# Chariot Privacy Engine Documentation

***Release 0.5.10***

**George Theofilis**

**Mar 05, 2019**

# Contents:

CHAPTER 1

# Chariot Base

Base utilities for chariot micro-services.

## 1.1 Development

Encrypt application secrets, with the folling command

```
$ gpg -c --batch --passphrase test config.json
```

## 1.2 Features

- Connection to Influx DB
- Connection to MQTT broker
- Connection to Cloudant
- Connection to IBM Watson IoT service

## 1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

# Installation

## 2.1 Stable release

To install Chariot Privacy Engine, run this command in your terminal:

```
$ pip install chariot_base
```

This is the preferred method to install Chariot Base, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for Chariot Privacy Engine can be downloaded from the Gitlab repo.

You can either clone the public repository:

```
$ git clone git@gitlab.com:chariot-h2020/chariot_base.git
```

Or download the tarball:

```
$ curl  -OL https://gitlab.com/chariot-h2020/chariot_base/-/archive/master/chariot_
↪base-master.zip
```

Then install requirements for a Debian like system, with the following command:

```
$ sudo apt-get install python-dev libgmp3-dev
```

Once you have a copy of the source and requirement are installed, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

# Usage

To use Chariot Privacy Engine in a project:

```
import chariot_base
```

CHAPTER 4

# Modules

**class** chariot_base.connector.local.**LocalConnector**
All subscribers/publisers at Chariot project should extend this class

**clear**()
Reinitialize the connector

**close_span**(*span*)
Close a logging span

**Parameters** **span** – Span to close

**inject_tracer**(*tracer*)
Inject an opentracing client

**Parameters** **tracer** – the opentracing client

**on_connect**(*client*, *flags*, *rc*, *properties*)
The handler run when the connections is established

**Parameters**

- **client** – the subscribed MQTT client
- **flags** –
- **rc** –
- **properties** – Custom properties

**on_disconnect**(*client*, *packet*)
The handler run when the connections is fineshed

**Parameters**

- **client** – the subscribed MQTT client
- **packet** –

**on_message**(*client*, *topic*, *payload*, *qos*, *properties*)
Handler for new message

**Parameters**

- **client** – the subscribed MQTT client
- **topic** – the MQTT topic
- **payload** – the message
- **qos** – MQTT broker quality of service
- **properties** – Custom properties

**on_subscribe**(*client*, *mid*, *qos*)
The handler run when the client subscribed to a new topic

**Parameters**

- **client** – the subscribed MQTT client
- **mid** –
- **qos** – MQTT broker quality of service

**publish**(*topic*, *msg*, *qos=1*)
Publish to a topic

**Parameters**

- **msg** – the published message
- **qos** – MQTT broker quality of service

**register_for_client**(*client*)
Register handlers to the client.

**Parameters** **client** – Client to register handlers

**set_up_tracer**(*options*)
Configure a new opentracing client

**Parameters** **options** – options to configure the new client

**start_span**(*span_id*, *child_span=None*)
Start a new logging span

**Parameters**

- **span_id** – identifier of a new logging span
- **child_span** – parent span

**subscribe**(*topic*, *qos=1*)
Subscribe to a topic

**Parameters**

- **topic** – Which topic the subscriber should listen for new message.
- **qos** – MQTT broker quality of service

chariot_base.connector.local.**create_client**(*options*, *postfix='_client'*)
Create a new GMQTT client

**Parameters** **options** – Options for client initialization

**Para postfix** Unique postfix for the client

**class** chariot_base.connector.watson.**WatsonConnector**(*options*)

**publish**(*point*)

**Parameters** **point** – A point represent received message.

**Returns** True if event publish is successfull.

**class** chariot_base.model.alert.**Alert**(*name*, *msg=None*, *severity=100*)
Describe each alert raised by Chariot

**dict**()
Converts alert object to dictionary

**class** chariot_base.model.point.**DataPointFactory**(*db*, *table*)
Converts to a new point

**from_json_string**(*msg*)
From JSON message payload

**from_mqtt_message**(*message*)
From mosquitto message payload

**class** chariot_base.model.message.**Message**(*sensor_id*, *value*)
Describe message passed between Chariot's components

**class** chariot_base.model.subscriber.**Subscriber**(*subscriber_id*)
Chariot gateway subscriber

**class** chariot_base.tests.utils.**Callbacks**

**on_message**(*client*, *topic*, *payload*, *qos*, *properties*)
Handler for new message

**Parameters**

- **client** – the subscribed MQTT client
- **topic** – the MQTT topic
- **payload** – the message
- **qos** – MQTT broker quality of service
- **properties** – Custom properties

**on_subscribe**(*client*, *mid*, *qos*)
The handler run when the client subscribed to a new topic

**Parameters**

- **client** – the subscribed MQTT client
- **mid** –
- **qos** – MQTT broker quality of service

CHAPTER 5

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://gitlab.com/chariot-h2020/chariot_base/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitLab issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitLab issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 5.1.4 Write Documentation

Chariot Base could always use more documentation, whether as part of the official Chariot Base docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://gitlab.com/chariot-h2020/chariot_base/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *chariot_base* for local development.

1. Fork the *chariot_base* repo on GitLab.
2. Clone your fork locally:

```
$ git clone git@gitlab.com:chariot-h2020/chariot_base.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv chariot_base
$ cd chariot_base/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 chariot_base tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitLab:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitLab website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

## 5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_chariot_base
```

## 5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

GitLab will then deploy to PyPI if tests pass.

CHAPTER 6

# Changelog

## 6.1 0.5.3 (2019-02-25)

- Add IoTLWrapper

## 6.2 0.5.0 (2019-02-19)

- Each data point gets a random guid.
- Add support for distributed tracing.
- Read JSON configuration.
- Migrate to gmqtt, drop use of python-mqtt.

## 6.3 0.2.3 (2018-11-12)

- Pass connection options to cloudant data storage
- Pass connection options to watson connector
- Modify how I write to the local storage
- Add test for data point
- Add test for sign & verify
- Custom (Integrated Encryption Scheme) IES encrypt/decrypt
- Bitwise permission checking

## 6.4 0.1.10 (2018-10-01)

- Add connector
- Update model
- Add various data source utilities

## 6.5 0.1.1 (2018-10-01)

- Add common model files

## 6.6 0.1.0 (2018-09-28)

- First release on GitLab.

CHAPTER 7

# Credits

## 7.1 Development Lead

- George Theofilis <g.theofilis@clmsuk.com>

## 7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## C

# Index

## W