
dojot Documentation

Release 0.0.0

Matheus Magalhaes

Feb 21, 2018

Contents:

1	Architecture	3
1.1	Components	3
1.2	Infrastructure	3
1.3	Communications	4
1.4	Deployment strategies	4
1.5	Comparative analysis	4
2	Operations Guide	5
2.1	Deployment	5
2.2	Device Management	5
2.3	User Management	6
2.4	Resources Management	6
2.5	System Dashboard	6
3	User Guide	7
3.1	Getting Started	7
3.2	Device Management	7
3.3	Flow Management	8
4	Installation guide - Docker compose	9
4.1	Dependencies	9
4.2	Installation	10
4.3	Configuration	11
5	Running dojot on VirtualBox	13
6	Frequently Asked Questions	21
6.1	General	22
6.2	Usage	23
6.3	Devices	24
6.4	Data Flows	25
6.5	Applications	27

This is the high-level documentation for dojot IoT platform developed by CPqD. This platform is largely based on [FIWARE](fiware.org), and aims to provide the application and device developers with a more concise and integrated interaction, while benefiting for the highly customizable and efficient infrastructure provided by FIWARE.

While based on FIWARE, this platform actually has a large set of components of its own, and interaction between components was modified to allow better packaging and performance for the solution as a whole.

While this does provide an overall glimpse of the platform, this documentation is not suited for middleware developers that might want to better understand the components that compose the solution themselves. For that, please check the component's own documentation repositories and ReadTheDocs pages.

This document describes the current architecture that guides the platform implementation, detailing the components that comprise the solution, as well as their functionalities and how each of them contribute to the platform as a whole.

While a brief explanation of each component is provided, this high level description does not explain (or aims to explain) the minutia of each component's implementation. For that, please refer to each component's own documentation.

Table of Contents

- *Components*
- *Infrastructure*
- *Communications*
- *Deployment strategies*
- *Comparative analysis*

1.1 Components

This section should describe - in high level - the components that comprise the solution, giving the reader enough detail to understand the responsibilities of the components, the technologies involved in its design and implementation and its relation with its peers.

1.2 Infrastructure

This section should describe the components that are used as ready-made pieces of working software that compose the solution, but have no implementation specific to the project. Relevant topics that might be discussed here are:

- The API gateway
- Storage components (mongo, redis, HDFS, CEPH, etc.)
- Processing libraries and environments (Spark, Flink, Storm, kafka-streaming, map-reduce, etc.)
- Broker components (rabbitMQ, mosquitto, kafka, verneMQ, emqtt, etc.)

1.3 Communications

This section should provide the reader with the communication strategy used to bind together the components that comprise the solution, as well as the interfaces (protocols, serialization formats) available to the applications and devices developers.

1.4 Deployment strategies

This section should list the deployment requirements and implementation decisions made to satisfy those requirements. “Why orchestrator platform ‘x’?”, “How can this be deployed on commercial cloud environments?”, “How can this be deployed on stand-alone environments?” are all questions that should be answered here.

1.5 Comparative analysis

This section should detail the features that differentiate the platform from a “stock” deployment of fiware, as well as a feature summary comparing the proposed solution with a reduced set of third-party implementations of IoT platforms available.

This document provides information on how to properly deploy and manage an instance of dojot. For documentation regarding the usage of the platform from the perspective of either an application, or device developer please refer to the [user guide]().

Table of Contents

- *Deployment*
- *Device Management*
- *User Management*
- *Resources Management*
- *System Dashboard*

2.1 Deployment

This section should describe the steps required to deploy the solution on all “homologated” environments (e.g. standalone, aws, google cloud, bluemix, etc.). For each environment, there’ll also be a link pointing to the environment-specific section of the *Resources Management* sub-section that describes how cloud resources are managed (allocated, released and pertinent configuration).

2.2 Device Management

This section should describe the steps required to configure a new device on the platform. While this information will also be presented on the user guide, here the idea is to give more focus to the specific infrastructure that has to be managed in order to guarantee the device’s authenticity and communication.

2.3 User Management

This section should describe the steps required to configure user roles and role permissions for the platform itself, as well as handle application authentication features.

2.4 Resources Management

For each “homologated” deployment scenario, this section should describe how the deployment is done, as well as which parameters are available for each of them.

2.5 System Dashboard

This should be a brief description of the system dashboard that is made available for the system administrator to check the system’s overall status and alerts.

This document provides information on how to use dojoy from a user perspective. On that regard, this should describe the steps required to install and operate the platform from a device developer or application developer point of view. For documentation regarding the operation of the platform itself, please refer to the [operations guide]()

Table of Contents

- *Getting Started*
- *Device Management*
- *Flow Management*

3.1 Getting Started

The idea here is to present the reader with a sample end-to-end usage of the platform, showcasing a relevant set of the features implemented by the platform, thus serving as an example of how to use the platform on a “real world” scenario.

To make things easier for the user, there may be a VM image with a pre-configured deployment of the platform. Should this be the case, instructions on how to deploy the solution might reside only on the Operations Guide.

3.2 Device Management

This section should detail how to integrate a new device with the system. That should encompass the both the communication requirements imposed on the device in order to allow its usage with the platform, as well as the steps (if any, depending on the protocol used) to configure this new device within the platform.

This could also explain (if indeed implemented) the device management functionalities made available by the platform to the device developer.

Regarding the requirements imposed on the devices, it is foreseen that, for each communication scheme (protocol/serialization format) officially supported by the platform, a step by step guide on how to “develop” a device is supplied. Such guide can, if applicable, make use of a platform-provided library or SDK.

3.3 Flow Management

Moving to the perspective of an application developer, this section should list and explain the usage of the information flow configuration process within the platform - how to use the provided gui, high level description of the APIs that can be used to configure such flows, available actions to be used when building the flows, so on and so forth.

Installation guide - 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.

Table of Contents

- *Dependencies*
 - *Docker engine*
 - *Docker Compose*
- *Installation*
- *Configuration*
 - *API gateway configurarion*
 - *User creation*

4.1 Dependencies

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

4.1.1 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/>

Note: 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/>

4.1.2 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 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 git@github.com:dojot/docker-compose.git
```

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]
```

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
```

Note: 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/>

4.3 Configuration

Once the environment is up, a few configuration steps are required to make it operational.

4.3.1 API gateway configurarion

In order to guarantee the proper mapping of API into processing services, the API gateway must be configured. To do so, please run `kong_config.sh`, present at the root of the repository.

```
./kong.config.sh
```

4.3.2 User creation

To be able to use the system's web front-end and make API calls, a user must be created. To create a first *admin* user, the following script can be run on the host machine of the platform (that is, the machine where docker-compose was run). The script is located at the root of the repository.

```
./create.user.sh
```

Running dojot on VirtualBox

This guide provides instructions to run dojot platform on VirtualBox.

You should only run dojot this way if you don't have any familiarity with docker and just want to learn how to use dojot. We don't recommend it for development and much less for experimental or real deployments.

The steps described here were checked on Windows 7, but you shouldn't have problems to run them in different operational systems.

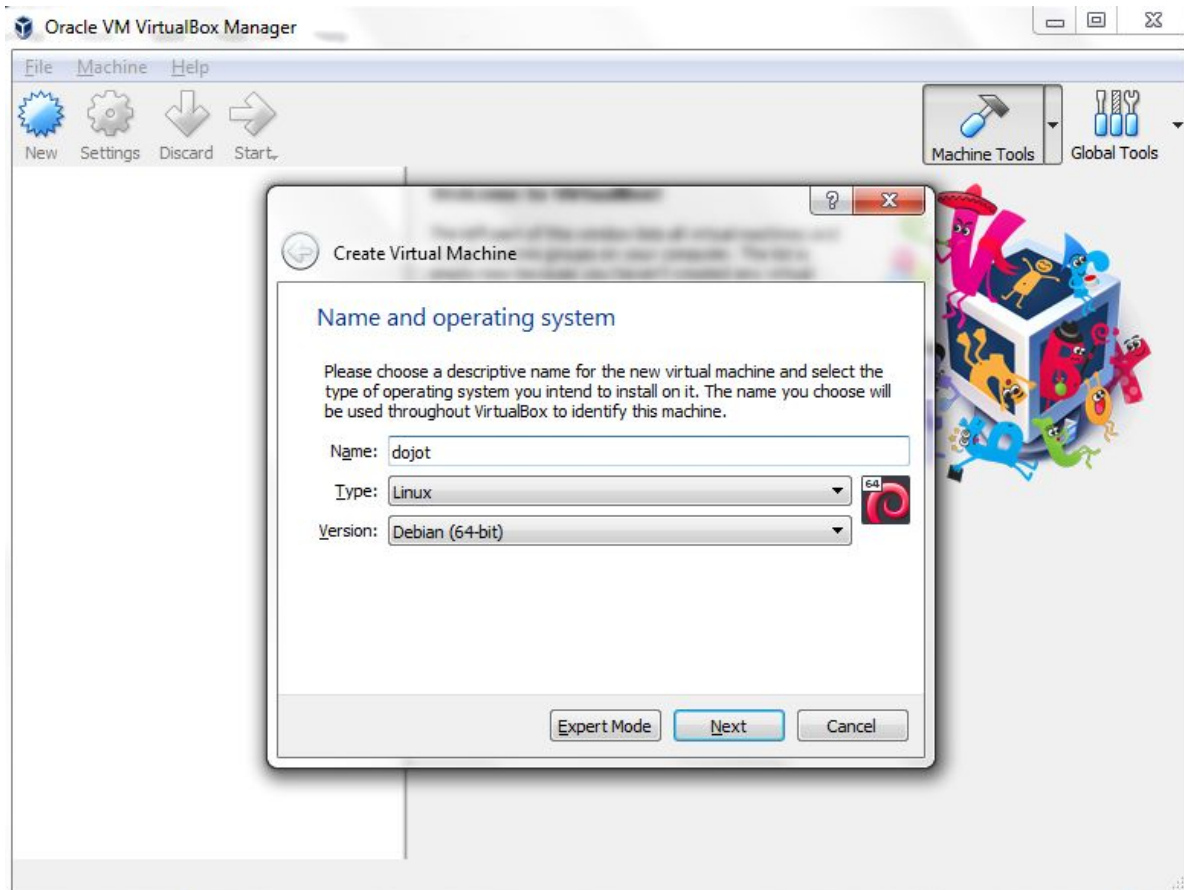
First of all, if you don't have VirtualBox you'll need to install it. Up to date information and installation procedures can be found at the project's documentation:

<https://www.virtualbox.org/>

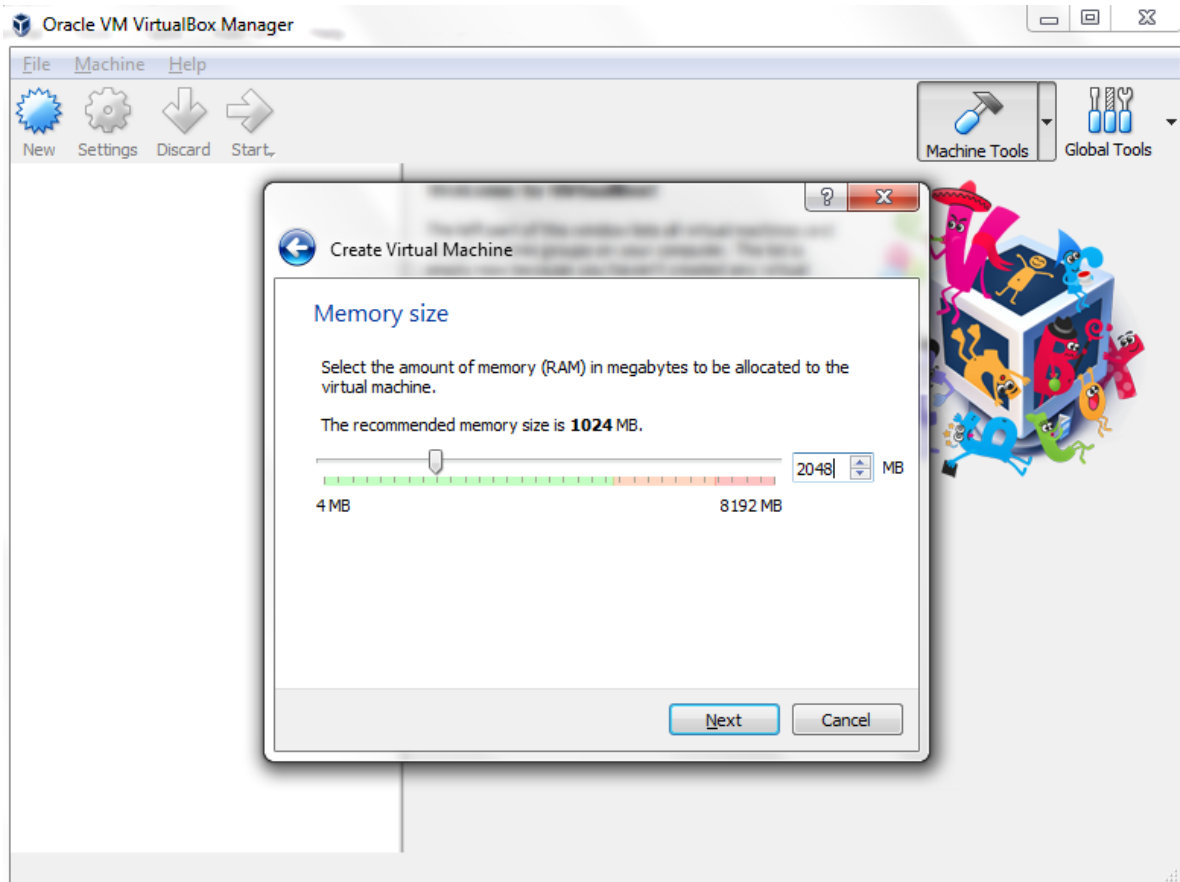
Then, you need to download a virtual machine image with dojot, which is available at:

<http://dojot-iso.s3.amazonaws.com/imagen/dojot.0.1.0-dojot.vdi>

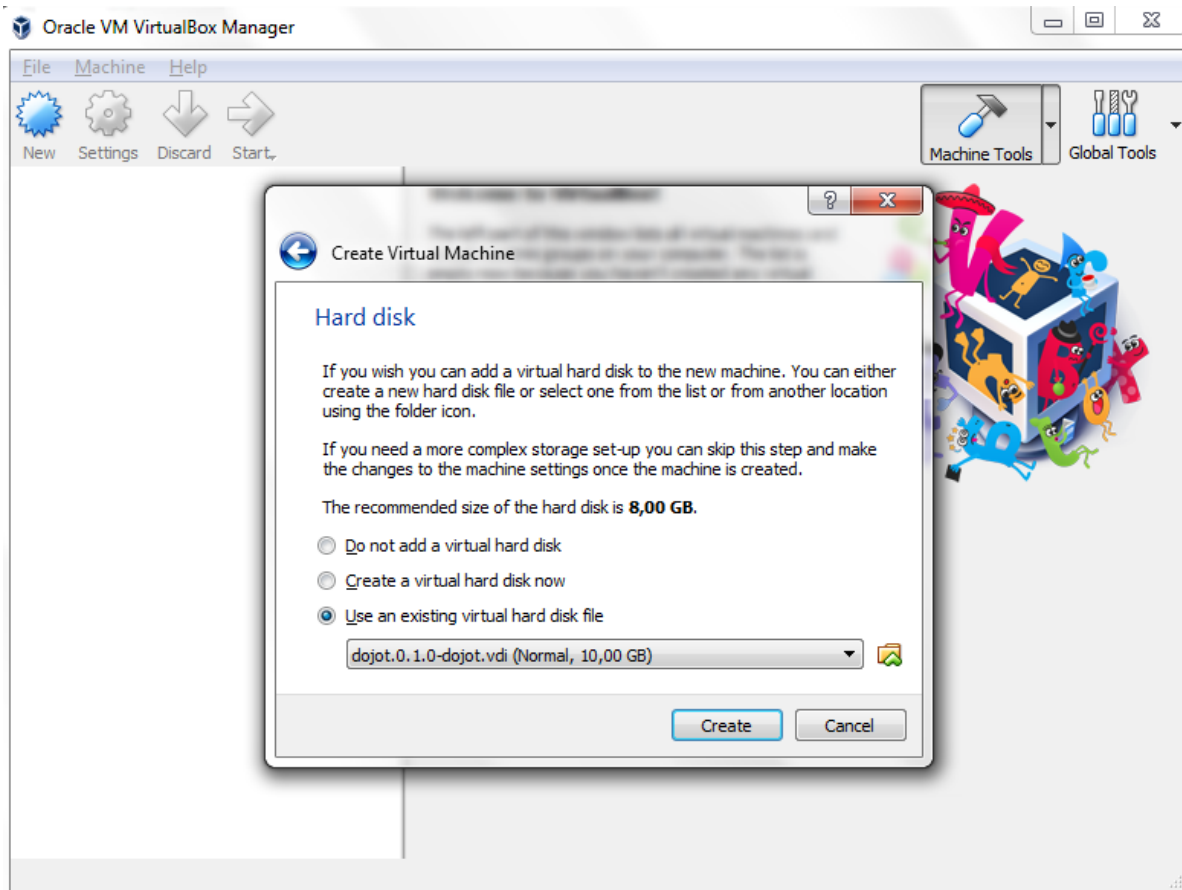
The next step is to create a virtual machine on VirtualBox. Click on the New button, then set the name as you wish, type to Linux and version to Debian (64-bit).



Click on Next, and set the memory size. We recommend at least 2048 MB.

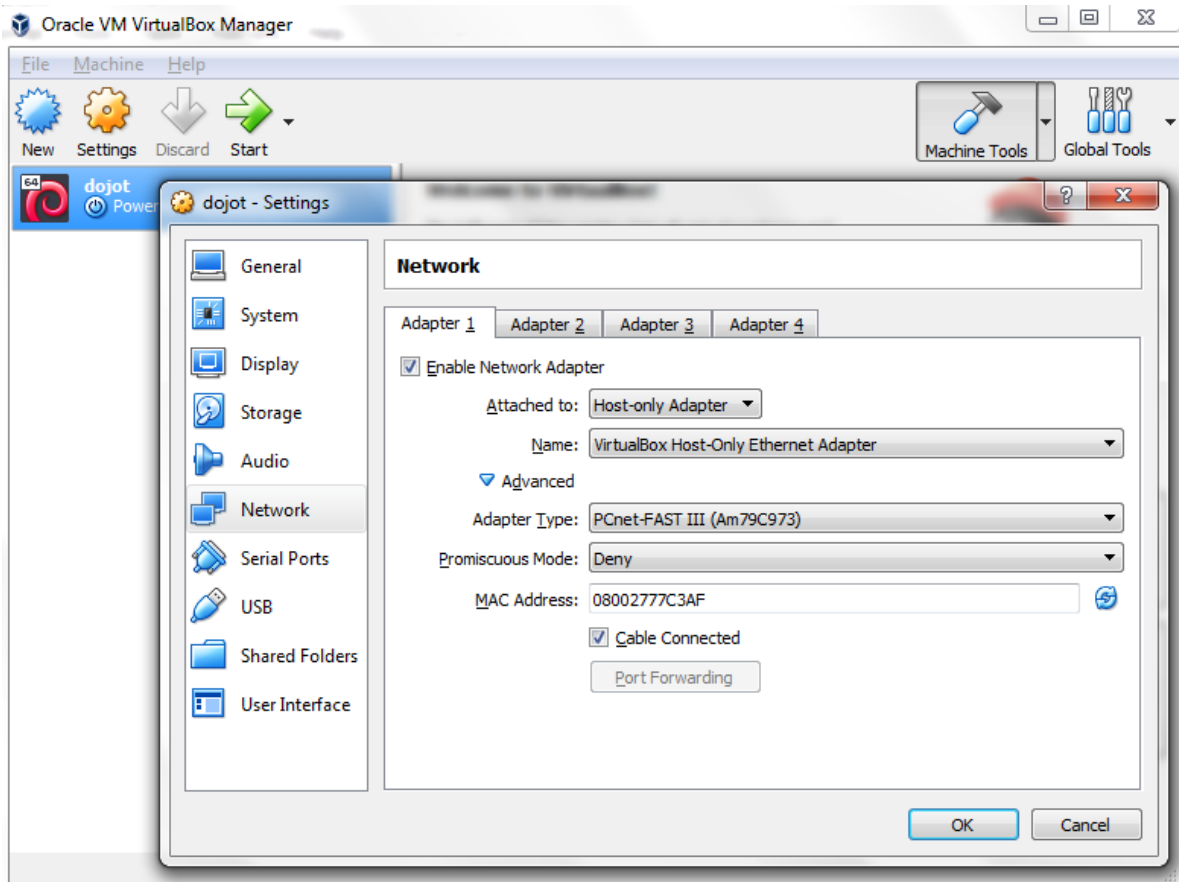


Click on Next, and set the hard disk to use an existing virtual hard disk file and choose the downloaded image.



Click on Create.

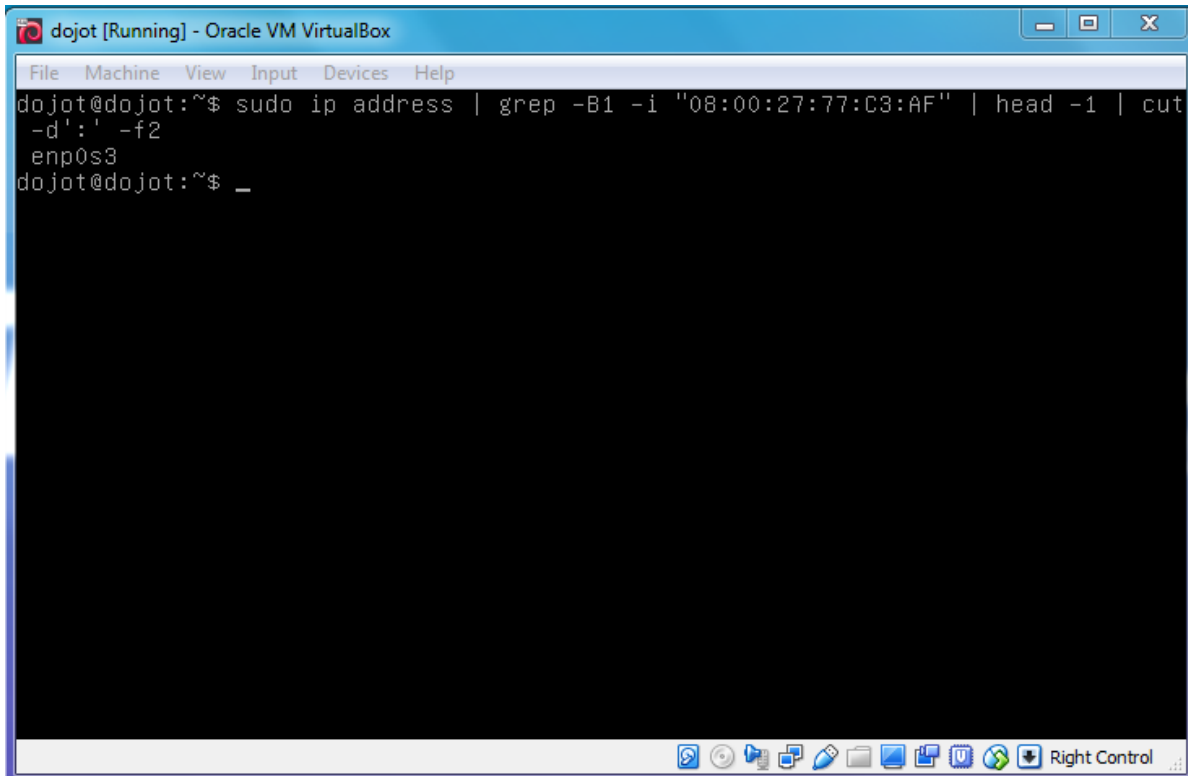
Next, click on Settings/Network and check whether the network adapter is enabled and set it to `Host-only`. This will allow host and guest to communicate to each other. Write down the MAC Address, you will need it later.



Click on OK and start the virtual machine.

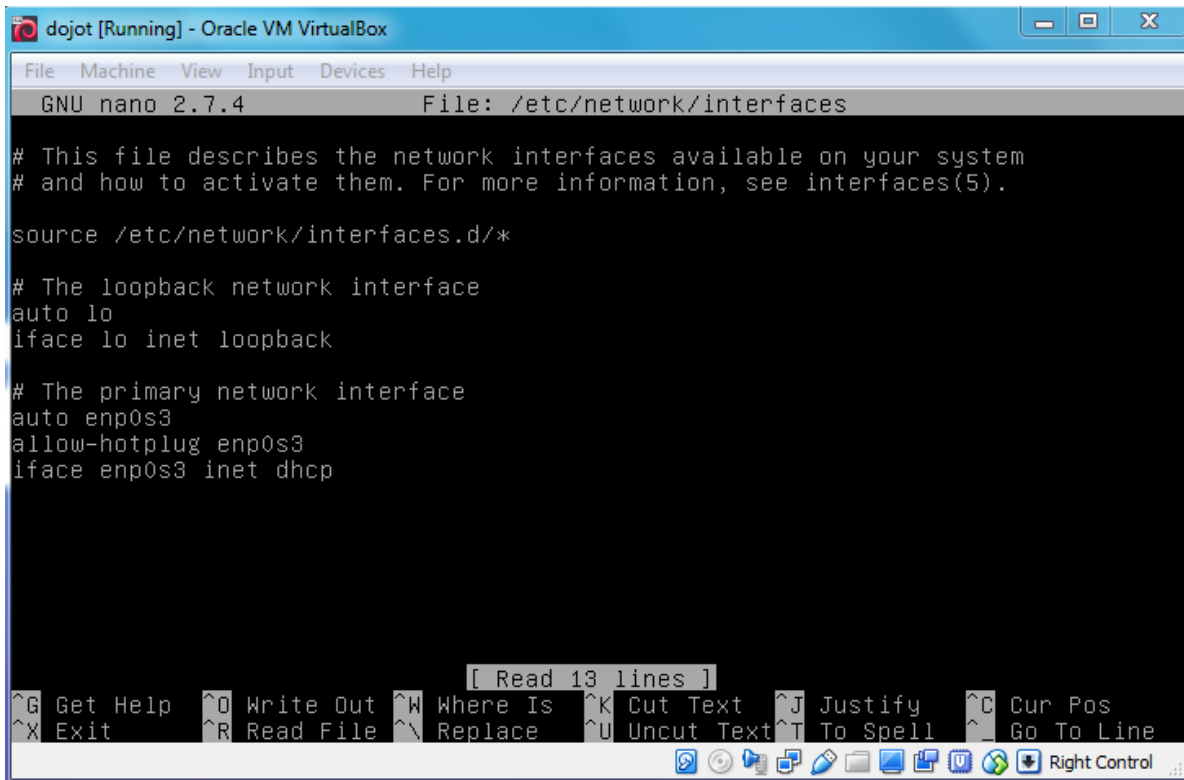
Login in the virtual machine (login/password are `dojot/dojot`) to set the network interface. Firstly, get the interface name:

```
$ sudo ip address | grep -B1 -i "<YOUR MAC ADDRESS>" | head -1 | cut -d':' -f2
```



Edit the file `/etc/network/interfaces`, adding

```
# The primary network interface
auto <YOUR INTERFACE NAME>
allow-hotplug <YOUR INTERFACE NAME>
iface <YOUR INTERFACE NAME> inet dhcp
```



```
dojot [Running] - Oracle VM VirtualBox
File  Machine  View  Input  Devices  Help
GNU nano 2.7.4      File: /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto enp0s3
allow-hotplug enp0s3
iface enp0s3 inet dhcp

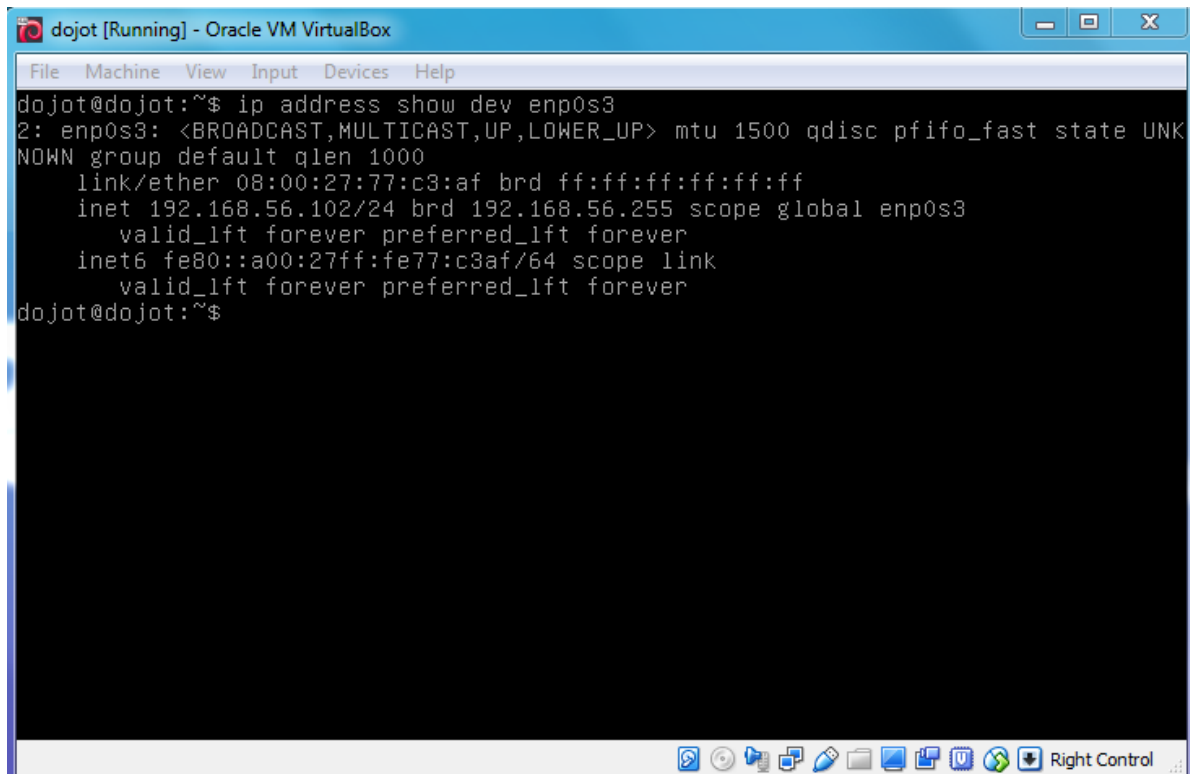
[ Read 13 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
Right Control
```

Restart the networking service:

```
$ systemctl restart networking.service
```

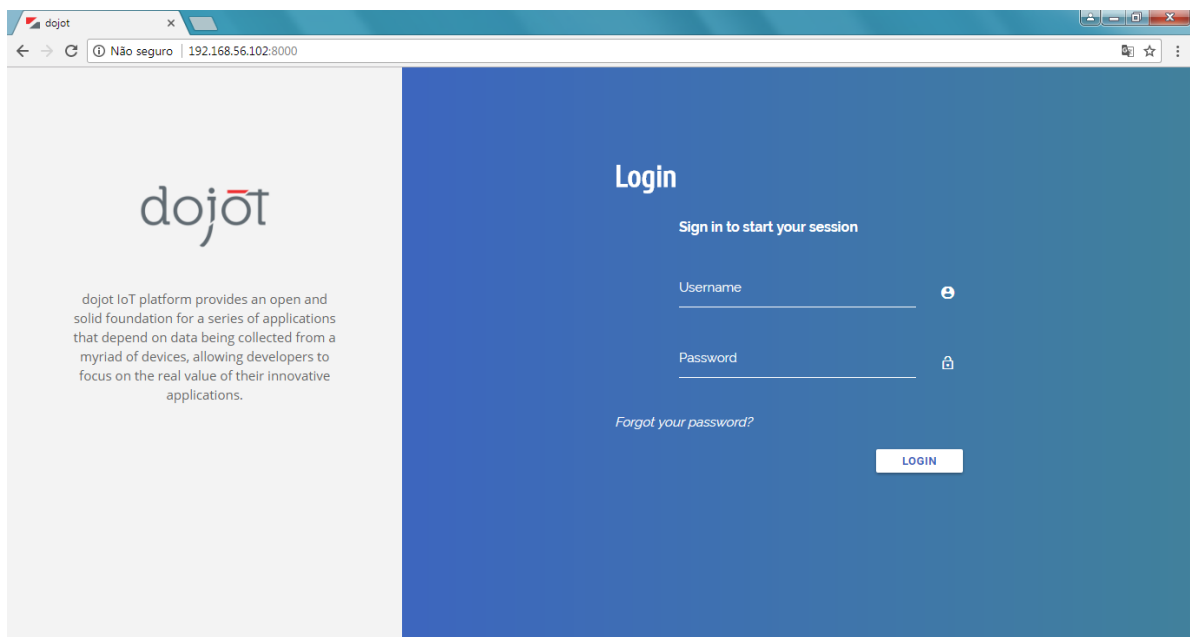
Get the ip address assigned to the interface:

```
ip address show dev <YOUR INTERFACE NAME>
```



```
dojot@dojot:~$ ip address show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNK
NOWN group default qlen 1000
    link/ether 08:00:27:77:c3:af brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global enp0s3
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe77:c3af/64 scope link
        valid_lft forever preferred_lft forever
dojot@dojot:~$
```

In the host machine, open a browser and type <YOUR IP ADDRESS>:8000.



Frequently Asked Questions

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

Got a question that isn't answered here? Please, open an issue on [dojot's Github repository](#).

Table of Contents

- *General*
 - *What is dojot? Why should I use it? Why open source it?*
 - *Where can I get it?*
 - *Which repository is the main one?*
 - *So, I found this pesky bug. How can I inform you about it?*
- *Usage*
 - *How do I start it? Is it CLI-based or it has a graphical user interface?*
 - *Ok, I started it and I logged in. Now what?*
 - *How can I update my deploy to dojot's latest version?*
- *Devices*
 - *What are devices for dojot?*
 - *What is the relationship between this device and my actual device?*
 - *What are virtual devices? How are they different from the other one?*
 - *How can I send MQTT data to dojot so that it appears on the dashboard?*
 - *On the dashboard some attributes are shown as tables and others as charts. How are they chosen/set?*
 - *I'm interested in integrating my super cool device with dojot. How can I do it?*
 - *Is there any restrictions about the message my device will send to dojot? Format, size, frequency?*

- *How can I send some commands to my device through dojot?*
- *I didn't find the protocol supported by my device in the type list, is there anything I can do?*
- *I saved an attribute, but it disappeared from the device. Is it a bug?*
- *Data Flows*
 - *What is data flow?*
 - *The data flow UI... really looks like node-RED. Are they related in some way?*
 - *Why should I use it?*
 - *What can it do, exactly?*
 - *So, how can I use it?*
 - *Can I apply the same flow to multiple devices?*
 - *Can I correlate data from different devices in the same flow?*
 - *I want to send an email, what should I do?*
 - *What about a HTTP POST request, how can I send it?*
 - *I want to rename the attributes of a device, what should I do?*
 - *I want to aggregate the attributes of multiple devices, what should I do?*
 - *It would be cool a WhatsApp node, is it in roadmap?*
- *Applications*
 - *What APIs are available for applications?*
 - *How can I use them?*
 - *I'm interested in integrate my application with dojot. How can I do it?*

6.1 General

6.1.1 What is dojot? Why should I use it? Why open source it?

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:

- Open APIs which makes the access to the platform resources easy.
- Capacity to store large volumes of data in different formats.
- Connectors to different types of devices.
- Graphical user interface with flow builder to prototype IoT solutions very quickly.
- Real time event processing with customizable rules.

dojot is based on Fiware, also an open source project, compromised to build an open and sustainable ecosystem grounded on open standards with the aim of easing the application development in different segments.

6.1.2 Where can I get it?

All components are available in dojot's GitHub repositories: <https://github.com/dojot>.

6.1.3 Which repository is the main one?

There are two main ones:

- <https://github.com/dojot/dojot>: this is where we keep track of all the things related to this project as a whole, such as architectural enhancements.
- <https://github.com/dojot/docker-compose>: repository for Docker compose files and configurations. This is what we would recommend to use to start with.

6.1.4 So, I found this pesky bug. How can I inform you about it?

We ask you to open an issue in [dojot's Github repository](#). If you know exactly which component is failing, you could open the issue in its repository (it will work the same way).

If you are able to analyze and fix this bug, please do so. Create a pull-request with a quick description of what you've done.

6.2 Usage

6.2.1 How do I start it? Is it CLI-based or it has a graphical user interface?

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, there are a few steps to start it up:

```
$ docker-compose up -d
$ ./kong-config.sh
$ ./create-user.sh
```

And that's it.

The web interface is available at `http://localhost:8000`. The user is `admin`, password `admin`.

REST APIs are explained in the [Applications](#) section.

6.2.2 Ok, I started it and I logged in. Now what?

Nice! Now you can add your first devices, described in [Devices](#), build some flows and subscribing to device events, both described in [Data Flows](#).

6.2.3 How can I update my deploy to dojot's latest version?

You need to follow some steps:

1. Update the docker-compose repository to the latest version.

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

2. Deploy the latest docker images.

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

This procedure also applies to the available virtual machines once they do use docker-compose.

6.3 Devices

6.3.1 What are *devices* for dojot?

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.

Consider, for instance, an actual device with temperature and humidity sensors; it can be represented into 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 actual ones, 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*.

6.3.2 What is the relationship between this *device* and my actual device?

It is 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. If you don't want some attributes to be available to applications or other elements, just don't add them in dojot.

6.3.3 What are *virtual devices*? How are they different from the other one?

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*? So, 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 (they won't accept messages addressed to them via MQTT, for example).

6.3.4 How can I send MQTT data to dojot so that it appears on the dashboard?

First of all, you create a digital representation for your actual device. Then, you configure it to send data to dojot so that it matches its digital representation.

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. Here, you need to follow some rules:

- MQTT topic must follow the pattern `/<service-id>/<device-id>/attrs`, where `<service-id>` is an identifier associated with the user account and the `<device-id>` is a unique identifier assigned by dojot. For example, topic `/admin/882d/attrs` must be used for user `admin` and device ID `882d`.
- MQTT payload must be a JSON with each key being an attribute of the dojot device, such as:

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

It's worth to point out that we are relaxing these rules so that you'll have more flexibility to configure both topic and payload. This feature will be available in the next official release.

6.3.5 On the dashboard some attributes are shown as tables and others as charts. How are they chosen/set?

The type of an attribute determines how the data is shown on the dashboard as follows:

- Geo: geo map.
- Boolean and Text: table.
- Integer and Float: line chart.

6.3.6 I'm interested in integrating my super cool device with dojot. How can I do it?

If your device is able to send messages using MQTT (with JSON payload), CoAP or HTTP, there is a good chance that your device can be integrated with minor or no modifications whatsoever. The requirements for such integration is described in the question *How can I send MQTT data to dojot so that it appears on the dashboard?*.

6.3.7 Is there any restrictions about the message my device will send to dojot? Format, size, frequency?

None but format, which is described in the question *How can I send MQTT data to dojot so that it appears on the dashboard?*.

6.3.8 How can I send some commands to my device through dojot?

This feature is not supported right now, but it is in roadmap and will be available in the next official release. If you are craving for this feature, please help us to develop it.

6.3.9 I didn't find the protocol supported by my device in the type list, is there anything I can do?

There are some possibilities. The first one is to develop a proxy to translate your protocol to one supported by dojot. The second one is to develop a connector similar to the existing ones for MQTT, CoAP and HTTP.

6.3.10 I saved an attribute, but it disappeared from the device. Is it a bug?

You might have saved the attribute, but not the device. If you don't click on the save button for the device, the added attributes will be discarded. We're improving the system messages to caveat the users and remember them to save their configurations.

6.4 Data Flows

6.4.1 What is data flow?

It's a processing flow for income messages/data of a device. With a flow you can dynamically analyse each new message in order to apply validations, infer information, and trigger actions or notifications.

6.4.2 The data flow UI... really looks like node-RED. Are they related in some way?

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

6.4.3 Why should I use it?

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

6.4.4 What can it do, exactly?

You can do things such as:

- Create virtual viewers of a device (rename attributes, aggregate attributes, change values, etc).
- Infer information based on switch rules.
- Infer information based on edge-detection rules.
- Infer information based on geo-fence rules.
- Notify through email.
- Notify through HTTP.

The data flows component is in constantly development with new features being added every new release.

6.4.5 So, how can I use it?

It follows the basic usage flow as node-RED. You can check its [documentation](#) for more details about this.

6.4.6 Can I apply the same flow to multiple devices?

Multiple devices can be used both as input and output of data flows. It's worth to point out that the flow is processed individually for each new input message, i.e. for each input device.

6.4.7 Can I correlate data from different devices in the same flow?

As the data flow is processed individually for each message, you need to create a virtual device to aggregate all attributes, then use this virtual device as the input of the flow.

6.4.8 I want to send an email, what should I do?

Basically, you need to add an email node and configure it. This node is pre-configured to use the Gmail server `gmail-smtp-in.l.google.com`, but you're free to choose your own. For writing an email body, you can use a template before the email.

It is important to point out that dojot contains no e-mail server. It will generate SMTP commands and send them to the specified e-mail server.

6.4.9 What about a HTTP POST request, how can I send it?

It is almost the same process as sending an e-mail.

One important note: make sure that dojot can access your server.

6.4.10 I want to rename the attributes of a device, what should I do?

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.

6.4.11 I want to aggregate the attributes of multiple devices, what should I do?

First of all, you need to create a virtual device to aggregate all attributes, then you build a data flow to map the attributes of each device to the virtual one. This can be done connecting a 'change' node after each input device to put the input values into an output, and finally connecting all changes to the virtual device and assigning to it the output.

6.4.12 It would be cool a WhatsApp node, is it in roadmap?

It's under analysis. We intend to support other notifications systems besides email, including WhatsApp, Twitter and Telegram. If you also have interest, please help us to develop them.

6.5 Applications

6.5.1 What APIs are available for applications?

- Authorization and user management
- Device management
- Subscriptions
- Flow management
- History

6.5.2 How can I use them?

First, you will need an access token, which can be retrieved sending a HTTP POST request to `/auth` endpoint with the following JSON content:

```
{  "username" : <>,  "passwd" : <> }
```

Obviously the values of each attribute should be correctly filled in. An example of such request using `curl` would be:

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

which gives us back:

```
{"jwt": "eyJhbGciOiJIUzI1..."}
```

This token (which is a lengthy alpha-numeric string) should be used in every request that is sent to dojot (excluding, of course this request). Each call for this API will generate a different token.

This token should be placed in a Authorization HTTP header, such as:

```
$ curl -X GET http://localhost:8000/device -H 'Authorization: Bearer eyJhbGciOiJIUzI1.
↪...'
```

A few endpoints requires two more headers, the Fiware-Service and Fiware-ServicePath. They are: /metrics/, /iot/ and /history/

Fiware-Service header should contain the service name associated to the user. In general, it should be the username. Fiware-ServicePath is always a forward slash (/). An example:

```
curl -X GET http://localhost:8000/metric/v2/entities -H 'Authorization: Bearer_
↪eyJhbGciOiJIUzI1...' \
-H 'Fiware-Service:admin' -H 'Fiware-ServicePath:/'
```

6.5.3 I'm interested in integrate my application with dojot. How can I do it?

This should be pretty straightforward. There are two ways that your application could be integrated with dojot:

- **Retrieving historical data:** you might want to periodically read all historical data related to a device. This can be done by using this API (one side-note: all endpoints described in this apiary should be preceded by /history/).
- **Subscribing to events related to devices:** if your application is able to listen to events, you might rather use subscriptions, which can be created using this API (also, all endpoints should be preceded by /metrics/).
- **Using mashup to pre-process data:** if you want to do something more, you could use flows. They can help process and transform data so that they can be properly sent to your application via HTTP request, by e-mail or stored in a virtual device (which can be used to generate notifications as previously described).

All these endpoints should bear an access token, which is retrieved as described in the question *How can I use them?*.