cloud.iO Documentation

Release 0.2

cloud.iO Team

Jul 26, 2018

Introduction

cloud.iO is a scalable **open source** Internet of Things solution licensed under the **MIT license**. It offers an infrastructure to **monitor** and **control** a huge number of I/O devices from a central cloud platform. cloud.iO can provide both, **real-time** monitoring and control data and **historic** data storage in a single place. Using state of the art **encryption** and certificate-based **authentication** for all connections (embedded Device Endpoints and applications accessing the actual data) and offering **flexible access control** mechanisms, cloud.iO is the ideal platform in order to drive your data acquisition projects on one single platform. It's **scalable** architecture allows it to grow with your needs.

cloud.iO is developed at the Industrial Systems Institute of the HES-SO Valais and is still in early development...

The main documentation is organized into a couple main sections:

- *Introduction*
- *User Documentation*
- *Admin Documentation*

Information for developers is also available:

- *Endpoint Developer Documentation*
- *Cloud Developer Documentation*

Finally the cloud.iO specification:

- *cloud.iO Specification*

CHAPTER 1

Motivation

Many research projects need an infrastructure to monitor and control devices deployed to many different locations whereas in the most cases an internet connection is available. In the past, each project handled those tasks differently which led to some issues:

Fig. 1: Redundant communication systems

- Each time a new solution for very similar problems were developed, which does not make sense from an economical point of view.
- Most of the projects do not dispose enough budget for the communication to create a stable/flexible solution.
- Data analyst has to work with a different database/control system for each single project.
- Data of different projects are not simple to compare and may have to be converted first.
- Scientists can not easily share their data with others.
- Remote control often is not possible, as a simple database is used at the center.
- Fine grained access control is not possible in the most cases.
- Monitoring is often very inefficient (Database Polling for example).

cloud.iO is our attempt to unify those recurring tasks in academic projects under in common base layer.

CHAPTER 2

Vision

The development of cloud computing platforms, ubiquitous networking and embedded/mobile systems has enabled the connection of various devices and appliances to the internet and the deployment of related services. Such systems where, tenths, hundreds or thousands of devices are connected to the internet are known as “Internet of Things” (IoT).

Of course, each individual project can assemble embedded sensors and actuators, develop its own communication protocols and security systems, setup specific databases and develop ad hoc services. This approach, which has been used and is still used in many projects, does not recognize the fact that there is a common pattern to all projects: each distributed “thing” feature a data model made up of measurement values, configuration parameters, status.

The vision of the cloud.iO project has been launched is to “industrialize” IoT systems deployment. To concrete this vision, the following options were considered in the design of cloud.iO:

1. The cloud.iO IoT platform is operated as SaaS (Software as a Service) and is meant to host multiple projects.
2. cloud.iO applications dispose of a syntactically uniform access to so-called data points in distributed embedded systems.
3. cloud.io does not impose any semantics for data model. Individual projects can either define their own semantics or use a free data model.
4. The cloud.iO platform manages access rights down to a single data point level, enabling the owner of a data point to provide applications with read and also possible right access to data points. Hence any privacy policy can be enforced.
5. cloud.iO keeps a history all read and write operations and provides an access to the history logs.
6. cloud.iO manages a searchable directory with the current status of all connected “things”.
7. cloud.iO defines secure connection rules for embedded devices as well as for applications.
8. cloud.iO provides APIs (Application Programming Interfaces) for the software development of embedded devices as well as for the development of cloud applications.
9. cloud.iO design philosophy is the following: reuse field proven open source software components and make minimum custom developments around them..
10. A provisioning system is responsible for the supervision of IoT devices and for triggering alarms when a problem is detected. cloud.iO provides a component letting individual projects define their own provisioning rules.

11. cloud.iO provides an interface for devices, applications and user management.
12. cloud.iO supports the plug and play deployment of large amounts of devices.

In its current status, many options listed above are not implemented, but the architecture does not put any barrier to possible future development of the not yet implemented options.

Fig. 1: cloud.iO as base for many applications

- **cloud.iO is**:
  - a low-level IoT framework allowing supervision and control of distributed systems.
  - suitable for integration either directly into end devices as well as into gateways.
  - based on proven open source technologies like , , , .
  - modular, it's micro-service architecture allows customisation to the finest granularity.
  - scalable by design.
  - very simple to use from the endpoint side (client).
  - in an early state of development.
- **cloud.iO is NOT**:
  - extremely lightweight, you will need at least a complete TCP/IP stack supporting TLS encryption and x509 certificate authentication and an implementation of the MQTT 3.1.1 protocol.
  - a high-level semantic data analysis tool, cloud.iO just offers the raw data a such a system could be build on.
  - a solution to any IoT problematic.
  - simple to deploy on the server side.

cloud.iO can be operated in two modes:

- **Private mode**: A cloud.io framework is deployed for an IoT project.
- **Public mode**: Several projects share a unique cloud.iO infrastructure.

CHAPTER 3

Elements in a cloud.iO ecosystem

A cloud.iO system is basically made up of three categories of elements as illustrated by the following figure:

Fig. 1: cloud.iO ecosystem

The following paragraphs will give a short introduction to the role of each category of elements.

3.1 Endpoint

Endpoints are distributed field devices featuring:

- a TCP/IP interface to access cloud.iO core services.
- a set of sensors and/or actuators either directly integrated in the endpoint device or connected by some (local) networking technology like Zigbee for example.

It needs a SSL certificate to communicate with cloud.iO (MQTT protocol). The definitions (interfaces, pre-processing steps, ...) are stored in CloudiO, but the implementation is done on back-end solutions, depending on the type of endpoint.

An endpoint does:

- publish his complete data model (including current values) when connecting to the cloud.
- publish a message when disconnected from the cloud.
- sends a message every time data of his data model has changed.
- allow applications or users to change the data from the outside (control, parameters).

3.2 Application

An applications is a computer programs that:

- can subscribe to input signals, which are typically results of sensors measurement.
- receive updates of subscribed input values without polling.
- set values of output signals, which are typically set points for actuators.
- access past values for any input or output signals.

It needs a SSL certificate to communicate with CloudiO (AMQP protocol).

An application can:

- search for actual data using schemes (interfaces or data classes).
- get actual and historical data for endpoint's attributes.
- control set points and parameters of endpoints.

3.3 Cloud.iO

The main objectives of the cloud.iO core services are:

- to decouple Applications from Endpoints by providing a syntactic abstraction layer.
- to keep up-to-date the topology of cloud.iO Endpoints and their current status.
- to log history values of all input signals and to make the allow Applications querying the.
- to enable Applications access to the current topology as well as the history logs.
- to enforce privacy rules based on access rights.

The history logs, the current topology and the access rights are stored in databases. These databases are named respectively *history database*, *process database* and *access rights database*.

Those databases are not part of the cloud.iO framework. The latter provides a database management system independent back-end connector for the three databases as well as a reference implementation for drivers of database management system compatible with the connectors.

3.4 User

A user is the owner of endpoints and applications. A user:

- owns one or more endpoints.
- can give other users access to his endpoints.
- can give applications access to his endpoints.
- can write his own applications.
- needs a login and password to communicate with cloud.iO (AMQP protocol)

We actually do not distinguish between different kind of users (simple users, developers, . . .).

CHAPTER 4

Building blocks

cloud.iO is based on existing and mature open source technology. The following chapters will introduce the building blocks used to develop cloud.iO.

4.1 Message based microservice architecture

cloud.iO is based on a microservice architecture as illustrated in the following figure:

This has the following advantages:

- **Simplicity**
  - A complex system can be composed by many of simple services.
  - Most of those services are completely stateless.
- **Scalability**
  - Services can be spread over multiple computing nodes.
  - Services under high load can be dynamically deployed to additional computing nodes.
- **Extensibility**
  - The System can be extended by new services or services can be modified/updated without any need for system downtime.

4.2 Spring framework

The is a mature, modular, open-source Java EE application stack with focus on web and cloud development. It offers the following features from a cloud.iO point of view:

- Lightweight and modular, ideal for microservices.

- Uses Java and Kotlin as main programming languages.
- Dependency injection and inversion of control allow decoupling of components and simplifies testing.
- Spring comes with an impressive amount of libraries and functionality to simplify developing.
- Spring offers excellent AMQP, JMS, Apache Kafka and STOMP messaging support.
- Spring offers support for many popular SQL and NoSQL databases.
- Spring is completely open source.

4.3 RabbitMQ

is a message **broker**: it accepts and forwards messages. You can think about it as a post office: when you put the mail that you want posting in a post box, you can be sure that Mr. Postman will eventually deliver the mail to your recipient. In this analogy, RabbitMQ is a post box, a post office and a postman.

The major difference between RabbitMQ and the post office is that it doesn't deal with paper, instead it accepts, stores and forwards binary blobs of data - messages.

RabbitMQ is stable, broadly used in production (Intagram as an example), scalable and has flexible message routing. User authentification and access rights are possible using multiple backends.

4.4 MQTT

(**M**essage **Q**ueue **T**elemetry **T**ransport) is a machine-to-machine (M2M) / IoT connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.

The design principles and aims of MQTT are much more simple and focused than those of AMQP; it provides publish-and-subscribe messaging (no queues, in spite of the name) and was specifically designed for resource-constrained devices and low bandwidth, high latency networks such as dial up lines and satellite links, for example. Basically, it can be used effectively in embedded systems.

One of the advantages MQTT has over more full-featured "enterprise messaging" brokers is that its intentionally low footprint makes it ideal for today's mobile and developing IoT style applications. In fact, companies like Facebook are using it as part of their mobile applications because it has such a low power draw and is light on network bandwidth.

Some of the MQTT-based brokers support many thousands of concurrent device connections. It offers three qualities of service.

MQTT's strengths are simplicity (just five API methods), a compact binary packet payload (no message properties, compressed headers, much less verbose than something text-based like HTTP), and it makes a good fit for simple push messaging scenarios such as temperature updates, stock price tickers, oil pressure feeds or mobile notifications.

4.5 AMQP

(**A**dvanced **M**essage **Q**ueuing **P**rotocol) is a messaging protocol that enables conforming client applications to communicate with conforming messaging middleware brokers.

Messaging brokers receive messages from publishers (applications that publish them, also known as producers) and route them to consumers (applications that process them).

Since it is a network protocol, the publishers, consumers and the broker can all reside on different machines.

The AMQP Model has the following view of the world: messages are published to exchanges, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to queues using rules called bindings. Then AMQP brokers either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand.

RabbitMQ uses AMQP at its core.

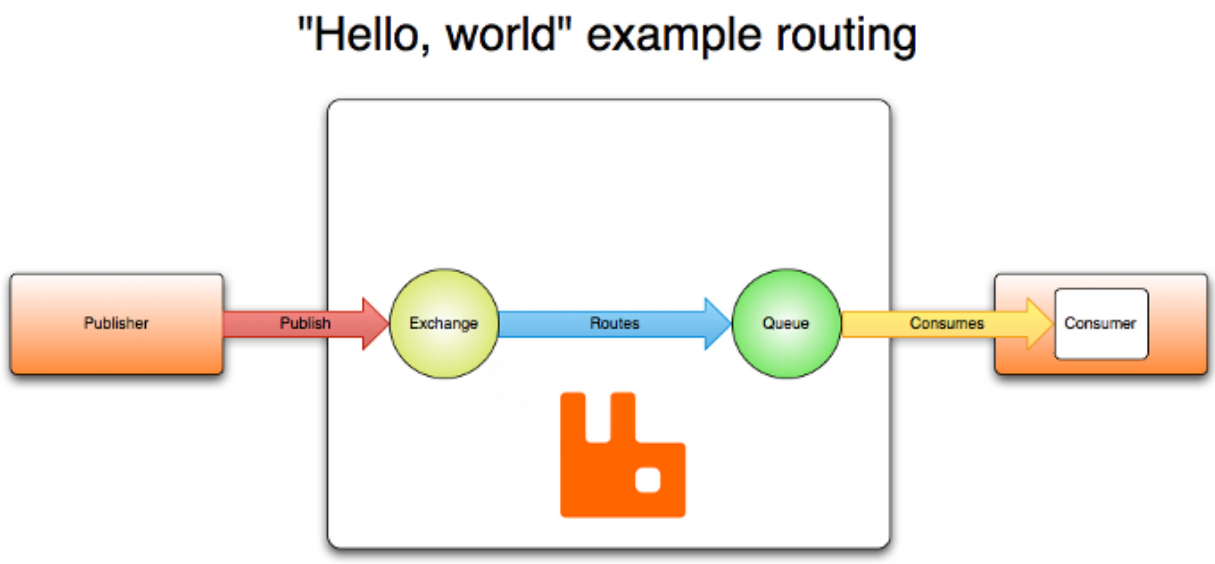

Fig. 1: https://www.rabbitmq.com/tutorials/amqp-concepts.html

CHAPTER 5

Architecture

CHAPTER 6

Getting started

Test

CHAPTER 7

Getting started

CHAPTER 8

Java endpoint development

CHAPTER 9

Python endpoint development

CHAPTER 10

Getting Started

CHAPTER 11

Protocols