dojot Documentation

Release 0.2.0

Matheus Magalhaes

abr 13, 2018

Conteúdo:

This is the high-level documentation for dojot IoT platform developed by CPqD. This platform aims to provide the application and device developers with a more concise and integrated interaction, while benefiting for a highly customizable and efficient infrastructure.

CAPÍTULO 1

Arquitetura

Este documento descreve a arquitetura atual que guia a implementação da plataforma, detalhando os componentes que compõe a solução, assim como as suas funcionalidades e como cada um deles contribui para a plataforma como um todo.

Enquanto uma breve explicação de cada componente é provida, esta descrição em alto nível não tem como objetivo explicar os detalhes de cada um dos componentes da implementação. Para isso, procure a documentação própria de cada um dos componentes.

Table of Contents

- *Comunicação*

1.1 Componentes

dojot was designed to make fast solution prototyping possible, providing a platform that's easy to use, scalable and robust. Its internal architecture makes use of many well-known open-source components with others designed and implemented by dojot team. This architecture is described on Fig. 1.1.

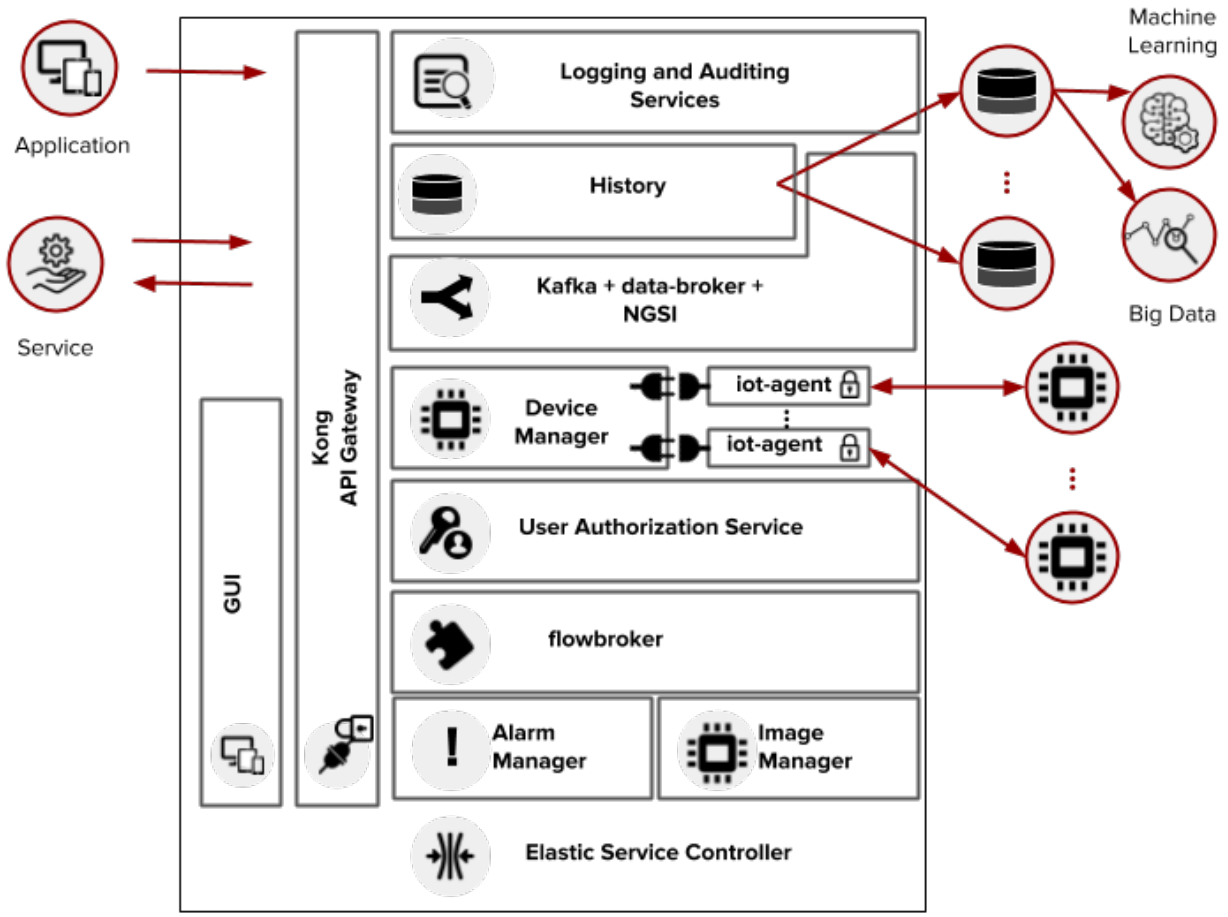

Fig. 1.1: Current Architecture

Using dojot is as follows: a user configures IoT devices through the GUI or directly using the REST APIs provided by the API Gateway. Data processing flows might be also configured - these entities can perform a variety of actions, such as generate notifications when a particular device attribute reaches a certain threshold or save all data generated by a device onto an external database. As devices start sending their readings to dojot, the user might want to receive these readings via notifications generated by subscriptions, consolidate all data into virtual devices, gather all data from historical database, and so on. These features can be used through REST APIs - these are the basic building blocks that any application based on dojot should use. dojot GUI provides an easy way to perform management operations for all entities related to the platform (users, devices, templates and flows) and can also be used to check if everything is working fine.

The user contexts are isolated and there is no data sharing, the access credentials are validated by the authorization

service for each and every operation (API Request). Once devices are configured, the IoT Agent is capable of mapping the data received from devices, encapsulated on MQTT for example, and send them to the context broker for internal distribution, reaching, for instance, the history service so it can persist the data on a database. If certain conditions are matched when rules are being processed, a new event is generated and sent to the broker service to be redistributed to the interested services.

For more information about what's going on with dojot, you should take a look at dojot GitHub repository. There you'll find all components used in dojot.

Cada um dos componentes que compõe a arquitetura são brevemente descritos nas sessões subsequentes.

1.1.1 Kafka + data-broker + NGSI

Apache Kafka is a distributed messaging platform that can be used by applications which need to stream data or consume/produce data pipelines. In comparison with other open-source messaging solutions, Kafka seems to be more appropriate to fulfil *dojot*'s architectural requirements (responsibility isolation, simplicity, and so on).

No Kafka, utiliza-se uma estrutura de tópicos especializada para garantir isolamento de dados de usuários e aplicações, viabilizando uma arquitetura multi-tenant.

The flow-broker service makes use of an in-memory database for efficiency. It adds context to Apache Kafka, making it possible that internal or even external services are able to subscribe or query data based on context. Flow-broker is also a distributed service to avoid it being a single point of failure or even a bottleneck for the architecture.

To keep a certain level of compatibility with NGSI-compatible components, it is possible to build an element that offers a NGSI interface for such components.

1.1.2 DeviceManager

DeviceManager is a core entity which is responsible for keeping device and templates data models. It is also responsible for publishing any updates to all interested components (namely IoT agents, history and subscription manager) through Kafka.

O serviço não mantém estados e tem seus dados persistidos em banco de dados, onde suporta isolamento de dados por usuários e aplicações, viabilizando uma arquitetura de middleware com multi-tenancy.

1.1.3 IoT Agent

An IoT agent is an adaptation service between physical devices and *dojot's* core components. It could be understood as a *device driver* for a set of devices. The *dojot* platform can have multiple iot-agents, each one of them being specialized in a specific protocol like, for instance, MQTT/JSON, CoAP/LWM2M and HTTP/JSON.

It is also responsible to ensure that it communicates with devices using secure channels.

1.1.4 Serviço de Autorização de Usuários

Serviço que implementa o gerenciamento de perfil de usuários e controle de acesso. Basicamente qualquer chamada de aplicação através do API Gateway é validada por este serviço.

Para ser capaz de atender a um grande volume de chamadas de autorização, faz uso de cache, não mantém estados e pode ser escalado horizontalmente. Seus dados são mantidos em banco de dados clusterizável.

1.1.5 flowbroker

This service provides mechanisms to build data processing flows to perform a set of actions. These flows can be extended using external processing blocks (which can be added using REST APIs).

1.1.6 History

The History component works as a pipeline for data and events that must be persisted on a database. The data is converted into an storage structure and is sent to the corresponding database.

Para armazenamento interno, utiliza-se uma base de dados não-relacional MongoDB que pode ser configurada em modo Sharded Cluster dependendo do caso de uso.

Os dados também podem ser armazenados em base de dados externa a plataforma dojot. Para isto, basta configurar o Logstash para enviar os dados para a base correspondente conforme a estrutura de dados desejada.

1.1.7 Serviço de Registro e Auditoria

All the services that are part of the *dojot* platform can generate usage metrics of its resources that can be used by a logging and auditing service, which process this registers and summarize then based on users and applications.

Os dados consolidados são disponibilizados para outros serviços do próprio dojot, permitindo-lhes, por exemplo, expor esses dados através de uma interface gráfica para os usuários, ára limitar o uso do sistema baseado no consumo de recursos e cotas associadas a usuários ou ainda pode ser usado por serviços externos de faturamento em função da utilização da plataforma por usuários.

Such components are currently in development.

1.1.8 Kong API Gateway

O Kong API Gateway é utilizado como um ponto de fronteira entre as aplicações e serviços externos e os serviços internos do dojot, isto resulta em inúmeras vantagens como, por exemplo, ponto único de acesso e facilidade na aplicação de regras sobre as chamadas de APIs como limitação de tráfego e controle de acesso.

1.1.9 GUI

The Graphical User Interface in *dojot* is responsible for providing responsive interfaces to manage the platform, including functionalities like:

- **Gerenciamento de perfil de usuários**: definir perfis e quais APIs podem ou não serem acessadas pelo respectivo perfil.
- **Gerenciamento de usuários**: operações de criação, visualização, edição e remoção.
- **Gerenciamento de aplicações**: operações de criação, visualização, edição e remoção.
- **Gerenciamento de modelos de dispositivos**: operações de criação, visualização, edição e remoção.
- **Gerenciamento de dispositivos**: operações de criação, visualização (dispositivo e dados em tempo real), edição e remoção.
- **Gerenciamento de fluxo de processamentos**: operações de criação, visualização, edição e remoção de fluxos de processamento de dados.

1.1.10 Controlador de Serviços Elástico

Serviço especializado para ambientes de nuvem que monitora a utilização da plataforma, diminuindo ou aumentando a sua capacidade de processamento e armazenamento de maneira automática e dinâmica de forma a se adaptar a variação da demanda.

Este controlador depende que os serviços que compõem o dojot possam ser escalados horizontalmente, assim como, que os bancos de dados utilizados sejam clusterizáveis, que é o caso da arquitetura adotada.

This component is currently scheduled for development.

1.1.11 Alarm Management

This component is responsible for handling alarms generated by dojot's internal components, such as IoT agents, Device Manager, and so on.

1.1.12 Image manager

This component is responsible for device image storage and retrieval.

1.2 Infraestrutura

A few extra components are used in dojot that were not shown in Fig. 1.1. They are:

- postgres: this database is used to persist data from many components, such as Device Manager.
- redis: in-memory database used as cache in many components, such as service orchestrator, subscription manager, IoT agents, and so on. It is very light and easy to use.
- rabbitMQ: message broker used in service orchestrator in order to implement action flows related that should be applied to messages received from components.
- mongo database: widly used database solution that is easy to use and doesn't add a considerable access overhead (where it was employed in dojot).
- zookeeper: keeps replicated services within a cluster under control.

1.3 Comunicação

All components communicate with each other in two ways:

- Using HTTP requests: if one component needs to retrieve data from other one, say an IoT agent needs the list of currently configured devices from Device Manager, it can send a HTTP request to the appropriate component.
- Using Kafka messages: if one component needs to send new information about a resource controlled by it (such as new devices created in Device Manager), the component may publish this data through Kafka. Using this mechanism, any other component that is interested in such information needs only to listen to a particular topic to receive it. Note that this mechanism doesn't make any hard associations between components. For instance, Device Manager doesn't know which components need its information, and an IoT agent doesn't need to know which component is sending data through a particular topic.

CAPÍTULO 2

User Guide

This document provides information on how to use dojot from a device developer or application developer point of view.

Table of Contents

2.1 Who should read this

- Users that want a deeper look at how dojot works;
- Application developers.

2.2 Getting Started

To start, please follow dojot's installation guide in *Installation Guide*. There you should find how to properly download a working copy of the components, how to minimally configure them, how to start them up and how to check whether they are working.

2.3 dojot basics

Before using dojot, you should be familiar with some basic operations and concepts. They are very simple to understand and use, but without them, all operations might become obscure and senseless.

In the next section, there is an explanation of a few basic entities in dojot: devices, templates and flows. With these concepts in mind, we present a small tutorial to how to use them in dojot - it only covers API access.

If you want more information on how dojot works internally, you should checkout the *Arquitetura* to get acquainted with all internal components.

2.3.1 User authentication

All HTTP requests supported by dojot are sent to the API gateway. In order to control which user should access which endpoints and resources, dojot makes uses of JSON Web Token (a useful tool is jwt.io) which encodes things like (not limited to these):

- User identity
- Validation data
- Token expiration date

The component responsible for user authentication is auth. You can find a tutorial of how to authenticate a user and how to get an access token in auth documentation.

2.3.2 Devices and templates

In dojot, a device is a digital representation of an actual device or gateway with one or more sensors or of a virtual one with sensors/attributes inferred from other devices. Throughout the documentation, this kind of device will be called simply as 'device'. If the actual device must be referenced, we'll be calling it as 'physical device'.

Consider, for instance, a physical device with temperature and humidity sensors; it can be represented in dojot as a device with two attributes (one for each sensor). We call this kind of device as regular device or by its communication protocol, for instance, MQTT device or CoAP device.

We can also create devices which don't directly correspond to their physical counterparts, for instance, we can create one with higher level of information of temperature (is becoming hotter or is becoming colder) whose values are inferred from temperature sensors of other devices. This kind of device is called virtual device.

All devices are created based on a *template*, which can be thought as a model of a device. As "model" we could think of part numbers or product models - one *prototype* from which devices are created. Templates in dojot have one label (any alphanumeric sequence), a list of attributes which will hold all the device emitted information, and optionally a few special attributes which will indicate how the device communicates, including transmission methods (protocol, ports, etc.) and message formats.

In fact, templates can represent not only "device models", but it can also abstract a "class of devices". For instance, we could have one template to represent all thermometers that will be used in dojot. This template would have only one attribute called, let's say, "temperature". While creating the device, the user would select its "physical template",

let's say *TexasInstr882*, and the 'thermometer' template. The user would have also to add translation instructions (implemented in terms of data flows, build in flowbuilder) in order to map the temperature reading that will be sent from the device to a "temperature" attribute.

In order to create a device, a user selects which templates are going to compose this new device. All their attributes are merged together and associated to it - they are tightly linked to the original template so that any template update will reflect all associated devices.

The component responsible for managing devices (both real and virtual) and templates is DeviceManager. DeviceManager documentation explains in more depth all the available operations.

2.3.3 Flows

A flow is a sequence of blocks that process a particular event or device message. It contains:

- entry point: a block representing what is the trigger to start a particular flow;
- processing blocks: a set of blocks that perform operations using the event. These blocks may or may not use the contents of such event to further process it. The operations might be: testing content for particular values or ranges, geo-positioning analysis, changing message attributes, perform operations on external elements, and so on.
- exit point: a block representing where the resulting data should be forwarded to. This block might be a database, a virtual device, an external element, and so on.

The component responsible for dealing with such flows is flowbroker.

2.3.4 Step-by-step device management

This section provides a complete step-by-step tutorial of how to create, update, send messages to and check historical data of a device. We will create a simple device with only one attribute, send a few messages emulating the physical device and check the historical data for the only attribute this device has.

Also, this tutorial assumes that you are using docker-compose, which has all the necessary components to properly run dojot (so all API requests will be sent to localhost:8000).

Getting access token

As said in *User authentication*, all requests must contain a valid access token. You can generate a new token by sending the following request:

```
curl -X POST http://localhost:8000/auth \
       -H 'Content-Type:application/json' \
       -d '{"username": "admin", "passwd" : "admin"}'

{"jwt": "eyJ0eXAiOiJKV1QiL..."}
```

If you want to generate a token for other user, just change the username and password in the request payload. The token ("eyJ0eXAiOiJKV1QiL...") should be used in every HTTP request sent to dojot in a special header. Such request would look like:

```
curl -X GET http://localhost:8000/device \
  -H "Authorization: Bearer eyJ0eXAiOiJKV1QiL..."
```

Remember that the token must be set in the request header as a whole, not parts of it. In the example only the first characters are shown for the sake of simplicity. All further requests will use a bash variable called `bash ${JWT}`, which contains the token got from auth component.

Device creation

In order to properly configure a physical device in dojot, you must first create its representation in the platform. The example presented here is just a small part of what is offered by DeviceManager. For more information, check the DeviceManager how-to for more detailed instructions.

First of all, let's create a template for the device - all devices are based off of a template, remember.

```
curl -X POST http://localhost:8000/template \
-H "Authorization: Bearer ${JWT}" \
-H 'Content-Type:application/json' \
-d ' {
  "label": "Thermometer Template",
  "attrs": [
    {
      "label": "temperature",
      "type": "dynamic",
      "value_type": "float"
    }
  ]
}'
```

This request should give back this message:


```
{
  "result": "ok",
  "template": {
    "created": "2018-01-25T12:30:42.164695+00:00",
    "data_attrs": [
      {
        "template_id": "1",
        "created": "2018-01-25T12:30:42.167126+00:00",
        "label": "temperature",
        "value_type": "float",
        "type": "dynamic",
        "id": 1
      }
    ],
    "label": "Thermometer Template",
    "config_attrs": [],
    "attrs": [
      {
        "template_id": "1",
        "created": "2018-01-25T12:30:42.167126+00:00",
        "label": "temperature",
        "value_type": "float",
        "type": "dynamic",
        "id": 1
      }
    ],
    "id": 1
  }
}
```

Note that the template ID is 1 (line 27).

To create a template based on it, send the following request to dojot:

```
1  curl -X POST http://localhost:8000/device \
2  -H "Authorization: Bearer ${JWT}" \
3  -H 'Content-Type:application/json' \
4  -d ' {
5    "templates": [
6      "1"
7    ],
8    "label": "device"
9  }'
```

The template ID list on line 6 contains the only template ID configured so far. To check out the configured device, just send a GET request to /device:

```
curl -X GET http://localhost:8000/device -H "Authorization: Bearer ${JWT}"
```

Which should give back:

```
{
  "pagination": {
    "has_next": false,
    "next_page": null,
    "total": 1,
    "page": 1
  },
  "devices": [
    {
      "templates": [
        1
      ],
      "created": "2018-01-25T12:36:29.353958+00:00",
      "attrs": {
        "1": [
          {
            "template_id": "1",
            "created": "2018-01-25T12:30:42.167126+00:00",
            "label": "temperature",
            "value_type": "float",
            "type": "dynamic",
            "id": 1
          }
        ]
      },
      "id": "0998",
      "label": "device_0"
    }
  ]
}
```

Sending messages

So far we got an access token and created a template and a device based on it. In an actual deployment, the physical device would send messages to dojot with all its attributes and their current values. For this tutorial we will send

MQTT messages by hand to the platform, emulating such physical device. For that, we will use mosquitto_pub from Mosquitto project.

> **Atenção:** Some Linux distributions, Ubuntu in particular, have two packages for mosquitto - one containing tools to access it (i.e. mosquitto_pub and mosquitto_sub for publishing messages and subscribing to topics) and another one containing the MQTT broker. In this tutorial, only the tools are going to be used. Please check if MQTT broker is not running before starting dojot (by running commands like `ps aux | grep mosquitto`).

The default message format used by dojot is a simple key-value JSON (you could translate any message format to this scheme using flows, though), such as:

```
{
  "temperature" : 10.6
}
```

Let's send this message to dojot:

```
mosquitto_pub -t /admin/0998/attrs -m '{"temperature": 10.6}'
```

If there is no output, the message was sent to MQTT broker.

As noted in the FAQ, there are some considerations regarding MQTT topics:

- If you don't define any topic in device template, it will assume the pattern `/<service-id>/<device-id>/attrs` (for instance: `/admin/efac/attrs`). This should be the topic to which the device will publish its information to.
- If you do define a topic in device template, then your device should publish its data to it and set the client-id parameter. It should follow the following pattern: `<service>:<deviceid>`, such as `admin:efac`.
- MQTT payload must be a JSON with each key being an attribute of the dojot device, such as:

```
{ "temperature" : 10.5,"pressure" : 770 }
```

For more information on how dojot deals with data sent from devices, check the *Integrating physical devices* section.

Checking historical data

In order to check all values that were sent from a device for a particular attribute, you could use the history APIs. Let's first send a few other values to dojot so we can get a few more interesting results:

```
mosquitto_pub -t /admin/3bb9/attrs -m '{"temperature": 36.5}'
mosquitto_pub -t /admin/3bb9/attrs -m '{"temperature": 15.6}'
mosquitto_pub -t /admin/3bb9/attrs -m '{"temperature": 10.6}'
```

To retrieve all values sent for temperature attribute of this device:

```
curl -X GET \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsIn...' \
  "http://localhost:8000/history/device/3bb9/history?lastN=3&attr=temperature"
```

The history endpoint is built from these values:

- `.../device/3bb9/...`: the device ID is `3bb9` - this is retrieved from the `id` attribute from the device
- `.../history?lastN=3&attr=temperature`: the requested attribute is temperature and it should get the last 3 values. More operators are available in history APIs.

The request should result in the following message:


```
[
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:47:07.050000Z",
    "value": 10.6,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:42.455000Z",
    "value": 15.6,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:21.535000Z",
    "value": 36.5,
    "attr": "temperature"
  }
]
```


This message contains all previously sent values.

2.4 Integrating physical devices

If you want to integrate your device within dojot, it must be able to send messages to the platform. There are two ways to do that:

- Use one of the available IoT agents: currently, there is support for MQTT-based devices. If your project is using (or allows changing to) this protocol, then it would suffice to check if the device is sending its data using a simple key/value JSON. If it isn't, then you might want to use iotagent-mosca (check iotagent-mosca documentation to check out how to do that). If it is indeed sending key/value JSON messages, then it can send its messages to dojot's broker and it will be recognized by the platform.
- Create a new IoT agent to support the protocol used by the device: if your device is using another protocol that is not yet supported, then it might be a good idea to implement a new IoT agent. It's not that hard, but there are a few details that must be taken into account. To help developers to do such thing, there is the iotagent-nodejs library which deals with most internal mechanisms and messages - check its documentation to know more.

After your device is able to communicate with dojot, you can start using it as described in *Step-by-step device management*.

CAPÍTULO 3

Components and APIs

3.1 Components

Tabela 3.1: Components

Component	GitHub repository	Documentation
mongodb		mongodb documentation
postgres		postgres documentation
Kong API gateway		Kong documentation
redis		Redis documentation
zookeeper		Zookeeper documentation
Kafka		Kafka documentation
auth	GitHub - auth	readthedocs - auth
History	GitHub - history-ws	
DeviceManager	GitHub - DeviceManager	readthedocs - DeviceManager
Image manager	GitHub - image-manager	
GUI	GitHub - GUI	
Flow broker	GitHub - flowbroker	
Data broker	GitHub - data-broker	
iotagent-mosca	GitHub - iotagent-mosca	
EJBCA-REST	GitHub - EJBCA-REST	
Alarm manager	GitHub - alarm-manager	

3.2 Exposed APIs

Tabela 3.2: APIs :header-rows: 1

Endpoint	Purpose	Component API	Repository
/device	Device management	API - DeviceManager	GitHub - DeviceManager
/template	Template management	API - DeviceManager	GitHub - DeviceManager
/flows	Flow management	API - flowbroker	GitHub - flowbroker
/auth	User authentication	API - auth	GitHub - auth
/auth/revoke	User authentication	API - auth	GitHub - auth
/auth/user	User authentication	API - auth	GitHub - auth
/history	Device historical data	API - history-ws	GitHub - history-ws
/metric	Context broker	API - data-broker	GitHub - data-broker
/gui	Graphical User Interface		GitHub - GUI
/sign	Public key signing	API - EJBCA-REST	GitHub - EJBCA-REST
/ca	Certification-Auth. functions	API - EJBCA-REST	GitHub - EJBCA-REST
/image	Device image management	API - image-manager	GitHub - image-manager

The API gateway used in dojot reroutes some of these endpoints so that they become uniform: all of them are accessible through the same port (default is TCP port 8000) and have the same naming scheme. Each component, though, might have something different in its configuration and API documentation. The following table shows which endpoint exposed by the API gateway is mapped to which component endpoint.

Tabela 3.3: Original endpoints

Service	Original endpoint	Endpoint
DeviceManager	host:5000/device	host:8000/device
DeviceManager	host:5000/template	host:8000/template
mashup	host:3000/	host:8000/flows
auth	host:5000/	host:8000/auth
auth	host:5000/auth/revoke	host:8000/auth/revoke
auth	host:5000/user	host:8000/auth/user
STH	host:8666/	host:8000/history
Data-Broker	host:1026/	host:8000/metric
GUI	host/	host:8000/gui
ejbca	host:5583/sign	host:8000/sign
ejbca	host:5583/ca	host:8000/ca

3.3 Kafka messages

These are the messages sent by components and their subjects. If you are developing a new internal component (such as a new IoT agent), see API - data-broker to check how to receive messages sent by other components in dojot.

Tabela 3.4: Original endpoints

Component	Message	Subject
DeviceManager	Device CRUD (Messages - DeviceManager)	`dojot.device-manager.device`
iotagent-mosca	Device data update (Messages - iotagent-mosca)	`device-data`

CAPÍTULO 4

Installation Guide

This page contains information about how to deploy dojot using Docker compose. Kubernetes and Google Cloud Platform support is on track to be implemented.

Table of Contents

4.1 Installation - Docker compose

This document provides instructions on how to create a trivial deployment environment on single host for *dojot*, using docker-compose as the processes orchestration platform.

While very simple, this deployment option is best suited to development and assessment of the platform and should not be used for production environments.

This guide has been checked on an Ubuntu 16.04 LTS environment.

4.1.1 Dependencies

This setup has two software requirements docker engine and docker-compose.

Docker engine

Up to date information and installation procedures for the docker engine can be found at the project's documentation:

https://docs.docker.com/engine/installation/

Nota: An optional step on the installation and configuration process of docker on any given machine is the setting of who is eligible for creating/spawning docker instances.

Should the post-installation steps (more specifically the "Manage docker as non-root user") have not been run, all docker and docker-compose commands should be run by the super user (root), or as sudo.

https://docs.docker.com/engine/installation/linux/linux-postinstall/

Docker Compose

Up to date information and installation procedures for the docker-compose can be found at the project's documentation:

https://docs.docker.com/compose/install/

4.1.2 Installation

To setup the environment, merely clone the deployment repository and run the commands below.

The docker-compose enabled deployment scripts and configuration repository can be found at:

https://github.com/dojot/docker-compose

or as git clone command::

```
git clone https://github.com/dojot/docker-compose.git
# Let's move into the repo - all commands in this page should be executed
# inside it.
cd docker-compose
```

Once the repository is properly cloned, select the version to be used by checking out the appropriate tag (do notice that the tagname has to be replaced):

```
# Must be run from within the deployment repo

git checkout tag_name
```

For instance:

```
git checkout 0.1.0-dojot
```

Or if you're brave enough:

```
git checkout master
```

That done, the environment can be brought up by:

```
# Must be run from the root of the deployment repo.
# May need sudo to work: sudo docker-compose up -d
docker-compose up -d
```

To check individual container status, docker's commands may be used, for instance:

```
# Shows the list of currently running containers, along with individual info
docker ps

# Shows the list of all configured containers, along with individual info
docker ps -a
```

Nota: All docker, docker-compose commands may need sudo to work.

To allow non-root users to manage docker, please check docker's documentation:

https://docs.docker.com/engine/installation/linux/linux-postinstall/

4.1.3 Usage

The web interface is available at `http://localhost:8000`. The user is `admin` and the password is `admin`. You also can interact with platform using the *REST API*.

Read the *User Guide* for more information about how to interact with the platform.

CAPÍTULO 5

Dúvidas Mais Frequentes

Here are some answers to frequently-asked questions from users of dojot platform.

Não encontrou aqui uma resposta para a sua dúvida? Por favor, abra uma *issue* no repositório da dojot no Github.

Sumário

5.1 Gerais

5.1.1 O que é a dojot? Por que eu deveria utilizá-la? Por que abrir o código?

It's a brazilian IoT platform launched as open source software with aims to ease the development of solutions and the IoT ecosystem with local resources geared towards brazilians needs.

It takes a role as an enabler platform with:

- APIs abertas tornando o acesso fácil das aplicações aos recursos da plataforma.
- Capacidade de armazenamento de grandes volumes de dados em diferentes formatos.
- Conectores para diferentes tipos de dispositivos.

- Construção de fluxos de dados e regras de forma visual, permitindo a rápida prototipação e validação de cenários de aplicações IoT.
- Processamento de eventos em tempo real aplicando regras definidas pelo desenvolvedor.

5.1.2 Onde eu posso baixar?

Todos os componentes estão disponíveis no repositório da dojot no GitHub: https://github.com/dojot.

5.1.3 Qual é o principal repositório?

Existem dois repositórios principais:

- https://github.com/dojot/dojot: é aqui que concentramos o acompanhamento de tudo relacionado a este projeto como decisões de arquitetura e melhorias.
- https://github.com/dojot/docker-compose: repositório com os arquivos e configurações para o docker-compose. Este é o repositório que recomendamos para começar com a dojot.

5.1.4 Então, encontrei um probleminha chato. Como posso informá-lo sobre isso?

Pedimos que você abra uma *issue* com o problema no repositório da dojot no Github. Se você souber exatamente qual componente está com o problema, você pode abrir a *issue* no respectivo repositório (funcionará do mesmo modo).

Se você puder analisar e resolver o problema, por favor faça isso e crie um *pull-request* com uma breve descrição do que foi feito.

5.2 Uso

5.2.1 Por onde eu começo? É baseado em CLI ou possui uma interface gráfica de usuário ?

dojot can be accessed by a nice web-based interface and by REST APIs. Considering that you installed `docker` and `docker-compose` and cloned the `docker-compose` repository, starting it up is done by just one command:

```
$ docker-compose up -d
```

E é isto.

A interface Web está disponível em `http://localhost:8000`. O usuário é `admin` e a senha é `admin`.

APIs REST são explicadas na seção *Aplicações*.

5.2.2 Pronto, já iniciei e fiz o *login*. E agora?

Legal! Agora você pode criar seus primeiros dispositivos, descrito em *Dispositivos*, criar alguns fluxos e registrar-se para eventos de dispositivos, ambos descritos em *Fluxos de Dados*.

5.2.3 Como posso atualizar o meu ambiente com a última versão da dojot?

Basta seguir alguns passos:

1 Update the docker-compose repository to the cutting-edge version (beware the bugs though)

```
$ cd <path-to-your-clone-of-docker-compose>
$ git checkout master && git pull

If you need a more stable version, you could checkout a tag instead:
```

```
$ git tag
0.1.0-dojot
0.1.0-dojot-RC1
0.1.0-dojot-RC2
0.2.0-aikido

$ git checkout 0.2.0-aikido -b 0.2.0
```

2 Deploy the latest docker images. This command might need `sudo`.

```
$ docker-compose pull && docker-compose up -d
```

Este procedimento também se aplica para as máquinas virtuais dojot uma vez que as mesmas utilizam *docker-compose*.

5.3 Dispositivos

5.3.1 O que são dispositivos para a dojot?

Na dojot, um dispositivo é uma representação digital para um dispositivo real ou gateway com um ou mais sensores ou uma representação para um dispositivo virtual com sensores/atributos inferidos de outros dispositivos.

Consider, for instance, an actual device with thermal and humidity sensors; it can be represented inside dojot as a device with two attributes (one for each sensor). We call this kind of device as *regular device* or by its communication protocol, for instance, *MQTT device* or *CoAP device*.

We can also create devices which don't directly correspond to their physical counterparts, for instance, we can create one with a higher level of temperature information (*is becoming hotter* or *is becoming colder*) whose values are inferred from temperature sensors of other devices. This kind of device is called *virtual device*.

5.3.2 Qual é a relação entre este *dispositivo* e um dispositivo real?

It is as simple as it seems: the *regular device* for dojot is a mirror (digital twin) of your actual device. You can choose which attributes are available for applications and other components by adding each one of them at the device creation interface.

5.3.3 O que são *dispositivos virtuais*? Como se diferenciam dos demais?

Regular devices are created to serve as a mirror (digital twin) for the actual devices and sensors. A *virtual device* is an abstraction that models things that are not feasible in the real world. For instance, let's say that a user has few smoke detectors in a laboratory, each one with different attributes.

Wouldn't it be nice if we had one device called *Laboratory* that has one attribute *isOnFire*? Therefore, the applications could rely only on this attribute to take an action.

Another difference is how virtual devices are populated. Regular ones will be filled with information sent by devices or gateways to the platform and virtual ones will be filled by flows or by applications.

5.3.4 And what are *templates*?

Templates, simply put, are "blueprints for devices" which serve as basis to create a new device. A single device is built using a set of templates - its attributes will be inherited from each template (their names must not be exactly the same, though). If one template is changed, then all associated devices will also be changed.

5.3.5 Como posso enviar dados via MQTT para a dojot de forma que apareçam no *dashboard*?

Primeiramente, crie uma representação digital para o seu dispositivo real. Depois, configure o seu dispositivo real para enviar dados para a dojot de maneira que os dados possam ser associados ao seu representante digital.

Let's take as example a weather station which measures temperature and humidity, and publishes them periodically through MQTT. First, you create a device of type MQTT with two attributes (temperature and humidity). Then you set your actual device to push the data to dojot.

In order to send data to dojot via MQTT (using iotagent-mosca), there are some things to keep in mind:

- If you don't define any topic in device template, it will assume the pattern `/<service-id>/<device-id>/attrs` (for instance: `/admin/efac/attrs`). This should be the topic to which the device will publish its information to.
- If you do define a topic in device template, then your device should publish its data to it and set the client-id parameter. It should follow the following pattern: `<service>:<deviceid>`, such as `admin:efac`.
- O *payload* MQTT precisa ser um JSON com as chaves correspondendo aos atributos definidos para o dispositivo na dojot, como:

```
{ "temperature" : 10.5,"pressure" : 770 }
```

5.3.6 No *dashboard* alguns atributos são exibidos como tabelas e outros como gráficos. Como são escolhidos/configurados?

O tipo do atributo determina o modo de exibição dos dados no *dashboard* como segue:

- `Geo`: mapa georreferenciado.
- `Boolean` e `Text`: tabela
- `Integer` e `Float`: gráfico de linha.

5.3.7 Estou interessado em integrar à dojot o meu dispositivo que é super legal. Como eu faço isso?

Se o seu dispositivo envia mensagens via MQTT (com *payload* do tipo JSON), CoAP ou HTTP, existe uma grande chance de ser possível intergrá-lo com mínima ou nenhuma modificação. Os requisitos para tal integração são descritos na questão *Como posso enviar dados via MQTT para a dojot de forma que apareçam no dashboard*?.

5.3.8 Existem restrições para as mensagens enviadas pelo meu dispositivo para a dojot? Formato, tamanho, frequência?

Nenhuma com exceção do formato, que é descrito na questão *How can I send MQTT data to dojot so that it appears on the dashboard?*.

5.3.9 Como posso enviar comandos para o meu dispositivo através da dojot?

For now, you can send HTTP requests to dojot containing a few instructions about which device should be configured and the actuation payload itself. More details on that can be found in Device-Manager how-to - sending actuation messages.

5.3.10 Não encontrei o protocolo suportado pelo meu dispositivo na lista de tipos, existe algo que eu possa fazer?

Existem algumas possibilidades. A primeira é desenvolver um *proxy* para traduzir o seu protocolo para um dos suportados pela dojot. A segunda é desenvolver um conector similar as existentes para MQTT, CoAP e HTTP.

5.3.11 Eu salvei um atributo, mas o mesmo sumiu do dispositivo. É um defeito?

Provavelmente você salvou o atributo, mas não o dispositivo. Se você não clicar no botão para salvar o dispositivo, o atributo adicionado será descartado. Estamos melhorando as mensagens da plataforma para avisar e lembrar os usuários de salvarem as suas configurações.

5.3.12 How can I retrieve historical data for a particular device?

You can do this by sending a request to /history endpoint, such as:

```
curl -X GET \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsIn...' \
  "http://localhost:8000/history/device/3bb9/history?lastN=3&attr=temperature"
```

which will retrieve the last 3 entries of *temperature* attribute from the device *3bb9*:

```
[
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:47:07.050000Z",
    "value": 29.76,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:42.455000Z",
    "value": 23.76,
    "attr": "temperature"
  },
  {
    "device_id": "3bb9",
    "ts": "2018-03-22T13:46:21.535000Z",
    "value": 25.76,
    "attr": "temperature"
```

```
  }
]
```

There are more operators that could be used to filter entries. Check history-ws API documentation to check out all possible operators.

5.4 Fluxos de Dados

5.4.1 O que é um fluxo?

It's a sequence of functional blocks to process incoming device messages. With a flow you can dynamically analyze each new message in order to apply validations, infer information and trigger actions or notifications.

5.4.2 A interface dos fluxos... ela se parece com o node-RED. Eles tem alguma relação?

It's based on the Node-RED frontend, but uses its own engine to process the messages. If you're familiar with Node-Red, it won't be difficult to use it.

5.4.3 Por que eu deveria usar um fluxo?

It allows one of the coolest things of IoT in an easy and intuitive way, which is to analyze data for extracting information and then take actions.

5.4.4 O que ele pode fazer, exatamente?

Você pode fazer coisas como:

- Create views from a particular device, by renaming, aggregating and changing values, etc).
- Infer information based on switch, edge-detection and geo-fence rules.
- Enviar notificações via email.
- Enviar notificações via HTTP.

O componente responsável pelo fluxo de dados está em desenvolvimento constante e novas funcionalidades são adicionadas a cada versão.

There are mechanisms to add new processing blocks to new flows. Check the *How can I add a new node type to its menu?* question for more information on that.

5.4.5 Pois bem, como eu posso usá-lo?

Ele segue o modo de uso do node-RED. Você pode ler a documentação para mais detalhes.

5.4.6 Posso aplicar o mesmo fluxo para múltiplos dispositivos?

You can use a template as input to indicate that the flow should be applied to all devices associated to that template. It's worth to point out that the flow is processed individually for each new input message, i.e. for each input device.

5.4.7 Posso correlacionar dados de diferentes dispositivos no mexmo fluxo?

Uma vez que os fluxos são aplicados individualmente para cada mensagem, você deve criar um dispositivo virtual para agregar todos os atributos e então usar este dispositivo como entrada de um novo fluxo.

5.4.8 Eu quero enviar uma notificação por e-mail, como devo fazer?

Basicamente, você deve adicionar um nó 'e-mail' e configurá-lo. Este nó tem como servidor pré-definido o gmail-smtp-in-l.google.com, mas você pode alterá-lo livremente. Para escrever o corpo do email, você deve usar um nó 'template' e associar a variável criada nestenó com o nó de e-mail através do campo 'source' deste último.

É importante notar que a dojot não contém um servidor de e-mail. A plataforma gera os comandos SMTP e os envia ao servidor especificado.

5.4.9 E se eu quiser enviar um HTTP POST?

É quase a mesma coisa de enviar um e-mail.

Um aviso importante: assegure-se de que a dojot consegue acessar seu servidor.

5.4.10 Eu quero renomear os atributos de um dispositivo. O que eu devo fazer?

First of all, you need to create a virtual device with the new attributes, then you build a data flow to rename them. This can be done connecting a 'change' node after the input device to map the input attributes to the corresponding ones into an output, and finally connecting the 'change' to the virtual device and assigning to it the output.

5.4.11 Quero agregar os atributos de múltiplos dispositivos. O que eu devo fazer?

Inicialmente, você deve criar um dispositivo virtual para agregar todos os atributos. Com este dispositivo criado, você deve criar fluxos para mapeamento dos atributos de cada dispositivo real de entrada neste dispositivo virtual. Isto pode ser feito em nós 'change' conectados a cada um dos dispositivos de entrada a fim de criar uma variável contendo todos os atributos de saída. Todos os nós change devem ser, por fim, conectados ao nó de saída representando o dispositivo virtual.

5.4.12 How can I add a new node type to its menu?

It's pretty easy, actually, although it needs a few commands in bash. To add a new node, you should send the following request:

```
curl -H "Authorization: Bearer ${JWT}" http://localhost:8000/flows/v1/node
-H "content-type: application/json" -d '{"image": "mmagr/kelvin:latest",
"id":"kelvin"}'
```

This will add a new node called 'kelvin' which is implemented by a docker image located at "mmagr/kelvin". There's only one caveat: you should pull this image in your target system (where dojot is installed) before adding it to the flow menu.

If you don't want this node anymore, you could delete it:

```
curl -X DELETE -H "Authorization: Bearer ${JWT}"
"http://localhost:8000/flows/v1/node/kelvin"
```

And that's it! In the flowbroker repository, there is an example of how to build a Docker image that could be added to flow node menu.

5.5 Aplicações

5.5.1 Quais APIs estão disponíveis para aplicações?

You can check all available APIs in the API Listing page

5.5.2 Como posso usá-los?

There is a very quick and useful tutorial in the User Guide.

5.5.3 I'm interested in integrating my application with dojot. How can I do it?

Isto deve ser bastante direto. Há duas formas de integrar sua aplicação à dojot:

- **Obtenção de dados históricos**: você pode querer ler todos os dados históricos relacionados a um dispositivo de forma periódica. Isto pode se feito usando esta API (um lembrete apenas: todos os serviços descritos neste apiary deve ser precedido de `/history/`).
- **Subscrição de eventos relacionados a dispositivos**: se sua aplicação é capaz de esperar por eventos, você poderá achar mais interessante usar subscrições, as quais podem ser criadas usando esta API (todos os serviços deste apiary devem ser precedidos por `/metrics/`).
- **Usar os fluxos de dados para pré-processar dados**: se for necessário realizar algum processamento extra, você pode usar os fluxos. Eles auxiliam no processamento e na transformação de dados para envio para sua aplicação via requisições HTTP ou e-mail. Uma outra forma é armazenar os dados em dispositivos virtuais e criar subscrições para enviar notificações toda vez que acontecer uma alteração em seus atributos.

Todas as requisições devem carregar um token de acesso, o qual pode ser obtido como descrito na pergunta *Como posso usá-los?*.