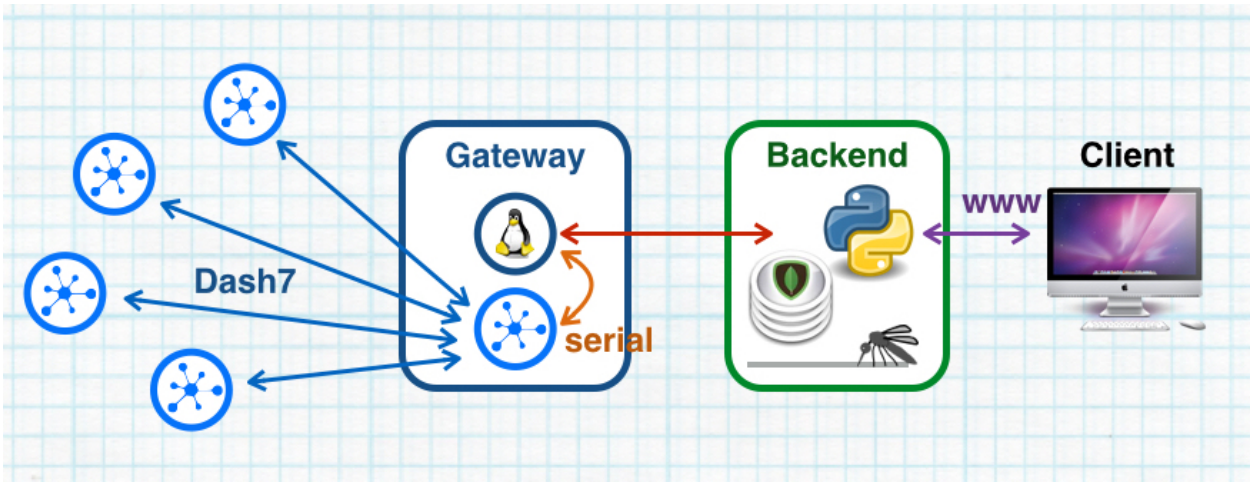
README

Dash7 BackEnd

Analysis and implementation of a generic data-gathering and command-proxying backend for a Dash7 network
Christophe VG (contact@christophe.vg)

Introduction

The overall architecture is depicted in the following diagram:



Backend Overview

A `Gateway` exposes a Dash7 network to a `Backend`, which exposes the data of the network to a web-based `Client`. The `Gateway` uses a serial connection to allow a `Linux` host to act as a bridge between the Dash7 network and a TCP/IP network.

Assumptions

1. The `Gateway` implements the physical serial connection between the local Dash7 node and the `Linux` host.
2. The Dash7 node mirrors all network communication (bi-directionally) to its serial port

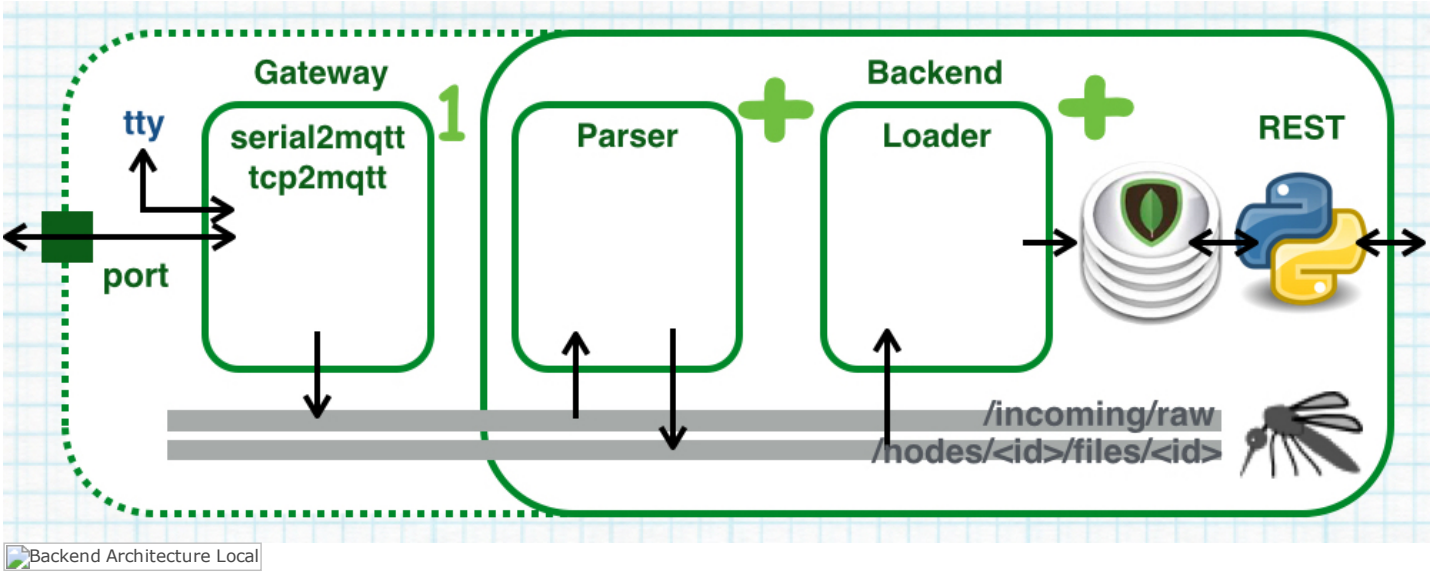
Requirements

1. Expose serial connection on a TCP/IP network
2. Connection between `Gateway` and `Backend`
3. Accept all file changes in the Dash7 network and build a data history
4. Propagate queries into the the Dash7 network
5. Provide user management for `Clients`
6. Provide entity management for `Gateways` and Dash7 networks

Backend Architecture

The `Backend` is constructed from a collection of loosely coupled components, each performing one specific task, allowing for reconfiguration of the entire backend over time and/or in different situations.

As a communication back-bone, an MQTT broker is proposed. The following diagram gives an overview of this architecture, implemented all within the `Backend`.



Backend Architecture Local

The MQTT back-bone also allows for the distribution of the different components, as shown in diagram, with the serial2mqtt implemented on the `Gateway`, connected directly to the serial device.

Serial2MQTT and TCP2MQTT

`serial2mqtt` and `tcp2mqtt` are two small components that performs two tasks:

1. monitor a TCP port or a serial device and post all messages received via this link to a topic on the MQTT broker, e.g. `/incoming/raw`.
2. monitor an MQTT topic for messages that should be send to the TCP-port/serial device (not depicted in the diagrams above)

A tool like `remserial` (see: http://lpccomp.bc.ca/remserial/) can also be used to implement a bridge from the serial port on the `Gateway` to a TCP port.

Parser

The `Parser` subscribes to e.g. the `/incoming/raw` topic and translates the byte oriented messages to a structured representation. These translations are then published to e.g. a `/nodes/<id>/files/<id>`-structured topic, for further processing.

Loader

The `Loader` subscribes to e.g. the `/nodes/<id>/files/<id>` topic and imports all structured messages into a datastore.

MongoDB

For the datastore, MongoDB is proposed. It fits the very flat data structure and the document-orientation of files and the variability of schemas.

REST API

To expose the gateways/nodes/files data a REST API is provided. During initial development, a Python implementation on top of web.py is used, but any other language or framework that can access MongoDB can be chosen. The implementation is currently so light, that any choice is valid.

Executor

Another example of a possible component in this processing stack is that of the `Executor`. An executor could be responsible for accepting commands that need to be executed and publish it to the right channel, where e.g. a `Encoder` first picks it up and converts it to the byte oriented message format, before it is picked up by one of the `serial2mqtt` or `tcp2mqtt` bridges to actually send it to the network.

On the Multiplicity of Components

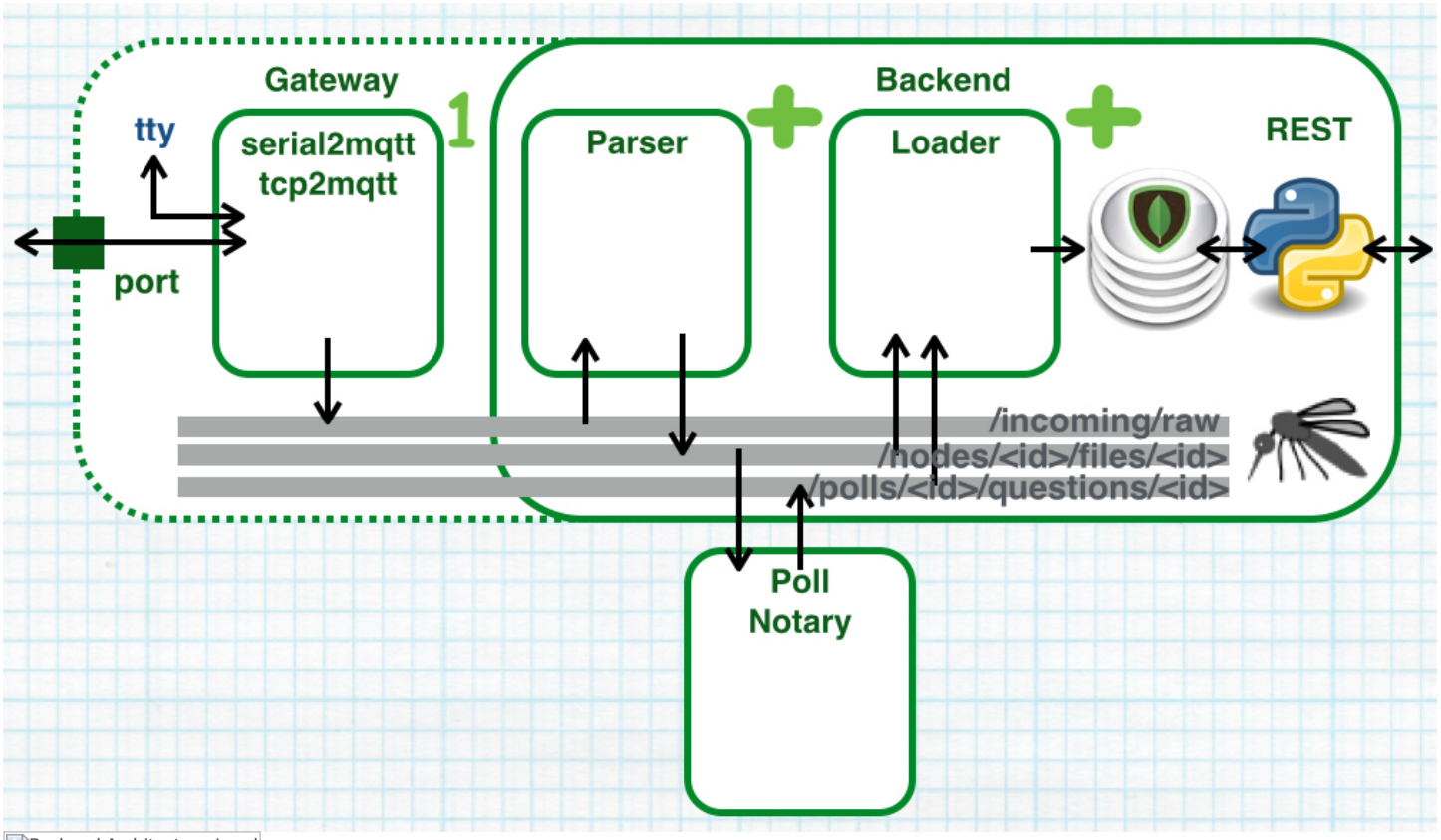
As indicated in the diagrams above, abstract components can be instantiated multiple times, meaning that different implementation of the same type of component can co-exist on the MQTT back-bone. This allows for splitting up functionally different parts of the entire solution, allowing new setups to be assembled, rather than (re)coded.

Regarding Topics

The MQTT protocol provides topics, aka channels, to which parties can publish and subscribe. Given a topic-scheme which includes all known information e.g. `/gateways/1/nodes/2/files/1`, one can construct subscriptions at different levels, allowing for different ways to process the data. Using wildcards, one component could subscribe e.g. to all messages from gateway 1 using `gateway/1/node/+/files/+`.

Case-Study: MOSAIC Conference Badge

Applications can now be build on top of this backend framework. If we for example want to implement a polls manager, that accumulates the votes of conference attendees, pressing buttons, we now simply can create a processor that listens for updates to files that are assigned for tracking the votes of badge-carriers.



Backend Architecture Local

Listening for `/gateways/+/nodes/+/file/123` might present the `Poll Notary` with all updates to the file used for answering question 1 in the second poll. The `Poll Notary` now simply has to map this to a new message that it publishes to `/polls/2/questions/1/yes`. The `Loader` will pick up this message and process it, updating the collection of polls, incrementing the yes votes for question 1 of the second poll.

Initial Evaluation

Selecting an MQTT broker as a back-bone, even for local IPC, incurs an overhead. A basic test implementation using Mosquitto (see: http://mosquitto.org) as a broker and Python (2.7.10) with the paho-mqtt module (see: https://pypi.python.org/pypi/paho-mqtt) provides the following information:

1. generating 10.000 messages at the `Serial Daemon`, and injecting them into the `incoming/raw` topic takes about 2.75s (on an old MacBook), thus reaching about 3600 msg/s
2. the first message reaches the next step, the `Translator` after 7.5ms and the `Loader` after another 6.5ms
3. the `Translator` processes the 10.000 messages in 4.8s or 2100 msg/s, the `Loader` takes 4.95s or 2000 msg/s
4. the entire process, from the first generated message in the `Serial Daemon` to the last loaded message in the `Loader`, takes about 4.97s or 2000 msg/s, which is of course roughly equal to the performance of the `Loader`, who receives its first message after 14ms and is the last to finish.

Given an example of 80.000 nodes, sending 1 message every 5 minutes, which is received by 3 gateways, the required performance boils down to 800 msg/s. The

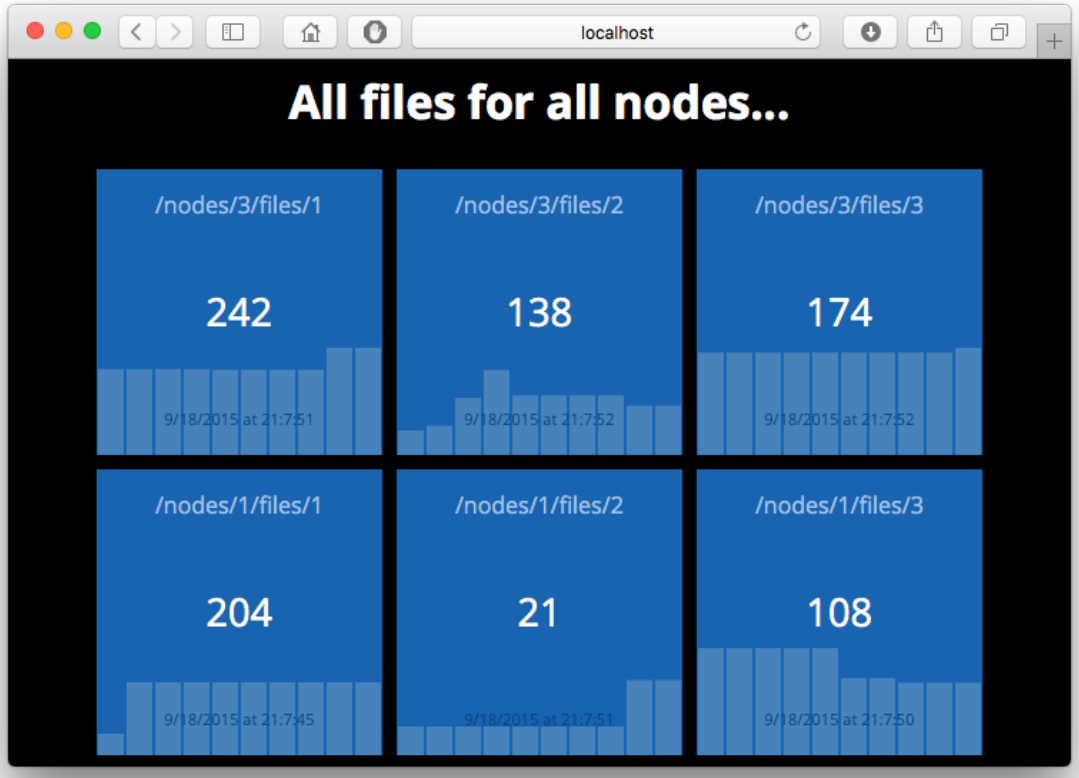
introduction of an MQTT broker seems affordable.

Front-ends

Given the REST API front-end can be implemented using a variety of technologies. For the initial development, basic HTML and Javascript are used to query the REST API and render a visualisation for the data.

Dashboard

A first demonstration of the ease of implementation is the dashboard:



The script driving it, queries the REST API for all nodes available, then queries all nodes for their files and creates widgets with the files' REST path, the current value, an update timestamp and a barchart presenting the last 10 values.

Installation

Prepare System

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Dependencies

The initial REST interface is build using `web.py`:

```
$ sudo pip install web.py
```

We use MQTT as a service-bus and currently use `mosquitto`.

To connect to MQTT we use `paho-mqtt`:

```
$ sudo pip install paho-mqtt
```

We use MongoDB as a datastore.

To connect to MongoDB we use `pymongo`:

```
$ sudo pip install pymongo
```

As an alternative to MongoDB, MySQL is also supported:

```
$ sudo pip install sqlalchemy
```

Backend - System setup

Backend User

Create a user `backend`:

```
$ sudo adduser --home /opt/backend --shell /bin/bash backend
$ sudo usermod -a -G dialout backend
```

Prepare Syslog

```
$ sudo echo "*.=alert    /opt/backend/logs/d7a-backend.log" >> /etc/rsyslog.conf
$ sudo service rsyslog restart
```

Backend Code

Clone the Backend repository and initialise it:

```
$ sudo -i -u backend
$ git clone http://redmine.cosys.be/git/dash7-backend
$ cd dash7-backend
$ make
*** initializing and updating submodules
```

Preparing a remote backend-end-point using Serial2MQTT

`Serial2MQTT` and `Serial2REST` are two scripts that enable bridging serial messages from the gateway to a central backend server. Currently one central server exists at IP `146.175.138.145`.

Before proceeding with this part, make sure that MQTT is accessible! Test connectivity and determine actual configuration parameters:

```
$ sudo -i -u backend
$ cd dash7-backend/backend
$ ./serial2mqtt.py -vp -b 146.175.138.145 -s /dev/ttyS2
connected to 146.175.138.145
subscribed to /outgoing
connected to /dev/ttyS2
```

If this command instead raises errors like this:

```
Traceback (most recent call last):
  File "./serial2mqtt.py", line 67, in <module>
    Serial2MQTT().process()
  File "./serial2mqtt.py", line 19, in __init__
    follow_topics=["/outgoing"]
  File "/Users/xtof/Workspace/UA/projects/dash7-backend/backend/processor.py", line 39, in __init__
    database      = self.config.database if self.database else None
  File "/Users/xtof/Workspace/UA/projects/dash7-backend/backend/client.py", line 29, in __init__
    self.connect_to_backend()
  File "/Users/xtof/Workspace/UA/projects/dash7-backend/backend/client.py", line 40, in connect_to_backend
    self.connect_to_mqtt()
  File "/Users/xtof/Workspace/UA/projects/dash7-backend/backend/client.py", line 58, in connect_to_mqtt
    self.mq.connect(self.mqtt_broker, 1883, 60)
  File "/Library/Python/2.7/site-packages/paho/mqtt/client.py", line 612, in connect
    return self.reconnect()
  File "/Library/Python/2.7/site-packages/paho/mqtt/client.py", line 734, in reconnect
    sock = socket.create_connection((self._host, self._port), source_address=(self._bind_address, 0))
  File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/socket.py", line 575, in create_connection
    raise err
socket.error: [Errno 61] Connection refused
```

... MQTT isn't accessible and a VPN must be set up OR you should use `Serial2REST` (see below).

If the `serial2mqtt` command succeeds, we can consolidate the configuration and create a local configuration file:

```
$ sudo -i -u backend
$ mkdir ~/etc
$ echo "-v"                       >  ~/etc/serial2mqtt.conf
$ echo "-b"                       >> ~/etc/serial2mqtt.conf
$ echo "146.175.138.145"          >> ~/etc/serial2mqtt.conf
$ echo "-s"                       >> ~/etc/serial2mqtt.conf
$ echo "/dev/ttySX"               >> ~/etc/serial2mqtt.conf
                ^^^
        (replace X with 2 or 3 depending on which serial device)
$ echo "-o"                       >> ~/etc/serial2mqtt.conf
$ echo "/gateways/X/incoming/raw" >> ~/etc/serial2mqtt.conf
                 ^^^
         (replace X with gateway number)
```

Test the complete setup:

```
$ cat /opt/backend/etc/serial2mqtt.conf
-v
-b
146.175.138.145
-s
/dev/ttyS2
-o
/gateways/6/incoming/raw

$ sudo -i -u backend /opt/backend/dash7-backend/src/backend/serial2mqtt.py @/opt/backend/etc/serial2mqtt.conf
```

Alternative if MQTT isn't accessible: `Serial2REST`

To enable bridging messages from the serial connection to the central backend, plain HTTP traffic is also possible, using the exposed REST interface of the backend.

Manual invocation:

```
$ sudo -i -u backend
$ cd dash7-backend/backend
$ ./serial2rest.py -vp -a http://146.175.138.145/api -s /dev/ttyS2
connected to /dev/ttyS2
```

The script only accesses the backend when new data arrives at the serial connection. So, make sure it receives such data. The script reports its activity every 15 seconds (if there was any activity), e.g.:

```
bridged 3 messages
```

The `services.conf` configuration

Each backend requires a configuration file, listing the services that should be handled. On the main backend this (currently) looks like this:

```
backend@dash7-backend:~$ cat /opt/backend/dash7-backend/etc/services.conf
logger
rest
loader
parser
birdtracker
sigfox
```

When using the generic, handle-all services, invocations of the `d7a-backend` script, these services will be handled in this order (in case of stop, the list is reversed).

In case of a remote backend-end-point, this file could look like:

```
backend@dash7-backend:~$ cat /opt/backend/dash7-backend/etc/services.conf
serial2mqtt
```

Enable the backend server functionality

Finally, make sure the backend scripts are started on system (re)boot:

Add the following line to `/etc/rc.local`:

```
sudo -i -u backend /opt/backend/dash7-backend/bin/d7a-backend start
```

Usage

Above the minimal setup and information about connecting a remote backend-end-point to a central backend. This section looks at the usage in more detail:

The `d7a-backend` script

All interaction with the backend can (and **should**) be done using the `d7a-backend` service script:

```
backend@dash7-backend:~$ d7a-backend help
USAGE: /opt/backend/dash7-backend/bin/d7a-backend <service-name> [start|stop|status|restart]
       supported services: logger rest loader parser birdtracker sigfox
USAGE: /opt/backend/dash7-backend/bin/d7a-backend [start|stop|status|restart|view|shell]

backend@dash7-backend:~$ d7a-backend rest status
 * rest is running

backend@dash7-backend:~$ d7a-backend status
 * logger is running
 * rest is running
 * loader is running
 * parser is running
 * birdtracker is running
 * sigfox is running

backend@dash7-backend:~$ d7a-backend rest restart
 * Stopping backend service rest.
 *
 * Starting backend service rest
 *
```

Logging

By default all functionality **does not** log its activity. To enable logging use the `-v` option:

```
backend@dash7-backend:~$ OPTS="-v" d7a-backend loader restart
 * Stopping backend service loader
 *
 * Starting backend service loader
 *
```

Watch logging being added to `/var/log/syslog` on the main/central backend (this should be `/opt/backend/logs/d7a-backend.log`, but for some reason, I can't find a way to configure it apparently?!)

```
root@dash7-backend:/opt/backend/logs# tail -f /var/log/syslog
Nov 17 11:19:07 dash7-backend loader.py: subscribing to /nodes/+/files/+
```

```
Nov 17 11:19:07 dash7-backend loader.py: subscribing to /birds/+
Nov 17 11:19:07 dash7-backend loader.py: subscribing to /signs/+
Nov 17 11:19:07 dash7-backend loader.py: connected to localhost with result code 0
Nov 17 11:19:41 dash7-backend loader.py: updating nodes 2653890122173749143 files 64
Nov 17 11:19:41 dash7-backend loader.py: updating birds 16758521300220227876 None None
Nov 17 11:19:41 dash7-backend loader.py: updating nodes 2633733879281649657 files 64
Nov 17 11:19:41 dash7-backend loader.py: updating birds 16758521300220227876 None None
Nov 17 11:19:41 dash7-backend loader.py: updating nodes 2633733900783163298 files 64
Nov 17 11:19:41 dash7-backend loader.py: updating birds 16758521300220227876 None None
Nov 17 11:19:41 dash7-backend loader.py: updating nodes 2633117044644197676 files 64
Nov 17 11:19:41 dash7-backend loader.py: updating birds 16758521300220227876 None None
```

To turn logging back off, restart without the `-v` switch.

```
backend@dash7-backend:~$ d7a-backend loader restart
 * Stopping backend service loader
 *
 * Starting backend service loader
 *
```

Of course, the switch can also be used when (re)starting all services.
WARNING This creates **a lot** of logging! Use with care and don't leave it running by default.

The Viewer

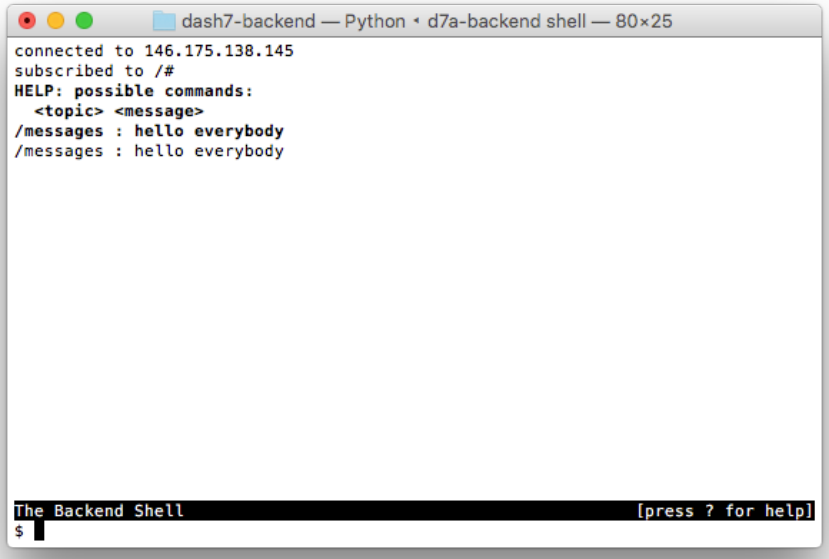
A special command that can be passed to the `d7a-backend` script is `view`:

```
backend@dash7-backend:~$ d7a-backend view
subscribing to /#
connected to localhost with result code 0
FILE UPDATE 2633733879281649657 64 [201, 43, 0, 0, 0, 0, 0, 0, 52, 80, 1, 171]
FILE UPDATE 2653890122173749143 64 [201, 43, 0, 0, 0, 0, 0, 0, 57, 80, 1, 171]
/birds/14495679825622073344 : {"battery": 3.36, "rssi": -52, "counter": 171, "subcontroller_id": "2633733879281649657"}
/birds/14495679825622073344 : {"battery": 3.36, "rssi": -57, "counter": 171, "subcontroller_id": "2653890122173749143"}
^Creceived KeyboardInterrupt... stopping processing
```

It displays all messages that are sent on the MQTT broker backbone. The is most of the time a good starting point for debugging.

The Shell

An extended implementation of the `viewer` is the shell. Starting it using `d7a-backend shell` opens up an ncurses-based terminal application:



Backend Architecture Local

The screen is split in two parts, separated by a black bar. Above the bar, the `viewer` functionality still logs all activity as seen on the subscribed topics in the MQTT broker. Below the bar is a command prompt accepting commands to be executed.

Initially the possible commands consists of a `topic` and some `text` terminated by a carriage return. The `text` is published to the corresponding `topic` on the broker, allowing to manually introduce messages in the system.

Commands are typically presented in the accumulating logging viewer in a bold face to indicate the activity of the user. They will of course also typically show up in the normal logging in the viewer as the message is also dispatched to the viewer.

The Loader

One script is in charge of loading records into the datastore. This datastore is by default MongoDB. But for the `Bondis` project, a minimal, transparant MySQL implementation was started.

To use the MySQL engine, use the following arguments to `loader.py`:

```
-e mysql
-s backend:backend@localhost
```

CAUTION !! - Currently only the `_historic` table is populated with records (aka. only `insert_one` is supported). An example:

```
mysql> describe bondis;
+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| id     | bigint(20) | YES  |     | NULL    |       |
| ts     | datetime   | YES  |     | NULL    |       |
| status | bigint(20) | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> describe bondis_historic;
+-----------+------------+------+-----+---------+-------+
| Field     | Type       | Null | Key | Default | Extra |
+-----------+------------+------+-----+---------+-------+
| bondis_id | bigint(20) | YES  |     | NULL    |       |
| ts        | datetime   | YES  |     | NULL    |       |
| status    | bigint(20) | YES  |     | NULL    |       |
+-----------+------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

```
$ ./loader.py -vp -e mysql -s backend:backend@localhost
connecting to MySQL instance mysql://backend:backend@localhost/d7db
connected to localhost
subscribed to /nodes/+/files/+
subscribed to /birds/+
subscribed to /signs/+
subscribed to /opinions/+
subscribed to /bondis/+
```

```
$ d7a-backend shell
$ /bondis/372829 { "status": 93848754 }
```

```
mysql> select * from bondis;
Empty set (0.00 sec)

mysql> select * from bondis_historic;
+-----------+---------------------+----------+
| bondis_id | ts                  | status   |
+-----------+---------------------+----------+
|    372829 | 2016-01-06 09:54:57 | 93848754 |
+-----------+---------------------+----------+
1 row in set (0.00 sec)
```

Using the default MongoDB engine ...

```
$ ./loader.py -vp
connected to localhost
subscribed to /nodes/+/files/+
subscribed to /birds/+
subscribed to /signs/+
subscribed to /opinions/+
subscribed to /bondis/+
```

```
$ d7a-backend shell
$ /bondis/372829 { "status": 93848754 }
```

```
$ mongo d7db
MongoDB shell version: 3.0.5
connecting to: d7db
> db.bondis.find().pretty()
{
    "_id" : "372829",
    "status" : 93848754,
    "ts" : ISODate("2016-01-06T10:01:00.002Z")
}
> db.bondis_historic.find().pretty()
{
    "_id" : ObjectId("568cd7ccd48d004043cb106d"),
    "status" : 93848754,
    "bondis_id" : "372829",
    "ts" : ISODate("2016-01-06T10:01:00.002Z")
}
```

root @ master

Naam	Grootte
backend	
bin	
etc	
lib	
media	
notes	
src	

.gitignore	40 Bytes
.gitmodules	384 Bytes
Makefile	855 Bytes
README.md	20.055 KB

Meest recente revisies

#			Datum	Auteur	Commentaar
e11d5bb9	◉		05-02-2016 09:40	Glenn Ergeerts	updated pyd7a external
f78d2f94	○	◉	08-01-2016 14:39	Christophe VG	added support for missing syslog (e.g. on Windows)
3ec06dd6	○	○	06-01-2016 10:04	Christophe VG	added preliminary support for MySQL db engine
af55b4e1	○	○	26-12-2015 22:37	Christophe VG	in python the extra hour is wrong ;-)
cd5ac6c6	○	○	26-12-2015 21:58	Christophe VG	improved logging and changed default interval
50d74ee6	○	○	26-12-2015 21:47	Christophe VG	javascript -> eval'ed python in validator queries
0416994d	○	○	26-12-2015 21:03	Christophe VG	ts entries are 1 hour ahead of ISODate/time
061a027c	○	○	26-12-2015 18:32	Christophe VG	added validator processor
21a92bdf	○	○	26-12-2015 18:32	Christophe VG	fixed issue when not following any topics
7f37552c		○	26-12-2015 17:26	Christophe VG	added a notifier processor (slack support for now)

Bekijk verschillen

Bekijk alle revisies | Bekijk revisies