A RESTful Sensor Data Back-end for the Internet of Things

Antti Iivari and Jani Koivusaari
VTT Technical Research Centre of Finland Ltd
Oulu, Finland
email: antti.iivari@vtt.fi, jani.koivusaari@vtt.fi

***Abstract*—As rapidly increasing amounts of smart communicating objects with sensing capabilities are generating raw measurement and observation data, scalable back-ends are needed to collect, store, marshal and process that influx of machine-generated data into actionable information. The data constantly flowing out from these embedded devices is very periodic and structured in nature, referred to as machine-generated data, beginning with a timestamp of some sort and then consisting of designated fields, such as measurement values, ranges and tags. Furthermore, as wireless sensing devices are in many cases battery operated and resource constrained, a mode of operation can be assumed where the device transmits measurement data at specific intervals between which it preserves power by sleeping or idling. This is only one of the reasons why a RESTful approach that has been more commonly associated with the World Wide Web, could be appropriate when dealing with the challenges brought forth by the Internet of Things (IoT) revolution. In this paper, initial findings concerning a proof-of-concept back-end implementation are presented in addition to discussing the benefits and technologies related to a RESTful approach in building a scalable sensor data back-end for the Internet of Things.**

Keywords-IoT; Sensor; Back-end; Data; REST.

I. INTRODUCTION

Today a rapidly increasing amount of smart communicating objects with sensing capabilities are generating raw measurement and observation data that in order to be useful, must be collected, stored, marshalled and processed in a back-end of some sort. This onslaught of small interconnected embedded devices and the messages they are transmitting is commonly referred to as the Internet of Things (IoT) [1]. Typically, the kind of data constantly flowing out from such systems is very periodic and structured in nature, referred to as machine-generated data [2], beginning with a timestamp of some sort and then consisting of designated fields, such as measurement values, ranges and tags Furthermore, as wireless sensing devices are in many cases battery operated and resource constrained, a mode of operation can be assumed where the device transmits measurement data at specific intervals between which it preserves power by sleeping or idling. This is only one of the reasons why a RESTful approach [3], which has been more commonly associated with the worldwide web, is appropriate when dealing with the systems and devices in an IoT context.

Representational state transfer, or REST [3], is a software architectural style for designing distributed systems and it is used for the World Wide Web. When distributed systems and services conform to the constraints of REST they can be called "RESTful". RESTful systems virtually always communicate via the Hypertext Transfer Protocol (HTTP) with the standard HTTP commands (GET, POST, PUT, DELETE). REST has gained widespread acceptance across the Web as an easier-to-use, resource-oriented alternative to more complex approaches such as SOAP or WSDL. When considering a RESTful approach for constrained very low-power sensor devices, the CoAP protocol is particularly interesting, as it is designed to interface with HTTP and the Web while meeting specialized requirements such as very low overhead and multicast support. In essence, CoAP aims to provide a more compact version of HTTP/REST with additional features optimized for M2M and IoT applications. As a reliable and scalable back-end solution is a requirement for most IoT-type applications where large amounts of rapidly streaming machine-generated data from multiple sources needs to be handled and stored for later processing, a RESTful approach for implementing such a back-end is discussed in this paper. The goal of the work described in this paper is to design and deploy a REST-style HTTP/POST -interface and enable IoT devices to communicate towards the backend as effortlessly as possible.

The paper is organised as follows: In Section II the most important characteristics of protocols and data formats for the Internet of Things are discussed, while Section III outlines the key building blocks of a sensor data back-end solution. Section IV presents the prototype implementations before we summarize and discuss some future work items in Section V.

II. MACHINE-GENERATED DATA FROM SMART CONNECTED OBJECTS

In order to build and design a viable back-end solution for reliably handling large amounts of machine-generated IoT data, the characteristics of such systems must be studied and understood. For the purposes of the work presented in this paper, there are two key facets of sensor data that must be considered. First is the format of the represented sensor data itself. Second is the message protocol with which this machine-generated data is transmitted. Some of the most common examples of sensor data formats in current systems are listed as follows:

- JSON: JavaScript Object Notation is a lightweight data-interchange format for storing and exchanging

 ISBN: 978-1-61208-478-7

data. It is easy to read and write by both humans and machines. While JSON is a text-based format and language independent, it is originally a subset of the JavaScript programming language.

- CSV: Basic comma separated values are by far the simplest and most rudimentary of commonly used sensor data formats.
- XML: Extensible Markup Language (XML) is essentially a set of rules for encoding documents in a format which is readable for both man and machine. XML also acts as a basis for some M2M protocols [4], such as XMPP and BitXML.

Similarly, the most essential messaging protocols [5] employed in application layer transfer of machine-generated data and IoT-type data transfer are listed below:

- Hypertext Transfer Protocol: HTTP the tried and true RESTful HTTP over TCP very familiar to us all from the web service world, is a particularly attractive option for constrained IoT devices, when considering the almost universal availability and compatibility of the legacy HTTP-stack on various platforms.
- Message Queuing Telemetry Transport: MQTT is a lightweight publish/subscribe based message protocol especially well-suited for running on limited computational power and lean network connectivity.
- Constrained Application Protocol: CoAP aims to be a generic web protocol for the special requirements of constrained sensor environments while easily integrating with HTTP and existing web technologies with a very low overhead.

It should be noted that any technologies related to the REST software architectural style, such as CoAP and HTTP, were given special consideration during this work, as RESTful architectures are clearly a promising and common approach employed in many contemporary IoT-platforms and sensor data platforms. The CoAP interaction model is similar to the client/server model of HTTP as a CoAP request is equivalent to that of HTTP and is sent by a client to request a resource. However, unlike HTTP, CoAP deals with these interchanges asynchronously over a lighter datagram-oriented transport such as UDP. Furthermore, employing RESTful APIs will also give the advantage of easy integration with existing web services and other popular http-based platforms.

III. INTERNET OF THINGS ON THE BACK-END

Raw sensor data in and of itself, consisting of measurement values or observational data corresponding to a short time-frame, is rarely useful or informative in an immediate or direct manner. Typically the data is transmitted to an application back-end to be marshalled and processed into something useful for the application at hand. The back-end consists of an interface (such as REST) to act as the collector towards which the sensing devices communicate either directly or via a gateway device [6].

In a typical IoT scenario, in addition to the embedded intercommunicating smart objects (the "things"), we have the server-side functionality where the actual application specific logic and data processing takes place. This is referred to as the back-end. The back-end usually consists of three main parts: a server, an application, and a database. In order to make the server, application, and database communicate with each other, server-side languages like PHP, Ruby, Python and JavaScript are employed, and database tools like MySQL or PostgreSQL are needed to find, save, or change data and serve it back to the users (either man or machine) of the service. MySQL is an open-source relational database management system (RDBMS) and one of the most popular ones used in modern web-applications. MySQL is also the database of choice prototype work discussed in this paper.

A. Representational State Transfer for IoT

In technical terms REST, or Representational State Transfer [3], is an architectural style for building networked applications. It is based on a stateless, client-server communications protocol and is almost always heavily tied to the HTTP protocol. Representational State Transfer (or REST) has become a widely accepted alternative software architecture approach for developing scalable web services. So called RESTful systems adhering to this principle communicate with each other simply by using the standard Hypertext Transfer Protocol (HTTP) with GET, POST, PUT and DELETE -queries. The basic idea is that simple HTTP is used to make resource calls between machines, instead of using complicated mechanisms such SOAP to connect services. This is essentially the same method that any web browser today uses to retrieve (GET) data and web pages from the internet and send (POST) input by the user to the remote server.

The benefits of REST from an IoT perspective are easily apparent, as it is a relatively lightweight approach to building intercommunicating services while also being fully-featured in the sense that there are not many things that can be done with Web Services that can't be realized with a RESTful software architecture in one way or another. Furthermore, REST itself is not a "standard" as there will never be a formal W3C specification for REST, for example. A concrete implementation of a RESTful distributed service always follows the following four key design principles:

- Resources expose easily understood directory structure-like URIs.
- Transfer JSON or XML to represent data.
- Messages use HTTP methods explicitly (GET, POST, PUT, DELETE).
- Based on stateless interactions. No client context information is stored on the server between requests.

While discussing the back-end prototype for a conditions monitoring system in the next section of this paper, it is important to note that the goal in this case is not to implement a fully functional REST-based architecture or interface strictly adhering to the specification. Instead, the approach is to design and deploy a REST-like HTTP/POST -

 ISBN: 978-1-61208-478-7

web interface to enable the sensing devices to communicate towards the backend as effortlessly as possible, as devices such as these sensors are often extremely resource constrained, not only in terms of processing power and memory, but also in terms of network capabilities and battery reserves. This has the added benefit of enabling any communication capable device, regardless of other operational or hardware characteristics, to push their measurement or log data towards the back-end by simply sending HTTP/POST-messages to the known address of the server. Formulating a HTTP-compliant POST message with the data included in the header as a payload is a simple matter and computationally light-weight operation. Some of the main benefits of implementing a RESTful service for the Internet of Things are as follows:

- Platform-independency
- Language-independency
- Standards-based (e.g. HTTP)
- Easy to work with firewalls

Utilizing RESTful architectures in the context of IoT or M2M applications is nothing new in and of itself. Indeed, others have successfully designed approaches for such systems before [7][8] based on REST.

IV. THE RESTFUL PROTOTYPE IMPLEMENTATIONS

During this work, a proof-of-concept pilot system has been established to measure, collect and store sensor data for the purpose of monitoring the conditions of an inhabited building. This paper focuses solely on the technical matters and preliminary findings concerning the back-end implementation, while the pilot system as a whole is left as the subject matter for another publication. In this section, the overall structure, main technological components and the chosen RESTful approach employed for the sensor data back-end system are outlined. It should also be noted that a simple but effective JSON-based sensor data format was designed at VTT for the purposes of this work.

A. *The traditional LAMP-stack*

First version of the RESTful sensor back-end was built on top of the traditional LAMP-stack [9]. The so called LAMP stack has become the tried and true basis for web-based applications now for two decades and the software components that make up the stack can be found in any of the default software repositories in most major Linux distributions. A standard LAMP stack consists of the following technologies: Linux as the underlying operating system, Apache as the Web server, MySQL as the relational database and PHP as the object-oriented scripting language. The components of LAMP are individually freely available Open Source Software, making them very attractive to potential users eliminating the need to purchase expensive commercial tools. The open licences make it possible for anyone to develop and distribute software based on the LAMP-stack without any licensing efforts or payments. The source code for any of the components in LAMP can be accessed by anyone, thus making it significantly easier to find faults and apply bug fixes, giving the users of the stack a degree of flexibility that is usually not available in comparable commercial alternatives.

While each of the technologies included in the LAMP stack are powerful and useful already in their own right, they are often used together and their compatibility towards each other has therefore been extended numerous times in the past to create a truly powerful and versatile platform for web-based applications. For these reasons, the LAMP stack was utilized as the key enabler and technological basis for the first prototype implementation of the back-end solution within the conditions monitoring prototype system. The PHP-based REST-interface for sensor data capturing implemented in the context of the prototype also includes support for HTTP Basic authentication as a relatively simple access control technique to ensure an elementary level of security in the exchange of measurement information. From the point of view of the sensor devices, applying HTTP basic authentication is also a very light-weight approach, as no handshakes, costly encryption calculations or similar procedures are required prior the transmission of data.

As discussed in the previous chapter, one of the key building blocks of a sensor data back-end is the database for storing the time-series data. The widely used and popular MySQL database, also used in the prototype discussed here, as a part of a web-based solution, works very well in combination with a number of modern programming languages (such as PERL, C, C++, Java, JavaScript and PHP) and various software development frameworks. From all of these languages, PHP is still the most popular one because of its convenience and capabilities in the domain of web-based application development. PHP provides a number of useful modules to access the MySQL databases and to manipulate data records and settings inside the database.

B. *First prototype*

To outline the structure of the LAMP-based first version of the prototype, a diagram illustrating the main components is given in Figure 1.

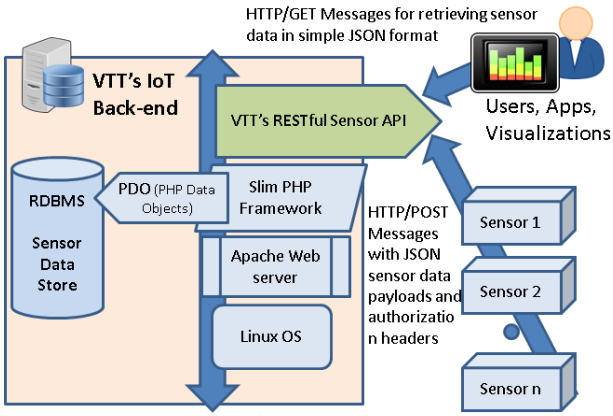

Figure 1. Overview of the first PHP-based RESTful IoT Back-end prototype.

Convenient and easily deployable interoperation with MySQL technology and the aforementioned capabilities geared towards web-applications and services led to the choosing of PHP as the programming language to implement

the required REST-like functionalities in the first version of the IoT back-end prototype. The PHP-functionalities were implemented by utilizing slimPHP [10], which is an open-source micro-framework designed for rapid development of responsive web applications and REST APIs with lots of useful functionalities built-in such as URL-routing and middleware architecture.

A REST –interface was programmed for storing and retrieving JSON-based sensor data payloads sent through standard HTTP GET and POST –messages. The sensor data is stored into a relational MySQL database.

C. *The Node.js environment*

Node.js is an open source JavaScript-based platform built on top of Google Chrome's JavaScript V8 Engine [11]. As it provides an event-driven architecture and a non-blocking I/O API making it very lightweight and efficient it is especially suitable for building data-intensive real-time applications that are scalable and run across distributed devices. Node.js facilitates the creation of highly scalable servers without using threading by using a simplified model of event-driven programming and providing a rich library of various usable JavaScript modules greatly simplifying the development of distributed applications. In the following, some of the key benefits of Node.js for IoT applications are listed.

- Fast code execution due to the underlying Google Chrome's V8 JavaScript Engine.
- The event driven asynchronous API ensures that the server never needs to wait for an API to return data.
- Highly scalable single threaded event mechanism scales better to a larger number of requests than traditional servers.
- Released under the open source MIT license.

D. *Second prototype*

The Node.js platform enables the developer to discard the traditional and somewhat cumbersome, LAMP-stack altogether while still providing excellent modules and built-in capabilities for development and interacting with various database technologies such as MySQL, as shown in Figure 2.

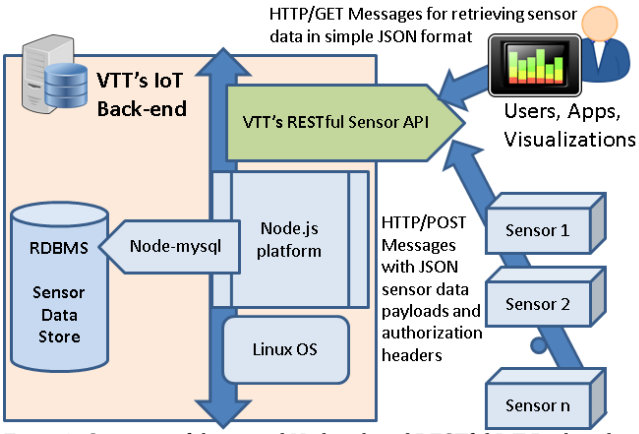

Figure 2. Overview of the second Node.js-based RESTful IoT Back-end prototype.

The potential of the platform for IoT-style application quickly becomes apparent, and as such, a second improved experimental version was built for this work by utilizing the more modern and scalable Node.js-platform. Functionalities in the second version of the back-end prototype correspond closely to the first version, but the scalability and maintainability of the system became superior. The data model for storing the sensor data from IoT devices into the database was also redesigned for the second version of the prototype system.

In addition to Node.js, some additional modules were employed for the implementation; with the most important one to mention being the Express framework [12], a flexible and minimalistic application framework for rapid development of HTTP-based web applications.

V. CONCLUSIONS

A RESTful approach for managing the rapidly incoming streams of machine-generated data in modern IoT systems is indeed a viable one. By harnessing modern software tools for web application development and server side application platforms, building scalable back-ends for the Internet of Things becomes more fluent and productive. When experimenting with the potential of these various technologies more familiar from the web-application world, it can be concluded that there is a lot of untapped potential for managing machine-generated sensor data and exploiting the true information value of the Internet of Things revolution. Scalability, security, reliability and easy deployment are just some of the observed benefits. In this paper, we have presented the first phases of the implementation work for a RESTful sensor data back-end.

The SlimPHP micro-framework proved to be an excellent tool in alleviating many of the problems and concerns with plain PHP-code or the heavier full-scale PHP frameworks, but as running PHP as the back-end code still required separate underlying Web server (such as Apache) there is a degree of cumbersomeness that can't be overcome with the LAMP-stack. The Node.js platform provided a flexible and dexterous alternative when implementing various features and interfaces for the second version of the prototype back-end. With the Node.js platform as a basis, superior flexibility and agility, in both deployment and maintaining phases, when compared to the standard LAMP-stack could be observed. Furthermore, due to the asynchronous and event driven nature of the Node.js technology, it is also expected to scale better for larger number of requests.

Some items are left altogether as next steps for the following phases of the work and future publications. Further comparisons and quantitative measurements on the performance and scalability of these back-end technologies are one topic of future interest. Enhanced security features for data privacy and system robustness are another item considered as next steps. Also, comparing the suitability of different database technologies as the amount of incoming sensor data starts nearing Big Data –volumes and different processing engines for data analytics become necessary, is another topic left for future work.

 ISBN: 978-1-61208-478-7

ACKNOWLEDGMENTS

The research from DEWI project (www.dewi-project.eu) leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n° 621353 and from TEKES.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions" Future Generation Computer Systems, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[2] K. Monash, "Examples and definition of machine-generated data", URL: http://www.dbms2.com/2010/12/30/examples-and-definition-of-machine-generated-data/ [retrieved: April, 2016]

[3] L. Richardson and S. Ruby, Restful Web Services, 1st ed. O'Reilly Media, May 2007.

[4] A. Iivari, T. Väisänen, M. Ben Alaya, T. Riipinen & T. Monteil, "Harnessing XMPP for Machine-to-Machine Communications & Pervasive Applications" Journal of Communications Software & Systems, Vol. 10 Issue 3, 2014, pp.163-178.

[5] V. Karagiannis, "A survey on application layer protocols for the internet of things." Transaction on IoT and Cloud Computing 3.1, 2015, pp.11-17.

[6] J. Latvakoski et al., "A survey on M2M Service Networks", Computers , vol.2, 2014, pp.130 - 173.

[7] W. Colitti, K. Steenhaut, N. De Caro, B. Buta and V. Dobrota, "REST Enabled Wireless Sensor Networks for Seamless Integration with Web Applications," Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, Valencia, 2011, pp. 867-872.

[8] D. Guinard, V. Trifa and E. Wilde, "A resource oriented architecture for the Web of Things," Internet of Things (IOT), 2010, Tokyo, 2010, pp. 1-8.

[9] G. Lawton, "LAMP lights enterprise development efforts", Computer, 2005, 9: 18-20.

[10] V. Vaswani, "Create REST applications with the Slim micro-framework" URL: http://www.ibm.com/developerworks/library/x-slim-rest/ [retrieved: April, 2016]

[11] J. R. Wilson, Node.js the right way. Pragmatic Programmers, 2014.

[12] A. Mardan, Pro Express. js: Master Express. js: The Node. js Framework For Your Web Development. Apress, 2014.

 ISBN: 978-1-61208-478-7