different types of data by instrumenting a polyglot data model. The different services follow a common philosophy which leads to a REST API that is similar in its core for all services. A first prototypical implementation that proves this concept and solves first development challenges is discussed.

3 Architectural Overview

The microservice based architecture used is inspired by the reference architecture of Gartner [2]. Figure 1 shows this reference architecture.

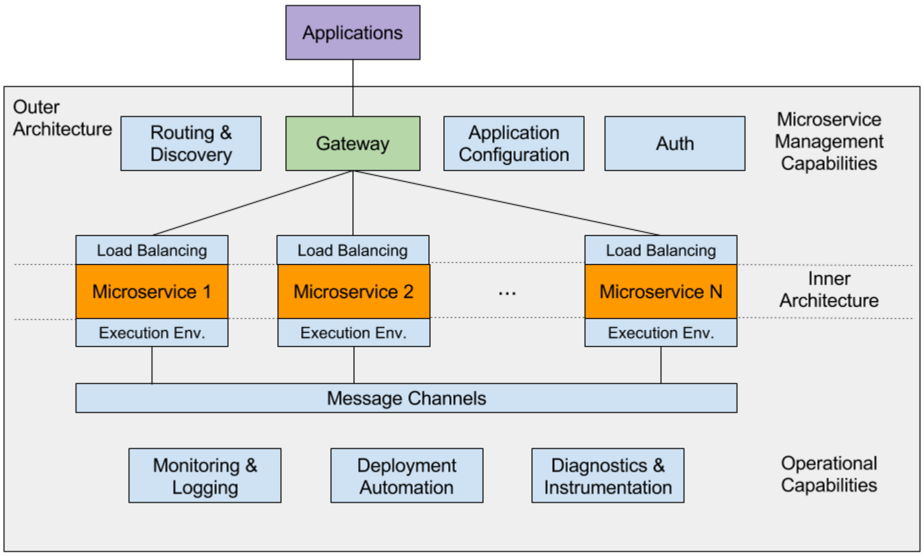


Fig. 1. Overview of microservice based architecture

The architecture can be split into three main functional areas: microservice management capabilities, actual microservices and operational capabilities. In the following only the second part is of major importance. Although management and operational capabilities are crucial for operating and maintaining a microservice architecture, this paper mainly focuses on which microservices are needed as part of the inner architecture to provide a scalable data management functionality for large scale information system applications.

Typically, microservices are designed to be load balanceable which leads to a (horizontally) scalable system. Therefore, the management layer of the Microservice Architecture contains an API gateway which distributes incoming client calls with the help of a discovery and routing service onto different instances of the inner services. Communication with the management layer and clients is generally achieved by using REST over HTTP(s) or WebSockets[3]. The communication between different microservices is implemented using a message channel (e.g. Apache Kafka[4]) to realize asynchronous messaging. This leads to more autonomous services because there are no synchronous and therefore blocking dependencies. Each microservice has its own execution environment which enables a service to be deployed as a container using container virtualization technologies (e.g. Docker[5]). This enables services to be automatically deployed, updated, and to run on many different platforms. Container automated services can also be scaled by increasing or decreasing the number of replicated instances. Service instances are registered with the discovery service and a heartbeat detection checks if instances are still alive. The execution environment of a service instance may

[3] https://tools.ietf.org/html/rfc6455

[4] https://kafka.apache.org

[5] https://www.docker.com

also include a private, internal database system but usually the database systems are separated from the microservices in order to keep the latter stateless. This enables databases to scale independently from the used services. As already mentioned for operating such microservice based applications a computing cluster with support for container virtualization is needed (this is typically the case in modern cloud environments). If the cluster or cloud environment also supports a Big Data stack (such as Apache Hadoop[6]), then Big Data tools, like NoSQL database technologies, can be used and integrated with the inner architecture microservices. As already explained the term polyglot data model means that each service is supposed to use the database technologies which are optimized for the data type managed by the service.

Figure 2 displays a more detailed architecture of the data management services. It shows the gateway as the single access point for client applications which allows to use a single harmonized URL space as well as a single REST API pattern to access the different services. This harmonized URL space and single API is key for the integration of the otherwise on several services distributed data. Each data object stored in a data management service is represented by a unique URI and URIs can be linked to each other to implement relations. Additionally, semantic services define the structure and interrelationship of the otherwise distributed data and provide this knowledge to the application and/or other services.

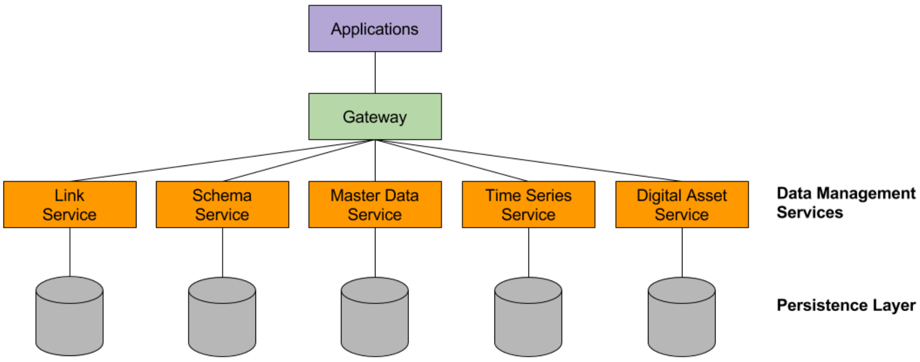


Fig. 2. Data management architecture

The inner architecture for data management consists of five microservices which are briefly described as follows:

- Link Service: a semantic service that is used internally to create relations between data. This service is crucial to create semantic links between different data objects. The service follows the concepts of linked data and semantic Web [3].
- Schema Service: a semantic service that manages schema descriptions (format, data types, etc.) of data objects stored in the different services.
- Master Data Service: manages structured data describing specific objects, i.e. master data.

[6] http://hadoop.apache.org

- Time Series Service: manages time series data. Persistence is usually achieved using a time series database. The separation of time series data into a separate service has many advantages as already depicted in [4].
- Digital Asset Service: manages digital assets similar to systems like Alfresco[7] that feature a CMIS[8] interface.

Another important feature of the architecture is shown in Figure 3 using e.g. the Time Series Service. Each of the data management services is connected to the persistence layer through one or more specific adapters that create a mapping between a concrete database technology used to implement a persistence layer and the respective service. This enables the independent development of persistent layers using different database technologies. The service itself is completely generic and has no dependencies on the underlying database technologies.

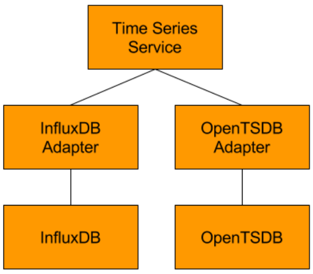


Fig. 3. Abstraction layer between the Time Series Service and time series databases

4 Services in Detail

This chapter focusses on the following services: Schema Service, Link Service and data discovery. The other services mentioned in the previous chapter are discussed in more detail in Schlachter et al. [6].

The Schema Service stores the structure, data types and other semantic information of the data stored in the basic data services. The Master Data Service benefits most from the Schema Service because master data is meant to be of a specific structure using specific data types. The main exchange format for the different services is JSON. Therefore, the Schema Service uses JSON schema[9] as default format to store and exchange schema information. An example for such a schema is depicted in Figure 4. It is a schema for an air measurement station. Beside the required attributes id and name this example contains two attributes that hold information about the particulate matter limit and current value. Additionally, the attribute *timeSeries* is a complex type that is not part of the official JSON schema standard. Such a complex type usually has an own

[7] https://www.alfresco.com

[8] http://docs.oasis-open.org/cmis/CMIS/v1.1/cs01/CMIS-v1.1-cs01.html

[9] http://json-schema.org

schema also described in the Schema Service. The service itself uses a document oriented database to store the JSON schemas efficiently.

The service is meant to be used internally mainly by the Master Data Service to validate incoming data and to add schema information to the data to implement linked data. The concept of linked data [5] can be used by applications that consume both; master data and their semantics.

Furthermore, the Schema Service can also be directly accessed from clients (through the gateway) to request a specific schema. This feature is used by applications that generate generic inputs and need the structure of the different fields to do so.

An additional requirement for the Schema Service is the support of different versions and namespaces. The structure of data can change over time which leads to an updated schema for the data. With introduced versions, the schema can exist in more versions and the Master Data Service can associate the data to the appropriate version depending on the changes. The namespace can be used to use the same schema name in different contexts.

```
{
 "title": "Air Measurement Station",
 "type": "object",
 "properties": {
   "id": { "type": "string" },
   "name": { "type": "string" },
   "pm10Limit": { "type": "number" }
   "pm10Current": { "type": "number" }
   ...
   "timeSeries": { "type": "timeSeries" }
 },
 "required": ["id", "name"]
}
```

Fig. 4. JSON schema of an air measurement station

Simple applications only need one of the services because they process a single type of data but more complex applications might need multiple data sets and most probably data sets that are related to each other. Using a microservice architecture, the data is distributed among different services which means that there is a need for a service that defines relationships between different data sets and objects. This service is called Link Service in our architecture. It allows to fully support linked data. The Master Data Service can only support the linked data concept for one data object but with the help of the link service data, corresponding schema information and relationships between data objects can be aggregated into one linked data description of complex interrelated data.

Links are implemented using URIs that point to the data. Additionally, relationships can define additional properties. Applications have the possibility to get a data object with all their links as URIs or with resolved links which lead to nested data objects. Figure 5 presents an example with both options in a pseudo format that is not part of the specification but only to explain the link resolving concept. The first option is the master data object for the air station mentioned above that contains two links that point to time series data for the measurements of air pollutants NO2 and PM10. These links have to be fetched separately from the application in order to get the actual time series data. The second option contains the same master data object but the links are already replaced by the data. Additionally, both objects have a PM10 limit and a current value that obviously matches the last value of the time series.

Object 1

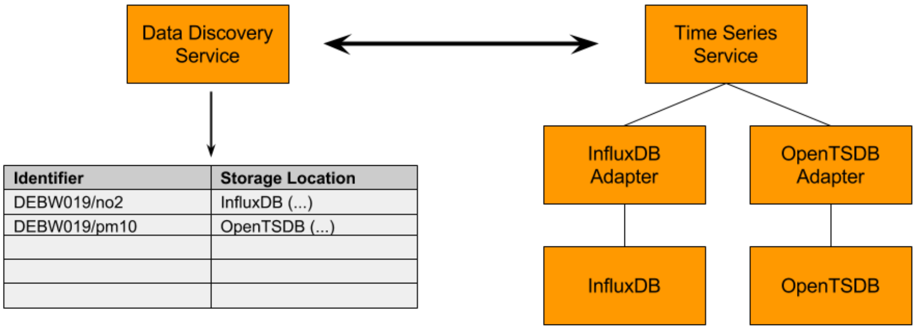
```
{
 "id": "DEBW019",
 "name": "air station Ulm",
   "pm10Limit": 50,
   "pm10Current": 20,
 ...
 "timeSeries": [
   timeseries/DEBW019/no2,
   timeseries/DEBW019/pm10
 ]
}
```

Object 2

```
{
 "id": "DEBW019",
 "name": "air station Ulm",
   "pm10Limit": 50,
   "pm10Current": 20,
 ...
 "timeSeries": [
   [26, 22, 21, 23, 31, 58],
   [26, 14, 11, 15, 26, 20]
 ]
}
```

Fig. 5. Time series link resolving

A third concept that will be discussed in this chapter is data discovery. Unlike the described services, the Data Discovery Service will only be used internally and has no external API. It closes the gap between data services (e.g. Master Data Service, Time Series Service) and a specific adapter. As discussed in the previous chapter, data services are implemented by internally using an abstract persistence interface which decouples them from specific implementations of the persistence layer using a certain database technology. One data service can use more than one persistence layer implementation. This abstraction leads to the requirement that a data service needs a way to find out in what persistence layer a specific data set (e.g. master data object or time series) is stored, and this information is provided by the Data Discovery Service. Figure 6 depicts an example for the Time Series Service. The Time Series Service gets the storage location of data using the identifier that is externally communicated as lookup key to the Discovery Service which returns the storage location. The request can then be forwarded to the appropriate adapter. This separation of core services and their underlying persistence layers allows the services to be much more flexible and generic. The Data Discovery Service can be used by every service that stores data in a location that is not known by this service itself.



Identifier	Storage Location
DEBW019/no2	InfluxDB (...)
DEBW019/pm10	OpenTSDB (...)

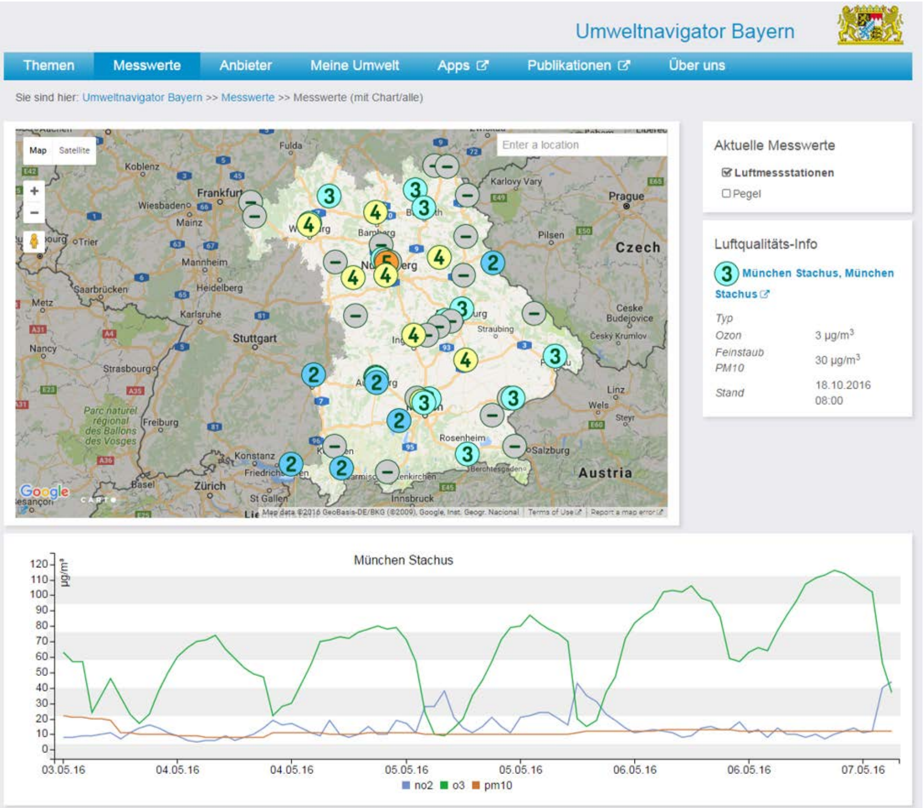
Fig. 6. Time series service and Data Discovery Service interaction

5 Prototype and Evaluation

The generic microservice based architecture was implemented as a prototype and evaluated in context of concrete information system applications, such as environmental information systems and smart energy systems control center software. The Gateway Service is implemented manually in the current prototype environment omitting load balancing features for the moment. In the future, it will likely be replaced by gateway tools like Netflix Zuul[10] for more complete operation. The data management services except the Digital Asset Service are already implemented to some extent. All microservices are implemented using the Java Framework Spring Boot[11] as implementation framework. The Schema Service and the Master Data Service use the document-oriented MongoDB NoSQL database as one persistence layer option by using specific adapters as described above. The Time Series Service is implemented with an OpenTSDB adapter. The adapters are implemented as own microservices which convert the HTTPS requests from database specific requests into abstract requests and vice versa. The communication between microservices as well as between the gateway and the client applications is realized using REST over HTTP(S). The asynchronous messaging channel mentioned while describing the architecture will be used in the future version to lower the synchronous coupling between services. Instead, services will cache data from other internal services. Decoupling and data actualization of cached data will be instrumented via the message channel. An appropriate solution using spring cloud stream and RabbitMQ was already tested in a separate prototype.

[10] https://github.com/Netflix/zuul/wiki

[11] http://projects.spring.io/spring-boot



Sie sind hier: Umweltnavigator Bayern >> Messwerte >> Messwerte (mit Chart/alle)

Aktuelle Messwerte

☑ Luftmessstationen

□ Pegel

Luftqualitäts-Info

3 München Stachus, München Stachus

Typ	
Ozon	3 µg/m³
Feinstaub PM10	30 µg/m³
Stand	18.10.2016 08:00

Fig. 7. Environmental information system Umweltnavigator

The Link Service is implemented using the Neo4j graph database[12] to model the different relations. To connect two data objects only their URI and a relation key are stored. Data can be fetched using the unique URI. The relation key can be used to further filter data by relation type. Such a relation key can describe the multiplicity of the relation or its semantic context.

The first prototype using the generic microservice architecture was implemented as background data service infrastructure for a showcase of the environmental information system “Umweltnavigator Bayern” of the “Bayerisches Staatsministerium für Umwelt und Verbraucherschutz”. As shown in figure 7, a web page of the Umweltnavigator displays a map of different air measurement stations. A specific station can be selected within the map resulting in the display of more information about the selected station in the right panel and the display of the related measurement data within the bottom visualization. Multiple backend services are accessed in parallel to aggregate this page. The map is filled with information from a geo service that contains the location and icon information of each measuring station. The panel to the right shows more detailed

[12] https://neo4j.com

information about the station which is fetched from the Master Data Service. The visualization at the bottom uses the same master data object and fetches the different available time series datasets referenced by the master data object. The URIs pointing to the time series datasets are resolved by the Master Data Service using the Link Service which contains the relationship information between measuring station and the measured time series. This example shows the interaction of the basic data services and more complex services like the Link Service or the Geo Service.

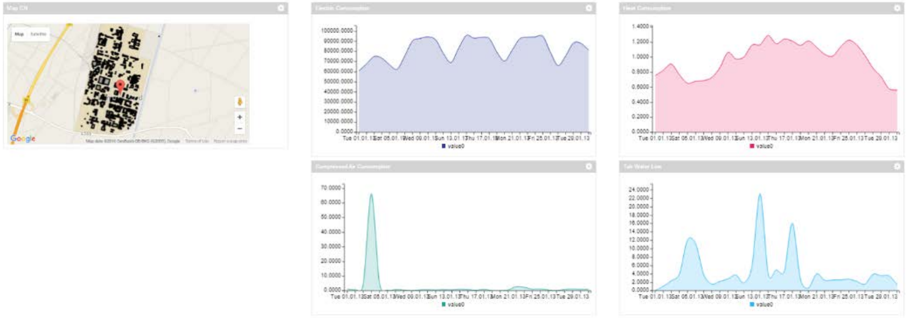


Fig. 8. Dashboard solution using the service infrastructure

The second prototype (figure 8) is used as a background data service infrastructure for a web based dashboard solution that displays the energy consumption of buildings. The prototype of the dashboard solution beside the backend services was developed in a diploma thesis [8]. The following description explains the usage of the web page from the point of view of a user and of a dashboard manager who creates and customizes pages. An end user accessing the dashboard web page with his browser can select a building on the map which then changes the visualizations of the measurement data accordingly to data that belong to that building. Similar to the previous prototype the visualization uses the master data and the Time Series Service to display the data. Furthermore, the dashboard solution offers additional functionality to customize the structure of the dashboard web page. A dashboard manager can change the information that is shown based on what the master data object that is displayed (in this case: a building) offers. The Schema Service provides this information for the dashboard configuration dialog. Therefore, the dashboard manager sees the different properties that a building can have, including the time series datasets. This allows him to customize every block of the dashboard page that can display information of an arbitrary master data object of a certain type (e.g. a building). In contrast, using a less generic solution, each building needs a manually customized page which does not scale for a large amount of buildings. This more complex scenario shows that the additional use of the Schema Service allows the development of smart applications that provide generic features.

Both prototypical implementations present first proofs of the concept and already show the great potential of the envisioned solution. The generic data service infrastructure is used in both use cases to provide application specific data to web applications belonging to different application domains. Both applications directly use the REST

API of the different services which result in an easy and lightweight communication. The scalability of the backend was not tested yet but overall the architecture is scalable because every microservice can be deployed multiple times behind the gateway without changes to the implementation of the prototypes. Furthermore, although the data services work with a very generic data model, the Link Service and the Schema Service can be used to implement applications that use additional meta information like relationships or schemas to fulfill very specific and more complex requirements regarding the data semantics of the stored data objects. This information is not hard coded within the implementing data services but can be configured as needed by using the semantic services.

6 Conclusion and Outlook

This paper shows a concept for a generic service oriented data management infrastructure for managing a large variety of data which can be implemented with state-of-the-art microservice frameworks and database technologies. The framework uses a polyglot data model and provides distinct services for managing large amounts of heterogeneous data in a generic way. Especially the separation of schemas, master data, time series data and digital assets was discussed in detail. An additional feature of the architecture discussed is the abstraction from database technologies. This enables the microservice architecture to be even more generic because the exposed interface is completely independent from underlying technologies. A prototype was implemented and used for an evaluation of the concepts with two web applications belonging to different application domains. The evaluation showed that the presented microservice architecture can be used in many application domains that have to deal with heterogeneous data, different databases, and that need a high performance and scalability which can be provided by horizontal scaling of the microservices.

The concept and the prototype can and will be extended in various ways. Firstly, the Gateway Service will be replaced by a tool like Netflix Zuul or NGINX Plus which is more reliable and has more features than the service implemented in the prototype. Secondly, an upcoming version of the prototype will also provide an additional binary and therefore faster communication interface than a pure REST API (e.g. by using web sockets). Tools like RabbitMQ[13] and Apache Kafka[14] can and will be used as messaging systems implementing a publish/subscribe protocol for decoupling clients and services which makes the whole system more elastic. Additionally, they will support distributing data to other services beside the Data Management Service, like dedicated data analytics pipelines. Another step towards the compatibility with more Web applications is to add services that can provide data in generic ways (e.g. in feed formats). Such services can be full text search or geo services (see [6]). Furthermore, the generic web visualization framework [7], which is also developed by the WebIS research group, is already integrated with the microservice architecture and it is already implemented

[13] https://www.rabbitmq.com

[14] http://kafka.apache.org

using a microservice backend. This will allow to easily explore data by using advanced visualizations.

References

1. Fowler, Martin; Lewis, James: “Microservices – definition of this new architectural term”; http://martinfowler.com/articles/microservices.html; visited September, 26th 2016
2. Gartner Blog Network: “Microservices : Building Services with the Guts on the Outside”; http://blogs.gartner.com/gary-olliffe/2015/01/30/microservices-guts-on-the-outside/; visited September, 26th 2016
3. Bizer, Christian et al.: “Linked Data – The Story So Far”; International Journal on Semantic Web and Information Systems, Vol. 5(3), Pages 1-22. DOI: 10.4018/jswis.2009081901
4. Leighton, Benjamin et al.: “A Best of Both Worlds Approach to Complex, Efficient, Time Series Data Delivery”; Environmental Software Systems. Infrastructures, Services and Applications, ISESS 2015, Melbourne, VIC, Australia, March 25-27, 2015; pp371-379; DOI: 10.1007/978-3-319-15994-2_37
5. W3C: “JSON-LD 1.0 W3C Recommendation”; https://www.w3.org/TR/json-ld/; visited September, 26th 2016
6. Schlachter, Thorsten et al.: “A Generic Web Cache Infrastructure for the Provision of Multifarious Environmental Data”; Extended Abstract submitted ISESS 2017
7. Braun, Eric et al.: “Generic Web Framework for Environmental Data Visualization”; Advances and New Trends in Environmental Informatics, EnviroInfo 2016, Berlin, Germany, September 14-16, 2016; pp.289-299; DOI: 10.1007/978-3-319-44711-7_23
8. Pathomkeerati, Kajorn: “A new generic Approach for Web based Dashboard Solutions in a Microservice Architecture”; Diploma Thesis, 2016