Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-5, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

IoT API System Structure for Handling Numerous Requests from Large Number of Endpoints

Gopal Chandra Bala[1], MD. Robiul Awal Maruf[2]
[1]Software Engineering Department, BitMascot Pvt. Ltd.
[2]Department of Computer Science and Engineering, World University of Bangladesh

Abstract*: The Internet of Things (IoT) is experiencing enormous growth around the world as the number of connected devices is increasing day by day. Besides IoT is getting matured and continues to be the least, most promoted topic in the information technology world. Over the last decade it has acquired attention by focusing the visual perception of a global infrastructure of inter-connected physical objects, enabling continuous communication between human to human, human to things and things to things. The IoT refers to the networking and coding of objects and things in order to render separately traceable on the internet. Electronic devices such as computers, smart phones, tablets, servers are linked through wired or wireless networks using IP address that connect the internet. These networks communicate by sending and receiving large volumes of data over the web API and information are extracted analyzing those data. A single restful API is responsible to process truly a large number of incoming requests from multiple IoT devices in every moment. Designing an API with an architectural style is much more essential in this case for processing millions of request efficiently.*

Keyword- *Internet of Things, API, Interconnected Object, Fragmentation, Request Channel*

1. Introduction

IoT systems network topology is the combination of simple nodes that collect and transmit an amount of data to a central gateway API which is responsible for providing connectivity to the internet and cloud service. In order to deploy a stable gateway API, technological and business model validation, large-scale request processing capability and proper security management play vital roles. Nodes and gateway API must be designed to minimize installation cost, cost of further logical changes and power consumption, provide large-scale network connections and extend wired or wireless connectivity range. Most of the IoT devices are typically powered and connect to the gateway API where data transmission and logical processing occurs. Sensor nodes generally transmit small volume of data and often have to run on batteries for many years. A process unit or microcontroller (MCU) of IoT systems process collected data and runs software stacks that make request to the central gateway API to store data based on the defined business validation rules. On the way toward "platforms for connected multiple devices" the biggest challenge is to overcome the fragmentation of vertically areas toward open systems and integrated environments and platforms, which support multiple applications of social values by bringing contextual knowledge of the surrounding world and events into complex business or social process. Data management is known as the root of the IoT Hub functionality and at present most of the IoT system does not provide require data management functionalities.

The REST API for IoT Hub allows logical and functional authorized access to the IoT device, data transmission, messaging and so on. It enables accessing the messaging service from within and IoT service running in web server from any kind of application that is able to send an HTTP web request and receive an HTTP response. The API here manages the device identities analyzing the request header. Each of the web requests that is send from the root IoT devices return an header than is used to obtain the device and request information.

It isn't possible to create any application that doesn't face performance issue at some point. Ensuring two facts- an application that work fast and perform reliably are much more essential while developing an application. It is important to access application resources that are both reliable and fast no matter what technologies are used. The ideal way to start thinking API performance is the single micro service. Even though the application will likely use more than just one. It brings out the best way to keep performance issue under control is to track the micro service individually rather than a group. The use of micro service also implies a distribute system. Using a simple method call in a monolithic application enable remote procedure call (RPCs). This means having to deal with issues such as network latency, fault tolerance, message serialization, unreliable networks and varying loads within the application tiers.

2. Common IoT Stack

A sensor is located at the root level of an IoT device. One or more sensors get connected with the

main device and collect environmental or associate data. The device management unit and device hub maintain a secure, robust and fault tolerant connection. The sensor data management unit processes the collected data that transmit to the API as parameterized web request. Figure 1 shows the common IoT stack sequence.

The IoT gateway acts as the aggregation point for a group of sensors and actuators to coordinate the connectivity of this device to each other and to an external network. IoT gateway will often processing of the data "at the edege" and storage capabilities to deal with network latency. For device to device connectivity, an IoT gateway deals with the interoperability issues between incompatible devices.

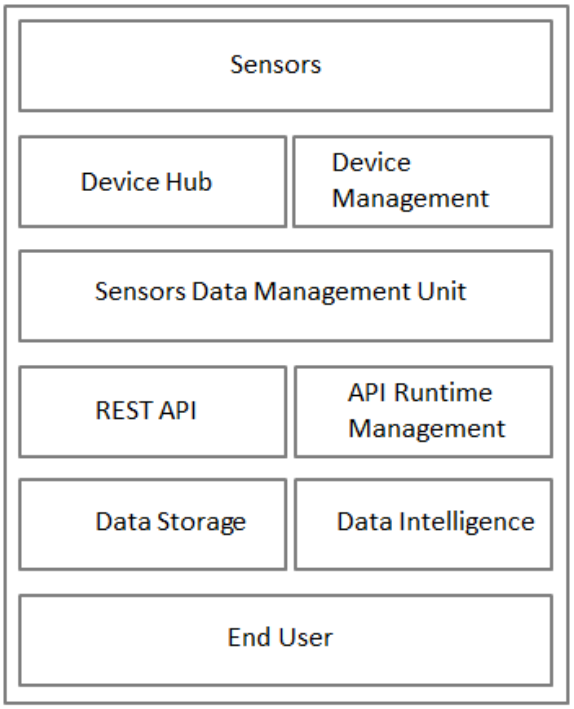

Fig.2.1. Steps of common IoT stack

3. REST API Architecture

A set of architectural principle is defined by REST. It encourages a focus on resources and is used to design web services illustrating how resource states are transferred over the HTTP protocol by a large range of end points. REST has such a huge effect on the web that in most of the case it has displaced WSDL and SOAP based interface structure which is a simple structure to use. Implementation of REST API follows four basic design principle- use HTTP methods explicitly, stateless nature, expose directory structure- like Urls and transfer XML, JSON or both. It used HTTP function explicitly and such a way that's stable with the definition of protocol. The basic design principles of the REST API generates a one to one mapping between GET, POST, PUT and DELETE verbs and HTTP functions.

It is important to note that REST API is a style of software architecture as opposed to a set of standards. As a result, such applications or architecture are sometimes referred to as a RESTful or REST style applications or architectures. An application or architecture considered RESTful or REST style is characterized by- 1. State and functionality are devided into distributed resources, 2. Every resource is uniquely addressable using a uniform and minimal set of commands. 3. The protocol is client/server, stateless, layered and supports caching.

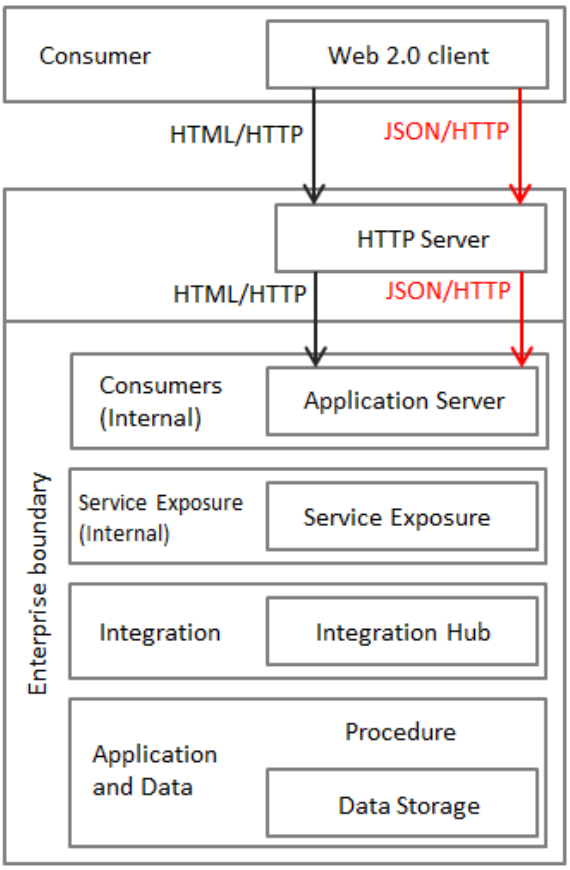

Fig.3.1. REST API Architectural overview

4. Experimental Details

Handling millions of request per minute from millions of IoT end points is the massive challenge. In this case, number of servers used, processing cost and power are the core factors that need to be considered. The goal of this research is to be able to process a large number of POST request from millions of IoT devices. Request as a service (RAAS) tools have been used for the experimental purposes. RAAS tools are effective for performance motoring. They send bulk request as multiple end points and alert immediately receiving a response.

Firstly, a worker tier architecture has created utilizing functional programming language GO, Sidekiq, Elasticbeanstalk worker tier, Rosque and Rabbit MQ with two different clusters, one for the IoT device end point and another one for the worker layer in order to scale up the volume of background task. Initially a naïve approach of GO routines is implemented to handle POST request parallelizing the mechanism of job processing. The process first read the HTTP response body into a string for JSON decoding and then goes through each data and queue items individually to write into the storage.

Secondly, the lifetime of the incoming POST request handler is kept short. In this case, instead of reading the body of HTTP request into a string for JSON decoding a buffered channel is entered to queue up several task and upload them to the server and since it's easy to maintain the maximum items in the queue. The process here only goes through each request parameter and queue item individually to be posted to the server.

Finally, a common request processing design is introduced at the time of using GO channels for creating a two layer channel system where one channel for pushing job into the queue and another one for managing the number of workers handle on the queue concurrently. The procedure parallelizes the entire workflow without generating the connection error from the associate server. At the time of initializing the web server a Dispatcher needs to be created and called in order to create the worker pool to process job from the job queue. Worker channels pool are registered with the dispatcher. Maximum number of workers is instantiated and added to the worker pool. The entire architecture follows the 12 factors methodology while configure in production environment and all associate values are read from the environment variables.

.

5. Results and Discussion

The performance of all of the approaches here is measured in Latency and Throughput. The time span between a web request and the response is known as Latency which is measured in units of time, such as seconds. One the other hand, throughput is the number of requests process per unit of times (bit transmitted per second, HTTP request handles per day, or millions of instructions per second). The approach works for moderate loads as there are no option to control the number of GO routines that are spawning and for a large number of POST request per minute both process crash very quickly. The second approach postpones the problem of serving millions of incoming POST request due to the flawed concurrency that occurs with a buffered queue. The synchronous processor is only responsible for uploading one request parameter at a time and when the rate of POST requests are larger then buffered channel reaches its maximum limit very fast blocking the functionalities of request handler (queue items).

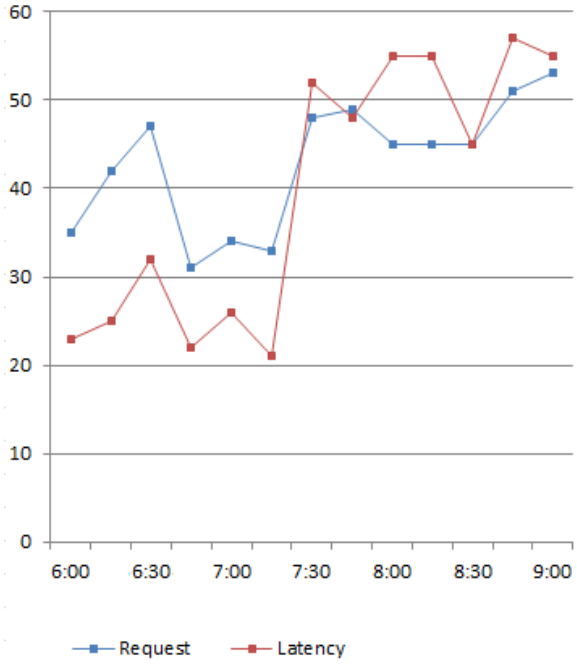

Fig.5.1. Web request vs. Latency chart- Latency increases for large number of web request

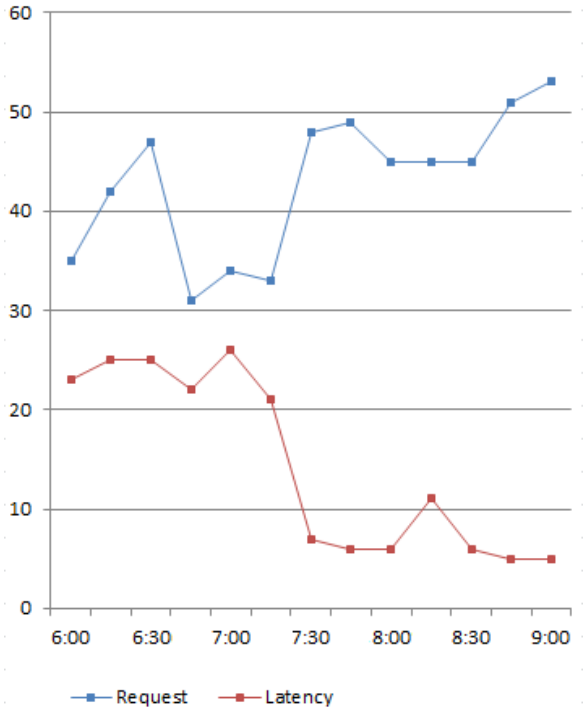

Fig.5.2. Web request vs. Latency chart- Latency decreases for large number of web requests

6. Conclusions

With the rise of web-based APIs, we have come to think of REST (Representational State Transfer) as being synonymous with JSON over HTTP. Unsurprisingly, JSON has supplanted XML as the data format of choice for the web. While early IoT technologies have embraced the JSON/HTTP mix, which could soon be changing. The concept of REST will survive, but JSON and HTTP may not be the common language of IoT data interchange for much longer. Loose coupling is one of the vital issues while thinking about the API performance. The expense is of course latency, particularly when IoT devices are not very close to the servers. Each device may need to submit several requests to the server in order get its job done.

7. Acknowledgements

The experiment and result discussion presented in here were collected while working with REST API performance issues for large number of IoT endpoints. This research was supported or partially supported by Software Ravine. Software Engineering team of Software Ravine who provided insight and expertise that greatly assisted the research. It is really essential to thank Ehshanul Hasan, Software Engineer for comments that greatly improved the manuscript. We would also like to show our gratitude to the (Name Surname, title, institution) for sharing their pearls of wisdom with us during the course of this research.

8. References

[1] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, “Research on the architecture of Internet of Things”.

[2] D. Lake, Rodolfo Milito, Monique Morrow, and Rajesh Vargheese, “Internet of Things: Architectural Framework for eHealth Security”, January 2014.

[3] T. Hejwowski, and A. Weronski, “The effect of thermal barrier coatings on diesel engine performance”, Vacuum,vol.65, pp.427-432, 2002.

[4] Vangelis Gazis, Manuel Görtz, Marco F. Huber, Florian Zeiger, “IoT: Challenges, Projects, Architectures”, February 2015.

[5] Computer and Information Sciences, King Saud University, “A survey on Internet of Things architectures”, 08 October 2016.

[6] Directions Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswamia “Internet of Things (IoT): A Vision, Architectural Elements, and Future”.

[7] G M Tere, R R Mudholkar, B T Jadhav, “Improving Performance of RESTful Web Services”, IOSR Journal of Computer Science (IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727 PP 12-16.

[8] Min Choi, “A Performance Analysis of RESTful Open API Information System”, DOI: 10.1007/978-3-642-35585-1_8.

[9] Snehal Mumbaikar, Puja Padiya, “Web Services Based On SOAP and REST Principles”, International Journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013 1 ISSN 2250-3153 www.ijsrp.org.

[10] Dominique Guinard, Iulia Ion, and Simon Mayer, “In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers’ Perspective”.

[11] Joel L. Fernandes, vo C. Lopes, Joel Rodrigues, Sana Ullah, “Performance evaluation of RESTful web services and AMQP protocol”, July 2013, DOI: 10.1109/ICUFN.2013.6614932, Conference: Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference on.