dojot Documentation

Release 0.3.0

Matheus Magalhaes

set 05, 2019

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.

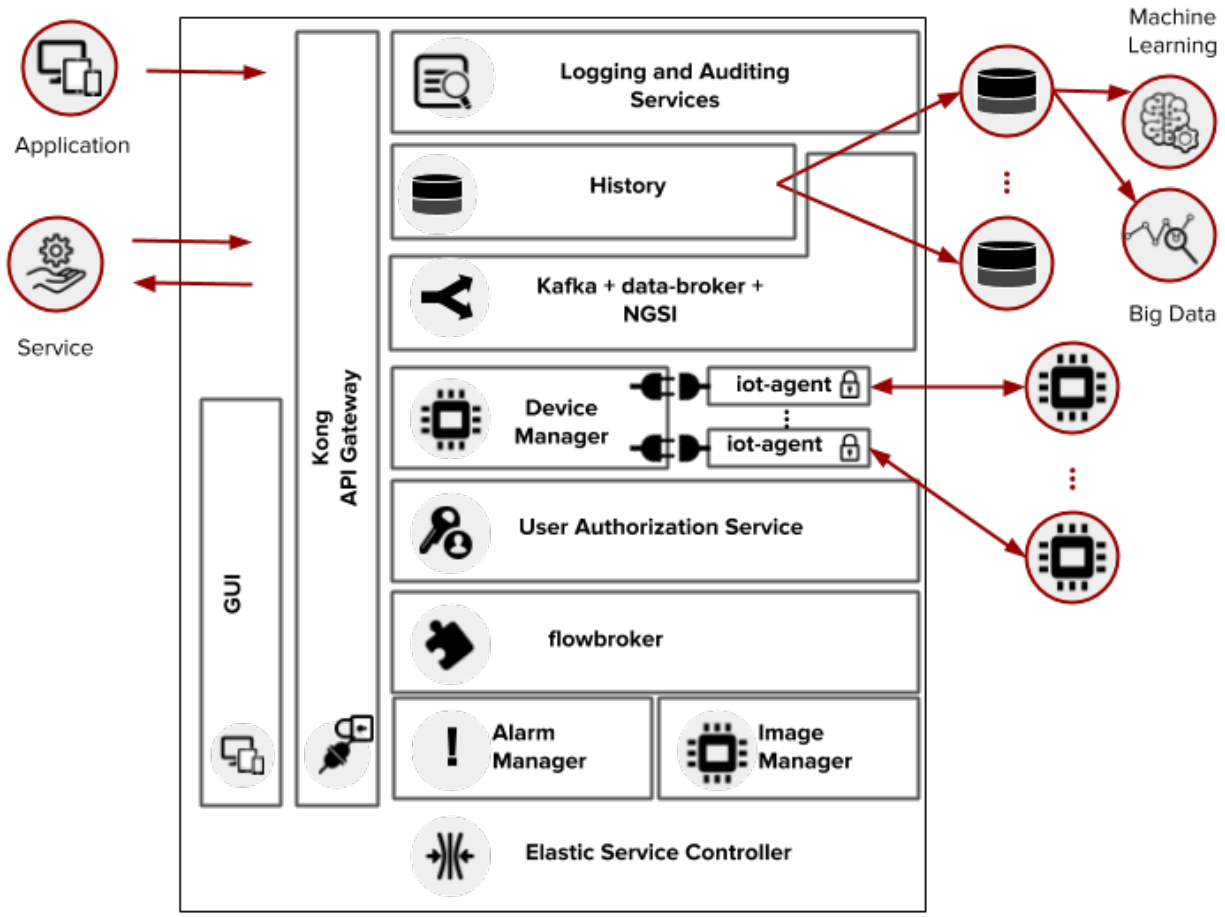


Fig. 1.1: Arquitetura Atual

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, a user can:

- 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 context are isolated and there is no data sharing, the access credentials are validated by the authorization service for each and every operation (API Request). Therefore, a user belonging to a particular context (tenant) cannot

reach any data (including devices, templates, flows or any other data related to these resources) from other ones.

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 message broker for internal distribution. This way, the data reaches the history service, for instance, so it can persist the data on a database.

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 + DataBroker

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).

The DataBroker 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. DataBroker is also a distributed service to avoid it being a single point of failure or even a bottleneck for the architecture.

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.

This component is also scheduled for development.

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 *Usando a interface WEB* 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	Repository / Main site	Documentation
MongoDB	MongoDB official site	MongoDB documentation
postgres	PostgreSQL official site	PostgreSQL documentation
Kong API gateway (Community Edition)	Kong official site	Kong documentation
redis	Redis official site	Redis documentation
zookeeper	Zookeeper official site	Zookeeper documentation
Kafka	Kafka official site	Kafka documentation
auth	GitHub - auth	readthedocs - auth
History	GitHub - history	
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	GitHub - history
/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
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`
auth	Tenants creation/removal (Messages - auth)	`dojot.tenancy`

CAPÍTULO 4

Guia de instalação

Esta página contém informação de como instalar a dojot utilizando o docker-compose. O suporte à instalação em ambientes Kubernetes e Google Cloud estão em andamento no projeto.

Table of Contents

4.1 Requisitos de hardware

Para que a dojot seja executada apropriadamente, os requisitos mínimos de hardware são:

- 4GB de RAM
- 10GB de espaço livre em disco
- Acesso à rede
- **As seguintes portas devem estar abertas:**
  - TCP (conexões de entrada): 1883 (MQTT), 8883 (MQTT seguro, se utilizado), 8000 (acesso à interface web)
  - TCP (conexões de saída): 25 (se envio de email nos fluxos for utilizado)

4.2 Docker-compose

Este documento contém instruções de como criar um ambiente para instalação trivial da dojot em um único host utilizando o docker-compose como o processo de orquestração da plataforma.

Muito simples, esta opção de instalação é a que melhor se adapta para desenvolvimento e verificação da plataforma dojot, mas não é aconselhável para ambientes de produção.

Este guia foi verificado utilizando-se o sistema operacional Ubuntu 16.04 LTS.

As seções seguintes descrevem todas as dependências do docker-compose.

4.2.1 Docker engine (motor do docker)

Informações atualizadas e procedimentos de instalação para o docker engine podem ser encontrados na documentação do projeto:

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

Nota: Um passo adicional no processo de instalação e configuração do docker em um determinado equipamento é definir quem será elegível para criar/iniciar instâncias do docker.

Caso os passos pós-instalação não tiverem sido executados (mais especificamente o "Manage docker como usuário não-root"), todos os comandos do docker e do docker-compose devem ser executados pelo super usuário (root), ou invocando o sudo.

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

4.2.2 Docker-compose

Informações atualizadas sobre procedimentos de instalação para o docker-compose podem ser encontradas na documentação do projeto:

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

4.2.3 Instalação

Para construir o ambiente, simplesmente clone o repositório e execute os comandos abaixo.

O repositório com os scripts de instalação e configuração do docker-compose podem ser encontrados em:

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

ou com o comando git clone::

```
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
```

Uma vez que o repositório esteja propriamente clonado, selecione a versão a ser utilizada por meio da tag apropriada (note que o tagname deve ser substituído):

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

git checkout tag_name -b branch_name
```

Por exemplo:

```
git checkout v0.3.1 -b v0.3.1
```

Ou se você for muito ousado:

```
git checkout master
```

Depois que o repositório estiver clonado e uma versão (tag) ou branch tiver sido selecionado, haverá ainda alguns módulos que devem ser instalados antes de utilizar a plataforma. Esses módulos podem ser obtidos executando o seguinte comando:

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

Feito isso, o ambiente pode ser iniciado assim:

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

Para verfificar o estado de um container individual, comandos do docker podem ser utilizados, como por exemplo:

```
# 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: Todos os comandos para docker e docker-compose podem requerer credenciais de super usuário (root) ou sudo.

Para permitir usuários "não-root" gerenciar o docker, confira a documentação do docker:

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

4.2.4 Utilização

A interface web está disponível em `http://localhost:8000`. O usuário é `admin` e a senha é `admin`. Você também pode interagir com a plataforma utilizando o *Components and APIs*.

Read the *Utilizando a API da dojot* and *Usando a interface WEB* for more information about how to interact with the platform.

4.3 Kubernetes

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

This deployment option when properly configured can be used for creating production environments.

The following sections describe all dependencies and steps required for this deployment.

4.3.1 Kubernetes Cluster

For this guide it is advised that you already have a working K8s cluster.

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

4.3.2 Kubernetes Requirements

- The minimum Kubernetes supported version is **v1.11**.
- Access to Docker Hub repositories
- (optional) a storage class that will be used for persistent storage

4.3.3 dojot Deployment

To deploy dojot to Kubernetes it is advised the use of ansible playbooks developed for dojot. The playbooks and all the related code can be found on the repository Ansible dojot.

The following steps will describe how to use this repository and its playbooks.

1. Cloning the repository

The first deployment step is cloning the repository. To do so, execute the command:

```
git clone https://github.com/dojot/ansible-dojot
```

2. Installing dependencies

The next step is installing the dependencies for running the ansible playbook, this dependencies include ansible itself with other modules that will be used to parse templates and communicate with kubernetes.

Enter the folder where the repository was downloaded and install the pip packages with the following commands:

```
cd ansible-dojot
pip install -r requirements.txt
```

3. Configuring the inventory

For deploying kubernetes with ansible, it is necessary to model your desired environment on an ansible inventory.

In the repository there is an '*inventory*' folder containing an example inventory called '*example_local*' that can be used as the starting point to creating the real environment inventory.

The first file that requires changes is the hosts.yaml. This file describes the nodes that will be accessed by ansible to perform the deployment. As the dojot deployment is done directly to K8s, only a node with access to the kubernetes cluster is actually required.

The node that will access the cluster might be a kubernetes cluster node that is accessible via SSH or event your local machine if it can reach the kubernetes cluster with a configuration file.

On the example file, the access is done via a local node, where the ansible script is executed. This node is described as localhost in the hosts item of the group **all**.

These same nodes must be added as children of the group dojot-k8s.

To configure a local access on the hosts file, follow the example below:

```
---
all:
  hosts:
    localhost:
      ansible_connection: local
      ansible_python.version.major: 3
  children:
    dojot-k8s:
      hosts:
        localhost:
```

To configure remote access via ssh to a node of the cluster, follow this other example:

```
---
all:
  hosts:
    NODE_NAME:
      ansible_host: NODE_IP
  children:
    dojot-k8s:
      hosts:
        NODE_NAME:
```

The next step is configuring the mandatory and optional variables required for deploying dojot.

There is a document describing each of the variables that can be configured at Ansible dojot variables.

This variables must be set for the group '*dojot-k8s*', to do so set their values on the file dojot.yaml on the folder '**group_vars/dojot-k8s/**'

4. Executing the deployment playbook

Now that the inventory is set, the next step is executing the deployment playbook.

To do so, run the following command:

```
ansible-playbook -K -k -i inventories/YOUR_INVENTORY deploy.yaml
```

Wait for the playbook execution to finish without errors.

5. Accessing the deployed dojot environment

Dojot access will be set using NodePorts, to view the proper ports to access the environment it is necessary to check service configuration.

```
kubectl get service -n dojot kong iotagent-mosca
```

This command will return the port used for external access to both the REST API and GUI via kong and the MQTT port via iotagent-mosca.

CAPÍTULO 5

Dúvidas Mais Frequentes

Aqui estão algumas respostas às dúvidas mais frequentes sobre a plataforma dojot.

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?

É uma plataforma brasileira para IoT que surgiu com uma proposta de código aberto, para facilitar o desenvolvimento de soluções e o ecossistema IoT com conteúdo local voltado às necessidades brasileiras.

Ela oferece

- 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 ?

A dojot pode ser acessada via interface Web ou APIs REST. Considerando que você já tenha instalado o `docker` e o `docker-compose` e tenha clonado o repositório dojot para o `docker-compose`, para iniciar todos os serviços, basta executar o comando abaixo:

```
$ 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 templates e 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 Atualize o repositório do docker compose com a última versão. (cuidado: bug bravo)

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

Se você precisar de uma versão mais estável, você pode fazer checkoutde uma tag:

```
$ 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
```

De tempos em tempos nós lançaremos novas versões para componentes dadojot. Eles podem ser lançados individualmente (no entanto, tentaremossincronizar todos os lançamentos). Uma vez que nós tenhamos um conjunto decomponentes que opere de forma estável, atualizaremos o repositório do docker-compose

Atualize o ambiente com as imagens dockers mais recentes.

```
$ 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.

Considere, por exemplo, um dispositivo real com um termômetro e umhigrômetro; ele pode ser representado na dojot como um dispositivo com doisatributos (um para cada sensor). Chamamos este tipo de dispositivo de*dispositivo normal* ou usando o seu protocolo de comunicação, como*dispositivo MQTT* ou *dispositivo CoAP*.

Nós também podemos criar dispositivos que não correspondem diretamente a dispositivos reais, por exemplo, podemos criar um dispositivo com informação em alto nível de temperatura (*está ficando mais quente* ou *está ficando mais frio*) cujos valores são inferidos a partir de sensores de temperatura de outros dispositivos. Este tipo de dispositivo é denominado de *dispositivo virtual*.

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

É simples como parece: o *dispositivo* para a dojot é um espelho (gêmeo/cópia digital) do dispositivo real. Você pode escolher quais atributos são disponibilizados para as aplicações e outros componentes, adicionando cada um deles através da interface de criação de dispositivo.

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

Dispositivos são criados para serem como espelhos (gêmeo/cópia digital) dos dispositivos e sensores reais. Um *dispositivo virtual* é uma abstração que modela coisas que não são factíveis no mundo real. Por exemplo, digamos que um usuário tenha alguns sensores para detectar fumaça em um laboratório, sendo que cada um tem diferentes atributos.

Não seria bom se exisitisse um dispositivo chamado *Laboratório* quepossui um atributo *emChamas*? Assim a aplicação dependeria apenas desteatributo para tomar alguma ação.

Uma outra diferença é a maneira como os dados dos dispositivos virtuais são populados. Os dispositivos são preenchidos com informações enviadas aplataforma dojot por dispositivos ou gateways e os virtuais são preenchidos por fluxos ou por aplicações.

5.3.4 E o que são *templates*?

Templates são "modelos para dispositivos" que servem como base para acriação de um novo dispositivo. Um dispositivo é construído usando um conjuntode templates - seus atributos serão herdados de cada template (não deve havernenhum atributo com mesmo nome, no entanto). Se um template é alterado, todosos dispositivos associados àquele template serão também alteradosautomaticamente.

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.

Tomemos como exemplo uma estação meteorólogica que monitora temperatura e umidade e publica essas medidas via MQTT periodicamente. Inicialmente, cria-se um dispositivo do tipo MQTT com dois atributos (temperatura e umidade) e, em seguida, configura-se a estação meteorológica para publicar os dados para a dojot.

Para enviar dados para a dojot via MQTT (usando o iotagent-mosca),existem algumas coisas para se ter em mente:

- `/admin/efac/attrs`). Dependendo de como o IoT agent foi inicializado (mais restritivo), o `client ID` deve ser também configurado para''<tenant>:<deviceid>'', como `"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?

Por enquanto, você pode enviar requisições HTTP para a dojot contendoalgumas instruções sobre qual dispositivo deve ser configurado e os dados paraatuação. Mais detalhes podem ser encontrados na seção 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 Compo eu posso obter dados históricos para um dispositivo em particular?

Você pode fazer isto enviando uma requisição para */history/*, como:

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

o qual retornará as últimas 3 entradas do atributo *temperature* dodispositivo *3bb9*

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

(continues on next page)

(continuação da página anterior)

```
    "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"
  }
]
```

Há mais operadores que podem ser usados para filtrar entradas. Check History API para ver todas os possíveis operadores e outros filtros.

5.4 Fluxos de Dados

5.4.1 O que é um fluxo?

É um processamento de mensagens de um dispositivo. Com um fluxo, você pode analisar dinamicamente cada nova mensagem para fazer validações, inferirinformações e tomar ações ou gerar notificações.

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

A interface dos fluxos é baseada no node-RED, mas a aplicação das regras e execução das ações é feita por um mecanismo próprio da dojot. Se o node-REDfor familiar para você, não será difícil usar o flowbroker.

5.4.3 Por que eu deveria usar um fluxo?

Ele permite uma das coisas mais interessantes do IoT de uma forma simples e intuitiva que é analisar dados para extração de informações e execução de ações.

5.4.4 O que ele pode fazer, exatamente?

Você pode fazer coisas como:

- Criar visões para um dispositivo (renomear atributos, agregá-los, alterá-los, etc.)
- Inferir informações baseadas em regras de detecção de borda e georreferenciamento.
- 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.

Há mecanismos para adicionar novos blocos de processamento a fluxos. Veja *How can I add a new node type to its menu?* para mais informações sobreisto.

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?

Você pode usar um template como entrada para indicar que ele deve serexecutado para todos os dispositivos associados àquele template. É válidoindicar que o fluxo é sempre executado para cada mensagem.

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.

Outra coisa que pode ser feita é construir um nó do flowbroker paralidar com contextos, os quais podem ser usados para armazenar e obter dadosrelacionados a um fluxo ou nó.

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?

Primeiramente, você deve criar um dispositivo virtual com os novos atributos, para então construir um fluxo de dados para renomeá-los. Isto pode ser feito conectando um nó 'change' após um dispositivo de entrada para mapear os atributos de entada a seus correspondentes na saída.

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 Como eu posso adicionar um novo tipo de nó no menu?

É simples, embora necessite de alguns comandos no terminal. Paraadicionar um nó novo, você deve executar o seguinte comando.

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

Este comando aidciona um novo nó chamado 'kelvin' o qual é implementadoem uma imagem do docker localizada no repositório "mmagr/kelvin".

Se você não quiser mais este nó, é possível removê-lo:

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

E é isso! No repositório do flowbroker, há um exemplo de como construiruma imagem do Docker que pode ser adicionada ao menu de nós do flowbroker.

5.5 Aplicações

5.5.1 Quais APIs estão disponíveis para aplicações?

Você pode ver todas as APIs disponíveis na página API Listing page

5.5.2 Como posso usá-los?

Há um tutorial simples e rápido na in the *Utilizando a API da dojot*.

5.5.3 Estou interessado(a) em integrar minha aplicação com dojot. O que devo fazer?

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/`).
- **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

Release history

6.1 battojutsu - 2018.10.03

- IoT agents:
  - Support for sigfox devices
  - Support for LoRa devices to be used with EveryNet networks.
  - Many improvements for IoT agent MQTT - performance, stability and documentation.
- GUI:
  - Map overlays
  - Pin color configuration on maps
  - Support for more screen resolutions
  - Filters (devices, templates)
  - Improved pagination
- Flows:
  - Support for global contexts: a new service, called ContextManager, was created to deal with contexts within a flow. They can be thought as data chunks that can be stored and retrieved by ContextManager when invoked within a flow node. They are split into four different access levels: tenant, flow, node and node instance. Check flowbroker node library to check how to use context within nodes or check flowbroker's get-context node to use it directly from flowbroker GUI you could just open the new flowbroker UI and check it in the node palette)
  - New configuration options for device actuation: send actuation message to the same device that triggered the flow or set which is the targed device dynamically, set while the flow is being processed.
  - Support for device information caching (improving performance)
- History:

  - Support for queries that retrieve all attributes from a particular device (without explicitly selecting which one should be returned). Check history API for more information.

- DeviceManager:
  - Now DeviceManager is able to generate a random key for devices (PSK)
- New libraries:
  - New library for dojot modules to accelerate development.
  - New log library to standardize all service logs.

CAPÍTULO 7

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

7.1 Gerenciamento de dispositivo

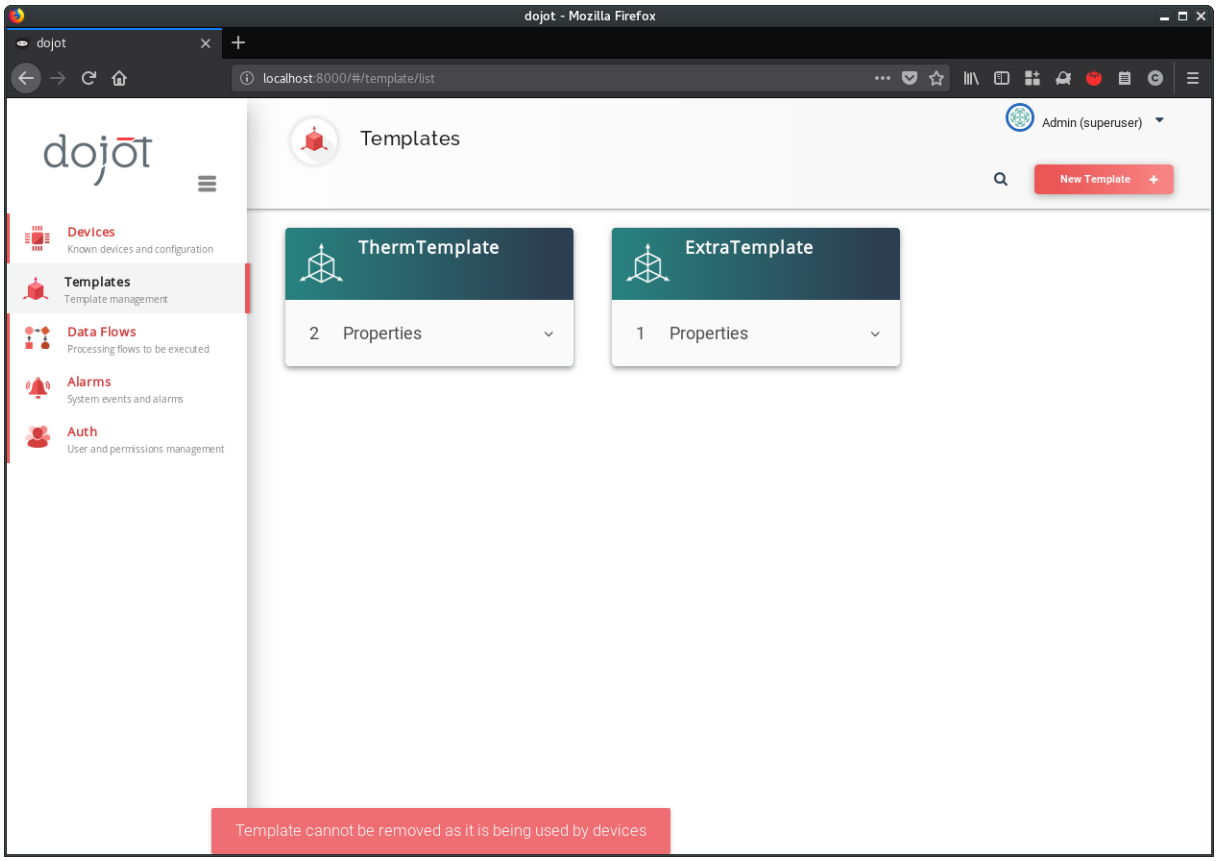
Esta seção mostra como gerenciar dispositivos. Para tal, serão utilizados dois dispositivos sensores de temperatura e um dispositivo virtual, esse último com a função de observar as temperaturas medidas nos dois primeiros e gerar alarmes em determinadas condições.

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.

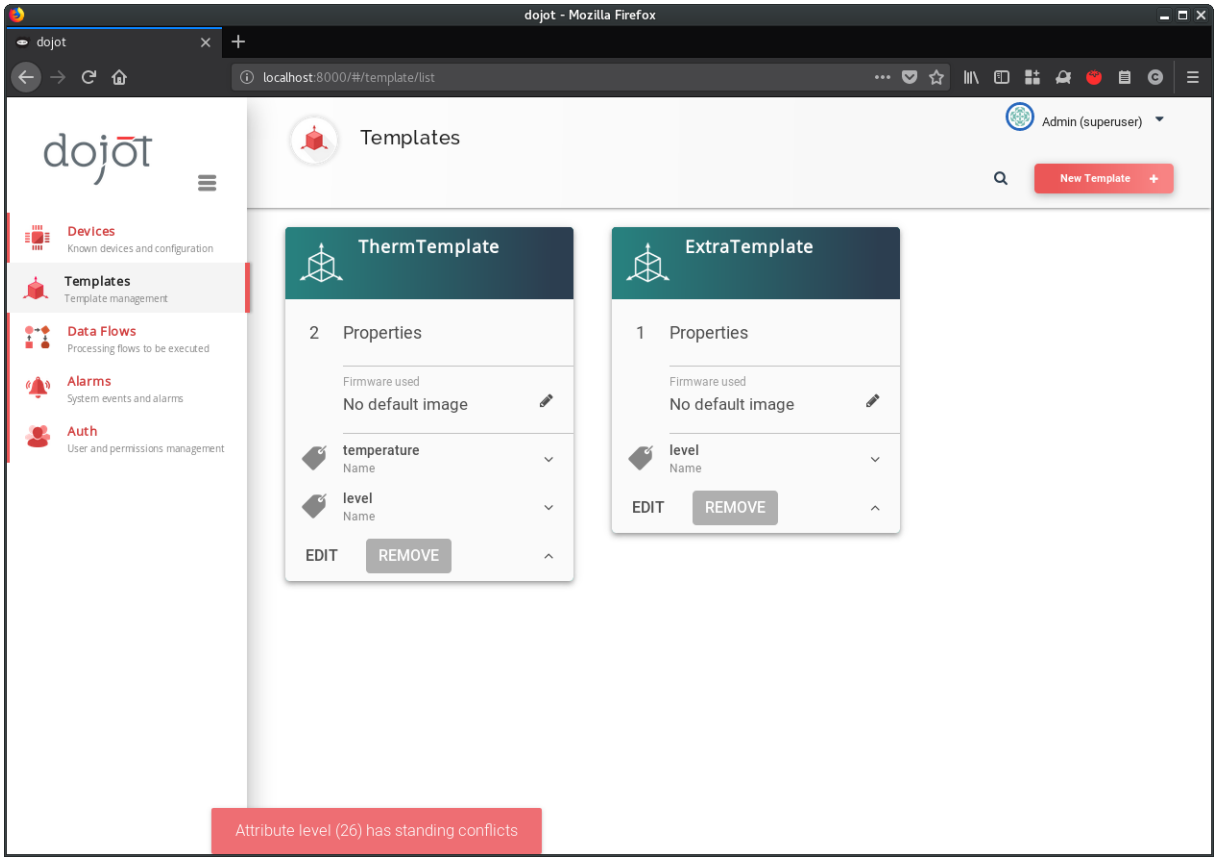
Agora há um modelo do qual dispositivos podem ser "instanciados". Todos dispositivos baseados nesse modelo aceitarão mensagens via protocolo MQTT que serão enviados para o tópico "/devices/thermometers". Para criar novos dispositivos, deve-se voltar para a opção Dispositivos (Devices) e criar um Novo 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.

7.2 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 8

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

8.1 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..."
```

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 an evironment variable called `${JWT}`, which contains the token got from auth component.

8.2 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
      }
    ],
    "id": 1
```

(continues on next page)

(continuação da página anterior)

```
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"
    }
  ]
}
```

8.3 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`).
- 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.

8.4 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/0998/attrs -m '{"temperature": 36.5}'
mosquitto_pub -t /admin/0998/attrs -m '{"temperature": 15.6}'
mosquitto_pub -t /admin/0998/attrs -m '{"temperature": 10.6}'
```

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

```
curl -X GET \
  -H "Authorization: Bearer ${JWT}" \
  "http://localhost:8000/history/device/0998/history?lastN=3&attr=temperature"
```

O endpoint do histórico é construído por meio desses valores:

- `.../device/0998/...`: the device ID is `0998` - this is retrieved from the `id` attribute from the device
- `.../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": "0998",
    "ts": "2018-03-22T13:47:07.050000Z",
    "value": 10.6,
    "attr": "temperature"
  },
  {
    "device_id": "0998",
    "ts": "2018-03-22T13:46:42.455000Z",
    "value": 15.6,
    "attr": "temperature"
  },
  {
    "device_id": "0998",
    "ts": "2018-03-22T13:46:21.535000Z",
    "value": 36.5,
    "attr": "temperature"
  }
]
```

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

CAPÍTULO 9

Using flow builder

This tutorial will show how to properly use flow builder to process messages and events generated by devices.

Nota:

- Who is this for: entry-level users
- Level: basic
- Reading time: 10 min

9.1 Dojot nodes

9.1.1 Device in

This node determine an especific device to be the entry-point of a flow. To configure the device in node, a window like Fig. 9.1 will be displayed.

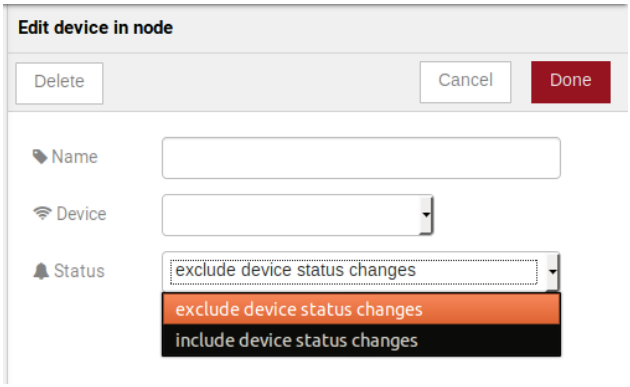


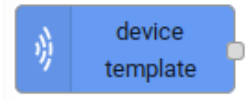
Fig. 9.1: : Device in configuration window

Fields:

- **Name** *(optional)*: Name of the node
- **Device** *(required)*: The *dojot* device that will trigger the flow
- **Status** *(required)*: *exclude device status changes* will not use device status changes (online, offline) to trigger the flow. On the other hand, *include devices status changes* will use these status to trigger the flow.

Nota: If the the device that triggers a flow is removed, the flow becomes invalid.

9.1.2 Device template in



This node will make that a flow get triggered by devices that are composed by a certain template. If the device template that is configured in **device template in** node is template A, all devices that are composed with template A will trigger the flow. For example: *device1* is composed by templates [A,B], *device2* by template A and *device3* by template B. Then, in that scenario, only messages from *device1* and *device2* will initiate the flow, because template A is one of the templates that compose those devices.

Fields:

- **Name** *(optional)*: Name of the node.
- **Device** *(required)*: The *dojot* device that will trigger the flow.
- **Status** *(required)*: Choose if devices status changes will trigger or not the flow.

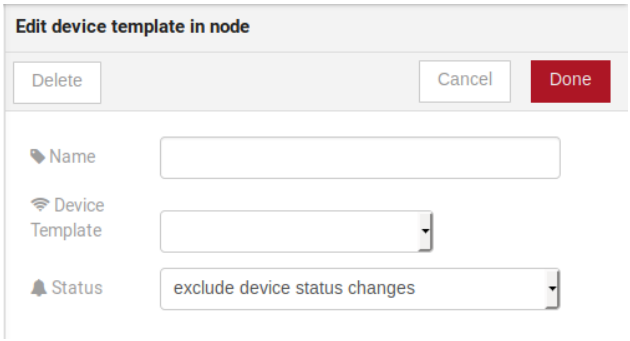
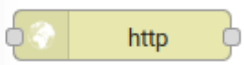


Fig. 9.2: : Device template in configuration window

9.1.3 http



This node sends an http request to a given address, and, then, it can forward the response to the next node in the flow.

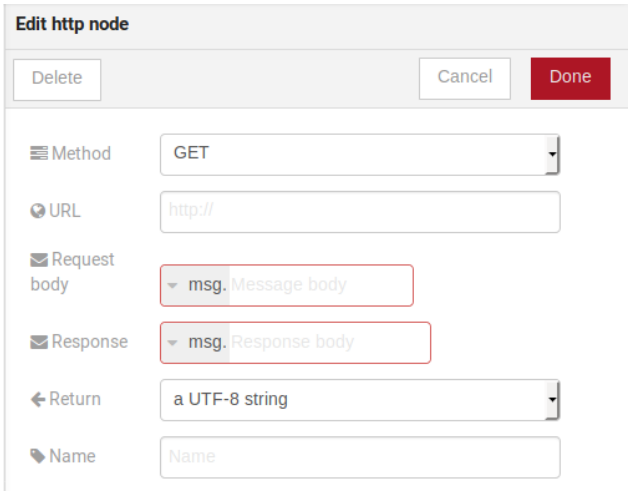
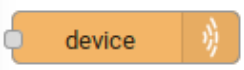


Fig. 9.3: : Device template in configuration window

Fields:

- **Method** *(required)*: The http method (GET, POST, etc. . .).
- **URL** *(required)*: The URL that will receive the http request
- **Request body** *(required)*: Variable that contains the request body. This value can be assigned to the variable using the **template node**, for example.
- **Response** *(required)*: Variable that will receive the http response.
- **Return** *(required)*: Type of the return.
- **Name** *(required)*: Name of the node.

9.1.4 Device out



Device out will determine wich device will have its attributes updated on *dojot* according to the result of the flow. Bear in mind that this node doesn't send messages to your device, it will only update the attributes on the platform. Normally, the chosen device out is a *virtual device*, which is a device that exists only on *dojot*.

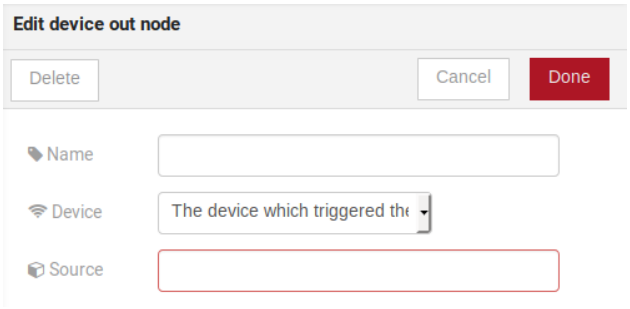


Fig. 9.4: : Device out config window

Fields:

- **Name** *(optional)*: Name of the node.
- **Device** *(required)*: Select "The device that triggered the flow" will make the device that was the entry-point be the end-point of the flow. "Specific device" any chosen device wil be the output of the flow and "a device defined during the flow" will make a device that the flow selected during the execution the endpoint.
- **Source** *(required)*: Data structure that will be mapped as message to device out

9.1.5 Actuate

Actuate node is, basically, the same thing of **device out** node. But, it can send messages to a real device, like telling a lamp to turn the light off and etc.

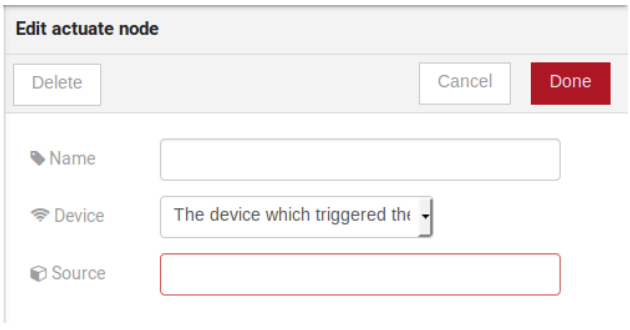


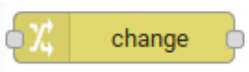
Fig. 9.5: : Actuate configuration

Fields:

- **Name** *(optional)*: Name of the node.

- **Device** *(required)*: A real device on dojot
- **Source** *(required)*: Data structure that will be mapped as message to device out

9.1.6 Change



Change node is used to copy or assign values to an output, i. e., copy values of a message attributes to a dictionary that will be assigned to virtual device.

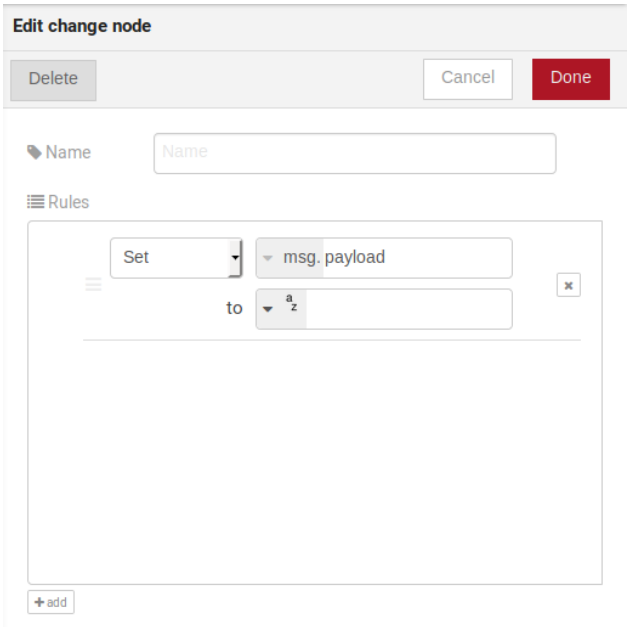


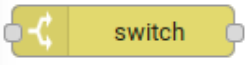
Fig. 9.6: : Change configuration

Fields:

- **Name** *(optional)*: Name of the node
- **msg** *(required)*: Definition of the data structure that will be sent to the next node and will receive the value set on the *to* field
- **to** *(required)*: Assignment or copy of values

Nota: More than one rule can be assign by clicking on *+add* below the rules box.

9.1.7 Switch



The Switch node allows messages to be routed to different branches of a flow by evaluating a set of rules against each message.

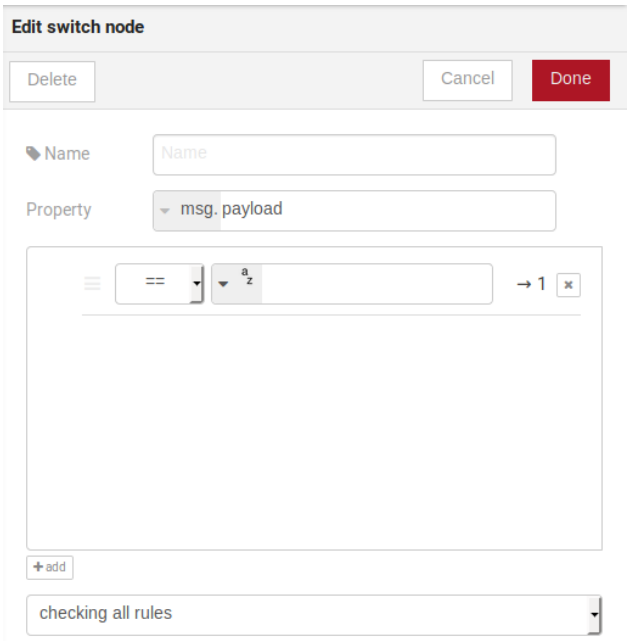


Fig. 9.7: : Switch configuration

Fields:

- **Name** *(optional)*: Name of the node
- **Property** *(required)*: Variable that will be evaluated
- **Rule box** *(required)*: Rules that will determine the output branch of the node. Also, it can be configured to stop checking rules when it finds one that matches other or check all the rules and route the message to the corresponding output.

Nota:

- More than one rule can be assign by clicking on *+add* below the rules box.
- The rules are mapped one-to-one to the output conectors. Then the first rule is related to the first output, the second rule to the second output and etc...

9.1.8 Template

Nota: Despite the name, this node has nothing to do with dojot templates

---

This node will assign a value to a target variable. This value can be a constant, the value of an attribute that came from the entry device and etc...

It uses the mustache template language. Check Fig. 9.8 as example: the field **a** of payload will be replaced with the value of the **payload.b**

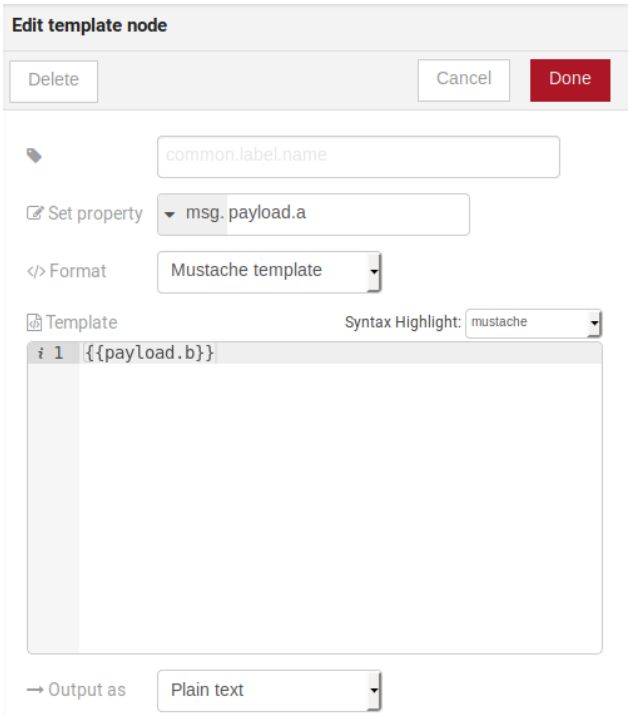
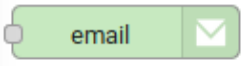


Fig. 9.8: : Template configuration

Fields:

- **Name** *(optional)*: Name of the node
- **Set Property** *(required)*: Variable that will receive the value
- **Format** *(required)*: Format template will be writen
- **Template** *(required)*: Value that will be assigned to the target variable set on **Set property**
- **Output as** *(required)*: The format of the output

9.1.9 Email



Sends an e-mail for a given address.

Fields:

- **From** *(required)*: The source email.
- **To** *(required)*: Destination email.
- **Server** *(required)*: The server of the email destination.
- **Subject** *(required)*: Subject of the email.
- **Body** *(required)*: Message on the email. The message can be writen in a variable using the **template node**, for example.
- **Name** *(optional)*: Name of the node.

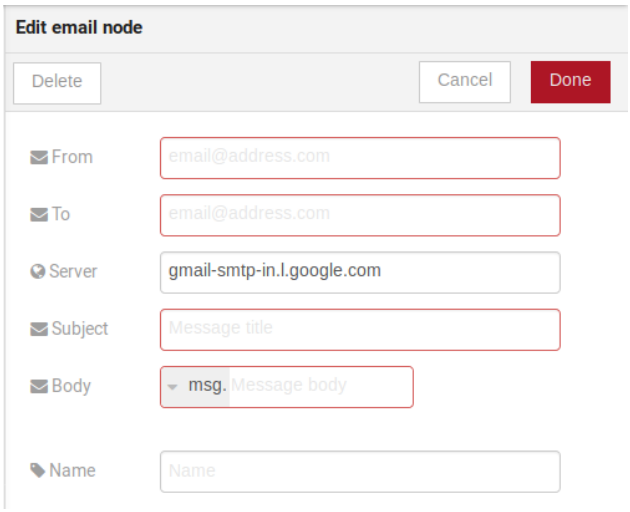


Fig. 9.9: : Email configuration

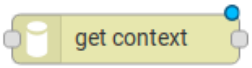
9.1.10 Geofence

Select an interest area to determine wich devices will activate the flow

Fields:

- **Area** *(required)*: Area that will be selected. It can be chosen with an square or with a pentagon.
- **Filter** *(required)*: Which side of the area will be picked: inside or outside the marked area in the field above.
- **Name** *(optional)*: Name of the node

9.1.11 Get Context



This node is used to get a variable that is in the context and assign its value to a variable that will be used in the flow.

Fields:

- **Name** *(optional)**: Name of the node
- **Context layer** *(required)**: The layer of the context that que variable is at
- **Context name** *(required)**: The variable that is in the context
- **Context content** *(required)**: The variable in the flow that will receive the value of the context

9.2 Learn by examples

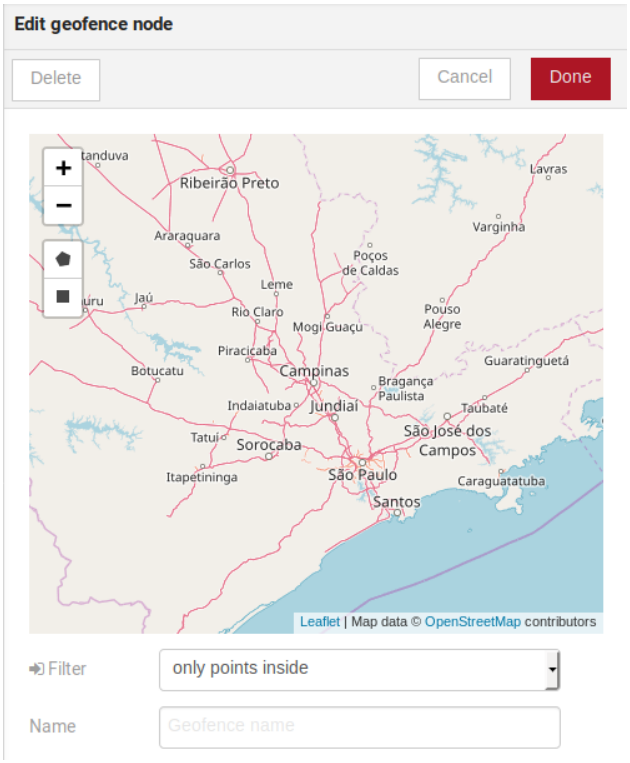
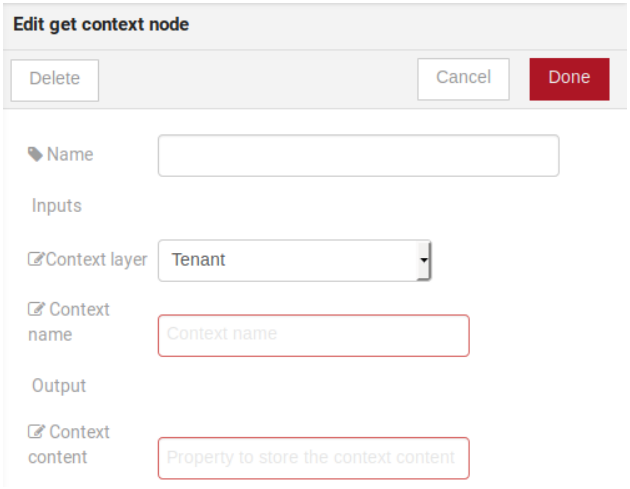


Fig. 9.10: : Geofence configuration



9.2.1 Using template and email nodes

To explain these nodes, the flow below will be used:

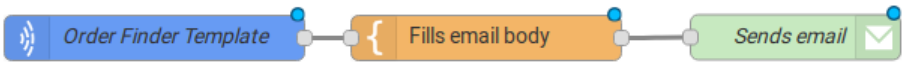


Fig. 9.11: : Flow using template and email nodes

Wonder a system that sends an email to somebody when an order arrive at his mail box. The email would be sent with the name of the sender, his phone number and the content of the order. A device with the order finder template has the attributes: *sender*, *phone* and *content*.

The template node will fill the message with the attributes that came in the message. The attributes sent by the entry-point device can be accessed on the variable **payload**. So, using the mustache template language, the node configuration would be like Fig. 9.12.

Then, the email body on the email node should be assigned to the variable that is on the field *Set property* on Fig. 9.12:

Then, the result of the flow, is an email arrive, problably at the spam box, to the destination address:

9.2.2 Using http node

Imagine this scenario: a device sends an *username* and a *password*, and from these attrs, the flow will request to a server an authentication token that will be sent to a virtual device that has a *token* attribute.

To send that request to the server, the http method should be a POST and the parameters should be within the requisition. So, in the template node, a JSON object will be assigned to a variable. The body (parameters *username* and *password*) of the requisition will be assigned to the **payload** key of the JSON object. And, if needed, this object can have a *headers* key as well.

Then, on the http node, the Requisition field will receive the value of the object created at the template node. And, the response will be assigned to any variable, in this case, this is *msg.res* .

Nota: If UTF-8 String buffer is chosen in the return field, the body of the response body will be a string. If JSON object is chosen, the body will be an object.

As seen, the response of the server is *req.res* and the response body can be accessed on **msg.res.payload**. So, the keys of the object that came on the responsy can be accessed by: **msg.res.payload.key**. On figure Fig. 9.18 the token that came in the response is assigned to the attribute token of the virtual device.

Then, the result of the flow is the attribute *token* of the virtual device be updated with the token that came in the response of the http request:

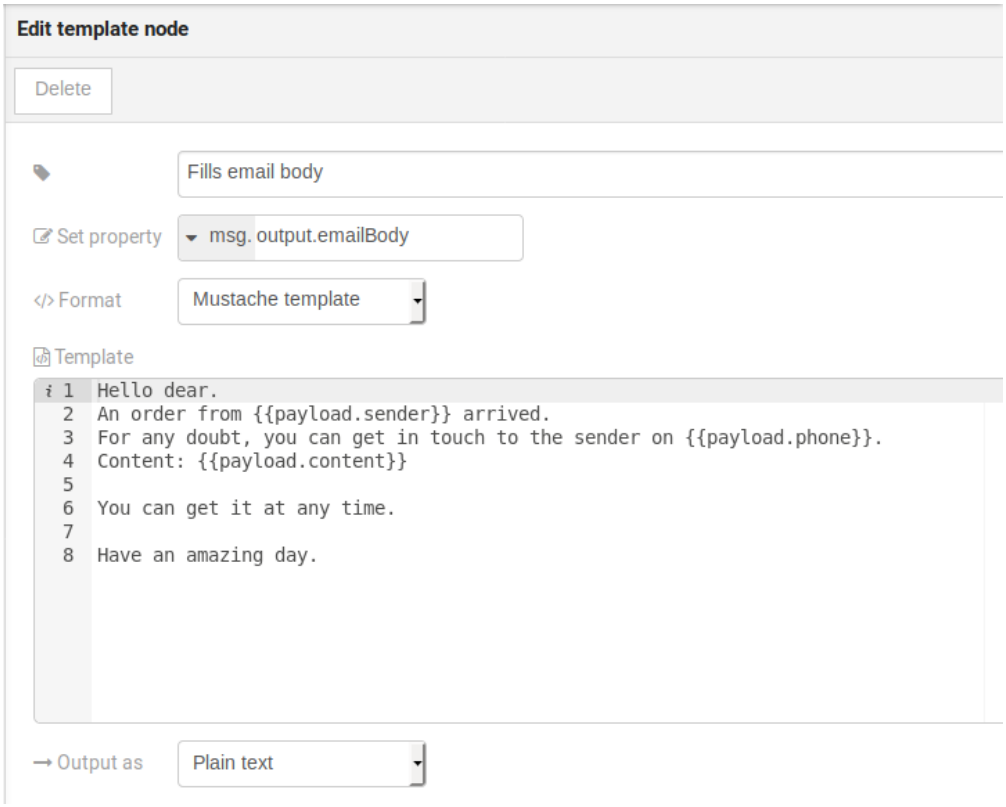


Fig. 9.12: : Template configuration

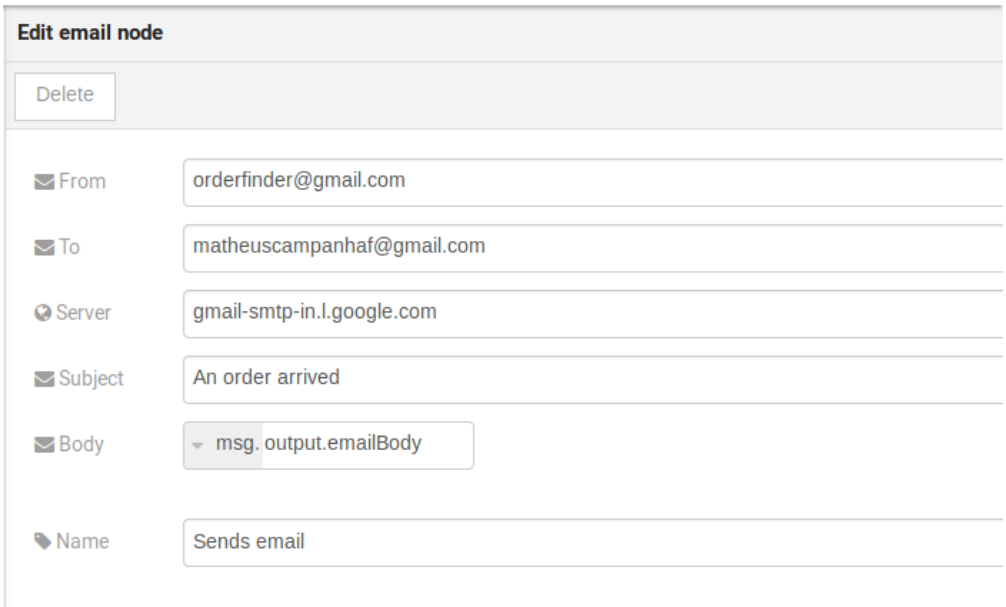


Fig. 9.13: : Email node configuration

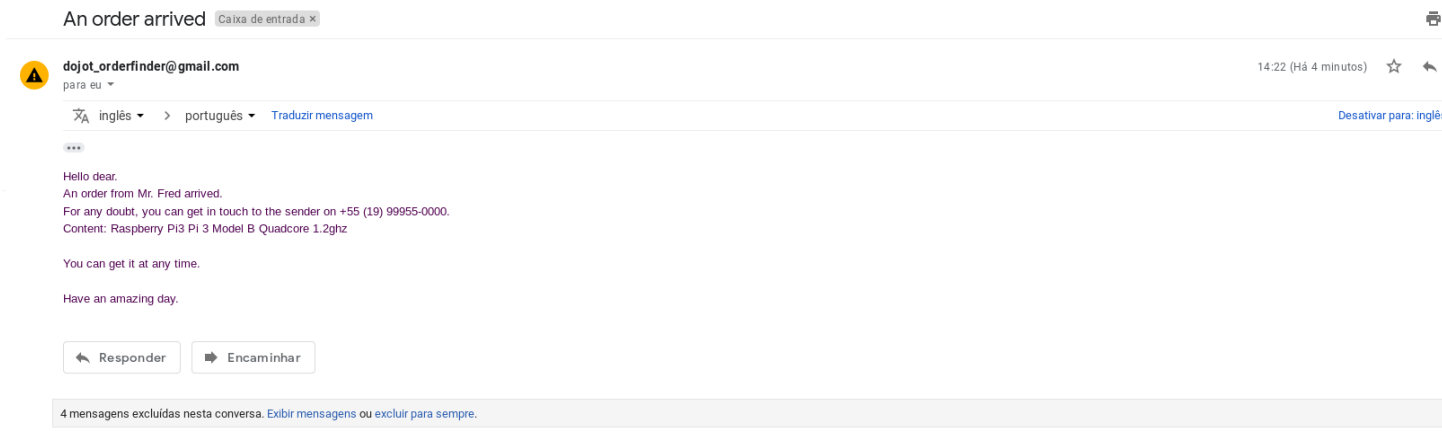


Fig. 9.14: : Sent email

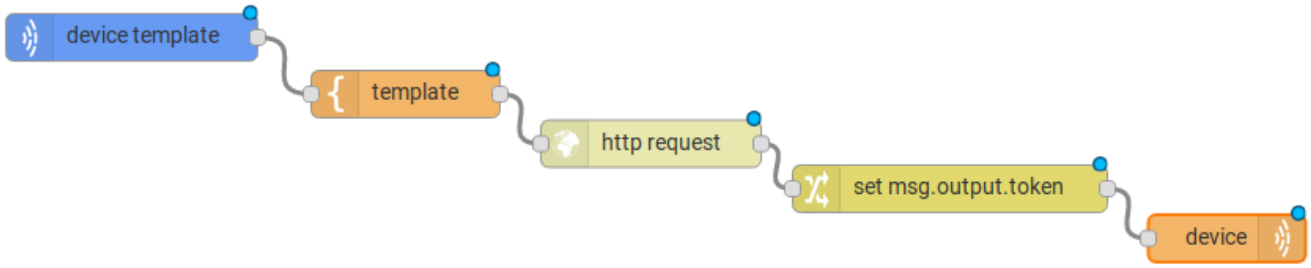


Fig. 9.15: : Flow used to explain http node

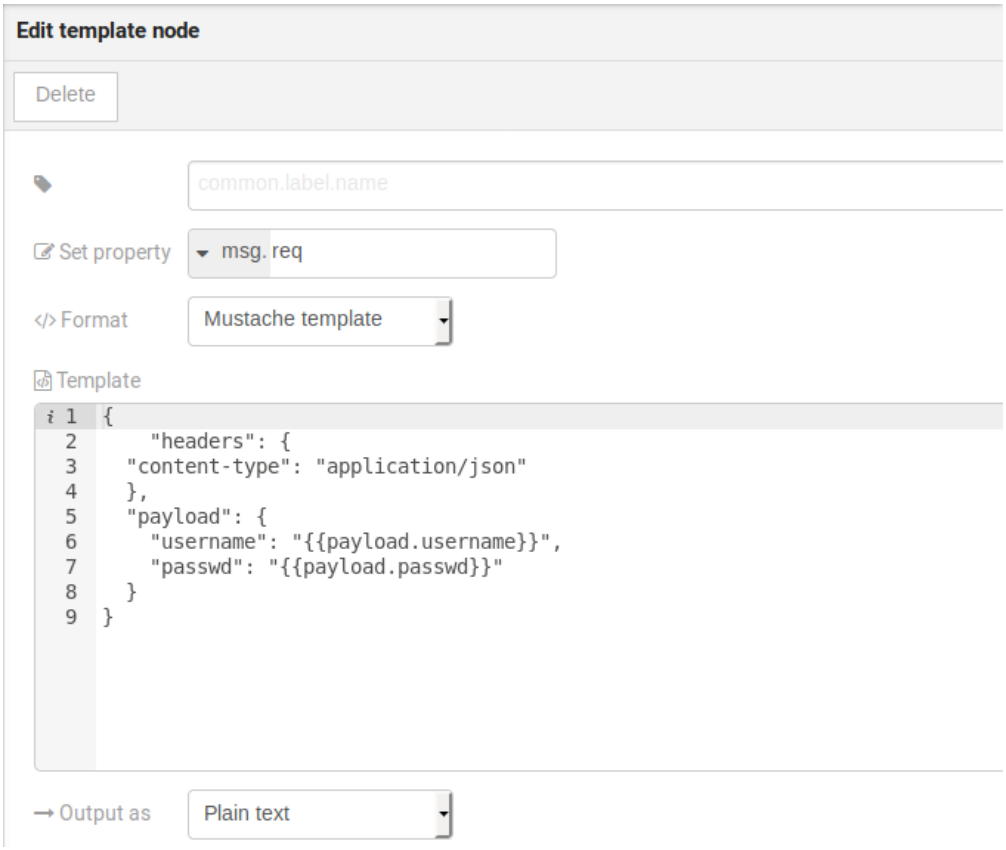


Fig. 9.16: : Template node configuration

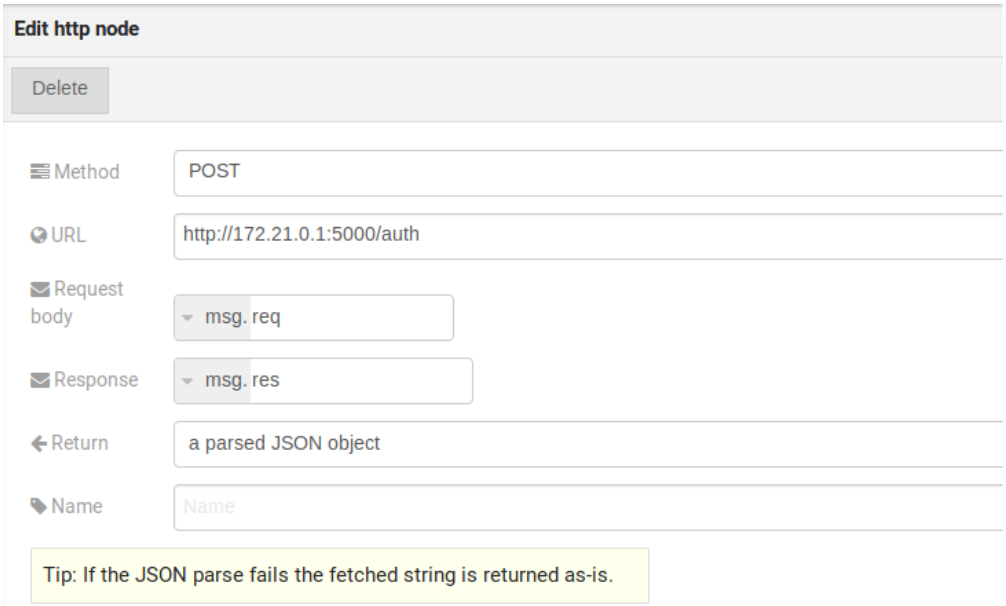


Fig. 9.17: : Template node configuration

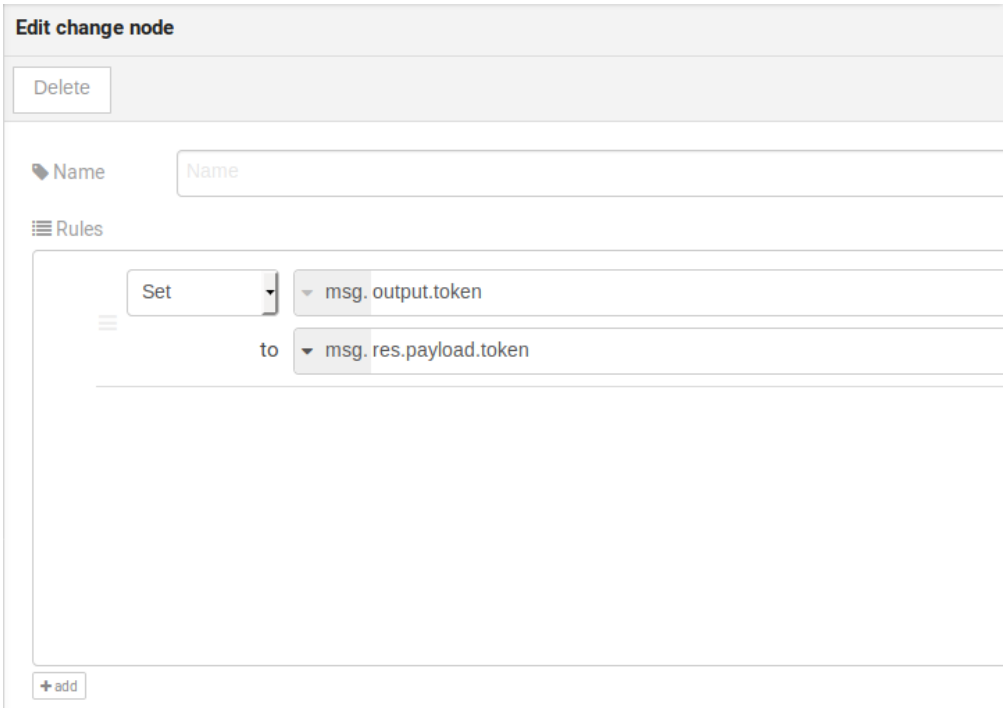


Fig. 9.18: : Template node configuration

Fig. 9.19: : Device out configuration

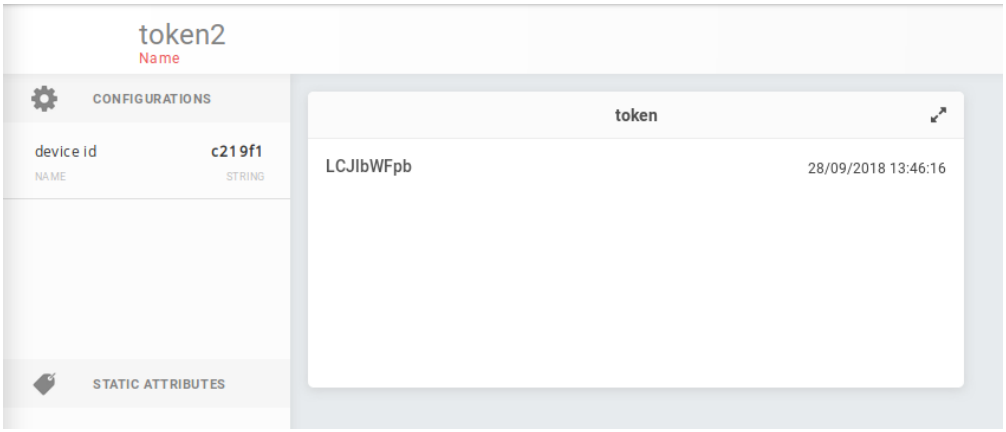


Fig. 9.20: : Device updated

9.2.3 Using geofence node

A good example to learn how geofence node works ia studying the flow below:

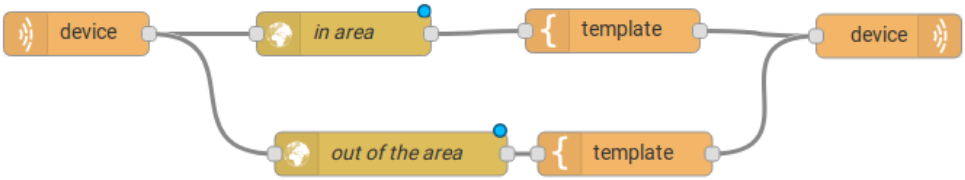


Fig. 9.21: : Flow using geofence

The geofence node named *in area* is set like seem in Fig. 9.22. The only thing that diffs the geofence nodes *in area* from *out of the area* is the field **Filter** that, in the first, is configured to *only points inside* and *only points outside* in the second, respectively.

Then, if the device that is set as *device in* sends a messagem with a geo attribute the geofence node will evaluate the geo point acoording to its rule and if it matches the rule, the node forward the information to the next node and, if not, the execution of the branch, which has the geofence that the rule didn't match, stops.

Nota: To geofence node work, the messege received **should** have a geo attribute, if not, the branches of the flow will stop at the geofence nodes.

Back to the example, if the car sends a message that he is in the marked area, like `{ "position": "-22.820156,-47.2682535" }`, the message received in device out will be "Car is inside the marked area", and, if it sends `{"position": "0,0"}` device out will receive "Car is out of the marked area"

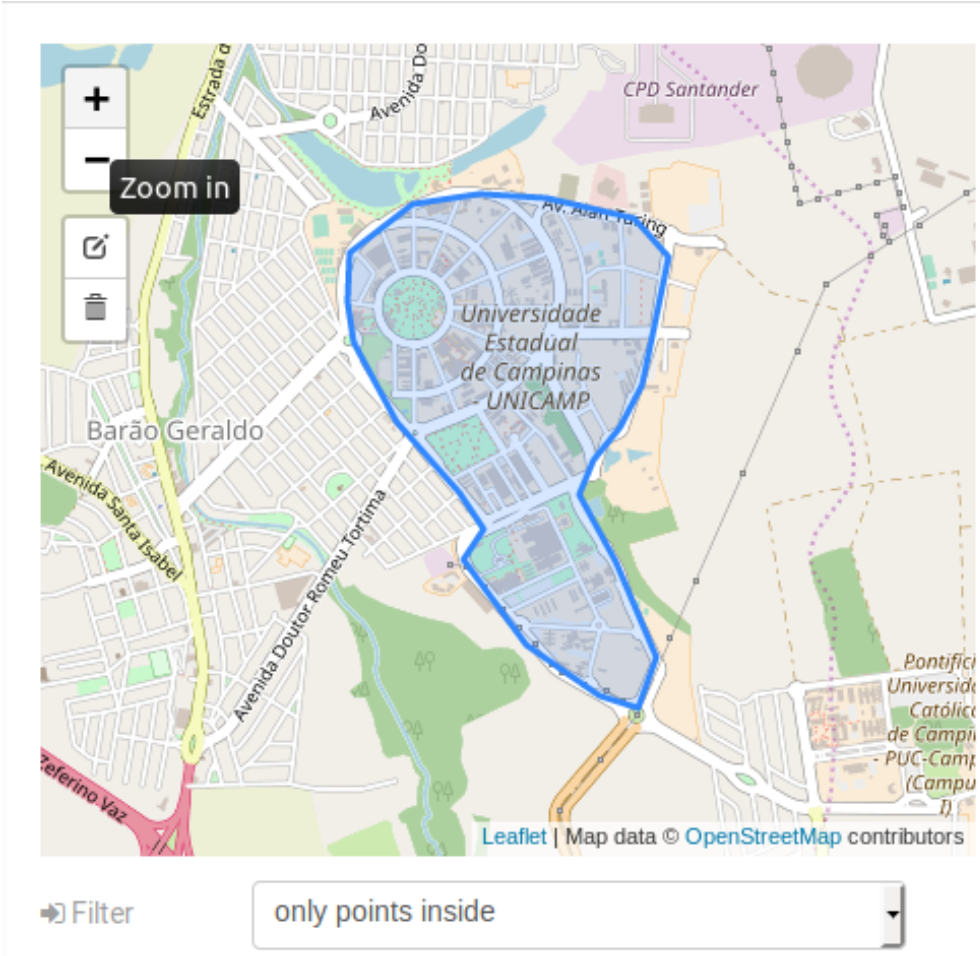


Fig. 9.22: : Geofence node configuration

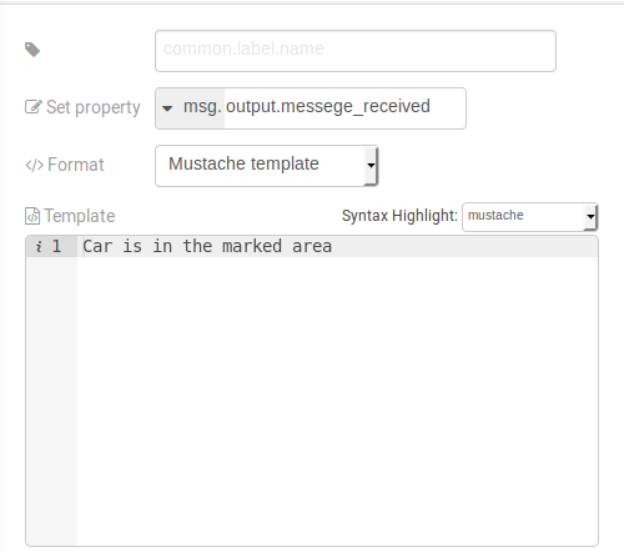


Fig. 9.23: : Template node configuration if the car is in the marked area

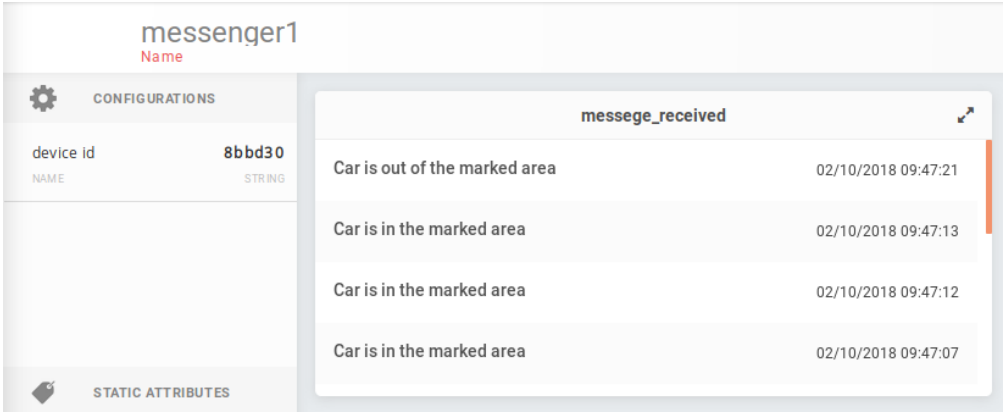


Fig. 9.24: : Output in device out