Four key factors to design a Web of things architecture

Francesco Bruni[1], Claudio Pomo[2] and Gaetano Murgolo[3]

[1]Planetek Italia SRL
bruni@planetek.it
[2]Polytechnic University of Bari
c.pomo@studenti.poliba.it
[3] Engineering Consulting SRL
info@ecmg.it

Abstract. The spreading of Internet accesses and latest developments of Web technologies simplified the process of building architectures capable of collecting and sharing data. On the other side, a massive use of smart devices helped increasing the overall quality of the monitored process. This report illustrates how sensed raw data collected over an electrical plant by a smart device have been turned into a market valuable knowledge. Implementing and extending the Back-end Data Sharing Pattern, the proposed architecture underlines some key factors to be appointed when designing this kind of infrastructures and how they have been implemented in a real world use case.

Keywords: Internet of Things, Software Architecture IoT, Web of Things, Back-end data sharing pattern

1 Scenario

The current *web of data era* made extremely easy collecting raw and semi-structured data and developing data processing pipelines capable of extracting metrics and insights. Indeed turning a raw information into a market valuable knowledge is the actual challenge and represents the goal of this work.

In order to monitor consumptions and prevent unforeseen events, the current work details an infrastructure capable of sensing and reporting collected data in a photo-voltaic plant. Exploiting a *smart device*, an extensive set of electrical parameters is collected and made available to other sophisticated designed to clean, normalize, aggregate and finally expose them over the Web.

This report details a **Back-End Data Sharing**[1] based architecture suitable for exposing smart devices (processed) sensed data to end users. This work focuses only on the overall infrastructure and does not cover internal processing algorithms details.

The proposed WoT[4] architecture overview is depicted in fig.1. The smart device splits up the gathered data into multiple JSON[1] encoded chunks. Uniquely

[1] http://www.json.org

identified by a *timestamp* field, the device performs a HTTP[2] request to the *Server* counterpart at user fixed rate:

```
{
        "timestamp": 1491250500,
        "data":{
        "FVARL": {
                "FVARL-MIN": 92897,
                "FVARL-MAX": 120115,
                "FVARL-MEAN": 106675
                }
        ...
}
```

Listing 1.1. JSON encoded raw data

After a basic validation, the Server publishes received data over multiple queues and hands over control to the *Processing* module to apply a well defined data processing pipeline. The *Analytics* module exposes the extracted knowledge (electrical parameters expressed as a time series) to end users. Furthermore, it allows administrator users the ability to assert the truthfulness and correctness of measurements.

2 Architecture

In order to enforce the software reuse and modularity, each server side deployed component has been wrapped in a container. This approach turns a monolithic structure into a set of independent pieces capable of communicating each other and simplifies the scaling process.

Given the scenario and the overall complexity of the problem, the proposed architecture addresses four different aspects detailed in the next subsections.

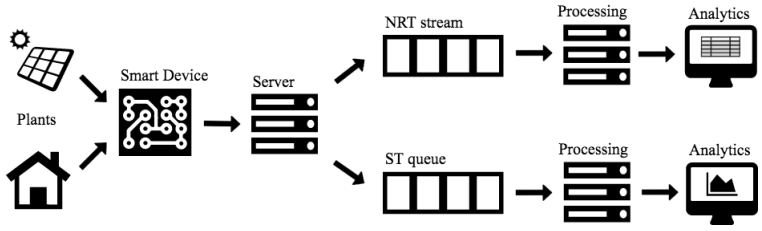

Fig. 1. Scenario overview

2.1 A scalable and fault tolerant system

To afford long time running computations, multiple queue-based data structures have been deployed. Processing incoming data or start producing user customized reports should not impact the overall system capabilities with blocking

[2] https://tools.ietf.org/html/rfc2616

calls and spinning loops. Nowadays, being asynchronous can be very effective if an immediate feedback could be delayed and handled later. As turning raw data into kwnoledge requires time, this process has been designed to be asynchronous. Specifically, data is sent to the Server component which, after a domain specific validation, replies with a well-defined response. Incoming server side data are then pushed over two different queues, as detailed below: an *Apache Kafka*[3] based *Near real time* (NRT) stream for admin oriented analytics and a *Redis*[4] based *Standard* queue (ST) for end users.

The involved queues have been spawned over multiple nodes and orchestrated by a designated component to be scalable and robust to unforeseen accidents since no single point of failure is present.

The underlined queue-based architecture recalls and partially implements the *Actor model* (AM) [3]: incoming messages are routed to specific actors (the request consumers) whose task is fixed and well designed. They can spawn other actors, edit received messages before retransmitting them, and make local decisions.

2.2 Per-user customized knowledge

Build on top of the *Angular.js* framework, two different web clients have been included in this architecture. Basically, they consume *RESTful* services exposed by the Server component.

To guarantee end user secure communications, client requests need to be authenticated. The JWT[2] protocol enforces this control and solves typical cross domain problem due to cookies. Besides, JWT encodes user information in the HTTP request *Authorization* header. This approach allows server side application to infer user identity and acting consequently.

As implied by fig.1, incoming data have been initially processed by the server component and then sent over two different queues. Indeed different kind of users should have access to a different knowledge. This means that admin oriented analytics have been exctracted in *real time* fashion and not shared with end users which have been granted access to a different set of informations. Thus superuser and user refer to different players and should be threated as different consumers.

2.3 React and Adapt

Sensing the physical world and pushing gathered perceptions to a server would be useless if the involved device was not able to adapt its behaviour to an external change.

Changing sensing rate or the Server component IP address should be reflected on the device. The provided architecture allows end user to reconfigure an operational smart device using a web client as well as monitoring this operation.

[3] https://kafka.apache.org/
[4] https://redis.io/

This feature masks the overall complexity and gives end user the power and the illusion to drive the entire system. Actually, the request is enqueued, validated by the Server component and, if still valid, notified to the smart device.

2.4 Handling connectivity issues and notifiyng anomalies

Detecting anomalies on sensed data can be challenging, since different types of anomalies require multiple acting approaches. Specifically, the architecture focuses on *connection unavailability* and *not available device*.

Missing data due to connectivity issues can be restored later if *fallback storage solutions* are implemented on the device side. Due to constrained storage capabilities, this approach could not easily implemented on typically IoT devices, but it is worth to consider it whenever such limit can be overcome. Storing data when Internet connection is not available implies focusing on two different aspects.

First of all, a correct evict data policy should be designed to avoid saturating the available capacity. Second, a retransmission logic needs to be implemented to assure stored data is sent when connectivity returns available.

From a user perspective, a delay will be experienced but no data will be lost in this scenario.

3 Conclusion and Future Work

In this work we presented a model for exposing electrical based measurements data over the Web to monitor consumptions, breakdown and preventing unforeseen events. We then discussed four aspects that characterized it. Actually, they all deal with the importance of *data* and it could make sense to explore three aspects. First of all, a set of *machine learning based algorithms* to detect anomalies; second, *harvesting multiple data sources* to publish an even more complete knowledge to end user; third, a *websocket based connection* between Server and Device as a more reactive way of keeping components chained each other.

By the designed infrastructure perspective, it could be useful to reorganize services into smaller components to match the *microservices architecture*[5] and thus deploying each service in a container.

References

1. Architectural Considerations in Smart Object Networking, `https://tools.ietf.org/html/rfc7452` (2015)
2. JSON Web Token (JWT), `https://tools.ietf.org/html/rfc7519` (2015)
3. Gul A. Agha, Actors: A Model of Concurrent Computation in Distributed Systems (1985)
4. Dominique Guinard, Vlad Trifa, Erik Wilde, A resource oriented architecture for the Web of Things (2010)
5. James Lewis, Martin Fowler, Microservices, `https://martinfowler.com/articles/microservices.html` (2014)