DESIGN AND IMPLEMENTATION OF A LIGHT WEIGHT, SCALABLE AND ASSISTIVE APPLICATION PROGRAMMING INTERFACE FOR INTERNET OF THINGS

Ahesh Perera

(168251M)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 2018

DESIGN AND IMPLEMENTATION OF A LIGHT WEIGHT, SCALABLE AND ASSISTIVE APPLICATION PROGRAMMING INTERFACE FOR INTERNET OF THINGS

Hetti Arachchige Ahesh Suranga Perera

(168251M)

Thesis submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

June 2018

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: …… Date: …….

Name: H.A.A.S. Perera

The above candidate has carried out research for the Masters Thesis under my supervision.

Name of the supervisor: Dr. Indika Perera

Signature of the supervisor: ……. Date: …….

Abstract

Cloud computing and Internet of Things (IoT) brings various physical devices which generate and exchange data with the services promoting the integration between the physical world and the computer world into a single common page. Together they have been providing various applications, use cases and services over the past few years, that has made a significant benefit on both industrial applications as well as day to day needs of humans.

On the other side of the coin, programming of the IoT based applications has become very challenging due to the vast knowledge base required in various technical domains, from low-power networking to the embedded operating systems, from low level calculations to the distributed algorithms and so on. It is certain that a well designed, reliable and scalable, easy configurable and high performance Application Programming Interfaces (APIs) are much needed in this paradigm to offer sophisticated services for an IoT cloud. APIs are generally exposed to its consumers as service endpoints to get pre-defined jobs done, and are offering convenient ways for developers to design and implement applications as well as vendors (OEMs) to design and manufacture their devices.

In this research I have mainly focused and discussed about the true challenges, issues and the concerns that we may face when designing and implementing high performance APIs for IoT cloud. I have also elaborated the technical and theoretical limitations come along with the performance issues in such APIs. Most importantly I have tried to design a platform for small start-ups who start developing their IoT based products with a limited knowledge, time, funds and resources so that they can build their products without worrying about the production level challenges in terms of scaling and performance once the business is grown up.

This research will provide a solution for most of the challenges when it comes to IoT cloud in terms of self configurations and elasticity with auto scaling whilst keeping better performance. Considering the massive variety of devices and the resource constraints we have in IoT, an architecture has been proposed for devices to be self-configured to the maximum extent with the API. The proposed solution will have a well designed RESTful API which comes in plug-and-play mode with developer convenience, supporting horizontal scaling as and when needed. In a nut-shell this gives a framework which takes care of all the architectural level challenges and best practices in IoT cloud where the engineering team focuses more on the business and the product.

ACKNOWLEDGEMENTS

I am grateful to Dr. Indika Perera, my supervisor for accepting my research under his supervision and for the guidance, continuous support and the direction given throughout to make this research a success.

My sincere appreciation goes to my family for the support and the motivation given for making this thesis a success.

I would also like to thank my colleagues at work place, Sysco Labs for spending their valuable time with me to discuss about my research and opening gates to discover new areas.

Finally, I wish to thank all the academic and nonacademic staff of Department of Computer Science and Engineering, University of Moratuwa and my colleagues of MSC'16 batch for the support and encouragement provided throughout past 2 years.

TABLE OF CONTENTS

DECLARATION i

Abstract ii

ACKNOWLEDGEMENTS iii

TABLE OF CONTENTS iv

LIST OF FIGURES viii

LIST OF TABLES x

LIST OF ABBREVIATIONS xi

Chapter 1 INTRODUCTION 1

1.1 Background 2

1.2 APIs are Driving the Internet of Things 3

1.3 Common IoT Challenges 4

1.4 Problem Statement 5

1.5 Motivation 6

1.6 High Level Research Objectives 8

Chapter 2 LITERATURE REVIEW 10

2.1 IoT based related work 11

2.2 Web Service Protocol for Interoperable IoT Tasking Capability 11

2.2.1 Capabilities of IoT 12

2.2.2 IoT Architecture 13

2.3 Integrated Middleware Framework for Heterogeneous Internet of Things 14

2.3.1 Requirements of M2M APIs for IoT Architecture 15

2.3.2 Convergence of M2M APIs to RESTFul Web services 16

2.4 Web API Management Meets the Internet of Things 17

2.4.1 Challenges for the Internet of Things and Web APIs 18

2.4.2 IoTGw - an API Gateway for IoT protocols 18

2.5 Problems and Limitations when designing a WEB-API of IOT 20

2.5.1 Implementation issues 20

2.6 A Self-Configuration Architecture for Web-API of IoT 21
2.6.1 Requirements of WEB APIs in IoT 21
2.6.2 Related work 24
2.6.2.1 ThingSpeak 25
2.6.2.2 NimBits 25
2.6.2.3 Cosm 25
2.6.2.4 SensorCloud 25
2.6.2.5 Evrythng 26
2.6.2.6 iDigi 26
2.6.2.7 GroveStreams 26
2.7 Ad Hoc Networks 27
2.7.1 Data Confidentiality 28
2.7.2 Privacy 28
2.7.3 Trust 29
2.8 RESTful Sensor Data Back-end 31
2.9 WSO2 IoT Server 33
2.9.1 Architecture 34
2.9.2 Limitations for the target group 36
Chapter 3 METHODOLOGY 37
3.1 Proposed Solution 38
3.2 Components 40
3.2.1 Cloud API 40
3.2.2 Front End Dashboard 40
3.2.3 Device / Agent 40
3.3 Cloud API 41
3.4 Technology stack 42
3.4.1 Node JS 42
3.4.2 JSON 43
3.4.3 Mongo DB 44
3.5 High Level Modular Architecture 44
3.6 How does this work? 46

3.6.1 IoT devices registration 46
3.6.2 Agents registration 46
3.7 Evaluation Plan 48
3.7.1 API performance 48
3.7.2 Ability to scale 48
3.7.3 Accuracy of the configuration scripts 49
Chapter 4 SOLUTION ARCHITECTURE AND IMPLEMENTATION 50
4.1 Solution Architecture 51
4.2 Implementation 52
4.2.1 Cloud API 53
4.2.2 Agent API 54
4.3 How does Scaling work? 55
4.3.1 Database layer 56
4.3.2 Configurations Scripts store 56
4.3.3 API Service layer 56
4.3.4 Congestion Controller / Queue Management Layer 56
4.3.5 Front End Application 57
4.3.6 Analytics Engine 58
4.4 Sample Results 58
4.4.1 How will Agent send data? 59
4.4.2 How will Cloud API receive data? 59
4.4.3 How will Database save data? 60
4.5 Best Practices 61
Chapter 5 SYSTEM EVALUATION 63
5.1 How was the evaluation done? 64
5.2 Tools used for the evaluation 65
5.2.1 How K6 works? 65
5.2.2 Sample Evaluation Results for API Performance 67
5.3 Ability to Scale 70

5.3.1 What is PM2? 70
5.3.2 Sample Evaluation Results in terms of Scaling ability 71
Chapter 6 CONCLUSION 77
6.1 Research Contributions 78
6.2 Research Limitations 78
6.3 Future Work 79
REFERENCES 80

LIST OF FIGURES

Figure 2-1 : Architecture of Internet of things 14
Figure 2-2 : Overall System Architecture 19
Figure 2-3 : Graphical representation of security challenges in Internet-of-Things 30
Figure 2-4 : Overview of the first PHP-based RESTful IoT Back-end prototype 32
Figure 2-5 : High level System Architecture of WSO2 IoT Server 35
Figure 3-1 : High-level Architecture of the system 39
Figure 3-2 : Modular Architecture of the API 45
Figure 3-3 : Registration process 47
Figure 4-1 : Components diagram of API 52
Figure 4-2 : Source Structure of Cloud API 53
Figure 4-3 : Source Structure of Agent API 54
Figure 4-4 : Components level scaling architecture 55
Figure 4-5 : LM-35 58
Figure 4-6 : Console logs when agent sends data to Cloud API 59
Figure 4-7 : Console logs when Cloud API receives the same data 60
Figure 4-8 : Database snapshot of current data set 61
Figure 5-1 : 10 Virtual users send request per second for 5 seconds 66
Figure 5-2 : 1000 Virtual users send request per second during 10 seconds for 4 times. 68
Figure 5-3 : Average values for 1000 users , send requests during 10 Seconds for 4 times 69
Figure 5-4 : PM2 shows a single node process is running in cluster mode 71
Figure 5-5 : HTOP shows how the CPU and the memory are utilized for single node process 71
Figure 5-6 : Average values for 1000 users , send requests in 10 Seconds in a single instance 72
Figure 5-7 : HTOP shows how the CPU and the memory are utilized for two node processes 72
Figure 5-8 : PM2 shows that 2 node processes are running in cluster mode 72
Figure 5-9 : Average values for 1000 users, send requests during 10 Seconds in 2 instances 73
Figure 5-10 : HTOP shows how the CPU and the memory are utilized for 4 node processes 73
Figure 5-11 : PM2 shows that 4 node processes are running in cluster mode 73
Figure 5-12 : Average values for 1000 users , send requests during 10 Seconds in 4 instances 74

Figure 5-13 : HTOP shows how the CPU and the memory are utilized for 8 node processes 74
Figure 5-14 : PM2 shows that 8 node processes are running in cluster mode 74
Figure 5-15 : Average values for 1000 users, send requests during 10 Seconds in 8 instances 75
Figure 5-16 : Sent and Received data variation against number of clusters 76

LIST OF TABLES

Table 3-1 Initially identified modules 38
Table 5-1: K6 Built in Matrices 67

LIST OF ABBREVIATIONS

Abbreviation	Description
OEM	Original Equipment Manufacturer
WWW	World Wide Web
IoT	Internet of Things
RFID	Radio Frequency Identification
ITU	International Telecommunication Union
API	Application Programming Interfaces
M2M	Machine to Machine
REST	Representational State Transfer
SDK	Software Development Kit
SLA	Service Level Agreement
HTTP	Hypertext Transfer Protocol
MQTT	MQ Telemetry Transport
CoAP	Constrained Application Protocol
IOT-OAS	IoT Open Architecture System
ROM	Read Only Memory
XML	Extensible Markup Language
YAML	Yet Another Markup Language
JSON	JavaScript Object Notation
URI	Uniform Resource Identifier
IP	Internet Protocol
IDE	Integrated Development Environment
LAN	Local Area Network
URL	Universal Resource Locator
XMPP	Extensible Messaging and Presence Protocol

Chapter 1 INTRODUCTION

Today Cloud computing, Semantic Web and mobile devices along with the enormous bandwidth capacity have come to the next level of the WWW which is now known as an enormous global computer. The Web has come a long way over the past 10,000 days, and the predictions of Kevin Kelly has become true in many ways [47]. According to him our mobile phones, tablets, watches, wearable, TVs and all portals are coming into this single supercomputer, known as the Web. The most impactful change that has happened since Kelly's Ted talk in 2007 [47], has been the massive explosion of mobile devices. Siri from Apple Inc. [48] is an amazing example for this where he predicted a years ago.

The Internet of Things (IoT) is an infrastructure that interconnects uniquely-identifiable devices though the Internet. It is not a brand new concept though it is attracting attention from various fields. From the early stage of the 21st century when the internet was not so popular and had no bandwidth like today, similar concepts were proposed, and some related communication technologies like the radio, barcodes, the Internet and radio frequency identification (RFID) were also invented [1,2]. But those ideas weren't so popular and come to an actual implementation level due to the limitations we had back in the days as above.

At the very early stage of the IoT, the main focus was to identify and track every physical thing, and many applications such as warehouse management and logistics applications applied RFID technology to prove the concept [1,3] but not in the mass scale level as an industry.

1.1 Background

The Internet of Things is the part of the Internet that is made up of "*Uniquely Identifiable Embedded Computing Devices*" as Wikipedia states. Just like the World Wide Web runs over the Internet, so does the IoT. Similarly, the way of becoming the Web is a mesh of computers, so does the IoT. The billions of devices in the IoT such as

the little computers embedded in thermostats, house keys, baby monitors, trash cans, fire extinguishers give us insights and access to a massive amount of data of the environments where those devices are placed.

With the advances of communication and sensor technologies in the recent past years, the definition and scope of IoT have been extended. For an example, the International Telecommunication Union (ITU) defined the IoT as "*A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*" [4].

For the IoT to take off and grasp the industry at a higher level, the programming of IoT devices needs to be as easy as scripting a simple Web application. At the same time for IoT to be useful, the devices that make up this mesh of computers must be connected to the cloud where users can access from anywhere. The easiest and efficient way that we can achieve this is through an Application Programming Interface (API).

Cloud-based services are the way in which the IoT is connected to the data we collected from various IoT devices. APIs are the bridges which have IoT on one side, useful information and plenty of data crunching capabilities on the other side. APIs make IoT useful, turning limited little things into a powerful portals of possibilities.

1.2 APIs are Driving the Internet of Things

APIs are the inter-connector which provides the interface between the Internet and the Things. Java World's Andrew Oliver [49] calls APIs "*The glue and interesting part where the Internet of Things starts to become useful and more than a buzzword.*" APIs are exposing the data that enables multiple devices to be combined and connected to solve new and interesting workflows. A solid, well-designed Machine to Machine (M2M) API will provide the basis for the simplified management of resources.

Additionally, the main advantage of such an API is that it is providing the abstraction layer necessary to realize the interactions between IoT devices uniformly. The starting point for defining the actual services for the IoT endpoints should be exposed via this M2M APIs.

Inevitably the Internet of Things will need to engage with Web APIs. Of course, many IoT devices have been already doing this, but the current usage has become very limited compared to the the full potential. Currently, most of the IoT devices are connected to services that are created by the provider of the hardware, and so they are using private APIs. There are a set of companies that are providing common cloud services and corresponding APIs for IoT such as Xively [5]. Also there are some emerging API standards for IoT devices to communicate with other sources such as HyperCat [6]. However, the strength of the IoT will be emerged when data from multiple sources can be aggregated, analyzed and acted upon. This will create a much greater demand for IoT devices to communicate with open Web APIs.

1.3 Common IoT Challenges

As more and more devices are connected to the internet, however, certain recurring problems must be solved. Overcoming the challenges presented by the IoT is hard. There is a lot of protocols to know, computer engineering to do, odd legacy systems to deal with, and a bunch of bit twiddling. Providing an API that achieves critical mass is also really hard.

Platform dependency and the necessity of having vast area of knowledge base are major issues we need to address. We can either continue to address these over and over again, or we can develop a common solution like a framework for the challenges introduced by the IoT everyday.

An important part of solving these problems is being addressed at the API layer. If we can make APIs interoperable, secure, scalable, well-documented, and discoverable, we have come a long way in solving many of the difficulties brought by the IoT. We also need to find reusable ways of building secure and persistent, real-time communication between these cloud-based services and the little devices running on the IoT.

1.4 Problem Statement

The research challenges I have identified of which will prevent us having a single self-configurable easy set-up platform or an Application Programming Interface for IoT can be pointed out as below.

1. What is the best architecture of the API where we have all the required modules at the minimal level to start off an IoT start-up?

2. Does our initial system allow us to continue adding more devices as we move on while supporting to different types of devices to be self-configured as easy as plug and play?

3. If the business needs to scale it up over time, does our initial product or the platform have all the required components which supports scaling up with a minimal time and effort?

4. Can our system be high performance all the time?

5. Does this product facilitate for a dashboard of reliable tracking and monitoring of the connected devices for end users?

Therefore, it's clearly visible that there is a high demand and a need of having a cloud based API where users can plug and play with their devices with minimal installations as well as auto configurations up to the maximum extent given that the API is capable enough of supporting different types of IoT devices and sensors.

At the same time the API should be able to scale up whenever there is a demand in compliance with the industry standard and the best practices so that the developers can pay more attention on their business rather than spending time to re-design the existing product.

1.5 Motivation

Even though there are many researches going on in this particular area of the field and many work have already been carried out by the other researchers, still there is a plenty of things left to discover. Also it's always a challenging to deal with IoT based applications where everyday we see new things coming in to the market and the evolving industry, hence people are always in a need of getting a one single platform where they can easily plug and play with the new devices, sensors and so on.

IoT will allow new business sectors to emerge and new products to be created. Embedded electronics and everyday objects will come together to create new products. When the smart devices are created to follow universal standards, it may perhaps also give rise to services similar to app-stores. With a number of interconnected smart devices available, the users will be able to buy, download and install software to get new functionality.

For manufacturers and companies, IoT will bring wide applicability in existing sectors and provide new opportunities. For instance, by adding Radio- Frequency Identification (RFID) to products, or carts moving products around, companies are able to track their journey through the supply-chain and monitor parameters like temperatures and bacterial composition. This allows retailers to keep track of their inventory in real-time and guarantee the required quality final products, which will benefit both companies and consumers.

Smart clothes monitoring health parameters, could serve workers in the health sector and the patients by for instance combining various historical data to provide better understanding of each individual's health. With a common underlying communication platform, IoT technologies will open up new business opportunities that will give rise to new cross-cutting applications and services.

This area has a lot of business value as IoT is getting more popular day by day. Today we see there is a lot of new start-ups are coming in in to the industry focusing on IoT, but on the other hand they are lacking of the right resources, right guidelines towards a successful and a reliable platform to start off at the beginning without considering the facts which will come up later when the products get matured. So it's a high cost for them to decide what platform they need to build their product on at the early stages in case if it needs a re-design or sometimes a re-engineering from the scratch.

Having said that, the exponential growth of the new technologies and the vast number of opportunities over there with the current trends, Internet of things forced me to think of a single, reliable and self-configurable API where most of the developers are dreaming of.

1.6 High Level Research Objectives

As per the research challenges I have identified and mentioned in the problem statement section, the following list will be the high level objectives and the outcome of my research.

- Identify the minimal set of components and the modules for the API to have in order to start off an IoT based applications,
    - provides the easiness of starting the development
    - all the required modules should be there in case of scaling up later on

- Identify the limitations and the possible issues we get when we want to add different devices from different vendors into the same system as we move on.

- Design a way of avoiding such limitations and a mechanism to continue adding different devices whilst the API functions with high performance.
- Design and implement configuration scripts which allow multiple IoT devices to be self configured up to the maximum extent as easy as plug and play with minimal manual inputs.

- Design and implement an API which fulfils all of above objectives and supports thousands of devices concurrently.

- Design and implement of a front end web application with a dashboard where users can track the devices in terms of health and related information as well as to generate reports as required.

- A comprehensive guideline and a documentation for the developers for easy integration.

The next chapters will walk you through about how I achieved above objectives across this research.

Chapter 2 LITERATURE REVIEW

2.1 IoT based related work

As many of the related research papers elaborates, the IoT is not only having a broad impact on our everyday life in the near future, but also create a new ecosystem involving a wide array of players such as device developers, service providers, software developers, network operators, and service users. It facilitates the entrance into the IoT related mass market, and establishing a global IoT ecosystem with the worldwide use of devices and software.

Since the approaches towards an Internet of Things span various research fields. Here, I have summarized in particular how existing IoT based applications have usually been developed, what are the limitations and problems the researchers have come across, how they have tried to resolve them while catering for the high demand with a reliable services and also the opportunities that are available.

2.2 Web Service Protocol for Interoperable IoT Tasking Capability

Currently, IoT devices created by different manufacturers follow different proprietary protocols and are locked in many closed ecosystems. This heterogeneity issue impedes the interconnection between IoT devices and damages the potential of the IoT.

To address this issue, this research [7] proposes an interoperable solution called tasking capability description that allows users to control different IoT devices using a uniform web service interface. This research demonstrates the contribution of the proposed solution by interconnecting different IoT devices for different applications.

2.2.1 Capabilities of IoT

In general, the IoT has two main capabilities [7]:

- Sensing capability

    - The sensing capability monitors devices' statuses or the environmental properties of their surroundings. People can use different sensors to collect not only the environmental properties like temperature, humidity, and location information, but also the status of devices such as "On" or "Off". Generally, the sensing capability allows users to remotely monitor device statuses and various properties through the internet, and consequently, users can utilize the sensor observations to support automatic and efficient applications [7].

- Tasking capability

    - The tasking capability allows other devices or users to actuate devices via the Internet so the users can easily control the devices to execute feasible tasks remotely. While the sensing capability allows users to continuously monitor the statuses of devices and the environmental properties, the tasking capability can help users to make adjustments accordingly by controlling devices remotely [7].

2.2.2 IoT Architecture

To understand and define the scope of this research, we need to look at the IoT architecture. As shown in Figure 2.1, four main layers can be seen as Device Layer, Gateway Layer, Web service Layer and the Application Layer [2].

- Device Layer

    - Contains the devices connecting to the Internet, such as appliances and smart sensors. With the ability to connect to the Internet, devices can upload sensor observations to a web service or be controlled by users via the Internet. However, devices can be divided into two types. The first type of device has enough computation resource to directly connect to the Internet. The second type of device is the device that is too resource-constrained to directly connect to the Internet by itself.

- Web Service Layer

    - Contains services that may receive data from gateways or directly from devices. Different services may provide different functionalities, such as data processing, data storing, data management and data querying. Application Layer is where applications retrieve the resources from the Web services and usually provide graphical user interfaces for users to operate and consume the IoT data.

- Gateway Layer

    - An additional layer called the Gateway Layer is required to serve as an intermediate layer to connect the resource-constrained devices on one end

and connect to the Internet on the other end [7]. Usually, gateways act as a translator converting the device protocol to the web service protocol and vice versa.

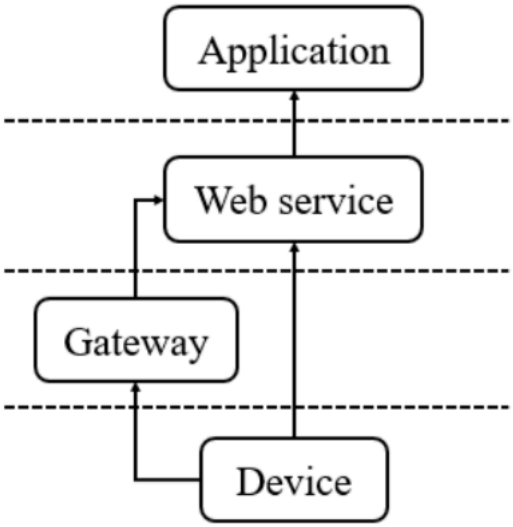


Figure 2-1 : Architecture of Internet of things

2.3 Integrated Middleware Framework for Heterogeneous Internet of Things

This paper basically talks about their journey to develop IoT framework starting from M2M APIs towards scalable service oriented architecture that leverages various opportunities to develop various applications using the same. The paper discusses about intrinsic characteristics of IoT with requirements of M2M APIs for IoT framework [10].

Some researchers think that IoT is for addressing physical objects and some think that IoT is ubiquitous with feature of “everything connected, intelligently controlled and

anywhere covered". In IoT, things can be classified as into two types i.e. physical things which include objects, behaviors, tendencies and physical events and the virtual things which include entities; actions that indicate the processing of virtual things and services that are offered for certain goals.

Machine to Machine (M2M) is paradigm in which end to end communication is executed without human intervention connecting various things to IT core network. Here, things involve commercial terminals that act automatically or on remote request [10].

2.3.1 Requirements of M2M APIs for IoT Architecture

Given the heterogeneous nature of the applications and device categories targeted in the IoT, the M2M API concept needs to be adaptable to the capabilities and requirements of the specific use case and can be categorized as:

- Requirements of communication
- Requirements of device control - includes the configuration of the device as well as support for remotely activating, deactivating or updating the device
- Requirements of server and client communication models
- Requirements of device status monitoring
- Communication failure notification
- Device Capabilities

To address these challenges, the emerging IEEE 802.11ah specifications are proposing a number of improvements and new features [13].

2.3.2 Convergence of M2M APIs to RESTFul Web services

According to the researches, we already witnessed the era of connecting machine-to-machine. Today more communication service providers are opening infrastructure to 3rd party developers through open APIs. Hence API growth rate converging towards services from Web 2.0 is increased in past few years.

- M2M
    - In M2M, the problem is constrained devices might not be connected all the time and thus they cannot immediately interact to all transactions in the network.
- REST
    - REST is based on concept of resources identified by URI, hence it provides placeholder to M2M device to store their states and data.

With the help of handling transactions in resource based communication, the REST based architecture provides efficient solution to the problem in M2M network.

2.4 Web API Management Meets the Internet of Things

In this paper they have outlined the challenges of working with Web APIs for large scale Internet of Things (IoT) projects. The key aspects of Web APIs and the active management of them are:

- Publication of metadata
- Access control and key management aspects
- Monitoring and Monetization of the interactions as well as throttling of usage

They have proposed a model for solving these issues and also have outlined the creation of a prototype system that implements that model with evaluated performance results [11].

API Management is an emerging area that aims to solve multiple challenges with Web APIs. These challenges include [11]:

- Publishing details of the APIs, documentation, SDKs and other human and machine-readable material in a portal aimed at developers.
- Allowing developers to sign up, define application clients, subscribe to APIs and test out Web APIs.
- Managing access control and authentication of API clients using “API keys” or tokens.
- Throttling traffic to specific clients based on a Service Level Agreement (SLA)
- Monitoring the usage of specific clients in order to be able to limit access or charge back for API usage.

In the area of providing API management for non-HTTP protocols, there is some work on identity and access management.

However, these do not address the wider issues around API Management including monitoring, key issuing, developer portals, and monetization.

2.4.1 Challenges for the Internet of Things and Web APIs

There is no accurate number of connected devices, but the best estimates all agree that there are more devices currently than humans on the planet. Cisco forecasts that there will be multi billion connected devices by 2020 [12].

The available approaches in the market do not address two main aspects.

1. Firstly, the Web APIs for IoT are all aiming to become the dominant or leading API for IoT. This is a common pattern in emerging technologies where there is a battle between competing standards to become dominant. However, the history of the Internet shows us that the power of the Web is having a heterogeneous set of APIs that work together in consort.

2. The second issue is that many IoT devices are using low-bandwidth binary protocols such as MQTT and CoAP to reduce energy and work with cheaper, smaller components. There is very little work exploring this area.

2.4.2 IoTGw - an API Gateway for IoT protocols

They have built a system that allows the capabilities of existing API management solutions to be utilized with IoT protocols.

Following figure illustrates the overall system architecture of API Gateway for IoT protocols.

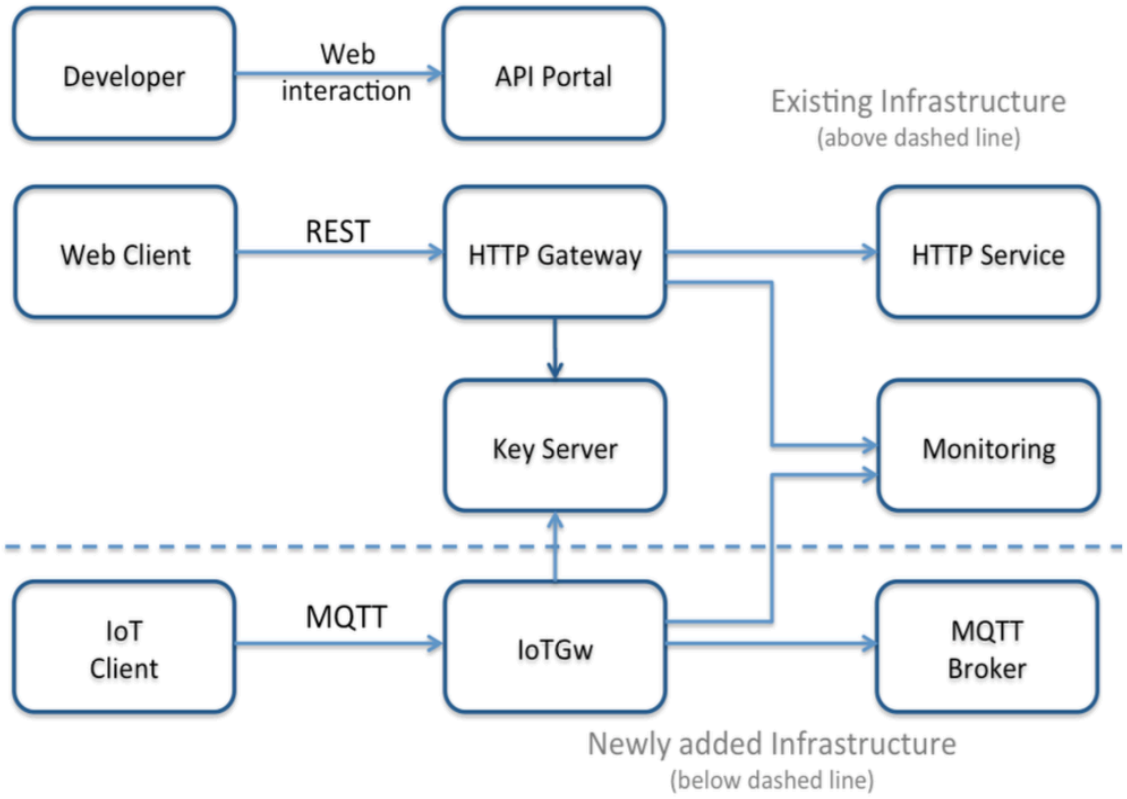


Figure 2-2 : Overall System Architecture

2.5 Problems and Limitations when designing a WEB-API of IOT

This paper talks about how different technologies like REST, cloud computing and embedded operating system in order to obtain mechanisms capable of self-configuration. As per the researches, it was possible to conclude that the Web-API proposed which increases useful techniques for the implementation of systems that want to run the self-configuration as well as assist in setting up networks of computers that work with wireless sensors and IoT. They have proposed a new Web-API for the internet of things that implements a self-configuring architecture devices [15].

2.5.1 Implementation issues

The paper states some problems were encountered when the implementation phase and only some of them have been resolved while others had to be circumvented.

1. When performing communication, it was necessary to use a proxy to perform the communication between the client and the server using the Web-API`s architecture. Once configured, the proxy, it was realized that the return of the proxy sent messages to the client did not occur. Therefore, it was found that the engine of CoAP-13 that was running on the client side was not operating with separate response [15].

   Solution: It was necessary to adapt the transaction control of *Erbium*, the programming language which had been used to implement this. That change was made through the creation of storage of all transactions in memory using a list of transactions that are now mapped mid, which is the variable responsible for the transaction ID.

2. The second problem was the implementation of the codes sent to the devices, even having received the code correctly when elf_load (file) function was called the ELFLOADER_NO_SYMTAB error occurred, which states that the symbol table was not found. [17]

   Solution: However, when generating a full symbol table using CoAP macros and other necessary overflow occurred again to the ROM memory of more than 1000 bytes. The solution to validate the architecture was to create manually a table of minimum symbols for the program to be run properly.

2.6 A Self-Configuration Architecture for Web-API of IoT

The configuration and installation of devices that will integrate a large and complex systems within the IoT is a challenge that is time consuming and error prone, even for the specialists [16].

This paper aims to introduce a mechanism of self-configuration for the Internet of Things, where the main idea is to make easier the configuration of devices and Web-APIs that will control the environment.

2.6.1 Requirements of WEB APIs in IoT

Fundamental features are described in this article in order to enumerate some of the concepts that can be used to serve as basis for self configuration mechanism, such as the form of communication with Rest (Zeng, 2011) [18], storage and standardization communication through the use of markup data languages (XML, YAML, JSON) (Xively 2013) [5].

- Open source
    - Even though this doesn't help directly the devices, it was regarded as important for that in the future people will work on top of existing Web-APIs and make your code to be improved and become Customer self-configurable.

- REST
    - The REST-based architecture is considered "the true architecture of the Web" (Zeng, 2011) [18], it is based on the concept that everything is modeled as resource using the HTTPURI. Thus, customers can identify the resources they need through the URI, manipulating them through traditional HTTP commands like: PUT, GET, POST and DELETE.

    - Another important feature is that REST works with stateless requests, treating each request independently, and this may not require a server to store session information or the status as is each of the multiple acquisitions. However, state full interactions can be supported in REST through the use of hyperlinks, so the states of the resources can be transferred by means of URIs for cookies or hidden fields (Zeng, 2011) [18].

- Standardization
    - As the APIs and the devices are usually developed in different languages, it must be pre-established a format of data communication between the receiver and transmitter and how they will exchange messages to inform how the data is separated and what the content within it represent.

- Centralized Architecture

- Due to the limitations of the devices many of the activities more robust need to be sent to a server that has capacity to perform a greater load of processing and storage. Therefore, currently the Web-APIs, tend to be centered on a server that is able perform this type of activity and to communicate with a server that is receiving data and managing the devices in the network.

- Security

  - In IoT, recognition of each device with the use of traditional IPs. Despite this, only a network identification is not sufficient to ensure the safety, it is necessary a profile control to inform if this equipment has access to the service that it is requesting. As in IoT these services are provided by APIs, the controls of inflows are usually made by API-Keys.

- Self-configuration

  - The autonomic computing is inspired in the human being's nervous system. Its main objective is to develop applications that can self-according to guidelines imposed by human beings at a high level. Thus with the policies established at a high level it is possible to make the systems self-reliant to self- configure, self-healing, self-optimization and the self-protection (Kephart, 2003) [19].

  - This is also responsible for automated configuration of system components, with it the system will automatically adjust and it always will adjust based on policies of self- configuration. The self-optimizing components and systems continually seeking opportunities to improve their own performance and efficiency.

Self- healing the system automatically detects, diagnoses and repairs problems of software and hardware located. The Self-Protection system automatically if defends against malicious attacks or cascading failures.

- Code-source Device
    - As each device needs to communicate with the Web-API through the REST, many of these offer codes-sources for which the user copy and paste in Integrated Development Environment (IDE) responsible for programming the device. It is important to realize that although there may be a useful source code available for the used equipment perform the copy and paste codes for multiple appliances can be an arduous task and subject to errors even for the specialists, once that may exist dozens of these to be configured in a single environment.

- Storage
    - As the WEB-APIs are on a server that contains high processing power and storage capacity, they are usually responsible for the storage of data that is captured and transmitted by devices. For this reason, it is used the concept of Feeds (system risers). These feeders are a specific part of the API that works with the reading and writing of data from the system.

2.6.2 Related work

This paper talks about the main existing Web-APIs and what features they have related to the requirements that were previously seen.

2.6.2.1 ThingSpeak

ThingSpeak is an API for "Internet of Things" open- source that stores and retrieves data from devices using the Hypertext Transfer Protocol (HTTP) over the Internet or simply of a LAN (Local Area Network). With this API it is possible to create applications in sensors for data records in a given environment, tracking, location and social networks of "things" (ThingSpeak, 2013) [20].

The data manipulation occurs by means of its channels, which have eight fields to be fed with data numeric and alphanumeric pagers, in addition to fields such as latitude and longitude, elevation and status.

2.6.2.2 NimBits

It is a collection of software components designed to record data of time series, such as for example, the changes in temperature read by a given sensor (NimBits, 2013) [21].

This API has the drive of events (triggers) during the recording of data. In This way, it is possible to perform calculations or trigger alerts along with your receipt.

2.6.2.3 Cosm

This tool (formerly called Pachube) was developed to be a platform as a service (PaaS) for the Internet of things. With it, you can manage multiple devices through the RESTful resources, thus it is possible to deal with all the components of the API (Feeds, triggers, datastreams and datapoint) using commands via HTTP URLs.

2.6.2.4 SensorCloud

The SensorCloud is a tool storage for sensors "things". SensorCloud provides a Rest API to allow the upload of data to the server. The API implementation is based on patterns of HTTP commands. Soon, it is easily adapted to any platform (SensorCloud, 2013) [22].

2.6.2.5 Evrythng

It is a platform for powering applications or services directed by dynamic information about physical objects. Your goal is that all things must be connected, thus sets a world where all 'Thng' have a digital presence of assets on the Internet, even in social networks if desired, allowing the rapid development of Web applications using real-time information flowing from, any object in the world (Evrythng 2013) [23].

2.6.2.6 iDigi

It is a platform in the cloud for managing network devices. It offers management gateways and endpoints on the network. It presents security policies of leaders in the industry, and great scalability for the exponential growth of devices on the network (Etherios, 2013) [24].

2.6.2.7 GroveStreams

GroveStreams is one of the most powerful platforms in clouds capable of providing real-time decision making for millions of users and devices. Among several of its qualities is the code generation per device. In this API it is possible that when you choose your device and the function that it will play a code that can be used to synchronize the device with the API is generated, thus there is only a need to copy this code paste in compiler used by the device and send to motto for which the code was generated (GroverStream, 2013) [25].

2.7 Ad Hoc Networks

Daniele Miorandi [45] presented a survey article giving an overview of IoT and its research challenges. They see IoT as a major trend that may represent the next big leap forward in the Information and Communication Technologies sector.

They refer to as smart devices as physical objects associated to at least one name or address with communication abilities. In addition to that, it should possess a unique identifier and may also be able to sense physical phenomena.

In this paper they talk about the Security in IoT in terms of Data confidentiality, Privacy and trust aspects. Security represents a critical component for enabling the widespread adoption of IoT technologies and applications. Without guarantees in terms of system-level confidentiality, authenticity and privacy the relevant stakeholders are unlikely to adopt IoT solutions on a large scale.

In early-stage IoT deployments, security solutions have mostly been devised in an ad-hoc way. This comes from the fact that such deployments were usually vertically integrated, with all components under the control of a single administrative entity. In the perspective of an open IoT eco-system, where by different actors may be involved in a given application scenario a number of security challenges do arise. In this section, we aim at revising and discussing the major security challenges to be addressed to turn Internet-of-Things technology into a mainstream, widely deployed one.

2.7.1 Data Confidentiality

Data confidentiality represents a fundamental issue in IoT scenarios, indicating the guarantee that only authorized entities can access and modify data. This is particularly relevant in the business context, whereby data may represent an asset to be protected to safeguard competitiveness and market values.

In the IoT context not only users, but also authorized objects may access data. This requires addressing two important aspects,

1. The definition of an access control mechanism

2. the definition of an object authentication process with a related identity management system

2.7.2 Privacy

Privacy defines the rules under which data referring to individual users may be accessed. The main reasons that makes privacy a fundamental IoT requirement lies in the envisioned IoT application domains and in the technologies used. Health-care applications represent the most outstanding application field, whereby the lack of appropriate mechanisms for ensuring privacy of personal and/or sensitive information has harnessed the adoption of IoT technologies.

In addition, in the IoT vision, a prominent role will be played by wireless communication technologies. The ubiquitous adoption of the wireless medium for exchanging data may pose new issue in term of privacy violation. In fact, wireless

channel increases the risk of violation due to the remote access capabilities, which potentially expose the system to eavesdropping and masking attacks. Hence privacy represents a real open issue that may limit the development of the IoT.

2.7.3 Trust

The concept of trust is used in a large number of different contexts and with diverse meanings. Trust is a complex notion about which no consensus exists in the computer and information science literature, although its importance has been widely recognized. Different definitions are possible depending on the adopted perspective. A main problem with many approaches towards trust definition is that they do not lend themselves to the establishment of metrics and evaluation methodologies.

In this paper they have illustrated the security challenges come in Internet-of-Things in a single frame as below.

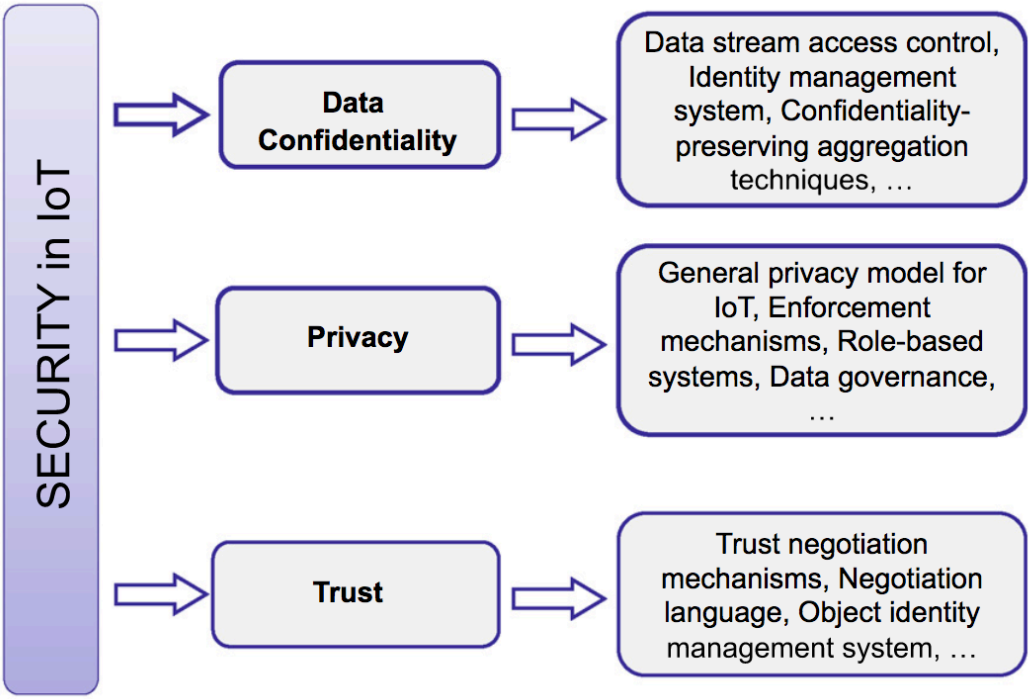


Figure 2-3 : Graphical representation of security challenges in Internet-of-Things

Summarizing, the open research challenges in terms of privacy-preserving mechanisms for IoT, as reported in above Figure 2-3 are given by:

- Definition of a general model for privacy in IoT

- Development of innovative enforcement techniques, able to support the scale and heterogeneity characterizing IoT scenarios.

- Development of solutions that balance the need of anonymity presented by some applications with the localization and tracking requirements of some other ones. This entails the definition of privacy policies, that specify under which

conditions it is possible to identify and localize a smart object. Moreover, it needs to specify when it is possible to access sensitive data.

2.8 RESTful Sensor Data Back-end

This paper [47] provides how PHP can be used to implement a RESTful back-end API for Internet of things. They talk about the benefits of REST from an IoT perspective which is 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.

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

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

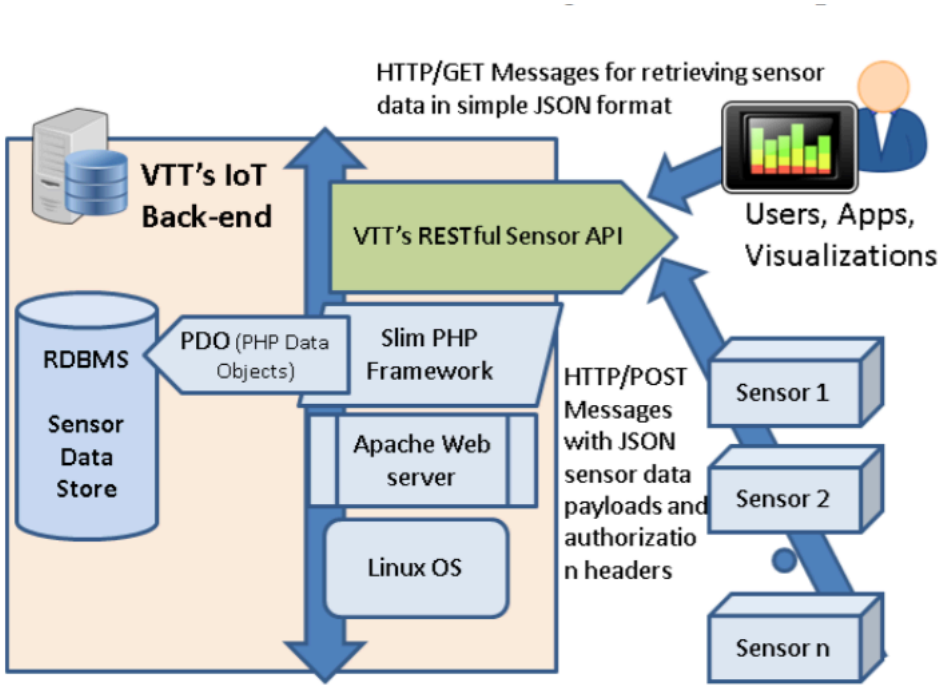


Figure 2-4 : Overview of the first PHP-based RESTful IoT Back-end prototype

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 based on REST.

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.

2.9 WSO2 IoT Server

WSO2 IoT Server is a comprehensive open source IoT solution which was released to public very recently even after I started this research. It enables enterprises to manage their mobile and Internet of Things (IoT) devices.

The WSO2 IoT platform is a combination of the following areas:

- Core offering

  The IoTS core offering is centralized around device management focusing on device plugins, event stream management and more.

- IoT Analytics

  The data gathered via the devices are analyzed to produce information that will be useful to the end-user.

- Extended Platform

  WSO2 IoTS can then be extended so that it can be used with the integration, machine learning, workflows and many other areas.

For example, the extended platform will involve the WSO2 Business Process Server (BPS) to handle the workflows and business processors in an organization [27].

WSO2 IoT Server is another product which was built on top of WSO2 Carbon therefore all the features come along with Carbon will be inherited on this too. All its capabilities are exposed through industry standard Swagger annotated REST APIs. It allows device

manufacturers to create their own device types and enroll and manage them securely. It is designed in a way to protect both devices as well as its data. It also provides analytics capabilities to gather sensor data, visualize them real time, identify patterns and convert these to responsive action. You can also extend its capabilities to securely manage mobile devices [27].

WSO2 IoT Server (IoTS) provides the essential capabilities required to implement a scalable server-side IoT Platform. These capabilities involve device management, API/App management for devices, analytics, customizable web portals, transport extensions for MQTT, XMPP and much more.

2.9.1 Architecture

WSO2 IoT server has first started out as a complete platform to manage mobile devices and later evolved to a more complex system by incorporating capabilities to manage mobile devices as well as all types of IoT devices. The initial release came with the ability to manage Android and iOS, and Windows mobile device management and application management was added later [28].

Following figure illustrates the architecture of WSO2 IoT server.

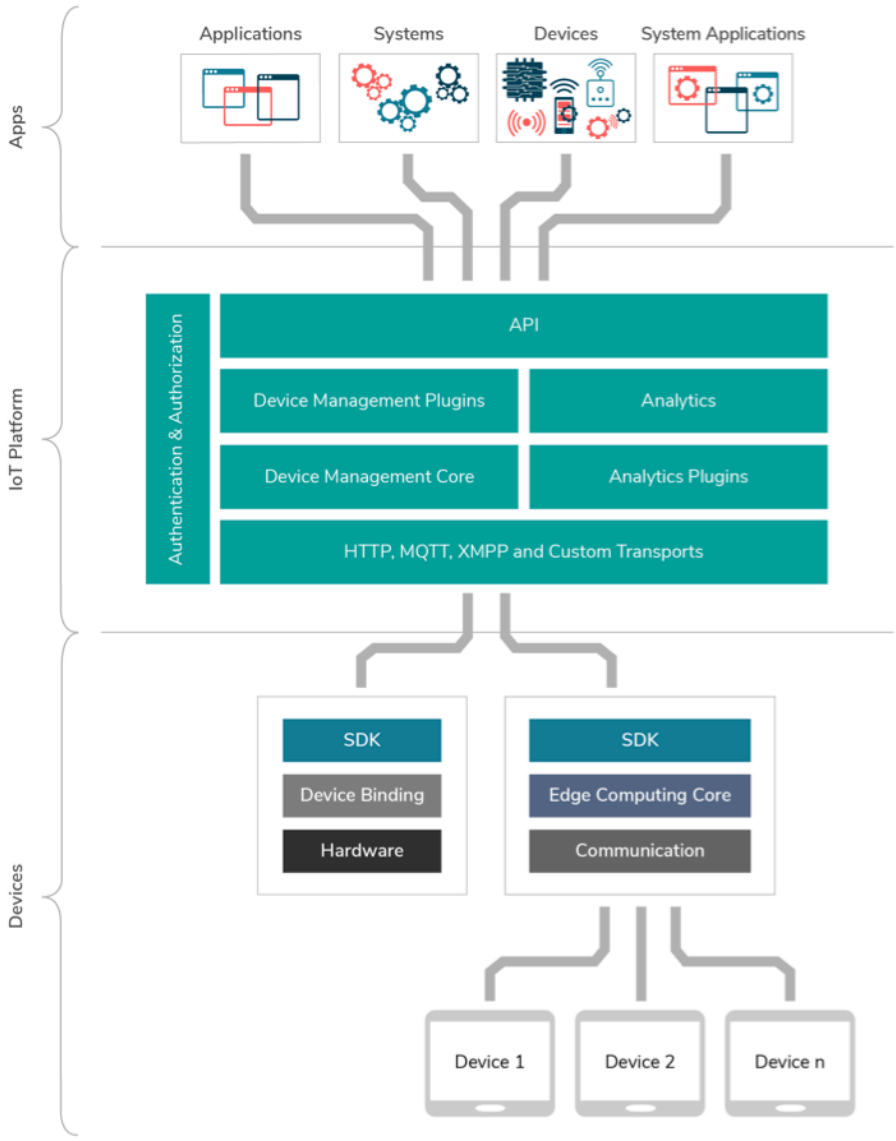


Figure 2-5 : High level System Architecture of WSO2 IoT Server

2.9.2 Limitations for the target group

As stated in WSO2 IoT server system requirements page, prior to installing any WSO2 Carbon based product, it is necessary to have the appropriate prerequisite software installed on your system. You need to make sure that your environment has the supported operating system and development platforms before starting the installation [29].

Following platform level limitations can be found at their official System Requirements page [29] where small start-ups will not be capable enough of providing all the time at their initial stages.

- *"All WSO2 Carbon-based products are Java applications that can be run on any platform that is JDK 7 or 8 compliant. Also, we do not recommend or support OpenJDK."*

- *"All WSO2 Carbon-based products are generally compatible with most common DBMSs. The embedded H2 database is suitable for development, testing, and some production environments. For most enterprise production environments, however, we recommend you use an industry-standard RDBMS such as Oracle, PostgreSQL, MySQL, MS SQL, etc. Additionally, we do not recommend the H2 database as a user store."*

- *"It is not recommended to use Apache DS in a production environment due to scalability issues. Instead, use an LDAP like OpenLDAP for user management*."

Chapter 3 METHODOLOGY

3.1 Proposed Solution

The primary objective of this research is to identify the minimal set of components and the modules for implementing a light weight, developer assistive API which is easy to start the development as well as supports scaling and allows easy integration with different IoT devices with a minimal configuration as a single platform. Therefore, the intended methodology will require a proper analysis and an identification of the minimal list of components and modules for the API in order to support scaling and the other objectives. This analysis will have a list of possible issues and challenges, performance barriers and modeling issues and also the technical limitations and relevant workarounds to overcome them.

Initially identified components and modules:

Module Name	**Description**
API Services	All the service end points exposed to out side
Databases	All the database connections / DAO layer
Consumers	Everything related to other consumers this API consumes
Helpers	Helper functions
Middleware	Middleware functionalities / Authentication & Validation functionalities
Utilities	All the utility functions
Tests	Unit testing modules / Mocked data
Configurations	All the configuration JSONs
Cluster Support	Node JS Cluster modules
Git Hooks / ES Lint	Code quality matrices
Deployment Scripts	Deployment scripts / CI CD scripts

Table 3-1 Initially identified modules

Based on the results of the initial analysis, this intended methodology will have a comprehensive architecture for the API. All the instructions required for installing and initial configurations will be scripted and stored in a common space in cloud so that any agent can make an API call for fetch the relevant configuration script.

Following figure provides a representation of the proposed **high level architecture design** and more descriptive diagrams will follow.

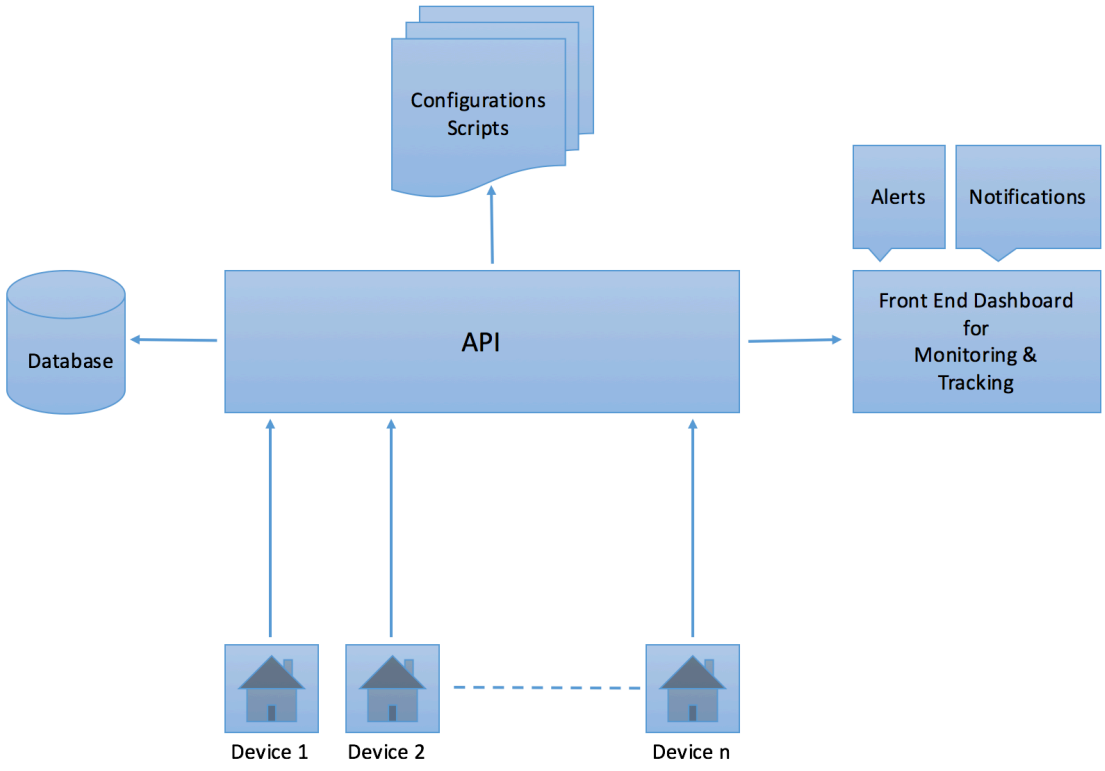


Figure 3-1 : High-level Architecture of the system

Device 1 to Device n represents the IoT layer which can be a sensor, a camera, smart home unit or a similar device which will connect to an **Agent** and will send data to the API periodically.

3.2 Components

As stated in Figure 3.1 the main components of this proposed solution are described as below.

3.2.1 Cloud API

- Includes all the API endpoints for IoT devices to connect and communicate, Database CRUD operations and serving the configurations scripts for devices to be self configured. This will also have framework level scaling support following industry level best practices.

3.2.2 Front End Dashboard

- This is a web based front end application which facilitates for tracking and monitoring all the connected devices. This application will display alerts, warnings based on the device health as well as it sends notifications where an anomaly device is detected.

3.2.3 Device / Agent

- This unit will be a minimal version of the same API as Cloud API which allows different type of IoT devices (example: sensors) to be registered

and connected with the Cloud API. Once the registration is done this can send data to Cloud API periodically. Single Agent can have one or more IoT devices connected to it as applicable.

3.3 Cloud API

The main and most important component of this IoT platform is the Cloud API. This API should have following capabilities in order to fulfil research objectives.

1. API should be light weight as well as developer friendly so that small start-ups don't hesitate to start off with it.

2. Architecture should be solid and should support elasticity, Auto scaling and micro services with minimal time and effort so that there is no re-engineering as and when the business grows up.

3. API should provide RESTFul services for IoT Agents to register with it and should be able to receive and process data with high performance.

4. API should provide RESTFul service end points for any Front End Web Application to be connected and fetch monitoring and health diagnosis information of the connected devices and relevant analytics data as well as generate reports as required.

5. API should resolve the common problems come in to picture when scaling such as session management, data consistency, high availability of the services and so on.

3.4 Technology stack

Considering the target group in which I am trying to address across this research and also the potential capabilities comes by default, Node JS seems to be the best approach for this API as the tech stack. Node JS has a massive open source community support available through NPM.

3.4.1 Node JS

Node.js is an open source JavaScript-based platform built on top of Google Chrome's JavaScript V8 Engine [40]. 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.

There are some other benefits for start-ups to go ahead with Node JS over any other tech stack,

- Node JS has a fast code execution due to the underlying Google Chrome's V8 JavaScript Engine and it has been released under the open source MIT license.

- All Node JS applications uses “Single Threaded Event Loop Model” architecture to handle multiple concurrent clients

- 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.

- When it comes to IoT there can be lots of various readings, so the application is essentially IO bound where Node JS gives much advantage.

- Relatively small code base and Regression happens quite easily. Most importantly Node JS helps Rapid Application development.

- Easy support for deployment tools and process management platforms like PM2 where it makes production deployments and scaling pretty easy and manageable.

3.4.2 JSON

A REST API needs to consume and produce data, encoded in a consistent manner. Whilst many choices exist (XML, YAML, etc.) a popular choice is JSON. JSON is well suited to data that needs to be both human and machine-readable. JSON data is, arguably, easier for a human reader to understand than XML; and can also be easily parsed by a computer. Tools and libraries to parse JSON exist in all major programming languages and environments.

JSON's key-value pair format is ideal for use with regular parametric data such as may be produced by a sensor device, or transmitted as a command-and-control message to a device or actuator.

From Node or another JavaScript implementation JSON data is already in the native format & as such requires no further parsing unlike, for example, XML.

3.4.3 Mongo DB

The advantage of the use of a document-oriented database, such as MongoDB is that it offers a dynamic schema rather than a fixed one [43]. This means that if the data structure of the database is required to change, as a result of needing to store additional parameters; new data with a different structure can be accommodated within the database, alongside earlier data, without the need to perform large scale data manipulation on the whole database.

MongoDB is also a good choice for storing JSON encoded data. MongoDB internally stores data in an efficient binary JSON format which allows for quick and easy import and export. It is also ideally suited to storing and processing large volumes of data. It can scale to thousands of nodes and petabytes of data [43].

MongoDB also exhibits better runtime performance for simple operations at a small-scale (single node), than Microsoft SQL Server Express [44].

3.5 High Level Modular Architecture

It is always a good practice to keep things simple and modular. This makes anything to be more readable and extensible.

In the high level architecture of this proposed approach, it can clearly be seen that almost all the major components are modularized as easy as to be separated out. Many

elements such as parameters, headers, and responses are shared among paths. Thus an enormous number of lines of code can be saved if proper re-use of components is used in the swagger specification [50]. Thus all shared definitions, models and parameters are reused. Code Reuse not only prevents inconsistencies and errors but also increases readability and extensibility.

The high level modularized architecture of the API is shown in the following figure.

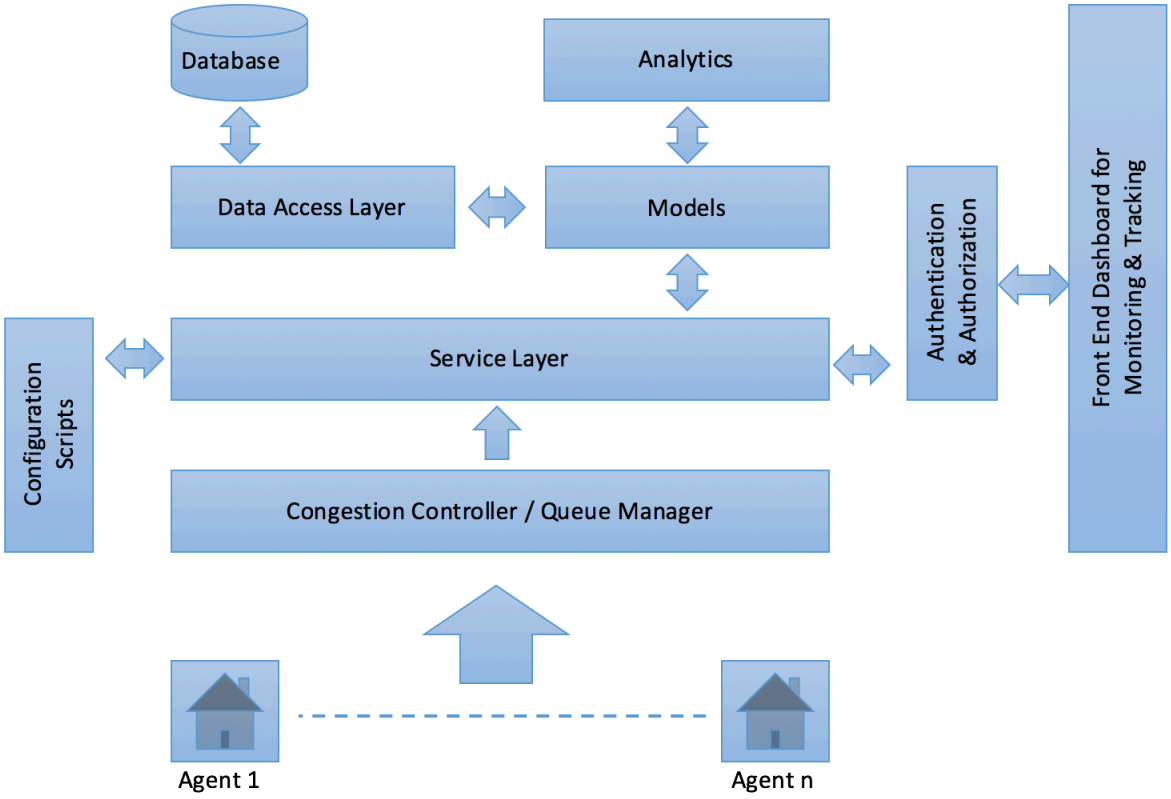


Figure 3-2 : Modular Architecture of the API

3.6 How does this work?

This system will have two main registrations:

3.6.1 IoT devices registration

When an IoT device wants to connect to the system it should register with its agent API. That registration is done by sending an authorized request and it's a token based authentication. After a successful registration the agent is ready for registering with the cloud API.

3.6.2 Agents registration

Now the agent will make a register call to the cloud API. Once the Cloud API returns the success code of REST [i.e. 200] it states that the request is authorized.
Then the required configuration modules, installations scripts will be started to download based on the information requested from the agent. This is specific to each device and it will depend on the vendor and the device types.

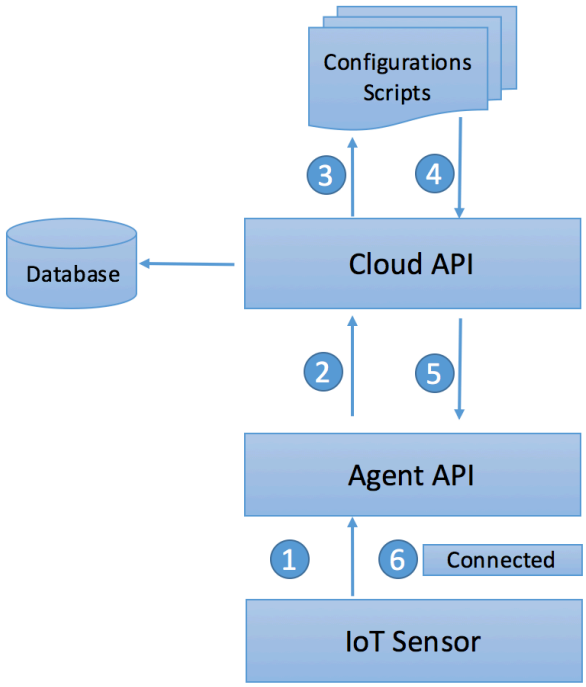


Figure 3-3 : Registration process

Once the agent who has a set of devices that has been successfully registered and configured with it, the set up is ready to function.

Next important task in this solution is to design and implement a front end web application which is more like a dashboard where users can track all the connected devices' health and monitor them on demand. React JS will be used for this application as it supports latest web browser rendering strategies, easy development as well as high performance.

This dashboard will have all the reports so that users can take some decisions based on the collected data.

Also a comprehensive guideline and a document will be provided for developers to make the integration as easy as plug and play.

3.7 Evaluation Plan

This proposed methodology can be evaluated by under three main categories in terms of API performance, ability to scale and the accuracy of the configuration scripts for different IoT devices.

3.7.1 API performance

API can be tested with a high load of requests created intentionally on virtual nodes in order to make sure it's responding as expected. In that case we can ensure that the congestion controller layer (queue manager) is capable enough of catering a large no of requests at a time.

Also we should be able to get the upper limit of no of requests where the API is getting failed to respond so that scaling can come to the action.

3.7.2 Ability to scale

When the business gets much traction it's clear that the system needs to be scaled up with a minimal time and an effort.

This proposed solution, we can consider a horizontal scaling mechanism by deploying in multiple servers with different database replications. This artifact can be evaluated by doing a critical architecture analysis along with current

trends and industrial best practices so that we can see the level of scaling ability as well as performance trade-offs if there is any.

3.7.3 Accuracy of the configuration scripts

This is kind of a tricky section but we can stick with the original configuration scripts provided by each vendor/OEM so that we can guarantee that the system will support for the devices to be self-configured as much as possible.

In this research, for the actual implementation of this methodology I have mocked the IoT device registration and data collecting layer assuming that the API receives the device data as expected so that I can focus more on the API performance and Scaling capabilities of the API.

When it comes to small start-ups the common problems they are having at their early stages is to select the right platform at the right time without worrying much and spending more time and money for analyzing available platforms. Since my main target group is someone who is lacking of such a platform, this research has skipped the configuration of IoT device layer and it has been paid more attention on resolving API level issues.

Chapter 4 SOLUTION ARCHITECTURE AND IMPLEMENTATION

4.1 Solution Architecture

The proposed solution will have the following system architecture in components level for the **Cloud API**. This will be same as for the **Agent API** also, but in small scale.

This architecture has been designed by considering a lot of industry level best practices. A few of major items can be highlighted such as;

1. Having a Daemon for initializing all the background services and handling their life cycles.

2. De-coupling all database related stuff so that database level scaling can be done easily.

3. Having a separate layer for managing the request load

4. De-coupling Authentication & Authorization as a middleware

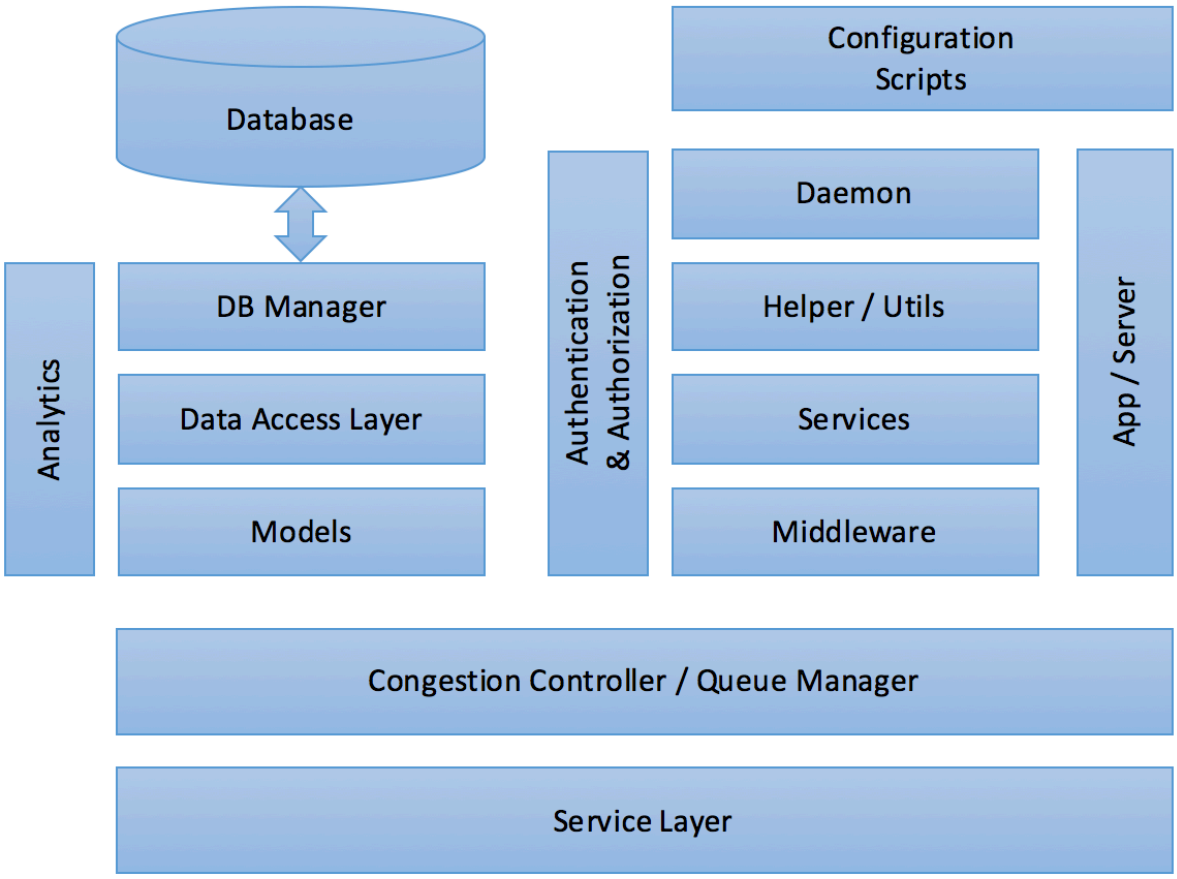


Figure 4-1 : Components diagram of API

4.2 Implementation

All JavaScript files in source root are trans-piled to be able to run on NodeJs V6. The root contains four modules, server, app, deamon and eslintrc. The server module creates the HTTP server and listens on port specified in platform config specific to the running environment. This also will kick start the daemon services and the ExpressJs application. The daemon services that run independent of the HTTP server are keeping all the required clients alive.

The app module defines the ExpressJs app configurations. All global express middle-ware are used within this module. The eslintrc module defines all the lint rules specific to the sources root. Services that are running independent of the ExpressJs app are initialized within Daemon module. These services include all daemon services that should be instantiated before starting the https server, or that can be initiated asynchronously.

4.2.1 Cloud API

The next figure illustrates how components are built and structured in Cloud API in order to be easy scalable, minimal configurations for kick off start-ups without considering much things at the beginning.

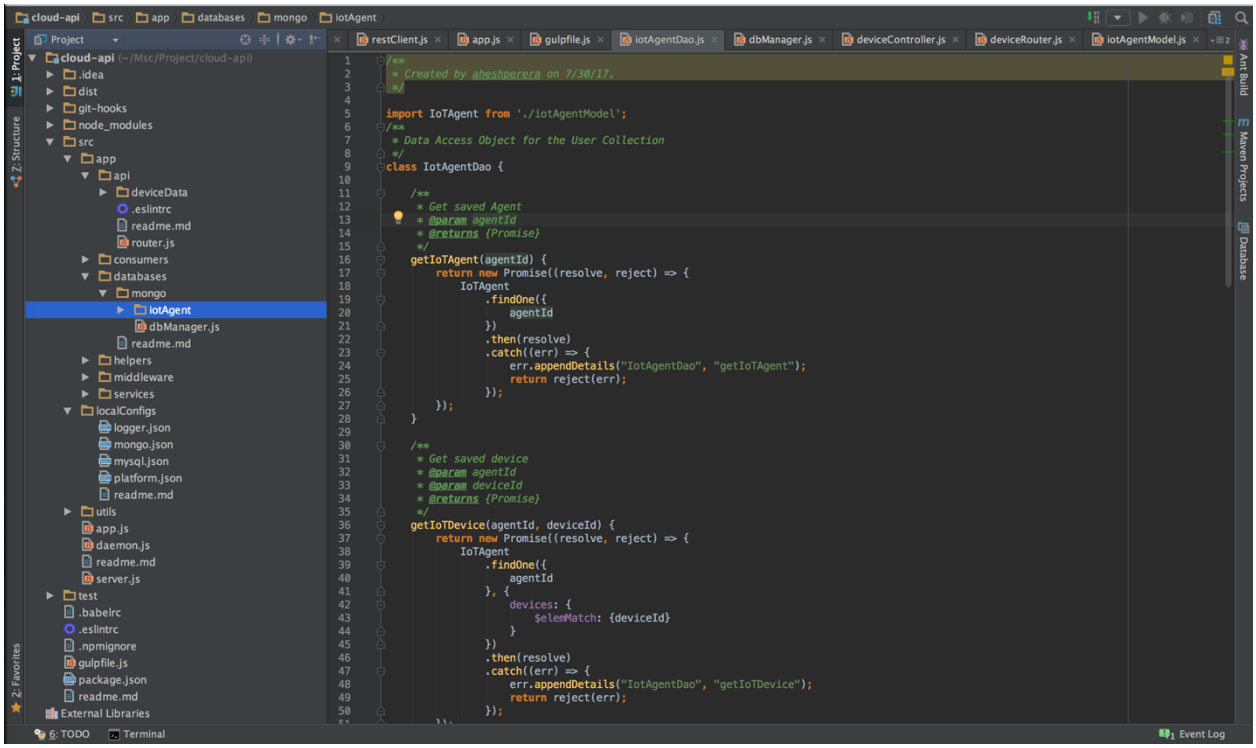


Figure 4-2 : Source Structure of Cloud API

This source will have all the required best practices such as code commit hooks, standard rules for following better coding practices and so on.

4.2.2 Agent API

Following figure illustrates the source structure of the minimal version of Agent API.

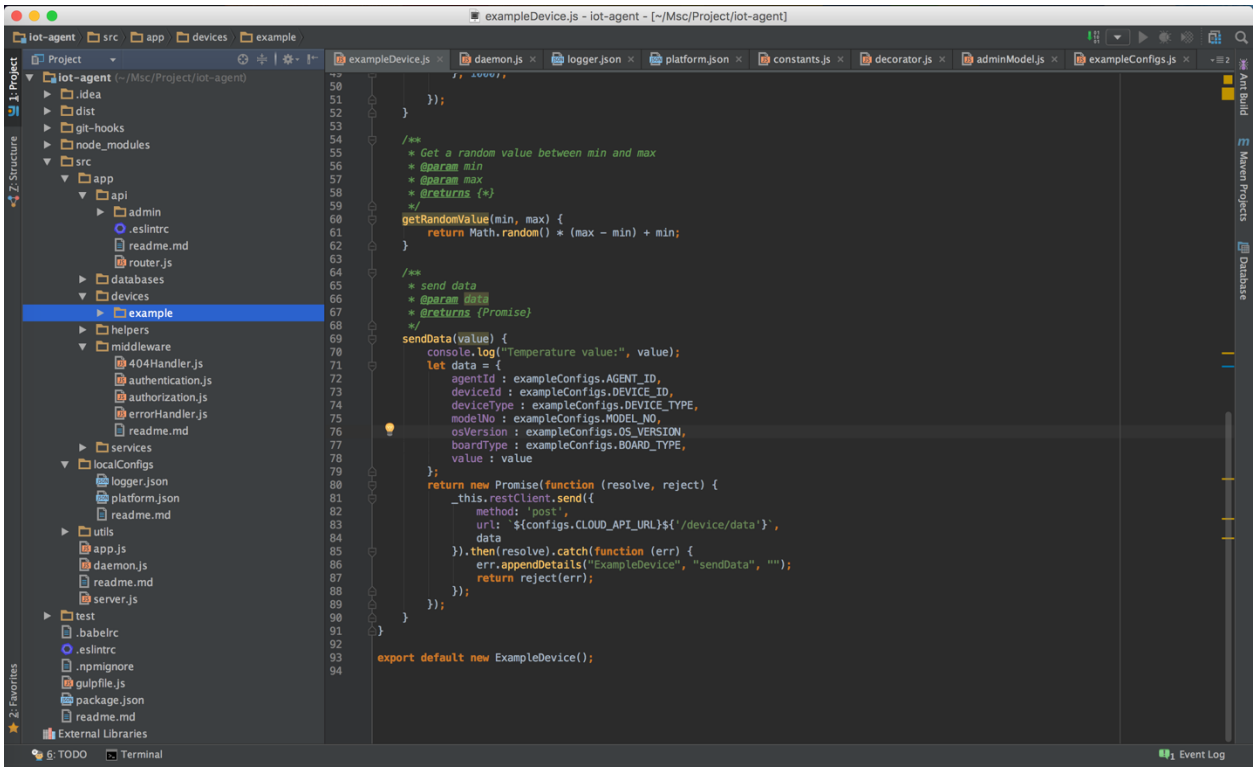


Figure 4-3 : Source Structure of Agent API

4.3 How does Scaling work?

The proposed API can be scaled up in components level as illustrated in following figure.

The boundaries are to show case what are the components from this system can be easily taken out and served in different/multiple instances depending on the deployment environment. Those instances can be AWS EC2 instances, local server machines or data centers.

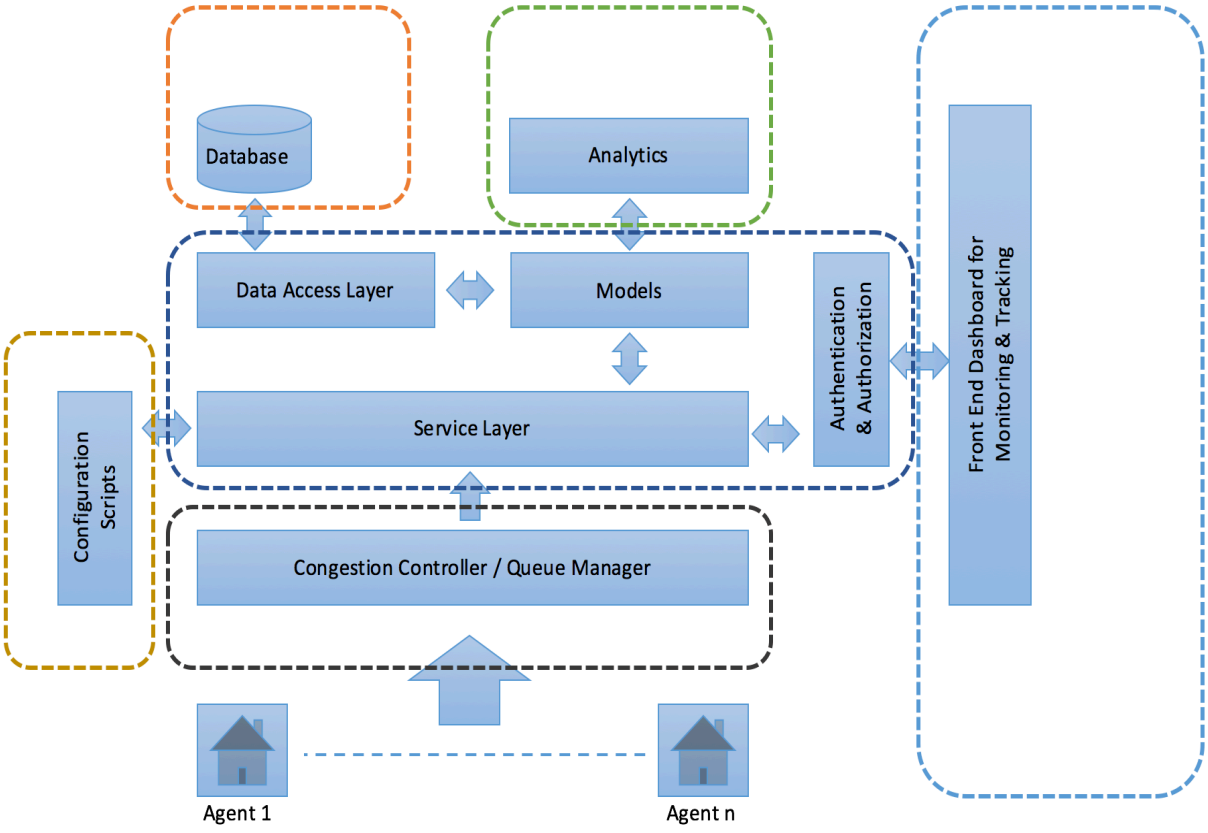


Figure 4-4 : Components level scaling architecture

4.3.1 Database layer

Multiple Databases can be hosted in different servers as Master/Slave backing up mechanism and replications so that it will avoid single point of failure as well as the data will be secured and available when a failure happens.

Also this separation provides an added advantage where we can have a proxy for different Databases with different connections so that it will allow the system to use a new database system on demand without touching any of the other sections of the API.

4.3.2 Configurations Scripts store

A cloud storage (ex: AWS S3 bucket) can be used for storing all the configurations scripts so that those scripts will be available for downloading as and when they are requested from the client level.

4.3.3 API Service layer

Horizontal scaling can be applied here to scale this layer with a minimal time and effort. Since the modular architecture contains everything required as modules the separation can be initiated and maintained easily.

Docker containers come in to the picture as the first option, but Clustering with Node JS is also another possibility for approaching this. But how we do it and when we do the scaling will depend on the business requirement of the application.

4.3.4 Congestion Controller / Queue Management Layer

This module will be extracted only in very large systems where performance really matters, otherwise this module can be combined with API level for most of the cases.

This layer takes care of all the requests in terms of handling congestion, request traffic and it makes sure that no request will get skipped or ignored. Depending on the requests load even another industry level queue management NPM module can be integrated easily with the system.

4.3.5 Front End Application

This is a React JS application and can be served from a different AWS instance as well as the same instance where the API is running. There will be a Backend for the Front end of application and it does the most when it comes to User Authentication. Almost all requests to Cloud API from Front end WebAPP are proxied using a middleware known as cloud-api proxy.

Apart from that, React Scripts infused with WebPack is used to bundle a production optimized build, ready for deployment. I have also made an effort to write a handy set of NPM and Gulp scripts which could drop development and CI/CD efforts enormously which is applicable for API deployment too. For instance, if we need to migrate from GOCD to Jenkins for CI, it only requires copying the relevant npm scripts.

When it comes to small scale start-ups this area is something they are lacking of at their initial stages. This proposed architecture boosts up the development as it takes care of all the developer best practices and assists developers to go beyond their knowledge.

4.3.6 Analytics Engine

When the business gets much more traction this module will add a huge value in to the system where we can take certain business decisions based on the analytics data we gather within the system.

This module can also be dockerized and made available easily so that the API will use this engine as and when it's needed.

As described above at each module level, the scaling of the system will not be a re-think or re-engineering effort whenever the business requires a scaling up. It is just a matter of following the instructions provided along with the system.

4.4 Sample Results

This section will have the results of proposed system. For easiness of implementation and obtain practical results, some of the functions have been mocked due to the limitation of actual IoT sensors.

These results are based on a mocked LM-35 temperature sensor [26, 37].

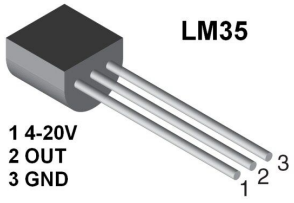


Figure 4-5 : LM-35

4.4.1 How will Agent send data?

When the agent API is started it will register with the Cloud API and will look for active sensors which are trying to connect and register. So once a device is registered it will send the data as follows.

```
> cloud-api@0.0.0 express:build /Users/aheshperera/Msc/Project/iot-agent
> npm install && gulp babel-source-maps

[22:53:05] Using gulpfile ~/Msc/Project/iot-agent/gulpfile.js
[22:53:05] Starting 'babel-source-maps'...
[22:53:06] Finished 'babel-source-maps' after 918 ms

> cloud-api@0.0.0 quick-start /Users/aheshperera/Msc/Project/iot-agent
> NODE_ENV='LOCAL' node dist/server.js

17:23:07.356Z  INFO cloud-api: Listening on port: 9090 (console-log=true)
Temperature value: 23.75
Temperature value: 21.74
Temperature value: 29.87
Temperature value: 31.95
Temperature value: 30.91
Temperature value: 27.77
Temperature value: 23.33
Temperature value: 29.64
Temperature value: 21.53
Temperature value: 27.87
```

Figure 4-6 : Console logs when agent sends data to Cloud API

4.4.2 How will Cloud API receive data?

At the same time the Cloud API will receive the same set of data as follows.

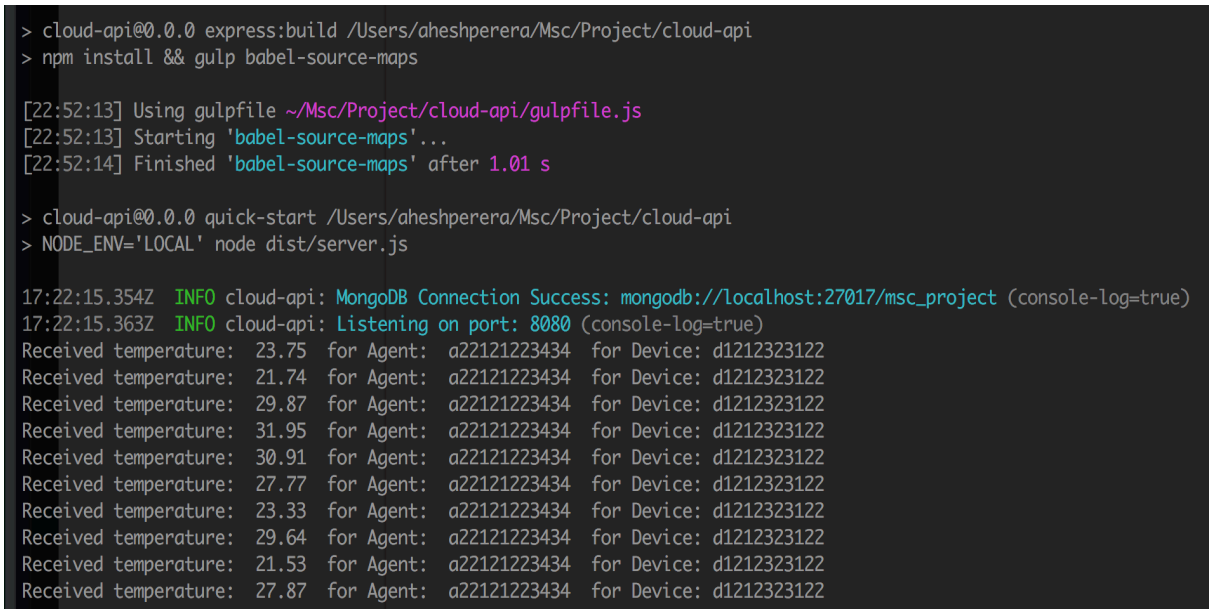


Figure 4-7 : Console logs when Cloud API receives the same data

4.4.3 How will Database save data?

If we cross check the database (in this case it's Mongo DB) it will show the received data as follows.

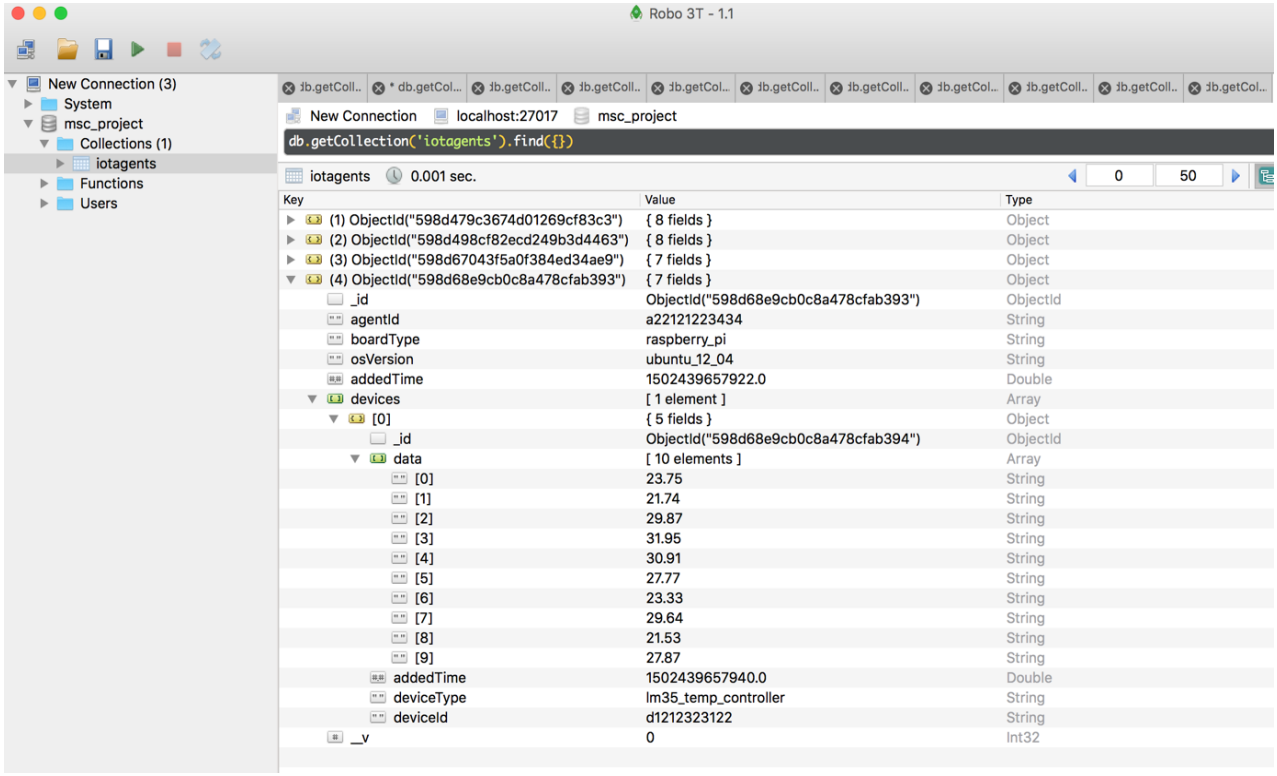


Figure 4-8 : Database snapshot of current data set

4.5 Best Practices

Following list describes the industry level best practices which have been applied through out this research and the implementation of the proposed system.

- UI components of the front end wed application are tested using Jest [51] and Enzyme[52].

- A custom helper component can be implemented to make testing of components which use React Router and Redux [53] to be much simpler.

- Reducers are tested using Jest and Deep Freeze to avoid accidental state mutations.

- For the backend modules, mocha and chai are used with sinon [54] as the test-double choice. Supertest [55] is another good solution for writing integration tests.

- As this system is heavily using metadata files for configurations and similar needs, we use the popular JSON Schema spec for testing the large metadata files.

- Unit test code coverage tools can be used for generating the code coverage reports.

- A carefully handpicked set of strict ESLint rules are used for maintaining code quality. SonarQube is also used as a more advanced complementary tool for eslint.

- Pre-Commit and Pre-Push hooks are also set up to prevent common mistakes by developers.

- Swagger is used for generating API docs as it’s enforced to use proper JS Docs in the code.

Chapter 5 SYSTEM EVALUATION

Since we have a thousands of different IoT devices from various vendors it's hard to evaluate in terms of the Accuracy of the configurations scripts. Therefore, considering the easiness of implementation and evaluation of the system, the IoT layer has been mocked so that we don't have to worry about the sensor data.

In system evaluation I have mainly focused on the ability to scale and API performance aspects.

5.1 How was the evaluation done?

With the rising number of IoT devices, the API can reach its resource limits. Therefore, the Congestion controller was introduced for handling the request queue and respond fast. I have come up with a highly customizable congestion control mechanism which uses the concepts such as memorization, throttling and queueing to control heavy spikes of the load. While the dirty-checkers can be written by hand, there are several inbuilt dirty-checker templates which make the solution even smarter.

One of the main advantages of having Node JS is we can utilize industry standard NPM modules with a very few lines of codes, but with a massive gain.

The best way to understand the benefits of this technique is to compare the prior and subsequent load average graphs. For this API I have made a high load of requests created intentionally in order to make sure it's responding as expected. In that case we can ensure that the congestion controller layer is capable enough of catering a large no of requests at a time. Also we should be able to get the upper limit of no of requests where the API is getting failed to respond.

5.2 Tools used for the evaluation

I have used K6 as the main tool for performance evaluation of this Cloud API [30].

K6 is an open source project aimed to provide the ability to test the performance of your backend infrastructure. Its written using Go and JavaScript. K6 is a modern load testing tool built on the experience of LoadImpact [32]. It is not the first tool that come up in the Google search results. But it's simplicity and ability to export data to InfluxDB to be visualized by Grafana makes it a powerful tool to load test your application [33].

5.2.1 How K6 works?

K6 uses the concept of virtual users (VU). You can have a multiple number of virtual users which runs the test script in parallel. Test scripts can be written for your application using modern ES6 syntax [31].

For testing the Cloud API with K6, I created 10 virtual users who send a request per second within for 5 seconds window using the following sample script.

The test script used for this load test can be found here:

```
import http from "k6/http";
import { check, sleep } from "k6";

export let options = {
  vus: 10,
  duration: "5s"
};

export default function() {
  var url = "http://<URL>/<end-point>";
  var payload = JSON.stringify({ <Build and set the Payload> });
  var params =  { headers: { "Content-Type": "application/json" } }
  http.post(url, payload, params);
  sleep(1);
};
```

When it was run the results were seen as below.

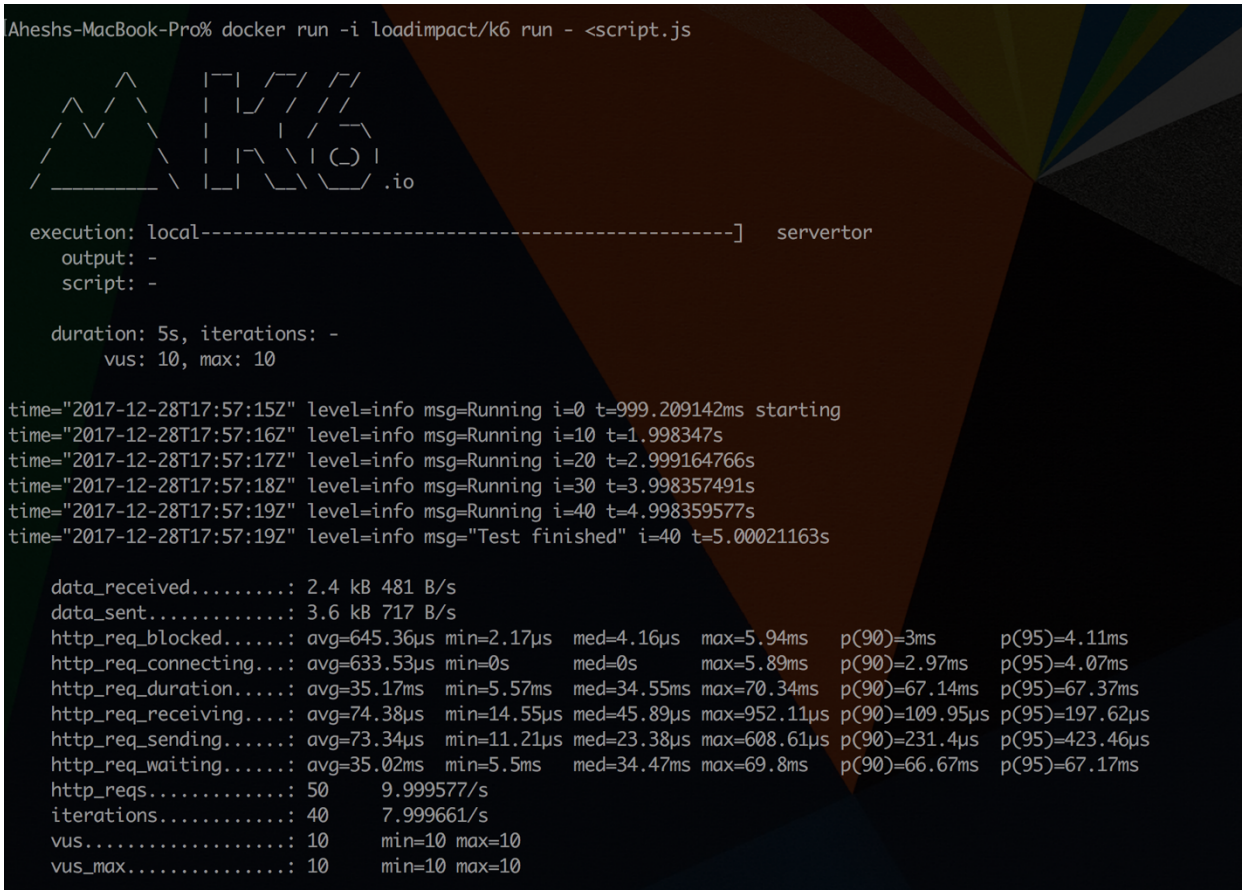


Figure 5-1 : 10 Virtual users send request per second for 5 seconds

Following table shows the matrices produced by K6 [34].

Matric	**Description**
http_req_blocked	Time spent blocked (waiting for a free TCP connection slot) before initiating request
http_req_connecting	Time spent establishing TCP connection to remote host
http_req_sending	Time spent sending data to remote host
http_req_waiting	Time spent waiting for response from remote host

http_req_receiving	Time spent receiving response data from remote host
http_req_duration	Total time for request, excluding time spent blocked (http_req_blocked), DNS lookup (http_req_looking_up) and TCP connect (http_req_connecting) time

Table 5-1: K6 Built in Matrices

5.2.2 Sample Evaluation Results for API Performance

Following set of screen shots will show the values I obtained for 1000 virtual users who send a request per second for 10 seconds time window and I repeated the same action for 4 times with a 1-minute rest.

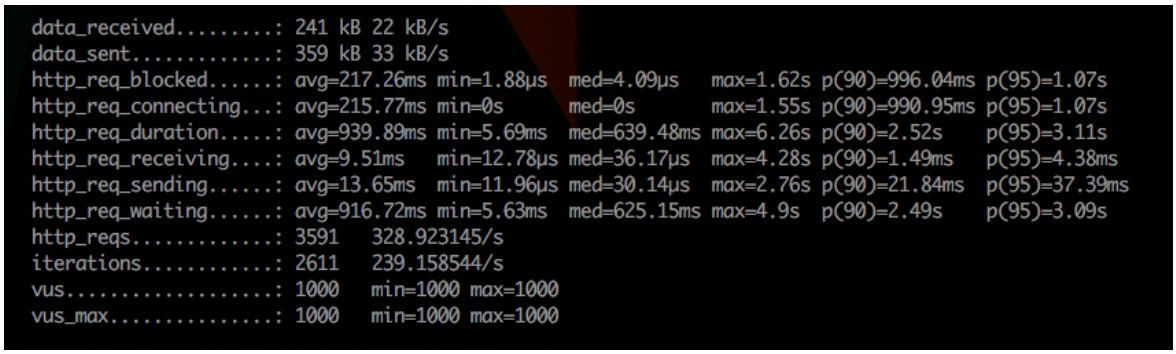
T=1

```
data_received.....: 241 kB 21 kB/s
data_sent.........: 359 kB 32 kB/s
http_req_blocked......: avg=273.49ms min=2.29µs  med=5.15µs   max=2.25s p(90)=1.22s    p(95)=1.25s
http_req_connecting...: avg=271.74ms min=0s      med=0s       max=2.25s p(90)=1.22s    p(95)=1.25s
http_req_duration.....: avg=642.18ms min=3.55ms  med=464.63ms max=5.42s p(90)=1.55s    p(95)=2.16s
http_req_receiving....: avg=3.64ms   min=10.58µs med=23.07µs  max=4.46s p(90)=680.28µs p(95)=2.71ms
http_req_sending......: avg=17.25ms  min=14.32µs med=43µs     max=1.2s  p(90)=40.58ms  p(95)=58.06ms
http_req_waiting......: avg=621.27ms min=3.49ms  med=441.31ms max=3.7s  p(90)=1.54s    p(95)=2.15s
http_reqs.........: 3947   351.281218/s
iterations........: 2960   263.438664/s
vus...............: 1000   min=1000 max=1000
vus_max...........: 1000   min=1000 max=1000
```

T=2

```
data_received.....: 241 kB 22 kB/s
data_sent.........: 359 kB 33 kB/s
http_req_blocked......: avg=437.16ms min=2.57µs  med=9.75µs   max=1.94s p(90)=1.24s  p(95)=1.37s
http_req_connecting...: avg=435.12ms min=0s      med=0s       max=1.94s p(90)=1.23s  p(95)=1.37s
http_req_duration.....: avg=1.28s    min=4.85ms  med=810.64ms max=7.58s p(90)=3.23s  p(95)=3.51s
http_req_receiving....: avg=3.27ms   min=12.4µs  med=44.78µs  max=5.32s p(90)=1.48ms p(95)=2.77ms
http_req_sending......: avg=40.75ms  min=14.68µs med=2.03ms   max=2.28s p(90)=34.6ms p(95)=364.24ms
http_req_waiting......: avg=1.23s    min=4.73ms  med=754.94ms max=5.15s p(90)=3.22s  p(95)=3.5s
http_reqs.........: 2424   225.787042/s
iterations........: 1444   134.503502/s
vus...............: 1000   min=1000 max=1000
vus_max...........: 1000   min=1000 max=1000
```

T=3



T=4

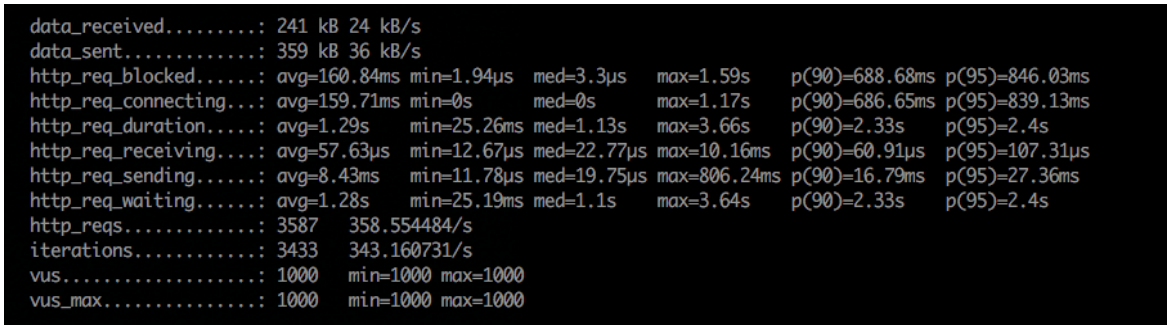
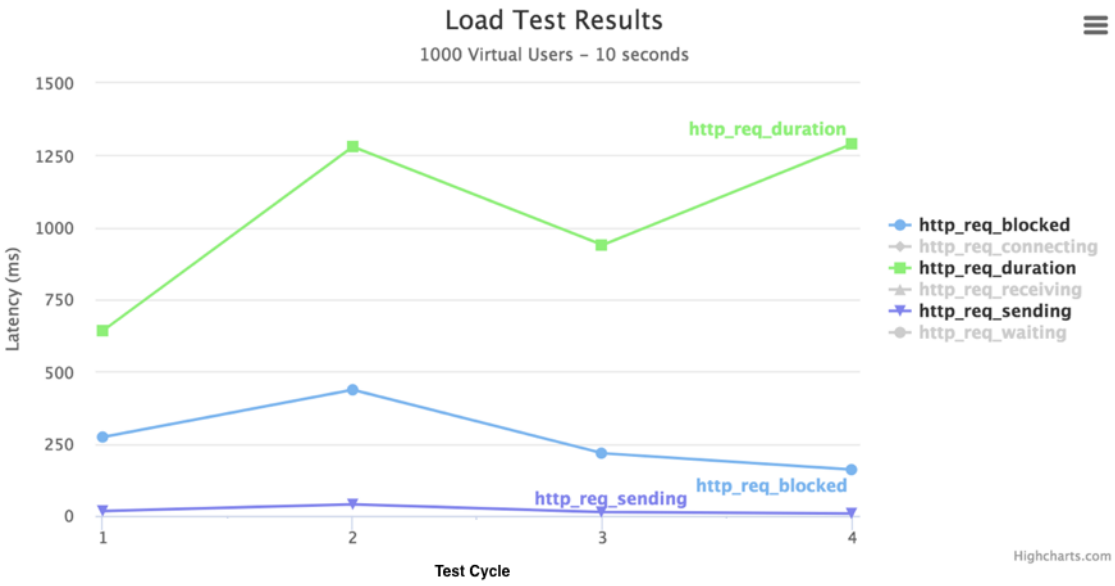


Figure 5-2 : 1000 Virtual users send request per second during 10 seconds for 4 times

Since the same load is applied for multiple times the data_received and data_sent values should be same in all four cases.

When the results are considered in terms of average values following graphs can be drawn.



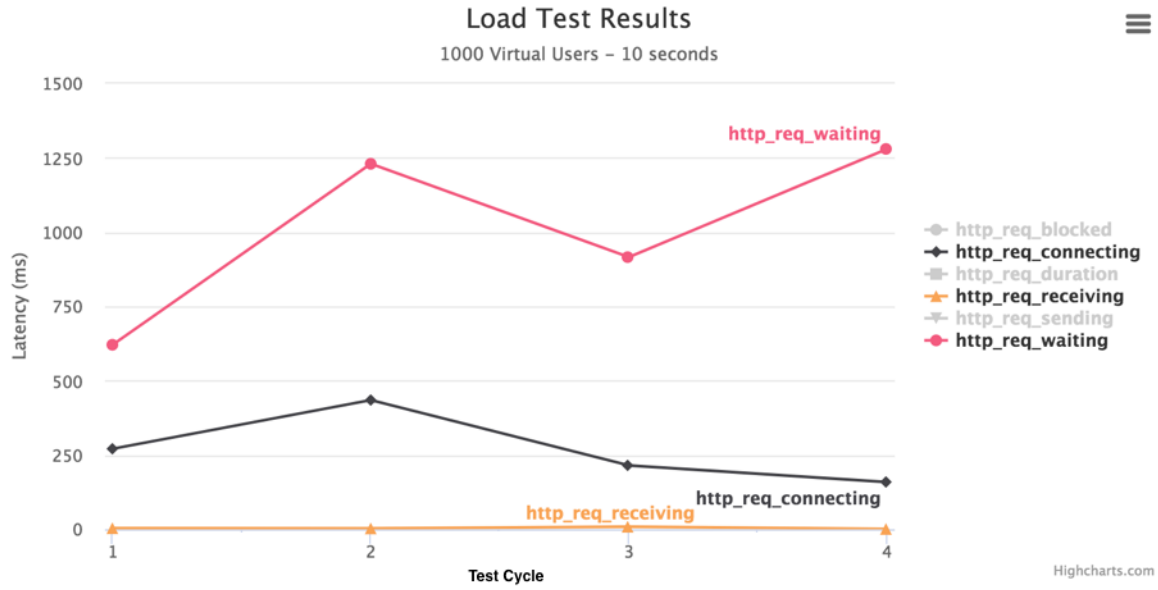


Figure 5-3 : Average values for 1000 users , send requests during 10 Seconds for 4 times

These graphs clearly show that the increasing number of requests make a trend for overall request duration as expected but the difference of two adjacent points are around 1 second which is a really good figure.

Some other researches has shown [41] that a Node server significantly outperforms both Apache and Nginx [42] for serving dynamic content. Also the implementation of JavaScript in Node JS is more than 2.5 times faster than the more traditional PHP approach, by efficiently utilizing available hardware.

5.3 Ability to Scale

This API was tested in terms of its scaling ability by using Node JS Clustering module with PM2 clustering feature.

5.3.1 What is PM2?

PM2 is a production level process manager tool specially built for Node.js applications. PM2 comes with a built-in load balancer where it allows the application to be kept alive forever. PM2 allows the application to be reloaded with zero-downtime as well as it facilitates common system admin tasks too [38].

The next series of figures will show the numbers I obtained by putting the same load intentionally as 1000 virtual users sending requests within 10 seconds duration for 1 cluster, 2 clusters, 4 clusters and 8 clusters respectively.

5.3.2 Sample Evaluation Results in terms of Scaling ability

Started the API using PM2 in the cluster mode on a single instance.

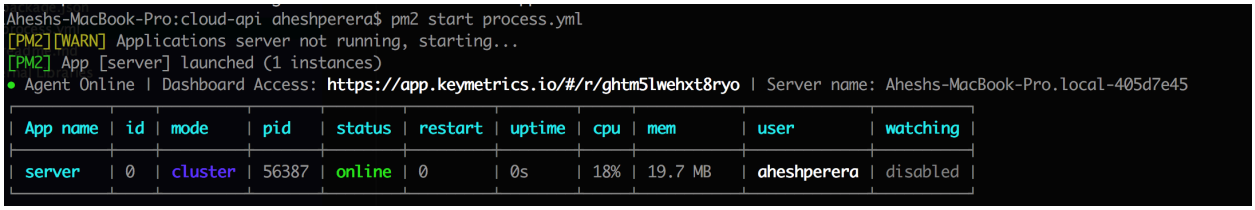


Figure 5-4 : PM2 shows a single node process is running in cluster mode

When the **htop** tool is executed it shows the API process is running as below [39].

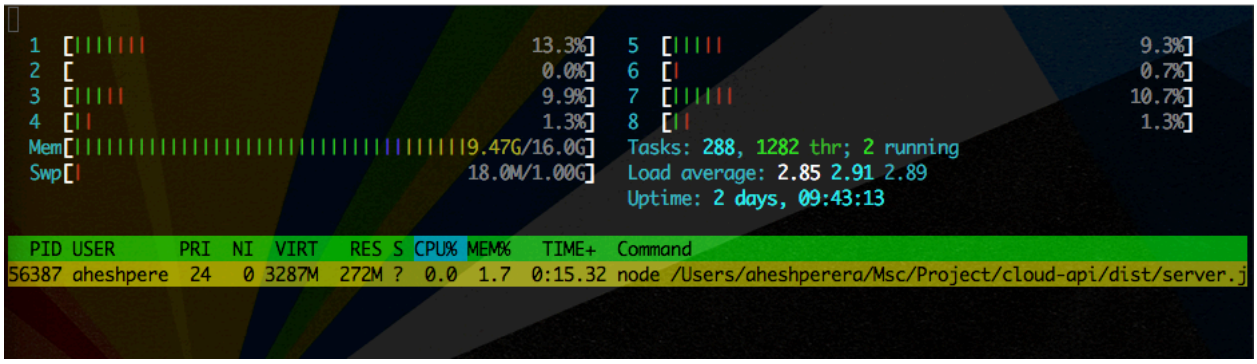


Figure 5-5 : HTOP shows how the CPU and the memory are utilized for single node process

Once the manual load is applied through K6 on the single instance following results were obtained.

```
data_received.....: 68 kB  6.8 kB/s
data_sent.........: 102 kB 10 kB/s
http_req_blocked......: avg=725.88ms min=2.08µs   med=697.66ms max=1.35s    p(90)=1.17s    p(95)=1.22s
http_req_connecting...: avg=724.96ms min=0s       med=697.06ms max=1.35s    p(90)=1.17s    p(95)=1.21s
http_req_duration.....: avg=5.67s    min=296.75ms med=6.29s    max=8.4s     p(90)=6.94s    p(95)=7.38s
http_req_receiving....: avg=100.77µs min=15.21µs  med=39.32µs  max=1.35ms   p(90)=273.79µs p(95)=357.22µs
http_req_sending......: avg=17.1ms   min=12.24µs  med=6ms      max=387.84ms p(90)=12.73ms  p(95)=146.48ms
http_req_waiting......: avg=5.66s    min=296.72ms med=6.28s    max=8.4s     p(90)=6.94s    p(95)=7.38s
http_reqs.............: 327    32.699325/s
iterations............: 293    29.299395/s
vus...................: 1000   min=1000 max=1000
vus_max...............: 1000   min=1000 max=1000
```

Figure 5-6 : Average values for 1000 users , send requests in 10 Seconds in a single instance

The same practical was repeated for the same API running in cluster mode on 2 instances on PM2 and the results were shown as follows.

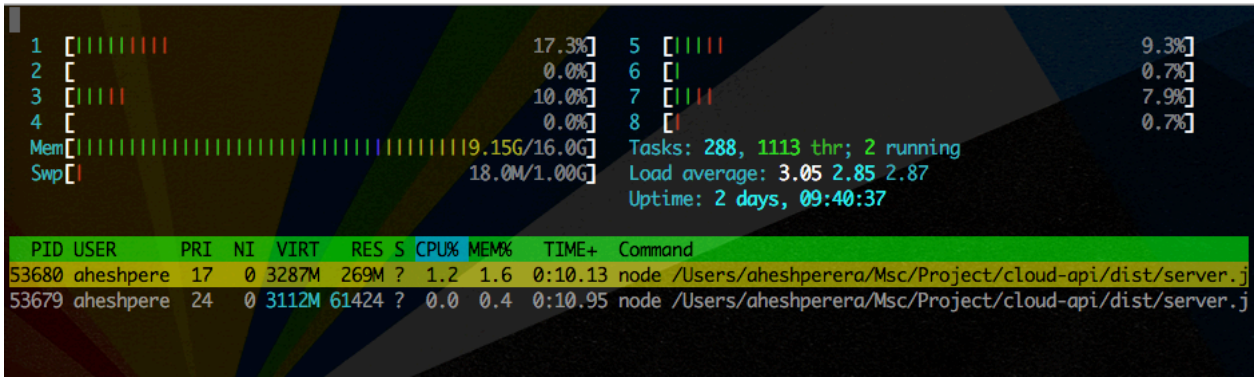


Figure 5-7 : HTOP shows how the CPU and the memory are utilized for two node processes

```
Aheshs-MacBook-Pro:cloud-api aheshperera$ pm2 start process.yml
[PM2][WARN] Applications server not running, starting...
[PM2] App [server] launched (2 instances)
• Agent Online | Dashboard Access: https://app.keymetrics.io/#/r/ghtm5lwehxt8ryo | Server name: Aheshs-MacBook-Pro.local-405d7e45
┌──────────┬────┬─────────┬───────┬────────┬─────────┬────────┬──────┬──────────┬─────────────┬──────────┐
│ App name │ id │ mode    │ pid   │ status │ restart │ uptime │ cpu  │ mem      │ user        │ watching │
├──────────┼────┼─────────┼───────┼────────┼─────────┼────────┼──────┼──────────┼─────────────┼──────────┤
│ server   │ 0  │ cluster │ 57913 │ online │ 0       │ 0s     │ 13%  │ 26.8 MB  │ aheshperera │ disabled │
│ server   │ 1  │ cluster │ 57914 │ online │ 0       │ 0s     │ 0%   │ 20.0 MB  │ aheshperera │ disabled │
└──────────┴────┴─────────┴───────┴────────┴─────────┴────────┴──────┴──────────┴─────────────┴──────────┘
```

Figure 5-8 : PM2 shows that 2 node processes are running in cluster mode

```
  data_received.....: 123 kB 12 kB/s
  data_sent.........: 183 kB 18 kB/s
  http_req_blocked......: avg=3.34s    min=4.57µs   med=3.56s   max=5.77s p(90)=3.76s    p(95)=3.77s
  http_req_connecting...: avg=3.33s    min=0s       med=3.55s   max=5.02s p(90)=3.76s    p(95)=3.77s
  http_req_duration.....: avg=3.47s    min=245.71ms med=3.18s   max=8.45s p(90)=4.78s    p(95)=4.95s
  http_req_receiving....: avg=6.65ms   min=35.18µs  med=64.58µs max=1.1s  p(90)=171.72µs p(95)=341.04µs
  http_req_sending......: avg=161.41ms min=29.17µs  med=82.57ms max=3.7s  p(90)=253.9ms  p(95)=299.22ms
  http_req_waiting......: avg=3.3s     min=245.56ms med=2.99s   max=5.47s p(90)=4.55s    p(95)=4.82s
  http_reqs.............: 658    65.794016/s
  iterations............: 511    51.095353/s
  vus...................: 1000   min=1000 max=1000
  vus_max...............: 1000   min=1000 max=1000
```

Figure 5-9 : Average values for 1000 users, send requests during 10 Seconds in 2 instances

To see the results in 4 instances, the same practical was performed on 4 instances as below.

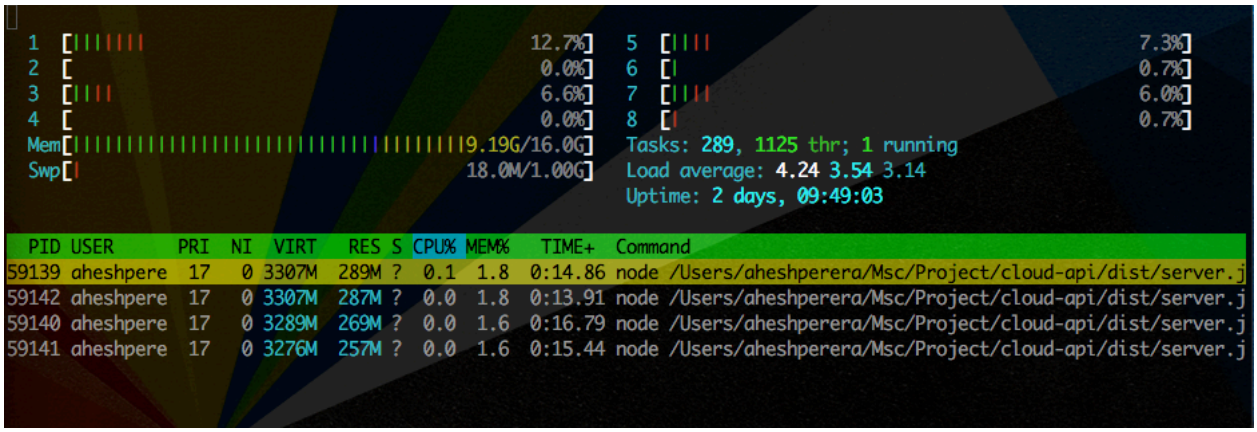


Figure 5-10 : HTOP shows how the CPU and the memory are utilized for 4 node processes

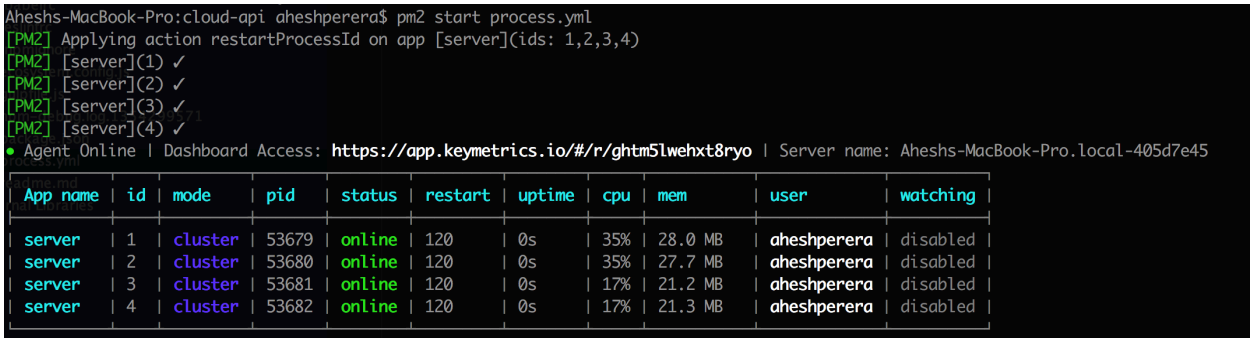


Figure 5-11 : PM2 shows that 4 node processes are running in cluster mode

```
  data_received.....: 146 kB 15 kB/s
  data_sent.........: 218 kB 22 kB/s
  http_req_blocked......: avg=857.66ms min=3.7µs   med=890.89ms max=1.63s    p(90)=1.25s    p(95)=1.3s
  http_req_connecting...: avg=856.06ms min=0s      med=889.89ms max=1.63s    p(90)=1.25s    p(95)=1.3s
  http_req_duration.....: avg=4.91s    min=94.85ms med=4.83s    max=8.4s     p(90)=7.72s    p(95)=8.13s
  http_req_receiving....: avg=154.76µs min=20.74µs med=61.13µs  max=30.46ms  p(90)=209.79µs p(95)=315.03µs
  http_req_sending......: avg=44.24ms  min=21.71µs med=15.51ms  max=755.81ms p(90)=57.82ms  p(95)=201.52ms
  http_req_waiting......: avg=4.86s    min=94.77ms med=4.76s    max=8.38s    p(90)=7.68s    p(95)=7.98s
  http_reqs.....: 759    75.881122/s
  iterations.....: 629    62.884355/s
  vus.....: 1000   min=1000 max=1000
  vus_max.....: 1000   min=1000 max=1000
```

Figure 5-12 : Average values for 1000 users , send requests during 10 Seconds in 4 instances

Finally, the same thing was performed on 8 instances to see the results.

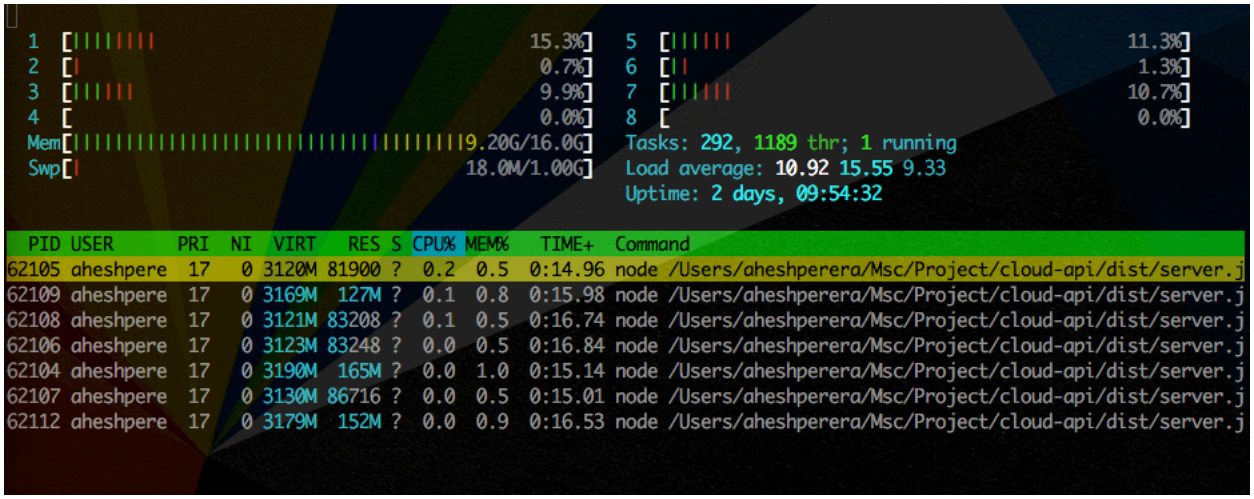


Figure 5-13 : HTOP shows how the CPU and the memory are utilized for 8 node processes

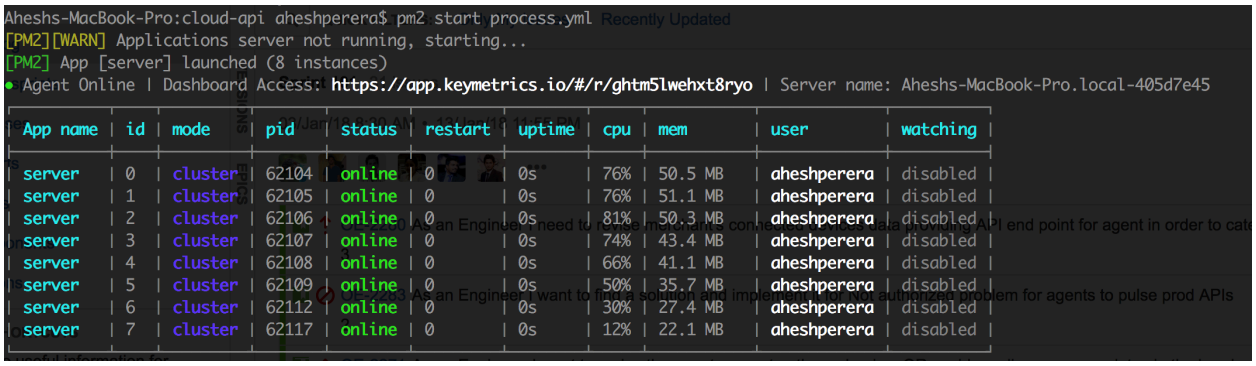


Figure 5-14 : PM2 shows that 8 node processes are running in cluster mode

```
data_received.....: 127 kB 13 kB/s
data_sent.........: 189 kB 19 kB/s
http_req_blocked......: avg=1.68s    min=3.08µs  med=1.69s   max=4.08s p(90)=1.91s    p(95)=2.8s
http_req_connecting...: avg=1.68s    min=0s      med=1.69s   max=4.08s p(90)=1.91s    p(95)=2.8s
http_req_duration.....: avg=3.72s    min=1.56s   med=3.02s   max=9.06s p(90)=6.18s    p(95)=6.84s
http_req_receiving....: avg=4.48ms   min=22.46µs med=43.31µs max=1.89s p(90)=149.14µs p(95)=429.53µs
http_req_sending......: avg=202.41ms min=19.58µs med=14.05ms max=2.14s p(90)=366.82ms p(95)=1.55s
http_req_waiting......: avg=3.51s    min=1.35s   med=2.89s   max=7.29s p(90)=5.69s    p(95)=6.44s
http_reqs.............: 587    58.699274/s
iterations............: 528    52.799347/s
vus...................: 1000   min=1000 max=1000
vus_max...............: 1000   min=1000 max=1000
```

Figure 5-15 : Average values for 1000 users, send requests during 10 Seconds in 8 instances

These figures give a clear idea about the practical I performed in order to measure the scaling ability of the API.

When the data_received and data_sent values are taken in to a bar chart the variance can be seen clearly as below.

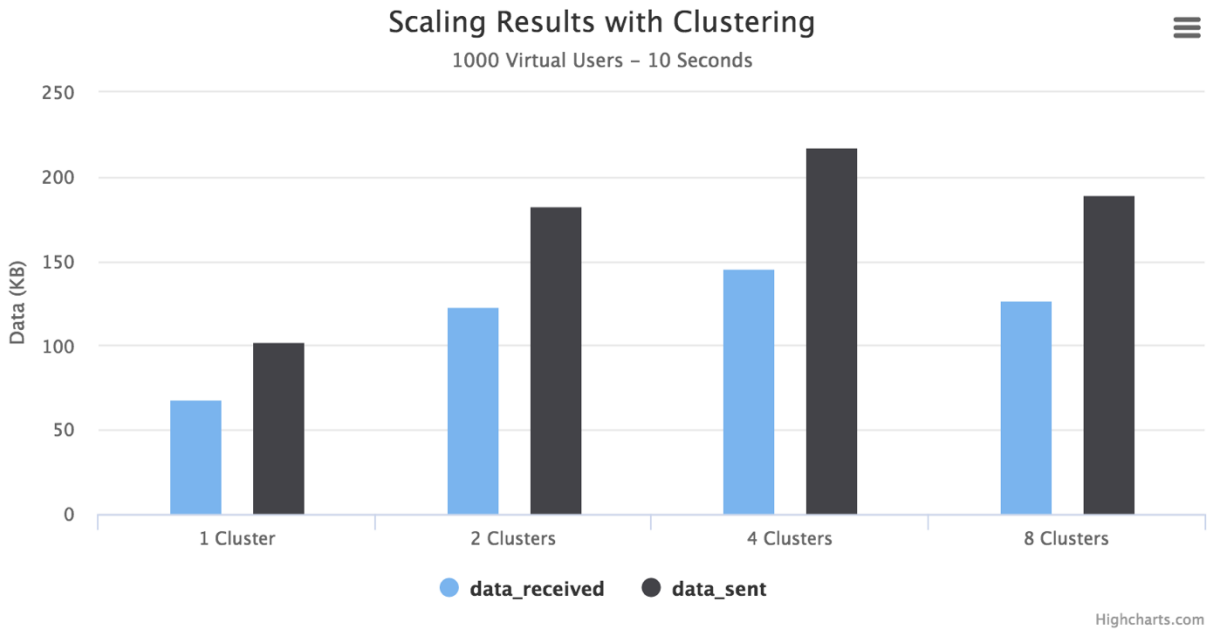


Figure 5-16 : Sent and Received data variation against number of clusters

As per the results in above figure 5-8, it's clear that 2 clusters double the amount of data being treated by the API than in 1 cluster. It increases more when we have 4 clusters but when it comes to 8 clusters the number will be a bit lower. That latency can be explained easily as it is a self introduced delay by the clusters because the workers will take a time when distributing the load among other workers.

So in that case running the API on 4 clusters would be the best case in terms of high performance.

Chapter 6 CONCLUSION

In the last few years, IoT technologies have been developing rapidly. Hence IoT cloud can provide enhanced services and effective utilization of resources for the end-users with the help of cloud technologies.

The goal of this research is to make building of Internet of Things applications significantly easier. My goals were to come up with a solution that would empower developers by hiding protocols details, to embrace system heterogeneity rather than avoid it and simplify service management and scaling by allowing decisions to be made autonomously by the runtimes.

6.1 Research Contributions

I have proposed an architecture for IoT applications that simplifies the Web of Things idea by merging IoT and cloud through a RESTful web service.

From an IoT based start-up perspective, this API resolves a few major challenges at the starting stage of building IoT applications. In particular, it is a way of consolidating the micro-service trend with single-purpose small devices waiting to be scaled up to the maximum extent as and when the business grows up.

For device manufacturers and vendors, this framework provides a way to increase the attractiveness of their products for developers by providing a self on board capability to expose their configurations via cloud service as a plug-and-play mode in addition to the traditional device driver installations.

Finally, several experiments have been carried out to evaluate the performance of the designed API under different conditions.

6.2 Research Limitations

This API is still in its early stages of development, and currently not all the desired functionality is fully implemented.

One of the main limitations I faced across this research was evaluating the elasticity with auto scaling capability of the API. There were many aspects to consider when designing the API in that manner but it was hard to measure scalability levels with figures. Clustering via PM2 was helpful to achieve this up to some extent but there is a room for further improvements too.

Since this has been started on Node JS, a community can form around this platform and explore its possibilities to the fullest by making this available under an Open Source license and making the architecture extensible.

6.3 Future Work

This phase of the work has mainly focused on making sure the foundation sounds well so that it will start getting traction. Next immediate piece of work would be making this available in a public Open Source Repository (example: github.com) as a starting template along with a developer documentation so that anyone who is interested in can try it out. Next phases of this research will be mainly on device level enhancements in terms of configuring them as easy as plug and play, security mechanisms required to make autonomous scaling smooth and so on but there are many aspects left to explore.

Further comparisons and quantitative measurements on the performance and scalability of these sort of APIs are one major 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.

Important parts such as auto scaling with AWS EC2 instances along with Kubernetes [36], adopting to micro services architecture, production deployment and service management enhancements are topics for future researches whilst this reference implementation are in place.

REFERENCES

[1] Weiser, M. The computer for the 21st century. Sci. Am. 1991, 265, 94–104.

[2] Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. Comput. Netw. 2010, 54, 2787–2805.

[3] Ashton, K. That 'Internet of Things' thing. RFiD J. 2009, 22, 97–114

[4] ITU Telecommunication Standardization Sector. Overview of Internet of Things; ITU-T: Geneva, Switzerland, 2012.

[5] Xively by LogMeIn Business Solutions for the Internet of Things: https:// xively.com/

[6] Evans,D. The internet of things. How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG) (2011).

[7] Chih-Yuan Huang and Cheng-Hung Wu: A Web Service Protocol Realizing Interoperable Internet of Things Tasking Capability

[8] Matthias Kovatsch, Martin Lanter and Simon Duquenno: Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications

[9] Geoffrey C. Fox, Supun Kamburugamuve, Ryan Hartman: Architecture and Measured Characteristics of a Cloud Based Internet of Things API

[10] Bhagyashri Katole , Suresh V., Gita Gosavi, Amit Kudale, Gokul Thakare, Girishchandra Yendargaye, Ch. Pradeep Kumar: The Integrated Middleware Framework for Heterogeneous Internet of Things (IoT)

[11] Paul Fremantle, Benjamin Aziz: Web API Management Meets the Internet of Things

[12] Zetta - An API-First Internet of Things (IoT) Platform - Free and Open Source Software, [Online] Available: http://www.zettajs.org/

[13] Ali Hazmi, Mikko Valkama and Juho Pirskanen, “IEEE 802.11AH: promising technology for IoT and M2M applications”, Internet-of-things magazine, Finland, pp. 22

[14] Kopecky, J., Fremantle, P., Boakes, R.: A history and future of web apis. Informa- tion Technology (2014)

[15] Eric Bernardes Chagas Barros, Admilson de Ribamar L. Ribeiro, Edward David Moreno: PROBLEMS AND LIMITATIONS FOR DESIGNING A WEB-API OF IOT

[16] Barros, Eric. Ribeiro, Admilson. A Self- Configuration Architeture for Web-API of Internet of Things.10th International Conference on Web Information Systems and Technologies. 2014.

[17] Orenstein, David. Application Programming Interface. COMPUTER WORLD Jan 2000. [Online] Available: http://www.computerworld.com/s/article/43487 /Application_Programming_Inter_face

[18] Zeng D., Guo S, and Cheng Z. The Web of Things: A Survey. Journal of Communications, vol. 6, setembro 2011.

[19] Kephart, Jeffrey O. The vision of Autonomic Computing. *IEEE Computer Society*. 2003.

[20] ThingSpeak [Online]. Available: https://www.thingspeak.com/ last accessed October 14, 2013.

[21] NimBits [Online]. Available: http://www.nimbits.com/ last accessed October 14, 2013.

[22] Sensor Cloud [Online]. Available: http://www.sensorcloud.com/sites/default/files/SensorCloud_Open_Data_API.pd f

[23] Evrythng [Online]. Available: http://www.evrythng.com/

[24] Etherios [Online]. Available: http://www.etherios.com/

[25] Grovestreams [Online]. Available: https://grovestreams.com/

[26] LM35 Precision Centigrade Temperature Sensors [Online]. Available: https://www.engineersgarage.com/sites/default/files/LM35.PDF

[27] WSO2 IoT Server [Online]. Available: https://docs.wso2.com/display/IoTS300/Overview

[28] WSO2 IoT Server Architecture [Online]. Available: https://wso2.com/library/articles/2017/07/an-introduction-to-wso2-iot-architecture/

[29] WSO2 IoT Server System Requirements [Online]. Available: https://docs.wso2.com/display/IoTS300/System+Requirements

[30] Welcome to the k6 Documentation [Online]. Available: https://k6.readme.io/docs/welcome

[31] How to Load Test Your Node.js App Using K6 [Online]. Available: https://medium.com/codeinsights/how-to-load-test-your-node-js-app-using-k6-74d7339bc787

[32] Load Impact [Online] Available: https://loadimpact.com/

[33] InfluxDB Grafana [Online] Available: https://docs.k6.io/docs/influxdb-grafana

[34] K6 Metrics [Online] Available: https://docs.k6.io/docs/result-metrics

[35] Auto Scaling Real-Time Node JS Applications on AWS [Online] Available: https://medium.com/@eyalronel1984/auto-scaling-real-time-nodejs-applications-on-aws-the-last-tutorial-youll-need-eba1d2c88a4c

[36] Kubernetes Documentation [Online] Available: https://kubernetes.io/docs/home/

[37] LM 35 Specification and Pin Diagram [Online] Available: https://cdn.instructables.com/FE0/DHQ4/HV2AIB01/FE0DHQ4HV2AIB01.MEDIUM.jpg

[38] Welcome to the PM2 Quick Start [Online]. Available: http://pm2.keymetrics.io/docs/usage/quick-start/

[39] htop - an interactive process viewer for Unix [Online]. Available: http://hisham.hm/htop/

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

[41] I. K. Chaniotis, K.-I. D. Kyriakou, and N. D. Tselikas, “Is Node.js a viable option for building modern web applications? A performance evaluation study,” Computing, pp. 1–22, 2014. [Online]. Available: http://dx.doi.org/10.1007/s00607- 014-0394-9

[42] Welcome to NGINX Wiki’s documentation. [Online]. Available: https://www.nginx.com/resources/wiki/

[43] MongoDB. [Online]. Available: https://www.mongodb.com/use-cases/internet-of-things

[44] Z. Parker, S. Poe, and S. V. Vrbsky, “Comparing NoSQL MongoDB to an SQL DB,” Proceedings of the 51st ACM Southeast Conference on - ACMSE ’13, 2013. [Online]. Avail- able: http://dl.acm.org/citation.cfm?id=2498328.2500047

[45] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. Ad Hoc Networks, 10(7):1497–1516, 2012.

[46] Antti Iivari and Jani Koivusaari VTT Technical Research Centre of Finland Ltd Oulu, Finland, A RESTful Sensor Data Back-end for the Internet of Things – 2016

[47] How technology evolves: Kevin Kelly on TED.com [Online] Available https://blog.ted.com/how_technology/

[48] Siri from Apple: [Online] Available https://www.apple.com/ios/siri/

[49] Andrew C. Oliver, the founder of the Apache POI project [Online] Available: https://en.wikipedia.org/wiki/Andrew_C._Oliver

[50] The Best APIs are Built with Swagger Tools [Online] Available: https://swagger.io/

[51] Jest [Online] Available: https://facebook.github.io/jest/

[52] Enzyme [Online] Available: https://github.com/airbnb/enzyme

[53] Redux [Online] Available: https://redux.js.org/

[54] Sinon [Online] Available: http://sinonjs.org/

[55] Supertest [Online] Available: https://www.npmjs.com/package/supertest