dojot Documentation

Release 0.2.0

Matheus Magalhaes

ago 02, 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 *dojot*, detalhando os componentes que compõem a solução, assim como as suas funcionalidades e como cada um deles contribui para a plataforma como um todo.

Aqui é feita uma breve explicação dos componentes, sendo esta descrição em alto nível e sem o objetivo de explicar os detalhes de implemenação de cada um deles. Para isso, procure a documentação própria do componente.

Table of Contents

1.1 Componentes

A *dojot* foi projetada para tornar possível uma prototipagem rápida, fornecendo uma plataforma fácil de usar, escalável e robusta. Sua arquitetura interna faz uso de muitos componentes conhecidos de código aberto e outros projetados e implementados pela equipe dojot. Essa arquitetura está descrita na figura abaixo.

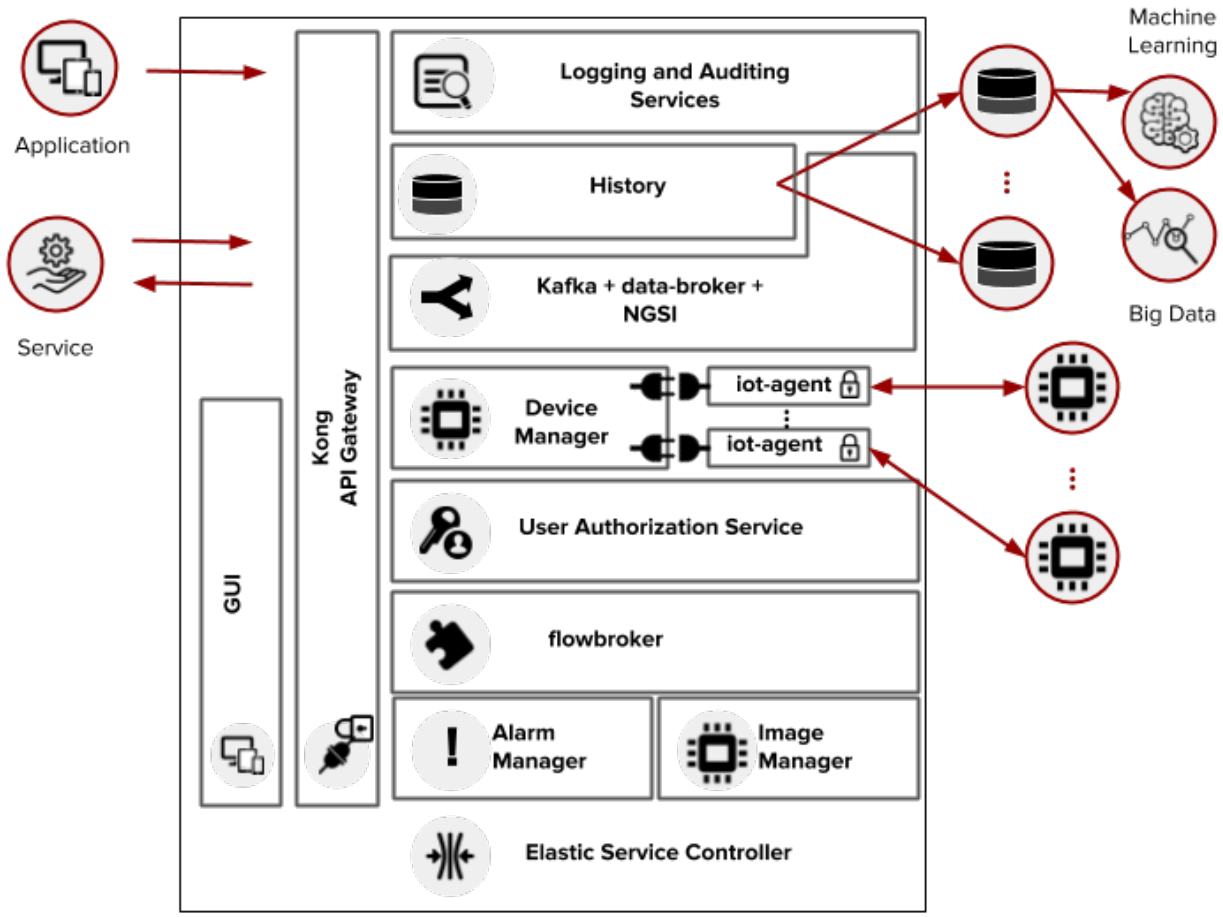

Fig. 1.1: Arquitetura Atual

O uso da *dojot* pode ser resumido assim: um usuário configura dispositivos IoT por meio da Interface Gráfica de Usuário (GUI) ou diretamente pela API REST fornecida pelo Kong API Gateway. Fluxos de processamento de dados também podem ser configurados dessas duas maneiras e suportam a execução de várias ações, como notificações geradas quando um determinado atributo de um dispositivo ultrapassa um certo limite, ou como a persistência de todos os dados gerados por um dispositivo em um banco de dados externo. Assim que os dispositivos começam a enviar seus dados para a *dojot*, o usuário pode ter interesse em receber esses dados via notificações (subscrição de notificações), em consolidar todos os dados em dispositivos virtuais, em reunir todos os dados do banco de dados histórico e assim por diante. Tais recursos podem ser utilizados por meio de APIs REST. Esses são os blocos básicos de construção que qualquer aplicação em cima da dojot poderia se valer. A Interface Gráfica de Usuário da dojot fornece uma maneira fácil de executar operações de gerenciamento para todas as entidades relacionadas à plataforma (usuários, dispositivos, modelos e fluxos) e pode também ser utilizada para verificar se tudo está funcionando bem.

Os contextos de usuários são isolados e não há compartilhamento de dados. As credenciais de acesso são validadas pelo serviço de autorização para cada operação (solicitação de API). Depois que os dispositivos são configurados, o agente IoT é capaz de traduzir os dados recebidos dos dispositivos, encapsulados no MQTT ou outro protocolo, e

enviá-los ao intermediador de contexto para distribuição interna, atingindo, por exemplo, o serviço de histórico, a fim de que ele possa persistir as informações no banco de dados. Se determinadas condições forem satisfeitas quando regras estão sendo processadas, um novo evento é gerado e enviado para o serviço de distribuição que repassa aos serviços interessados.

Para maiores informações sobre o que acontece na dojot, você pode conferir os *repositórios GitHub do projeto <https://github.com/dojot>*. Lá você encontrará todos os componentes utilizados pela plataforma.

Cada um dos componentes que compõem a arquitetura é brevemente descrito 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-inquilino (multi-tenant).

O serviço de gerenciamento de subscrições faz uso de um banco de dados em memória para ser eficiente. Ele agrega contextos ao Apache Kafka, permitindo que serviços internos ou até mesmo externos, se subscrevam ou consultem dados baseados em contexto. O gerenciador de subscrição também é um componente distribuído para não ser um gargalo ou ainda um ponto único de falha na arquitetura.

A fim de se manter um certo nível de compatibilidade com componentes do tipo NGSI (Next Generation Service Interfaces), é possível construir um elemento que ofereça uma interface NGSI para tais componentes.

1.1.2 Gerenciador de Dispositivos

O Gerenciador de Dispositivos é uma entidade central responsável por manter as estruturas de dados de dispositivos e modelos (templates). Também é responsável por publicar quaisquer atualizações para todos os componentes interessados (agentes IoT, histórico e gerenciador de subscrição) através do 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 Agente IoT

Um agente IoT é um serviço de adaptação entre dispositivos físicos e componentes principais da *dojot*. Pode ser entendido como um *driver de dispositivo* para um conjunto de dispositivos. A plataforma *dojot* pode ter vários agentes IoT, cada um deles especializado em um protocolo específico, como, por exemplo, MQTT / JSON, CoAP / LWM2M e HTTP / JSON.

O agente IoT também é responsável por garantir que a sua comunicação com dispositivos seja feita por meio de canais seguros.

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 Orchestrador

Esse serviço provê mecanismos para construir fluxos de processamento de dados para execução de um conjunto de ações. Os fluxos podem ser estendidos usando um bloco de processamento externo (que pode ser incluído utilizando APIs REST).

1.1.6 Histórico

O componente histórico funciona como um condutor de dados e eventos que devemser persistidos em um banco de dados. Os dados são convertidos em uma estrutura de armazenamento que é enviada para o banco de dados correspondente.

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 à 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

Todos os serviços que fazem parte da plataforma dojot podem gerar métricas de uso de seus recursos. Tais métricas podem ser utilizadas por serviços de Registro e Auditoria, que processam esses dados sumarizando-os por usuários e aplicativos.

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

Observação: Componentes atualmente em desenvolvimento.

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. Isso 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 Interface Gráfica de Usuário

A Interface Gráfica de Usuário na *dojot* é uma aplicação WEB que provê interfaces responsivas para gerenciamento da plataforma, incluindo funcionalidades como:

- **Gerenciamento de perfil de usuários**: permite definir perfis e quais APIs podem ou não ser acessadas pelo respectivo perfil.
- **Gerenciamento de usuários**: permite operações de criação, visualização, edição e remoção.
- **Applications Management**: Creation, Visualization, Edition and Deletion Operations
- **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 fluxos de processamento**: permite 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.

Este componente está programado para entrar em desenvolvimento.

1.1.11 Gerenciamento de Alarmes

Este componente é responsável por tratar alarmes gerados pelos componentes internos da dojot, tais como os oriundos de Agentes IoT, Gerenciador de Dispositivos e outros.

1.1.12 Image manager

Este componente é responsável pelo armazenamento e recuperação de imagens de firmware de dispositivos.

1.2 Infraestrutura

Alguns outros componentes são utilizados na dojot e não estão representados em Fig. 1.1. São eles:

- postgres: esse banco de dados é utilizado para persistir informações de vários componenentes, como do gerenciador de dispositivos.
- redis: é um banco de dados em memória usado como cache em vários componentes, como o serviço de osquestração, gerenciador de subscrição, agentes IoT e outros. É bem leve e fácil de usar.
- rabbitMQ: intermediador de mensagens utilizado no serviço de orquestração para implementar fluxos de ações relacionados que podem ser aplicados a mensagens recebidas dos componentes.
- Banco de dados mongo: solução de banco de dados amplamente utilizada que é fácil de usar e não adiciona esforço de acesso considerável (nos locais onde foi empregado na dojot).
- zookeeper: mantém sob controle serviços replicados em cluster.

1.3 Comunicação

Todos os componentes se comunicam de duas maneiras:

- Por meio de requisições HTTP: se um componente necessita recuperar dados de outro, como um agente IoT que precisa a lista de dispositivos configurados do gerenciador de dispositivos, ele pode enviar uma requisição HTTP para o componente apropriado.
- Por meio de mensagens Kafka: se um componente precisa enviar novas informações sobre um recurso controlado por ele (como novos dispositivos criados no gerenciador de dispositivos), o componente pode publicar esses dados através do Kafka. Utilizando esse mecanismo, qualquer outro componente que esteja interessado

em tal informação precisa apenas ouvir um tópico específico para recebê-la. Note que este mecanismo não faz quaisquer associações difíceis entre componentes. Por exemplo, o gerenciador de dispositivos não sabe quais componentes precisam de suas informações e um agente IoT não necessita saber qual componente está enviando dados através de um tópico específico.

CAPÍTULO 2

Concepts

This document provides information about dojot's concepts and abstractions.

Table of Contents

Nota:

- **Audience**
  - Users that want to take a look at how dojot works;
  - Application developers.
- Level: basic

2.1 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. There a GUI oriented tutorial in tutorials/using-web-interface tutorial.

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

2.1.1 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.

2.1.2 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.

2.1.3 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.

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 Hardware requirements

In order to properly run dojot, the minimum hardware requirements are:

- 4GB of RAM
- 10GB of free disk space
- Network access

- **The following ports should be opened:**
  - TCP (incoming connections): 1883 (MQTT), 8883 (Secure MQTT if used), 8000 (web interface access)
  - TCP (outgoing connections): 25 (if send e-mail node is used in a flow)

4.2 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.

The following sections describe all Docker compose dependencies.

4.2.1 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/

4.2.2 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/

4.2.3 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 -b branch_name
```

For instance:

```
git checkout 0.2.0 -b 0.2.0
```

Or if you're brave enough:

```
git checkout master
```

After the repository is cloned, and a release (or branch) has been selected, there are still a few external modules that must be gathered before using the platform. These modules can be retrieved by executing the following command:

```
git submodule update --init --recursive
```

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/

4.2.4 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 *Components and APIs*.

Read the tutorials/using-api-interface and tutorials/using-web-interface for more information about how to interact with the platform.

4.3 Kubernetes

This section provides instructions on how to create a simple dojot deployment environment on a multi-node environment, using Kubernetes as the orchestration platform.

This deployment option as presented in this document is best suited for testing and platform assessment. With appropriate changes, this option can be also be used in production environments.

This guide has been checked on a Kubernetes cluster with Ceph as the underlying storage infrastructure and it has also been tested on a Kubernetes cluster over the Google Cloud Platform

The following sections describe all Kubernetes dependencies.

4.3.1 Kubernetes Cluster

For this guide it is advised that you already have a working cluster.

If you desire to prepare a Kubernetes cluster from scratch, up to date information and installation procedures can be found at Kubernetes setup documentation.

4.3.2 Persistent Storage

To make sure that all the data from the containers running databases is persisted when containers fail or are moved to different nodes of the Kubernetes environment it is necessary to attach persistent storage to the database pods.

Kubernetes requires that an infrastructure for persistent storage already exists on the cluster. As an example for how to configure your persistent storage we provide files for two different kind of deployments, the first is for a local deployment where a Ceph Cluster is used as storage backend, more information on Ceph may be found at: http://ceph.com/. The second example is based on a Google Cloud deployment and use the existing persistent storage services that are provided by Google Cloud. If you're deploying dojot using Kubernetes to a different cloud provider, some adjustments to fit the different deployments might be necessary.

Information about the currently supported persistent storage for Kubernetes can be found at persistent-volumes page.

4.3.3 Kubernetes Client

To install the Kubernetes client on your machine before proceeding with this guide, follow the proper instructions as presented on the Kubernetes documentation.

Also, verify that your client is capable of connecting to the cluster.

For providing access for a local cluster, follow the documentation below:

https://kubernetes.io/docs/tasks/access-application-cluster/access-cluster/

If the Kubernetes cluster is running on a specific cloud platform like Google Cloud, follow the steps as presented by your cloud provider.

4.3.4 Deployment

To deploy dojot to a Kubernetes environment, we provide a script for clusters with Ceph as storage solution.

To download the required files using git, run the following command:

```
git clone https://github.com/dojot/kubernetes.git
```

or, to download a compressed zip file containing the data, use the following link: https://github.com/dojot/kubernetes/archive/master.zip

This repository contains all the scripts and deployment files necessary to properly setup dojot's containers. There is one file that must be changed: `config.yaml`, which contains all the parameters used by these scripts. An example of such file is this:

```
1  ---
2  version: 0.2.0-nightly20180319
3  namespace: dojot
4  storage:
5    type: ceph
6    cephMonitors:
7    - '10.0.0.1:6789'
8    - '10.0.0.2:6789'
9    - '10.0.0.3:6789'
10   cephAdminId: admin
11   cephAdminKey: AQD85Z5a/wnlJBAARNISUDpC6RHc8g/UkUcDLA==
12   cephUserId: admin
13   cephUserKey: AQD85Z5a/wnlJBAARNISUDpC6RHc8g/UkUcDLA==
14   cephPoolName: kube
15 externalAccess:
16   type: publicIP
17   ips:
18   - '10.0.0.1'
19   - '10.0.0.2'
20   - '10.0.0.3'
21   ports:
22     httpPort: 80
23     httpsPort: 443
24     mqttPort: 1883
25     mqttSecurePort: 8883
26 services:
27   zookeeper:
28     clusterSize: 3
29   postgres:
30     clusterSize: 3
31   mongodb:
32     replicas: 2
33   kafka:
34     clusterSize: 3
35   auth:
36     emailHost: 'smtp.gmail.com'
37     emailUser: 'test@test.com'
38 emailPassword: 'password'
```

From line 5 to 14, we have Ceph configuration parameters. The `cephMonitors` attribute specifies how many monitors are going to be used and by which address they can be accessed. For more information about this element, check ceph monitors documentation. `cephAdminId`, `cephAdminKey`, `cephUserId` and `cephUserKey` attributes refers to user information. These values are set/generated in user creation.

In `externalAccess` section we have what addresses and ports should be exposed for external access. In `services` section, we can configure how many replicas we want to each service and a few other parameters to configure that service (for instance, auth taks an `emailHost` and `emailUser` parameters).

To configure and start the kubernetes cluster, just install all python requirements and start the deploy.py script:

```
pip install -r ./requirements.txt
python ./deploy.py
```

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,
```

(continues on next page)

(continuação da página anterior)

```
    "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 *Utilizando a API da dojot*.

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?*.

CAPÍTULO 6

Usando a interface WEB

Este tutorial descreve as operações básicas na dojot, como criar dispositivos, conferir seus atributos e criar fluxos.

Nota:

- Para quem é esse tutorial: usuários iniciantes
- Nível: básico
- Tempo de leitura: 15 minutos

6.1 Gerenciamento de dispositivo

Esta seção mostra como gerenciar dispositivos. Para tal, serão utilizados dois dispositivos sensores de temperatura e um dispositivo virtual, esse último com a função de observar as temperaturas medidas nos dois primeiros e gerar alarmes em determinadas condições.

Como descrito em *Concepts*, todos os dispositivos são baseados em um ou mais modelos (templates). Para a criação de um modelo, você deve acessar a opção Modelos (Templates) na lateral esquerda da tela e então criar um Novo Modelo (New Template), como mostrado abaixo.

Agora há um modelo do qual dispositivos podem ser "instanciados". Todos dispositivos baseados nesse modelo aceitarão mensagens via protocolo MQTT que serão enviados para o tópico "/devices/thermometers". Para criar novos dispositivos, deve-se voltar para a opção Dispositivos (Devices) e criar um Novo Dispositivo (New Device), selecionando os modelos nos quais o dispositivo será baseado, como mostrado abaixo.

Note que, quando um modelo é selecionado no painel direito da tela de criação de dispositivo, todos os atributos são herdados para aquele dispositivo. É possível adicionar mais de um modelo, tendo em mente que modelos que compõem o dispositivo não podem compartilhar atributos com o mesmo nome.

Atenção: Os dispositivos são fortemente atrelados aos a modelos. Para remover um modelo, deve-se remover primeiro todos os dispositivos a ele associados. Caso contrário, a seguinte mensagem aparecerá:

Atenção: É possível adicionar e remover atributos dos modelos, fazendo com que as alterações sejam imediatamente refletidas nos dispositivos associados. No caso de novos adição, no entanto, deve-se observar que os atributos dos modelos que compõem um determinado dispositivo não podem possuir o mesmo nome. Se isso acontecer, a seguinte mensagem aparecerá:

Essa imagem da tela foi capturada quando um novo modelo foi criado (`ExtraTemplate`) com um atributo chamado `level`. Depois um novo dispositivo baseado em ambos os modelos foi criado e um novo atributo também chamado `level` foi adicionado ao modelo `ThermTemplate`.

Quando isso ocorre, nenhuma modificação é aplicada ao modelo (nenhum atributo com nome "level" relativo ao "ThermTemplate" é criado). Contudo, o atributo é mantido no cartão do modelo para que o usuário perceba o que está acontecendo. Se o usuário atualizar a tela, as informações serão revertidas para o estado que estava antes da modificação.

Agora os dispositivos físicos podem enviar mensagens à plataforma dojot. Existem algumas coisas a serem observadas: como foi definido o tópico MQTT (todos os dispositivos enviarão mensagens para o tópico `/devices/thermometer`), os dispositivos devem se identificar utilizando o parâmetro `client-id` do protocolo MQTT. Outra maneira de se fazer isso é utilizar o esquema de tópico default (que é `/{SERVICE}/{DEVICE_ID}/attrs`).

Por questão de simplicidade, será emulado um dispositivo utilizando-se a ferramenta mosquito_pub. O parâmetro `client-id` será configurado utilizando a opção `-i` do mosquito_pub.

Estando criados os sensores de temperatura, falta agora a criação do dispositivo virtual. Ele será a representação de um alarme de sistema disparado quando algo ruim for detectado pelos sensores. Por exemplo, se os sensores de temperatura estivessem instalados em uma cozinha, a medição de uma temperatura acima de 40ºC poderia indicar que o local estaria em chamas. Essa representação do alarme poderia ter dois atributos: nível de severidade e mensagem textual, para que o usuário pudesse ser informado do acontecimento.

Assim como "dispositivos regulares", dispositivos virtuais também são baseados em modelos. Portanto, um modelo será criado, como mostrado abaixo.

6.2 Configuração de fluxo

Uma vez criado o dispositivo virtual, pode-se adicionar um fluxo para implementar a lógica por detrás da geração de alarmes. A ideia é: se a temperatura medida for menor ou igual a 40ºC, o sistema de alarmes será atualizado com uma notificação de severidade 4 (média) e uma mensagem indicando que a cozinha está OK. Caso a temperatura medida seja maior que os 40ºC, uma notificação de severidade 1 (muito alta) será enviada com a mensagem que a cozinha está em chamas. Isto é feito como mostrado abaixo.

É importante notar que os nós do tipo "change" têm uma referência a uma entidade "output". Isso pode ser visto como uma simples estrutura de dados, onde existem os atributos `message` e `severity` que casam com aqueles do dispositivo virtual. Este "objeto" é referenciado no nó de saída (output) como uma fonte de dados para o dispositivo

que será atualizado (nessa caso, o dispositivo virtual criado). Em outras palavras, pode-se dizer que há uma informação que é transferida dos nós do tipo "change" para o "dispositivo virtual" com os nomes "msg.output.message" e "msg.output.severity", onde "message" e "severity" são atributos do dispositivo virtual.

Vamos, agora, enviar mais algumas mensagens e ver o que acontece para aquele dispositivo virtual.

Se está interessado em como usar os dados gerados por esses dispositivos em sua aplicação, confira o tutorial Building an application.

CAPÍTULO 7

Utilizando a API da dojot

Esta seção descreve o passo a passo completo de como criar, alterar, enviar mensagens e conferir dados históricos relativo a um dispositivo. Este tutorial assume que está sendo utilizada a instalacao docker-compose e que todos os componentes necessários estão sendo executados corretamente na dojot.

Nota:

- Audiência: desenvolvedores
- Nível: básico
- Tempo de leitura: 15 minutos

7.1 Obtendo um token de acesso

Como mencionado em *User authentication*, todas as requisições devem conter um token de acesso que seja válido. É possível gerar um novo token enviando a seguinte requisição:

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

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

Se o intuito for gerar um token para outro usuário, é necessário somente mudar o username e passwd no corpo da requisição. O token ("eyJ0eXAiOiJKV1QiL...") deve ser usado em toda a requisição HTTP enviada para a dojot, colocando-o no cabeçalho da mensagem. A requisição seria algo desse tipo:

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

É importante ressaltar que o token deve estar inteiro no cabeçalho da requisição, não apenas parte dele. No exemplo, somente os primeiros caracteres foram mostrados por questão de simplificação. Todas as demais requisições serão compostas da variável de ambiente chamada `bash ${JWT}` que contém o token obtido da dojot (mais especificamente do componente de autorização da dojot).

7.2 Criação de dispositivo

A fim de configurar um dispositivo físico na dojot, é necessário criar sua representação na plataforma. O exemplo mostrado aqui é apenas uma parte pequena do que é oferecido pelo componente DeviceManager. Para mais informações sobre esse componente, confira o documento DeviceManager how-to.

Primeiramente vamos criar um modelo (template) para o dispositivo, pois todos os dispositivos são baseados em modelos, não esqueça.

```
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"
    }
  ]
}'
```

Esta requisição deve retornar a seguinte mensagem:

```
{
  "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
      }
    ],
```

(continues on next page)

(continuação da página anterior)

```
27      "id": 1
28    }
29  }
```

Note que o ID do modelo é 1 (linha 27)

Para criar um dispositivo baseado nesse modelo (ID 1), envie a seguinte requisição para a 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 }'
```

A lista de IDs de modelos na linha 6 contém um único ID do modelo configurado até o momento. Para conferir os dispositivos configurados, basta enviar uma requisição do tipo GET para /device:

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

Que deve retornar:

```
{
  "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"
    }
  ]
}
```

7.3 Enviando mensagens

Até o momento um token de acesso foi obtido, um modelo e um dispositivo (baseado no modelo) foram criados. Em um sistema real, o dispositivo físico envia mensagens para a dojot com todos os seus atributos contendo valores correntes. Nsse tutorial serão enviadas mensagens MQTT montadas "na mão" para a plataforma, emulando um dispositivo físico. Para tal, será utilizado o mosquito_pub do projeto Mosquitto.

Atenção: Algumas distribuições Linux, o Ubuntu em particular, tem dois pacotes para o mosquitto - um contendo ferramentas para acessá-lo (por exemplo, mosquito_pub e mosquito_sub para publicação de mensagens e subscrição a tópicos) e outro contendo o broker MQTT. Neste tutorial, somente as ferramentas serão utilizadas. Certifique-se que o broker MQTT não está sendo executado antes de iniciar a dojot (para isso, pode-se utilizar o comando `ps aux | grep mosquitto`).

O formato padrão de mensagem usado pela dojot é um simples "chave-valor" JSON (é possível traduzir qualquer formato para esse esquema utilizando fluxos), como abaixo:

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

Vamos enviar essa mensagem para a dojot:

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

Se não houver saída (output), a mensagem é enviada ao broker MQTT.

Como descrito no *Dúvidas Mais Frequentes*, existem algumas considerações a respeito dos tópicos MQTT:

- Pode-se configurar o ID do dispositivo origem da mensagem utilizando o parâmetro MQTT `client-id`. Deve seguir o seguinte padrão: `<service>:<deviceid>`, como em `admin:efac`.
- Se por algum motivo você não pode fazer tal coisa, então o dispositivo deve configurar seu ID no tópico utilizado para publicar as mensagens. O tópico deve assumir o padrão `/<service-id>/<device-id>/attrs` (por exemplo: `/admin/efac/attrs`).
- Se for definido um tópico no modelo de dispositivo, então o dispositivo deve publicar seus dados em tal tópico e configurar o parâmetro client-id.
- Os dados da mensagem MQTT (payload) deve ser um JSON com cada chave sendo um atributo do dispositivo cadastrado na dojot, como:

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

Para mais informações sobre como a dojot trata os dados enviados por dispositivos, veja o tutorial integrating-physical-devices. Lá você poderá verificar como trabalhar com dispositivos que não publicam mensagens neste formato e como traduzi-las.

7.4 Conferindo dados históricos

A fim de se conferir todos os valores que foram enviados pelo dispositivo para um atributo particular, pode-se utilizar as history APIs. Vamos, então, enviar agora alguns outros valores à dojot para que possamos conseguir resultados um pouco mais interessantes:

```
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}'
```

Para recuperar todos os valores enviados do atributo temperature desse dispositivo:

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

O endpoint do histórico é construído por meio desses valores:

- `.../device/3bb9/...`: o ID do dispositivo é `3bb9` - isso é obtido do atributo `id` do próprio dispositivo
- `.../history?lastN=3&attr=temperature`: o atributo requerido é temperature e deve ser recuperado os 3 últimos valores. Mais operadores são descritos em history APIs.

  A requisição deve resultar na seguinte mensagem:


```
[
  {
    "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"
  }
]
```


A mensagem acima contém todos os valores previamente enviados pelo dispositivo.