UNIVERSITY OF THESSALY

MASTER THESIS

Study & Implementation of an IoT Service for Device Provisioning & Controlling, Data Collection & Visualization

Author: Antonis KALKANOF

Supervisor: Prof. Athanasios KORAKIS

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in

Computer Science Department of Electrical and Computer Engineering

September 11, 2018

Abstract

Designing an IoT framework requires a multilayered approach that includes device programming, provisioning and deployment, as well as connecting those devices, monitoring and configuring their status and also executing commands on them. Finally we have data collection of the metrics being monitored from those devices and illustration of said data in a meaningful and appropriate way so that analysis can take place not just by devices but also humans that can hopefully reach to a conclusion about the data being monitored. Most free open source solutions for IoT frameworks target prototyping and fast deployment of an IoT idea, providing bare bones utility for the user who is assumed to be an expert and can master the framework - not an average user. On the other hand advanced and well configurable IoT products require a licence and are closed-source. They offer a lot of scalability and configurability which come at a cost. IoT technology and its community has advanced and can provide a lot of open source tools, that if carefully picked, configured and integrated can provide an open source and free solution. For the purposes of this thesis and the reasons described above, we created agnoNIT, an open source IoT framework for device provisioning and management, data collection and monitoring. agnoNIT was also created for the purposes of learning the inner workings of an IoT framework.

Περίληψη

Ο σχεδιασμός μιας IoT σουίτας απαιτεί μία πολυεπίπεδη προσέγγιση η οποία περιλαμβάνει προγραμματισμό, ταυτοποίηση και εγκατάσταση συσκευών, όπως επίσης και την συνδεσμολογία, παρακολούθηση και επεξεργασία της κατάστασής τους, καθώς και την εκτέλεση εντολών πάνω τους. Ακολούθως, έχουμε την συλλογή δεδομένων από τις μετρήσεις που παίρνουν οι συσκευές και την οπτικοποίηση αυτών των δεδομένων με ένα χρήσιμο και κατάλληλο τρόπο ούτως ώστε να γίνει ανάλυση όχι μόνο από συσκευές αλλά και από ανθρώπινο παράγοντα ώστε να βγούνε χρήσιμα συμπεράσματα. Οι περισσότερες λύσεις δωρεάν ελεύθερου λογισμικού στοχεύουν στην γρήγορη δημιουργία και εγκατάσταση μιας IoT ιδέας, προσφέροντας τα βασικά μόνο εργαλεία στον χρήστη. Οι λύσεις αυτές απευθύνονται σε καταρτησμένους και όχι απλούς χρήστες. Από την άλλη πλευρά τα προηγμένα και αρκετά παραμετροποιήσιμα IoT προϊόντα απαιτούν άδεια, είναι κλειστού λογισμικού και απευθύνονται κυρίως σε εταιρίες. Προσφέρουν αρκετές δυνατότητες κλιμάκωσης και παραμετροποίησης αλλά με το ανάλογο κόστος. Η IoT τεχνολογία και κοινότητα έχει προοδεύσει αρκετά ώστε να μπορεί να προσφέρει πληθώρα εργαλείων ελεύθερου λογισμικού, τα οποία εάν επιλεχθούν, παραμετροποιηθούν και ενσωματωθούν σωστά, πιστεύουμε ότι μπορούν να προσφέρουν μια ολοκληρωμένη λύση. Για τους σκοπούς της διπλωματικής αυτής και ορμώμενοι από τους παραπάνω λόγους, υλοποιήσαμε το agnoNIT, μία IoT σουίτα ελεύθερου λογισμικού για αναγνώριση και έλεγχο συσκευών, καθώς και συλλογή και οπτικοποίηση δεδομένων. Το agnoNIT δημιουργήθηκε και για τον επιπλέον λόγο της εκμάθησης των μερών που αποτελούν μία IoT σουίτα.

Acknowledgements

I am grateful to my family and friends for their support over the years.

I sincerely thank my supervisor, professor Athanasios Korakis, whose advice and knowledge helped me a lot on and off this thesis.

IoT

The Internet of Things (IoT), also called the Internet of Everything or the Industrial Internet, is a new technology paradigm envisioned as a global network of machines and devices capable of interacting with each other over a network without requiring human-to-human or human-to-computer interaction. The IoT is recognized as one of the most important areas of future technology and is gaining vast attention from a wide range of industries.

An IoT ecosystem consists of web-enabled smart devices that use embedded processors, sensors and communication hardware to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analyzed or analyzed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices -- for instance, to set them up, give them instructions or access the data. The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.

The internet of things offers a number of benefits to organizations, enabling them to:

- monitor their overall business processes
- improve the customer experience
- save time and money
- enhance employee productivity
- integrate and adapt business models
- make better business decisions and
- generate more revenue

The adoption of this technology is rapidly gaining momentum as technological, societal, and competitive pressures push firms to innovate and transform themselves. As IoT technology advances and increasing numbers of firms adopt the technology, companies are urged to rethink the ways they approach their businesses, industries and markets and gives them the tools to improve their business strategies.

At the same time IoT enhances individuals' experience at home, by offering remote monitoring of both video and sensor data such as the house's temperature and humidity, as well as smart regulated devices (such as lamps, thermostats etc)

In general IoT enables each component connected to the network to become 'smart' and interconnect/interface with the rest of the devices by exchanging data.

Essential IoT technologies

Five IoT technologies are widely used for the deployment of successful IoT-based products and services:

1. Radio frequency identification (RFID)
2. Wireless sensor networks (WSN)
3. Middleware
4. Cloud computing
5. IoT application software

Radio frequency identification (RFID)

Radio frequency identification (RFID) allows automatic identification and data capture using radio waves, a tag, and a reader. The tag can store more data than traditional barcodes. The tag contains data in the form of the Electronic Product Code (EPC), a global RFID-based item identification system developed by the Auto-ID Center. Three types of tags are used. Passive RFID tags rely on radio frequency energy transferred from the reader to the tag to power the tag; they are not battery-powered. Applications of these can be found in supply chains, passports, electronic tolls, and item-level tracking. Active RFID tags have their own battery supply and can instigate communication with a reader. Active tags can contain external sensors to monitor temperature, pressure, chemicals, and other conditions. Active RFID tags are used in manufacturing, hospital laboratories, and remote-sensing IT asset management. Semi-passive RFID tags use batteries to power the microchip while communicating by drawing power from the reader. Active and semi-passive RFID tags cost more than passive tags.

Wireless sensor networks (WSN)

Wireless sensor networks (WSN) consist of spatially distributed autonomous sensor-equipped devices to monitor physical or environmental conditions and can cooperate with RFID systems to better track the status of things such as their location, temperature, and movements (Atzori, Iera, & Morabito, 2010). WSN allow different network topologies and multihop communication. Recent technological advances in low-power integrated circuits and wireless communications have made available efficient, low-cost, low-power miniature devices for use in WSN applications (Gubbi, Buyya, Marusic, & Palaniswami, 2013).

WSN have primarily been used in cold chain logistics that employ thermal and refrigerated packaging methods to transport temperature-sensitive products (Hsueh and Chang, 2010, White and Cheong, 2012). WSN are also used for maintenance and tracking systems. For example, General Electric deploys sensors in its jet engines, turbines, and wind farms. By analyzing data in real time, GE saves time and money associated with preventive maintenance.

Likewise, American Airlines uses sensors capable of capturing 30 terabytes of data per flight for services such as preventive maintenance.

Middleware

Middleware is a software layer interposed between software applications to make it easier for software developers to perform communication and input/output. Its feature of hiding the details of different technologies is fundamental to free IoT developers from software services that are not directly relevant to the specific IoT application. Middleware gained popularity in the 1980s due to its major role in simplifying the integration of legacy technologies into new ones. It also facilitated the development of new services in the distributed computing environment. A complex distributed infrastructure of the IoT with numerous heterogeneous devices requires simplifying the development of new applications and services, so the use of middleware is an ideal fit with IoT application development. For example, Global Sensor Networks (GSN) is an open source sensor middleware platform enabling the development and deployment of sensor services with almost zero programming effort. Most middleware architectures for the IoT follow a service-oriented approach in order to support an unknown and dynamic network topology.

Cloud computing

Cloud computing is a model for on-demand access to a shared pool of configurable resources (e.g., computers, networks, servers, storage, applications, services, software) that can be provisioned as Infrastructure as a Service (IaaS) or Software as a Service (SaaS). One of the most important outcomes of the IoT is an enormous amount of data generated from devices connected to the Internet (Gubbi et al., 2013). Many IoT applications require massive data storage, huge processing speed to enable real-time decision making, and high-speed broadband networks to stream data, audio, or video. Cloud computing provides an ideal back-end solution for handling huge data streams and processing them for the unprecedented number of IoT devices and humans in real time.

IoT applications

The IoT facilitates the development of myriad industry-oriented and user-specific IoT applications. Whereas devices and networks provide physical connectivity, IoT applications enable device-to-device and human-to-device interactions in a reliable and robust manner. IoT applications on devices need to ensure that data/messages have been received and acted upon properly in a timely manner. For example, transportation and logistics applications monitor the status of transported goods such as fruits, fresh-cut produce, meat, and dairy products. During transportation, the conservation status (e.g., temperature, humidity, shock) is monitored constantly and appropriate actions are taken automatically to avoid spoilage when the connection is out of range. For example, FedEx uses SenseAware to keep tabs on the

temperature, location, and other vital signs of a package, including when it is opened and whether it was tampered with along the way.

While device-to-device applications do not necessarily require data visualization, more and more human-centered IoT applications provide visualization to present information to end users in an intuitive and easy-to-understand way and to allow interaction with the environment. It is important for IoT applications to be built with intelligence so devices can monitor the environment, identify problems, communicate with each other, and potentially resolve problems without the need for human intervention.

IoT platform

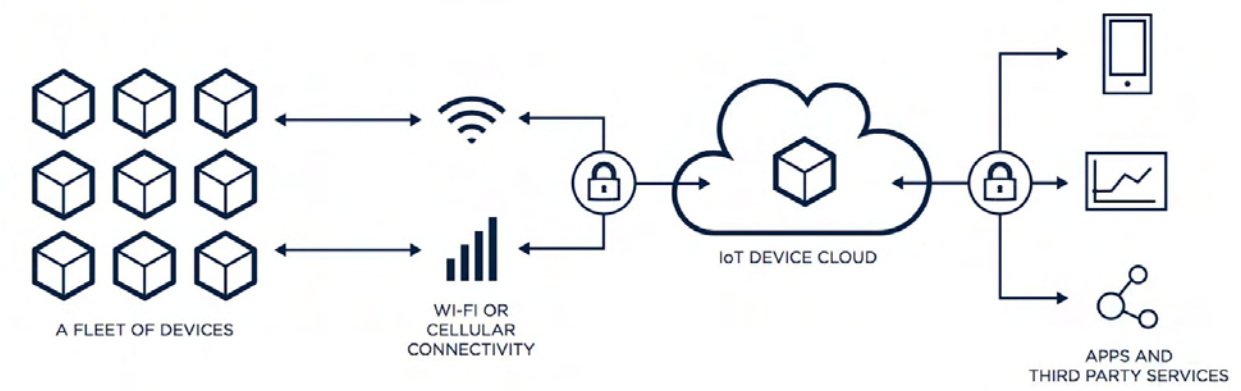

Fig.1 IoT platform architecture

An IoT platform is a multi-layer technology that enables straightforward provisioning, management, and automation of connected devices within the Internet of Things universe. It basically connects your hardware, however diverse, to the cloud by using flexible connectivity options, enterprise-grade security mechanisms, and broad data processing powers. For developers, an IoT platform provides a set of ready-to-use features that greatly speed up development of applications for connected devices as well as take care of scalability and cross-device compatibility.

Thus, an IoT platform can be wearing different hats depending on how you look at it. It is commonly referred to as *middleware* (Essential IoT technologies no.3) when we talk about how it connects remote devices (*WSN*: Essential IoT technologies no.2) to user applications (or other devices) and manages all the interactions between the hardware and the application layers. It is also known as a *cloud enablement* platform (Essential IoT technologies no.4) or IoT enablement platform to pinpoint its major business value, that is empowering standard devices with cloud-based applications and services. Finally, under the name of the *IoT application enablement* platform, it shifts the focus to being a key tool for IoT developers.(Essential IoT technologies no.5)

IoT platforms originated in the form of IoT middleware, which purpose was to function as a mediator between the hardware and application layers. Its primary tasks included data collection from the devices over different protocols and network topologies, remote device configuration and control, device management, and over-the-air firmware updates.

To be used in real-life heterogeneous IoT ecosystems, IoT middleware is expected to support integration with almost any connected device and blend in with third-party applications used by the device. This independence from underlying hardware and overhanging software allows a single IoT platform to manage any kind of connected device in the same straightforward way. Modern IoT platforms go further and introduce a variety of valuable features into the hardware and application layers as well. They provide components for frontend and analytics, on-device data processing, and cloud-based deployment. Some of them can handle end-to-end IoT solution implementation from the ground up.

IoT platform technology stack

In the four typical layers of the IoT stack, which are things, connectivity, core IoT features, and applications & analytics, a top-of-the-range IoT platform should provide you with the majority of IoT functionality needed for developing your connected devices and smart things.

Your devices connect to the platform, which sits in the cloud or in your on-premises data center, either directly or by using an IoT gateway. A gateway comes useful whenever your endpoints aren't capable of direct cloud communication or, for example, you need some computing power on edge. You can also use an IoT gateway to convert protocols, for example, when your endpoints are in LoRaWan network but you need them to communicate with the cloud over MQTT.

An IoT platform itself can be decomposed into several layers. At the bottom there is the infrastructure level, which is something that enables the functioning of the platform. You can find here components for container management, internal platform messaging, orchestration of IoT solution clusters, and others.

The communication layer enables messaging for the devices; in other words, this is where devices connect to the cloud to perform different operations.

The following layer represents core IoT features provided by the platform. Among the essential ones are data collection, device management, configuration management, messaging, and OTA software updates.

Sitting on top of core IoT features, there is another layer, which is less related to data exchange between devices but rather to processing of this data in the platform. There is reporting, which allows you to generate custom reports. There is visualization for data representation in user

applications. Then, there are a rule engine, analytics, and alerting for notifying you about any anomalies detected in your IoT solution.

Importantly, the best IoT platforms allow you to add your own industry-specific components and third-party applications. Without such flexibility adapting an IoT platform for a particular business scenario could bear significant extra cost and delay the solution delivery indefinitely.

Advanced IoT platforms

There are some other important criteria that differentiate IoT platforms between each other, such as scalability, customizability, ease of use, code control, integration with 3rd party software, deployment options, and the data security level.

- Scalable (cloud native) – advanced IoT platforms ensure elastic scalability across any number of endpoints that the client may require. This capability is taken for granted for public cloud deployments but it should be specifically put to the test in case of an on-premises deployment, including the platform's load balancing capabilities for maximized performance of the server cluster.
- Customizable – a crucial factor for the speed of delivery. It closely relates to flexibility of integration APIs, louse coupling of the platform's components, and source code transparency. For small-scale, undemanding IoT solutions good APIs may be enough to fly, while feature-rich, rapidly evolving IoT ecosystems usually require developers to have a greater degree of control over the entire system, its source code, integration interfaces, deployment options, data schemas, connectivity and security mechanisms, etc.
- Secure – data security involves encryption, comprehensive identity management, and flexible deployment. End-to-end data flow encryption, including data at rest, device authentication, user access rights management, and private cloud infrastructure for sensitive data – this is the basics of how to avoid potentially compromising breaches in your IoT solution.

Cutting across these aspects, there are two different paradigms of IoT solution cluster deployment offered by IoT platform providers: a public cloud IoT PaaS and a self-hosted private IoT cloud.

agnoNIT

Our desire to build an IoT platform from scratch stems from the fact that we design our own hardware for IoT nodes, using a range of different wireless interfaces (IEEE 802.15.4, LoRa, etc) and IoT gateways on boards such as Raspberry Pi, Beaglebone & Arduino. agnoNIT, the framework we built, covers sectors 2 through 5 from the essential IoT technologies (WSN,

middleware, Cloud and IoT apps). We've already made several unique working deployments for smart cities, smart agriculture etc. and decided we needed a framework that orchestrates and automates the whole process of a setting up and monitoring a WSN. Since there are several individuals in our lab working on different projects with limited time, this framework shouldn't rely on a single admin, but rather each user should be able to set up their own topology of sensors and acquire data without much hassle. Because of the diversity of the projects involved in our lab, agnoNIT shouldn't take anything for granted regarding the type of data being collected, the type of IoT nodes, or anything in general really. It should be agnostic to all stages of the deployment and offer ways for the user to specify and config parameters on each stage. Speaking of stages, here are all the layers the agnoNIT framework is comprised of.

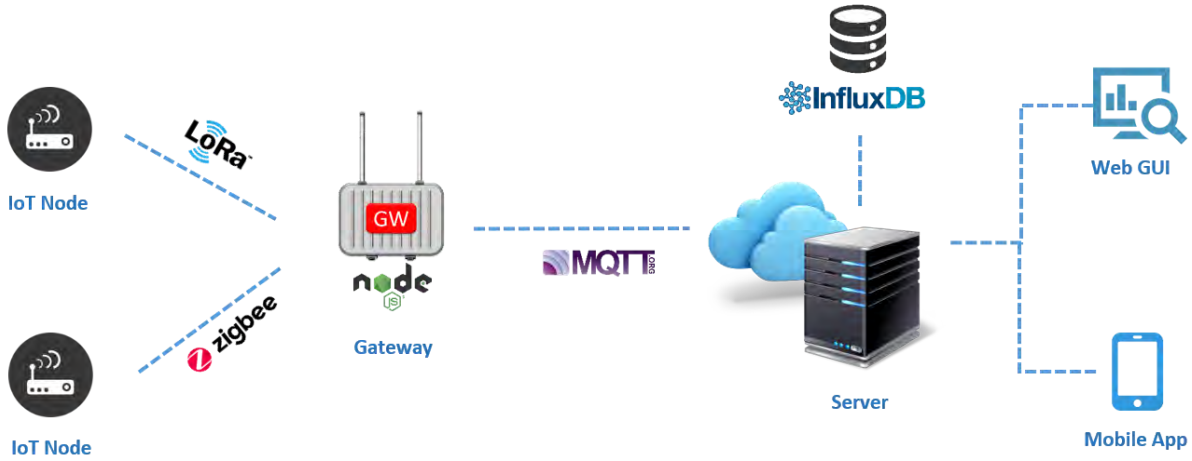

Fig.2 agnoNIT's architecture

Connectivity

Connectivity is all about messaging between the cloud and the devices. That is how the devices connect to the cloud to perform different operations.

agnoNIT is built on top of MQTT. MQTT (Message Queuing Telemetry Transport) is a publish/subscribe messaging protocol built on ton of the TCP/IP stack, designed for constrained Internet of Things devices and low-bandwidth, high-latency or unreliable networks. Because MQTT specializes in low-bandwidth, high-latency environments, it is an ideal protocol for machine-to-machine (M2M) communication.

MQTT defines basic rules of communication between the platform and the devices. The protocol is fully open, asynchronous, and allows for arbitrary message formats. Furthermore, you can choose between encrypted and unencrypted channels. Use the encrypted channel to secure sensitive data or the unencrypted channel for open data. MQTT Mosca broker is used in the heart of the pub-sub network. (fig. 3)

In case your device does not have an IP connectivity or already implements some communication capabilities, agnoNIT employs a gateway architecture where a gateway talks to the device over a local network protocol and performs transport-level message conversion or even represents them to the cloud. For example if the IoT node has direct access to the Internet via WiFi or 3G/4G, then the gateway acts as an MQTT client and connects to the MQTT broker which resides on the central server. A second way of communication may be through the LoRaWAN interface where the Gateway communicates with a LoRaWAN gateway which in turn acts as an MQTT client.

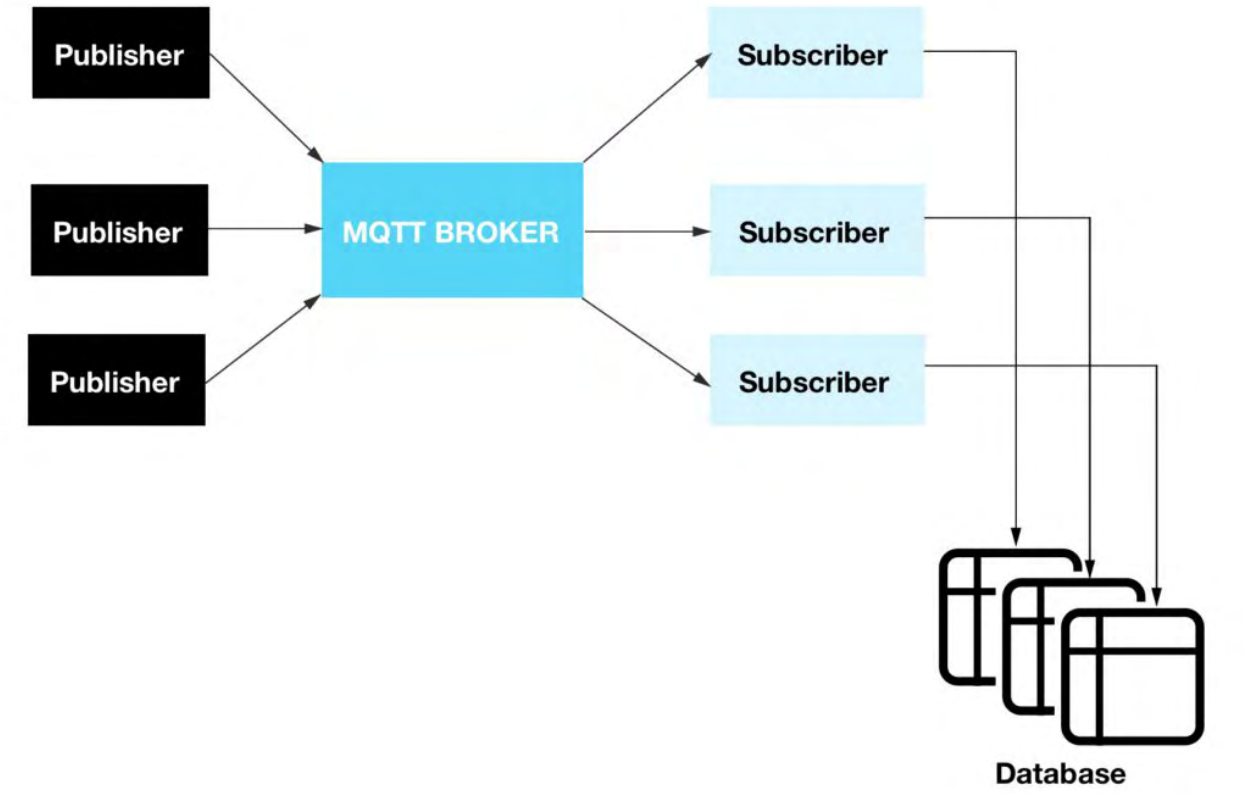

Fig.3 MQTT pub-sub system with db

Device management

At software level agroNIT provides a register of digital twins, which represent things, devices, and other entities managed by the platform. agnoNIT also allows you to store device attributes, which provide more detailed information about any characteristic of the device. Examples of such attributes could be serial number, MAC address, location, software version, etc. In addition

to simple data types, attributes can contain more complex, structured objects, such as a list of connected peripherals and their properties.

The devices we program in our lab shown below (fig. 4). There's a variety of wireless interfaces (LoRa, ZigBee, WiFi etc) and sensors (humidity, temperature, etc) to choose from for each deployment.

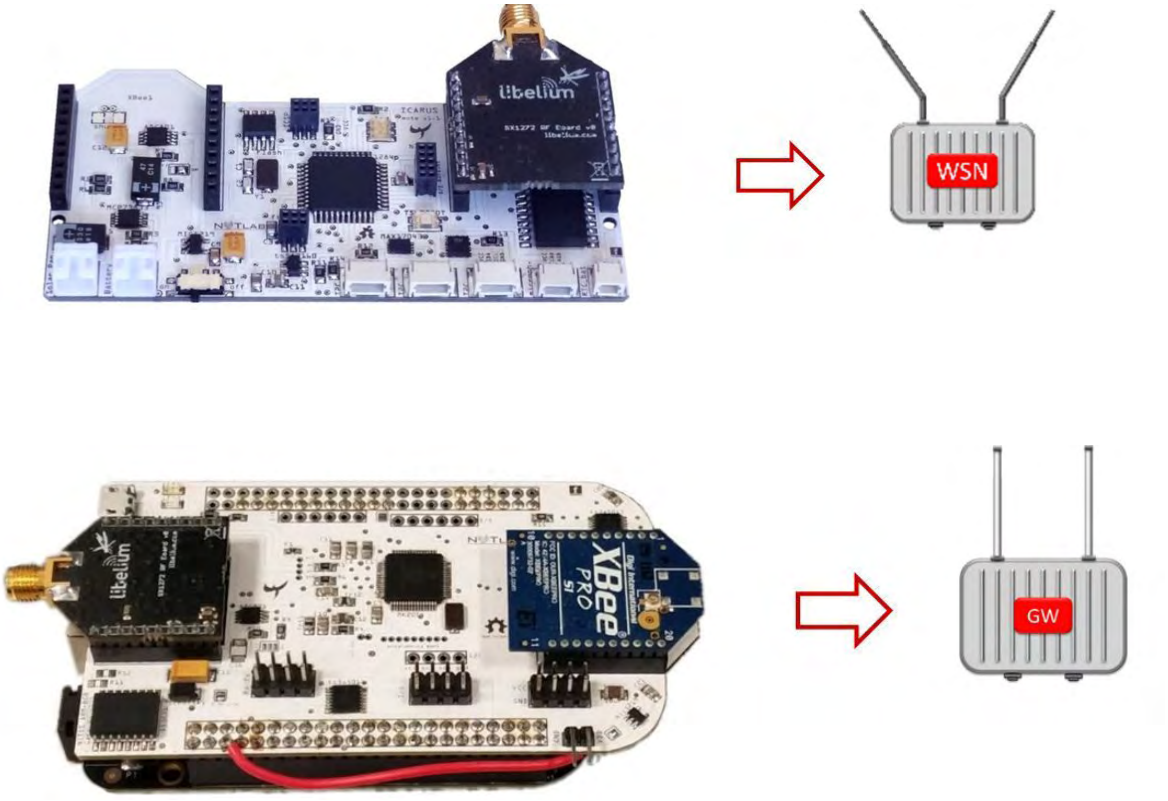

Fig.4 agnoNIT's monitoring devices

To connect to the platform, a device has to present valid credentials, such as pre-shared keys, tokens, login and password combinations, certificates, etc. You can use agnoNIT credential management APIs to provision, suspend, or revoke access (fig .5). JWT tokens are being used for authorization and authentication of devices which integrate nicely with the MQTT protocol.

agnoNIT tracks the device throughout its lifecycle, from the initial provisioning and connectivity events to software updates and final decommissioning. You can create and assign custom handlers to automate the corresponding workflows.

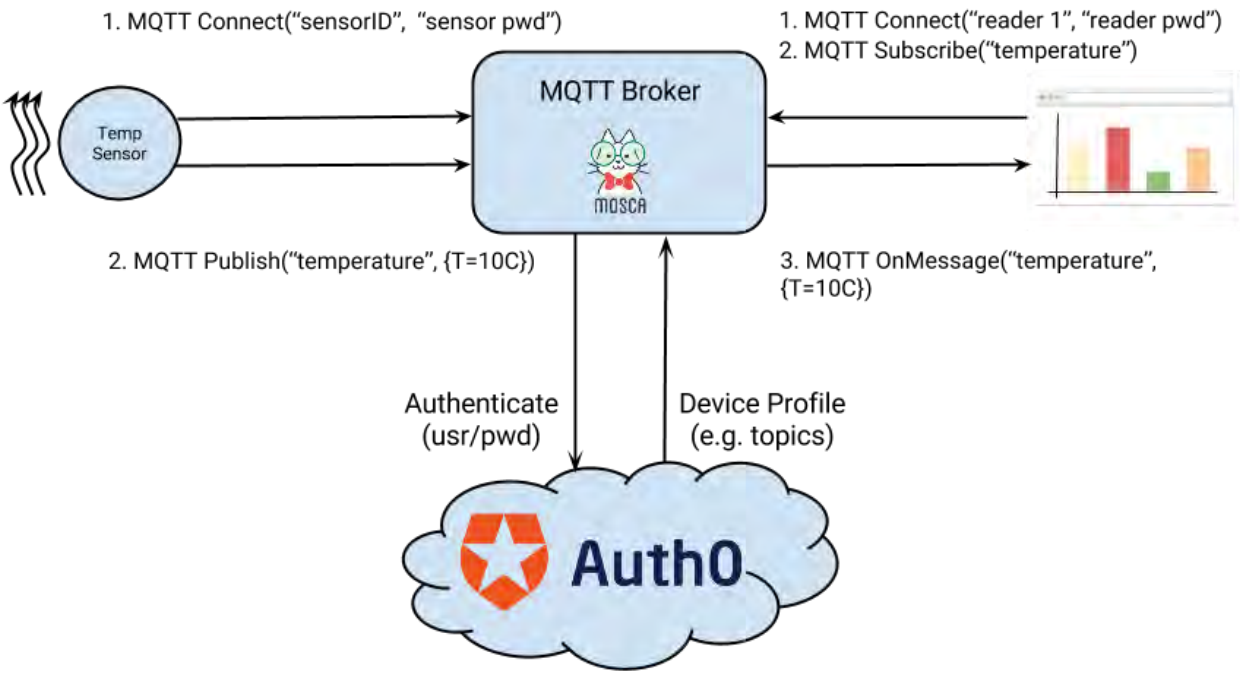

Fig.5 Device auth with JWT tokens

Data collection

agnoNIT's protocol for collecting data ensures reliable data delivery, leveraging MQTT's protocol QoS and topic structure, with response codes, which indicate the result of data processing by the platform. Once received by the platform, the device data can be dispatched to multiple processing pipelines. In case there is any error in the middle of processing, disk crash, or processor overload, the device is notified of that. As a result, the device always knows whether the submitted data is safe to delete or should be resent.

To minimize network usage and improve the data throughput, the protocol supports batching. It provides your devices with the capability to buffer data locally before uploading it in one message. Additionally, intermediary gateways can perform the store and forward function. Besides optimizing the network efficiency, this capability is useful in IoT deployments with intermittent connectivity and helps preserve the device battery life.

agnoNIT allows you to collect both structured and unstructured data. It can be of primitive types, such as plain numbers or text, or compound, such as key-value maps, arrays, or nested objects.

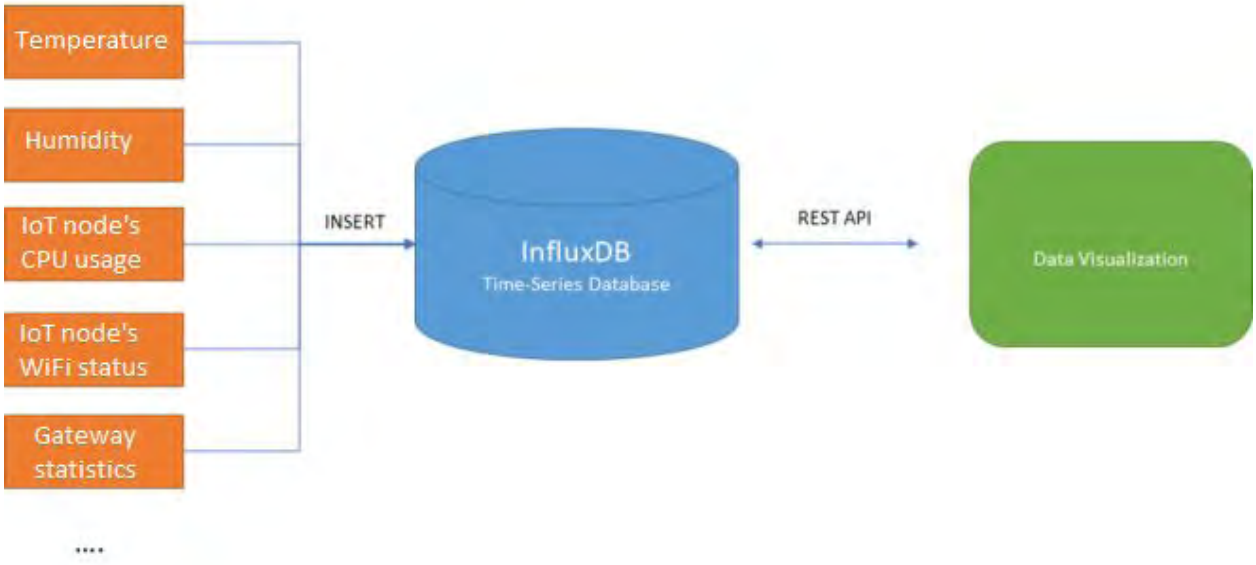

Fig.6 Data collected from different devices (IoT nodes, IoT sensors, IoT gateway) and stored at the same place.

Data processing and analytics

For the purposes of storing IoT data we concluded that time-series databases were the way to go, since they provide flexibility for such metrics. InfluxDB(fig .2 and 6) in particular is our database of choice. InfluxDB is an open-source time series database developed by InfluxData. It is written in Go and optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics. InfluxDB has no external dependencies and provides an SQL-like language with built-in time-centric functions for querying a data structure composed of measurements, series, and points. Each point consists of several key-value pairs called the fieldset and a timestamp. When grouped together by a set of key-value pairs called the tagset, these define a series. Finally, series are grouped together by a string identifier to form a measurement. Values can be 64-bit integers, 64-bit floating points, strings, and booleans. Points are indexed by their time and tagset. Retention policies are defined on a measurement and control how data is downsampled and deleted. Continuous Queries run periodically, storing results in a target measurement.

Raw (unstructured) or normal data can be transformed into well-structured time series, convenient for analytics, pattern analysis, visualization, charting, etc.

Working with time series is very flexible with agnoNIT. Apart from displaying the main value for a time series, e.g., temperature, the platform allows users to set up tags, which allow viewing some additional data, such as location, light intensity, humidity, etc. Tag values are extracted

from the collected raw data and attached to each data point in a time series. Also, it is possible to build multiple time series from one data sample.

Time series consumers (fig. 3) can be configured to listen to new data points and trigger particular actions, for example, send mobile push notifications.

Data visualization

The data visualization component of agnoNIT comprises a rich set of widgets, such as gauges, charts, maps, tables, etc. You can use these widgets to visualize different types of data, whether telemetry, statistics, geolocation, metadata, or other—both historical and current. All widgets are configurable and allow you to change their data sources as well as visual representation. To address special use cases, agnoNIT visualization component allows you to easily plug in custom widgets.

agnoNIT makes extensive use of the open source tool Chronograf (fig .7). which pairs nicely with InfluxDB as the front-end.

Besides data visualization, widgets allow you to interact with devices by sending commands, changing configuration and metadata, etc.

Dashboards help you organize widgets into logical groups and define their layout. Dashboards can be hyperlinked to streamline navigation in the complex multi-device data sets. Moreover, agnoNIT supports dashboard templating, which allows you to reuse one configuration for multiple device dashboards.

Fig. 7 power/humidity/temperature and door status for a home

Configuration management

Configuration management is essential for controlling the device behavior, managing data processing parameters, edge analytics, feature flagging, and other functions. The agnoNIT platform allows you to implement all of this functionality by providing the configuration management feature that works with arbitrary data structures. Thus, you can apply the configuration data that is as simple as a set of key-values or as complex as nested objects. Since IoT devices might not be constantly connected, agnoNIT tracks already applied configuration data as well as pending delivery.

For the purposes of this setup, we used Kapacitor. Kapacitor is a native data processing engine. It can process both stream and batch data from InfluxDB. It lets a user plug in his own custom logic or user-defined functions to process alerts with dynamic thresholds, match metrics for patterns, compute statistical anomalies, and perform specific actions based on these alerts like dynamic load rebalancing.

The default configuration management protocol of agnoNIT supports both the push and pull modes. In other words, a device may subscribe to be notified whenever the configuration is changed on the agnoNIT server or, alternatively, may periodically poll for changes itself. To ensure reliability, the configuration delivery is based on configuration application confirmations and result codes.

When the device changes its current configuration, agnoNIT generates a configuration change event. These events may be used to trigger different actions and automation processes inside and outside the platform.

The agnoNIT platform allows you to manage configuration of the devices individually or at scale. You can define configuration for a specific device or for a group of devices based on their individual characteristics, such as software version, location, or other attributes.

Command execution

Command execution is the agnoNIT platform feature that allows you to deliver messages with the arbitrary payload to connected devices, execute commands, and receive near-real time responses. For example, you can remotely check current temperature on a home thermostat.
agnoNIT implements the two-way communication that allows devices to send a response back to the server. The caller can wait for the response either synchronously or asynchronously. For lightweight commands you may go with the synchronous option meaning that the caller will be put on hold until the command execution result is delivered. For the resource-consuming commands you may choose the asynchronous option, which enables the platform to notify the caller about the command execution result.

From the device perspective, the agnoNIT platform supports both push and pull models of delivery: a device can either periodically check for new commands or rely on the server to push them. In the case of constrained devices that are unable to maintain a persistent session, agnoNIT can buffer scheduled messages until they are successfully delivered or timed out.

Over the air updates

agnoNIT allows you to reliably deliver software updates by utilizing the confirmation response codes sent by devices upon the update result.
Out of the box, agnoNIT allows tracking the current software version installed on the managed devices. Each software version is represented in agnoNIT by a flexible descriptor that is delivered to the device as an instruction to perform an update. For example, common descriptor fields include the download URL, the new software version name, etc. With this information, the device can easily download the new software, install, and report back success.

Software Architecture

General Design

Due to the nature of agroNIT, which is highly distributed and interconnected, we followed the pattern of microservices for our software development. A microservice is a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop, and test and more resilient to architecture erosion. It also allows the architecture of an individual service to emerge through continuous refactoring. Microservices-based architectures enable continuous delivery and deployment.

All the services (and microservices) we created are RESTful due to the simplicity and asynchronous nature of the HTTP protocol.

Backend

A microservice architecture requires a lot of communication between its components. This is the reason we selected Node.js as the underlying back-end technology, which has asynchronous events at its core. Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js

lets developers use JavaScript to write Command Line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server side and client side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

Node is paired nicely with ExpressJS, a lightweight web framework that provides good abstraction for core HTTP actions and backend services in general.

Front-end

agnoNIT is front-end agnostic, the same way it is data agnostic. For this reason we wanted a technology that can be used and is supported for both desktop/PC and mobile devices. We chose AngularJS which can be used for Desktop versions as is, and through Hybrid app frameworks like Ionic and Meteor for mobile development.

AngularJS is built on the belief that declarative programming should be used to create user interfaces and connect software components, while imperative programming is better suited to defining an application's business logic.The framework adapts and extends traditional HTML to present dynamic content through two-way data-binding that allows for the automatic synchronization of models and views. As a result, AngularJS de-emphasizes explicit DOM manipulation with the goal of improving testability and performance. AngularJS design goals include:

- to decouple DOM manipulation from application logic. The difficulty of this is dramatically affected by the way the code is structured.
- to decouple the client side of an application from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
- to provide structure for the journey of building an application: from designing the UI, through writing the business logic, to testing.

AngularJS implements the MVC pattern to separate presentation, data, and logic components. Using dependency injection, Angular brings traditionally server-side services, such as view-dependent controllers, to client-side web applications. Consequently, much of the burden on the server can be reduced.

For the mobile versions of the front-end we used Ionic an open-source SDK for hybrid mobile app development built on top of AngularJS and Apache Cordova. Ionic provides tools and services for developing hybrid mobile apps using Web technologies like CSS, HTML5, and Sass.

Apps can be built with these Web technologies and then distributed through native app stores to be installed on devices by leveraging Cordova.

Conclusions & Future work

Our framework has been used for pilot deployments in several fields, some of each are:

- Smart Agriculture for smart irrigation purposes and growth monitoring of fruit.
- Power and environmental monitoring of homes.
- Underwater monitoring of substances such as CO, NO2, SO2 etc.
- Presence and environmental monitoring for buildings.

We managed to have a clearer understanding of what is required to build a functioning IoT framework and reduced the overhead of setting up a new IoT monitoring deployment.

What's missing from the framework at the moment is a mechanism for acquiring specific intelligence for each unique deployment. We are working on some Machine Learning techniques for incorporating that. Also an area that needs improvement is the range of the devices we support. We work mainly with Arduino, Raspberry Pi and Beaglebone boards but we plan to support more. Finally the security sector can benefit from improvements such as secure OTA updates and OAUTH2.

References

[1] White-paper-IoT-platforms-The-central-backbone-for-the-Internet-of-Things-Nov-2015-vfi5

[2] IEEE-IoT-Towards-Definition-Internet-Of-Things

[3] https://thingsboard.io/docs/reference/architecture/

[4] https://primalcortex.wordpress.com/2015/02/25/setting-up-an-iot-frameworkdashboard-with-nodered-moscamosquitto-and-freeboard-io-dashboard/

[5] https://www.hivemq.com/mqtt-essentials/

[6] https://mqtt.org/

[7] https://docs.influxdata.com/influxdb/v1.6/

[8] https://jwt.io/