auth Documentation

Release 0.2.0

Giovanni Curiel dos Santos

Sep 04, 2020

Contents:

This service handles user authentication for dojot. Namely this is used to maintain the set of known users, and their associated roles. Should a user need to interact with the platform, this service is responsible for generating the JWT token to be used when doing so.

CHAPTER 1

Installation

This service depends on a couple of python libraries to work. To install them, please run the commands below. These have been tested on an ubuntu 16.04 environment (same used when generating) the service's docker image.

```
# you may need sudo for those
apt-get install -y python3-pip
python3 setup.py
```

Another alternative is to use docker to run the service. To build the container, from the repository's root:

```
docker build -t <tag> -f docker/Dockerfile .
```

In order to run this command, you may need sudo on your machine: https://docs.docker.com/engine/installation/linux/linux-postinstall/

CHAPTER 2

Configuration

2.1 Database related configuration

Some auth configuration is made using environment variables. On a Linux system one can set a environment variable with the command

```
export VAR_NAME=varvalue
```

on a docker-compose schema, one can set environment variables for a container Append the following configuration

```
environment:
  VAR_NAME: "varvalue"
```

The default value is used if the configuration was not provided The following variables can be set

Table 2.1: Environment variable

Variable	Description	Default value
AUTH_DB_NAME	database type. Current only postgres is supported	postgres
AUTH_DB_USER	The username used to access the database	auth
AUTH_DB_PWD	The password used to access the database	empty password
AUTH_DB_HOST	The URL used to connect to the database	http://postgres
AUTH_KONG_URL	The URL where the Kong service can be found. If set to 'DISABLED' Auth won´t try to configure Kong and will generate secrets for the JWT tokens by itself.	http://kong:8001
AUTH_TOKEN_EXP	Expiration time in second for generated JWT tokens	420
AUTH_TOKEN_CHECK_SIGN	Whether Auth should verify received JWT signatures. Enabling this will cause one extra query to be performed.	False
AUTH_CACHE_NAME	Type of cache used. Currently only Redis is suported. If set to 'NOCACHE' auth will work without cache. Disabling cache usage considerably degrades performance.	redis
AUTH_CACHE_USER	username to access the cache database	redis
AUTH_CACHE_PWD	password to acces the cache database	empty password
AUTH_CACHE_HOST	the hostname where the cache can be found	redis
AUTH_CACHE_TTL	Cache entry time to live in seconds	720
AUTH_CACHE_DATABASE	Cache database name (or number)	'0'

If you are running without docker, You will need to create and populate the database tables before the first run. This can be done by executing the following commands in python3 shell:

```
>>> from webRouter import db
>>> db.create_all()
```

Create the initial users, groups and permissions

```
python3 initialConf.py
```

CHAPTER 3

REST API

This is the REST API documentation for DeviceManager. This page is automatically generated from these files:

- `auth`
- `CRUD API`
- `Relations`
- `Report`

All APIs are available in Github pages API description

CHAPTER 4

Internal messages

There are some messages that are published by Auth through Kafka. These messages are related to tenancy lifecycle events.

Table 4.1: Kafka messages

Event	Subject	Service	Message type
Tenant creation	dojot.tenancy	internal	*Tenant creation*
Tenant removal	dojot.tenancy	internal	*Tenant removal*

4.1 Tenant creation

This message is published whenever a new tenant is created. Its payload is a simple JSON:

```
{
  "type": "CREATE",
  "tenant": "admin"
}
```

And its attributes are:

- *type* (string): “CREATE”
- *tenant*: New tenant

4.2 Tenant removal

This message is published whenever a new tenant is removed. Its payload is a simple JSON:

```
{
  "type": "DELETE",
  "tenant": "admin"
}
```

And its attributes are:

- *type* (string): “DELETE”
- *tenant*: Tenant to be removed

CHAPTER 5

How to build/update/translate documentation

If you have a local clone of this repository and you want to change the documentation, then you should follow this simple guide.

5.1 Build

The readable version of this documentation can be generated by means of sphinx. In order to do so, please follow the steps below. Those are actually based off ReadTheDocs documentation.

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme sphinx-intl
export READTHEDOCS_VERSION=latest
make html
```

The *READTHEDOCS_VERSION* environment variable should be set to the component version being built, such as *latest* or *0.2.0*. In the automated build process from readthedocs, this exact variable will be set as the name of the branch/tag being built.

For that to work, you must have pip installed on the machine used to build the documentation. To install pip on an Ubuntu machine:

```
sudo apt-get install python-pip
```

To build the documentation in Brazilian Portuguese language, run the following extra commands:

```
sphinx-intl -c conf.py build -d locale
make html BUILDDIR=build/html-pt_BR O='-d build/doctrees/ -D language=pt_BR'
```

5.2 Update workflow

To update the documentation, follow the steps below:

1. Update the source files for the english version
2. Extract translatable messages from the english version

```
make gettext
```

3. Update the message catalog (PO Files) for pt_BR language

```
sphinx-intl -c conf.py update -p build/gettext -l pt_BR
```

4. Translate the messages in the pt_BR language PO files

This workflow is based on the Sphinx i18n guide.