Leveraging the Spring Boot Framework to Facilitate Microservice REST Communications Between IoT Edge Deployments and the Cloud

Lee M. Brunovsky
Graduate School of Computer Science
Georgia Southwestern State University
Americus, Georgia
lbrunovs@gsw.view.usg.edu

***Abstract*—Given the popularity of cloud containerization, edge computing, micro-services, and IoT, I propose to integrate several related work initiatives that fall short of showcasing the full spectrum end-to-end capabilities of combining the Spring Boot framework and cloud services in an open source GitHub hosted repository. More specifically, in this paper I seek to implement a full stack micro-services use case in two phases, where the objective in phase one is to utilize the Spring boot application Rest-controller to facilitate web server servo position commands being sent over HTML in the form of application.JSON formatted POST requests to an Arduino Uno, which will then pass tilt and pan angle byte data to a custom rig containing SG90 servo motors (one for the x and y dimension), as well as a HC-SR04 ultrasonic sensor. The phase two objective is a stretch goal that involves passing measured distance data captured from an HC-SR04 ultrasonic sensor back to the application for routing into cloud Docker containerized storage using the Kubernetes Engine on Google Cloud, as well as exploration of further development initiatives such as load balancing multiple instance HTTP requests and token based security implementation. This project will then be posted in a repository on GitHub to facilitate further open source collaboration.**

***Index Terms*—IoT, Kubernetes, Microservice, REST, Spring Boot**

I. Introduction

IoT applications are in high demand in part by an ever evolving availability of cheap and effective sensor solutions capable of measuring acceleration, humidity, motion, light, position, sounds temperature, vibration etc. Once such example is the HC-SR04 ultrasonic sensing module, which is commonly coupled to Arduino Uno edge devices on numerous IoT initiatives on account of their low cost, availability, relative high degree of accuracy for tracking targets and ease of use [1]. In order to integrate the data collected from these sensors, a fault tolerant and secure middleware in the form of micro-services can effectively facilitate communication between edge devices and assist with cloud integration[2].

II. Related Work

Several studies have explored the performance strengths of micro-services relative to monolithic architectures for working with large numbers of IoT sensors across heterogeneous systems, which can be independently deployed, scaled and tested: Furthermore, Spring Boot offers an annotation based framework to facilitate configuration, reduce development time, easily utilize databases, implement REST endpoints in a secure manner, apply load balancing, and integrate micro-services with popular cloud resources [2]. This project is founded on a deprecated GitHub Spring Boot project built for the purposes of controlling a turret via a RESTful web service capable of receiving JSON POST requests in order to relay serialized commands to an Arduino Uno, which I have updated using Spring Initializer.

III. REST Microservice and IoT and Cloud Implementation

Prior to explaining phase one and two implementation, figure 1. provides context of the overall architecture, with the green box representing the core of the Spring application and the black box reflecting the temporary design prior to completion.

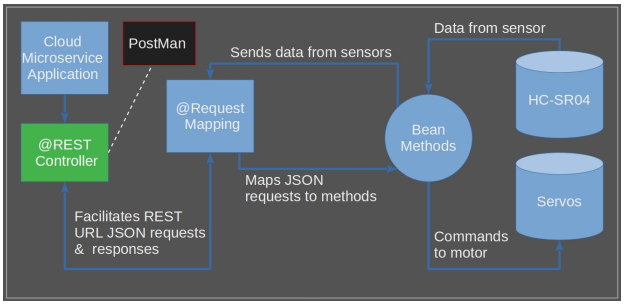

Fig. 1. Cloud Hosted Spring Boot IoT Architecture

In this Spring Boot application, the REST controller is designed to consume JSON formatted commands sent from the Postman agent to local host port 8090 and maps them to the Arduino edge device over a serial connection via a self injecting bean that leverages Spring's IoC container, Autowired constructor properties, lifecycle, and usage dependencies. A spring service method generated object will then be sent back to the client verifying that the command ID was sent, which will be orchestrated by utilizing HttpMessageConverter

implementation to automatically generate client responses. Furthermore, error handling callbacks will be facilitated with the message and exceptions properties inherent to the REST controller. Future development of this open source initiative will include a front-end user interface (UI) to replace the Postman agent, which will initially be leveraged for proof-of-concept in phase one of the development life-cycle.

Phase two will utilize the RXTX Java native library to capture raw distance byte data sent over the serial connection established by the same bean that presented servo angle commands in order for the REST controller to provide both the UI and Docker storage with distance figures at the time of a command. This will not only provide user-level range assessments, but also serve as a means of persistent data capture, which may be a necessary historical artifact in several use cases such as a military drone pilot firing a weapon, or for analytic modeling of historic data. The Kubernetes Engine on Google Cloud an established method to host Spring Boot applications and is fully supported with a generous free trial; however, some challenges such as managing the impacts of dynamic pod storage pertaining to the networking configuration effect on the Spring Boot application.properties will need to be overcome. Finally, phase two will conclude with a bench-marking of load balancing multiple instance HTTP requests, and token based security implementation, some of which is offered inherently by both the Spring Boot framework and Google Cloud Services.

IV. Results

Phase one went relatively smoothly after getting acquainted with the established code base, and configuring the package for the appropriate JDK and Gradle versions, as well as the IP of the PC running the application. Moreover, the aforementioned phase one aspects of the implementation are fully completed, and the application includes a Gradle wrapper so that it can be executed from a single jar file, and hosted on Google Cloud Services upon completion of phase two. Unfortunately, several issues were experienced in phase two. The primary roadblock revolved around reading the Sonar raw bytes being sent from the Arduino back to the REST controller portion of the application over serial com port. Several methods were exhausted over many hours, to include post mortem analysis of RXTX thread exemptions that pointed to a potential bug in the libNRJavaSerial.dll pertaining to the SerialImp.c dependency jobj char type being set to long, opposed to +size_t. Accordingly, a ticket was raised to the developers for a hopeful resolution. In the meantime, Python was utilized as a mitigating measure to read the sonar distance provided by the edge device. Changes to the Arduino sketch were also made to more easily facilitate reading of the stream. Figure 2. demonstrates the python code written to display the sonar data in the terminal and figure 3. reflects the changes made to the Arduino sketch from the baseline included n the original package.

```
import serial
import struct

serialPort = 'COM6'
serialBaud = 9600
bufferLength = 1

# Connect to serial port
print('Trying to connect to ' + str(serialPort) +
.... ' at ' + str(serialBaud) + ' BAUD.')
try:
.... s = serial.Serial(serialPort, serialBaud, timeout=4)
.... print('Connected!')
except:
.... print("Failed to connect with " + str(serialPort) +
..... ' at ' + str(serialBaud) + ' BAUD.')

s.reset_input_buffer() # flush input buffer

while(True):
.... #time.sleep(0.2)
.... data = s.readline()
.... dis = str(data)
.... dis = dis.strip().replace("b'",")
.... print("Sonar Diastance: " + dis[:-5])
```

Fig. 2. Python code written: sonarReader.py

```
void loop() {

.... digitalWrite(triggerPin, LOW);
.... delayMicroseconds(5);
.... digitalWrite(triggerPin, HIGH);
.... delayMicroseconds(10);
.... digitalWrite(triggerPin, LOW);

.... pinMode(echoPin, INPUT);
.... duration = pulseIn(echoPin, HIGH);

.... // Convert the time into a distance
.... cm = (duration/2) / 29.1; // Divide by 29.1
.... inches = (duration/2) / 74; // Divide by 74

.... Serial.print(inches);
.... Serial.print("in, ");
.... Serial.print(cm);
.... Serial.print("cm");
.... Serial.println();
```

Fig. 3. Arduino Sketch Changes

Although the Java RXTX issues resulted in most of the cloud based implementation aspects of phase two not being accomplished by the deadline, I did manage to to run the

snapshot.jar file of the Spring Boot application on Google cloud services, as depicted in Figure 4. below . Similarly, the delays left the bench-marking of load balancing multiple instance HTTP requests, and token based security implementation unexplored.

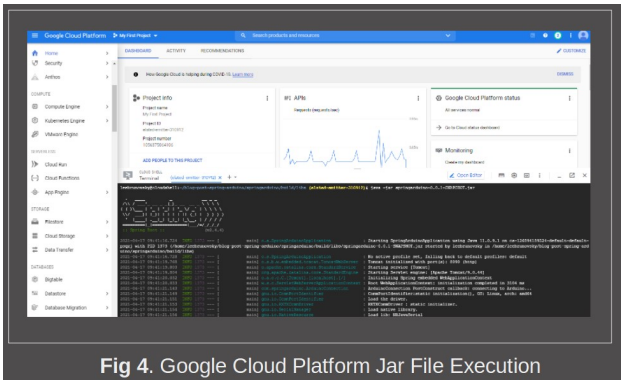

Fig. 4. Google Cloud Platform Jar File Execution

V. Future Work

Much thanks and credit to the original content creator Milosz Orzel, whom has added my GitHub repository fork to the original project link README.md, as a currently functioning update. Accordingly, I am optimistic that the serial read headwinds will be overcame given the open source exposure of this project, which will hopefully accelerate the completion of phase two. Furthermore, load balancing multiple instances and token based security remain to be explored and adopted. Finally, data processing and storage of the microserve features will need to be further developed, perhaps by leveraging MongoDB.

VI. Conclusion

While I am dissatisfied that time and budgetary resources resulted in the in-completion of phase two, I strongly feel that this paper has demonstrated the effectiveness of the Spring Boot framework to facilitate the RESTful concept for use in IoT edge deployments. While this project is still in its infancy, I remain bullish on the open-source community providing further contributions. Please reference figure 5. to access the GitHub repository, which also contains a video file demonstrating the full scope of the current capabilities of the project.

Fig. 5. GitHub Repository & Full Scope Demonstration Video

References

[1] Raza, K. A., Monnet, W. Moving objects detection and direction-finding with hc-sr04 ultrasonic linear array. 2019 International Engineering Conference (IEC), 153-158. doi:10.1109/iec47844.2019.8950639.

[2] Modugu S.R., Farhat H. Implementation of the Internet of Things Application Based on Spring Boot Microservices and REST Architecture. In: Silhavy R., Silhavy P., Prokopova Z. (eds) Software Engineering Perspectives in Intelligent Systems. CoMeSySo 2020. Advances in Intelligent Systems and Computing, vol 1294. Springer, Cham. https://doi.org/10.1007/978-3-030-63322-6 3.