ConsenSys-Utils

Release 0.2.0-dev

Aug 09, 2018

Contents

Welcome to ConsenSys-Utils's documentation.

CHAPTER 1

User's Guide

1.1 Guide

1.1.1 About ConsenSys-Utils

ConsenSys-Utils is a library including a set of utility resources used on a daily basis by ConsenSys France Engineering team.

1.1.2 Create a Flask Application with Factory pattern

Quickstart

ConsenSys-Utils provides multiple features to create a Flask application. In particular ConsenSys-Utils helps you implement the Application factory pattern

1. Create a `app.py`

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup

# Create an application factory
>>> app_factory = FlaskFactory(__name__)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

2. Define an entry point in `setup.py`:

```
from setuptools import setup

setup(
    name='my-app',
```

(continues on next page)

(continued from previous page)

```
    ...,
    entry_points={
        'console_scripts': [
            'my-app=app:cli'
        ],
    },
)
```

3. Install the application and start the application

```
$ pip install -e .
$ my-app run --config config.yml
```

Note that

- `config.yml` is your .yml configuration file
- you don't need to set `FLASK_APP` environment variable
- `run` command reads `FLASK_ENV` environment variable. If `FLASK_ENV=production` the application will be run using a `gunicorn` server otherwise it uses `werkzeug` default development server

Advanced usage

Class *`consensys_utils.flask.FlaskFactory`* allows you to

- *provide a specific yaml configuration loader*
- *provide specifics WSGI middlewares*
- *initialize specifics Flask extensions*
- *set application hooks*
- *register specifics Flask blueprints*

Change configuration loader

By default *`consensys_utils.flask.FlaskFactory`* uses a .yml configuration that validates against `consensys_utils.config.schema.flask.ConfigSchema`. If you like you can define your own configuration loader.

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup
>>> from cfg_loader import ConfigSchema, YamlConfigLoader
>>> from marshmallow import fields

# Declare you configuration schema and config loader
>>> class MySchema(ConfigSchema):
...     my_parameter = fields.Str()

>>> yaml_config_loader = YamlConfigLoader(config_schema=MySchema)

# Create an application factory
>>> app_factory = FlaskFactory(__name__, yaml_config_loader=yaml_config_loader)
```

(continues on next page)

(continued from previous page)

```
# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

Add WSGI Middlewares

You can define your own WSGI middlewares and have it automatically applied on your application

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup
>>> import base64

>>> class AuthMiddleware:
...     def __init__(self, wsgi):
...         self.wsgi = wsgi
...
...     @staticmethod
...     def is_authenticated(header):
...         if not header:
...             return False
...         _, encoded = header.split(None, 1)
...         decoded = base64.b64decode(encoded).decode('UTF-8')
...         username, password = decoded.split(':', 1)
...         return username == password
...
...     def __call__(self, environ, start_response):
...         if self.is_authenticated(environ.get('HTTP_AUTHORIZATION')):
...             return self.wsgi(environ, start_response)
...         start_response('401 Authentication Required',
...             [('Content-Type', 'text/html'),
...              ('WWW-Authenticate', 'Basic realm="Login"')])
...         return [b'Login']

>>> middlewares = [AuthMiddleware]

# Create an application factory
>>> app_factory = FlaskFactory(__name__, middlewares=middlewares)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

Add Flasks Extension

You can declare your own flask extensions

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup
>>> from flasgger import Swagger

>>> swag = Swagger(template={'version': '0.3.4-dev'})

>>> my_extensions = [swag]
```

(continues on next page)

(continued from previous page)

```
# Create an application factory
>>> createapp_factory_app = FlaskFactory(__name__, extensions=my_extensions)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

consensys_utils.flask.FlaskFactory also extensions given as a function taking a `flask.Flask` application as an argument

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup

>>> def init_login_extension(app):
...     if app.config.get('LOGIN'):
...         from flask_login import LoginManager
...
...         login_manager = LoginManager()
...         login_manager.init_app(app)

>>> my_extensions = [init_login_extension]

# Create an application factory
>>> app_factory = FlaskFactory(__name__, extensions=my_extensions)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

It allows you to implement advanced extension initialization based on application configuration. In particular in the example above it allows to allows user having 'Flask-Login' installed on option, only users having activated a `LOGIN` configuration need to have 'Flask-Login' installed.

Set Application Hooks

```
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup

>>> def set_log_request_hook(app):
...     @app.before_request
...     def log_request():
...         current_app.logger.debug(request)

>>> my_hook_setters = [set_log_request_hook]

# Create an application factory
>>> app_factory = FlaskFactory(__name__, hook_setters=my_hook_setters)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

Register Blueprints

```
>>> from flask import Blueprint
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup


>>> my_bp1 = Blueprint('my-bp1', __name__)
>>> my_bp2 = Blueprint('my-bp2', __name__)

>>> blueprints = [
...         my_bp1,
...     lambda app: app.register_blueprint(my_bp2),
... ]

# Create an application factory
>>> app_factory = FlaskFactory(__name__, blueprints=blueprints)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

Declare custom CLI commands

It is highly recommended that you declare custom CLI commands directly on the `consensys_utils.flask.cli.FlaskGroup` object. It automatically injects a `--config` option to the command for configuration file.

```
>>> from flask import Blueprint
>>> from flask.cli import with_appcontext
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup

# Create an application factory
>>> app_factory = FlaskFactory(__name__)

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)

>>> @cli.command('test')
... @with_appcontext
... def custom_command():
...     click.echo('Test Command on %s' % current_app.import_name)
```

1.1.3 Properly manage process to execute an iterator

Quickstart

ConsenSys-Utils provides some resources to properly maintain the execution of an iterator. In particular it allows to

1. Run the iterator with a Gunicorn worker in a properly maintained process
2. Connect a Flask application to the iterator enabling external control on iterator state

It relies on two main resources

- `consensys_utils.flask.extensions.iterable.FlaskIterable()` that allows to transform a Flask application into an Iterable

- `consensys_utils.gunicorn.workers.SyncIterableWorker()` that allows to properly maintain a loop on an iterable WSGI object

1. Create a `app.py`

```
>>> from flask import Flask
>>> from consensys_utils.flask.extensions.iterable import FlaskIterable
>>> from consensys_utils.flask import FlaskFactory
>>> from consensys_utils.flask.cli import FlaskGroup

# Create an iterator
>>> iterator = iter(range(3))

# Create an app factory and extend it to make it with a FlaskIterable␣
↪extension
>>> iterable = FlaskIterable(iterator)
>>> app_factory = FlaskFactory(__name__, extensions=[iterable])

# Declares a click application using ConsenSys-Utils click group
>>> cli = FlaskGroup(app_factory=app_factory)
```

2. Set a `config.yml` choosing a `consensys_utils.gunicorn.workers.SyncIterableWorker()` Gunicorn worker allowing to iterate ono the

```
flask:
  base:
    APP_NAME: Iterating-App
gunicorn:
  worker-processes:
    worker_class: consensys_utils.gunicorn.workers.SyncIteratingWorker
```

3. Define application entry point and start application as described in *Create Flask Application Quickstart*

Advanced usage

For an advance use-case you can refer to the next example

```
"""
    examples.iterable
    ~~~~~~~~~~~~~~~~~

    Implement an example of properly managing an iterator using Flask-Iterable and␣
↪Gunicorn

    :copyright: Copyright 2017 by ConsenSys France.
    :license: BSD, see LICENSE for more details.
"""

import logging
import os

from cfg_loader.utils import parse_yaml_file
from flask import current_app, jsonify
from gunicorn.app.base import BaseApplication

from consensys_utils.flask import Flask
from consensys_utils.flask.extensions.iterable import FlaskIterable
```

(continues on next page)

(continued from previous page)

```
from consensys_utils.gunicorn.workers import PauseIteration

logger = logging.getLogger('examples.iterable')
LOGGING_FILE = os.path.join(os.path.dirname(__file__), 'logging.yml')


# Declare an iterator that we want to properly managed using Gunicorn
class Iterator:
    def __init__(self):
        self.meter = 0

    def set_config(self, config):
        self.meter = config['meter']

    def __iter__(self):
        return self

    def __next__(self):
        logger.info('Iterator.__next__ meter=%s' % self.meter)
        self.meter += 1
        if self.meter % 2 == 0:
            # Indicating the running loop to pause iteration for 2 secs
            raise PauseIteration(2)

        if self.meter >= 100:
            raise StopIteration()


# We declare a Flask application and extend it to make it iterable
iterable_app = Flask(__name__)
iterable_app.config['meter'] = 10
FlaskIterable(Iterator, iterable_app)


# We declare routes on Flask application to interact with the iterator
@iterable_app.route('/get')
def get():
    """Get current value of the iterator meter"""
    logger.info('app.get meter=%s' % current_app.iterator.meter)
    return jsonify({'data': current_app.iterator.meter})


@iterable_app.route('/set/<int:meter>')
def set(meter=0):
    """Set current value of the iterator meter"""
    current_app.iterator.meter = rv = meter
    logger.info('app.set meter=%s' % current_app.iterator.meter)
    return jsonify({'data': rv})


# We declare a custom Gunicorn application for the only matter of the example
class Application(BaseApplication):
    def load(self):
        return iterable_app

    def load_config(self):
        self.cfg.set('logconfig_dict', parse_yaml_file(LOGGING_FILE))
```

(continues on next page)

(continued from previous page)

```
        # We use specific ConsenSys-Utils worker class
        self.cfg.set('worker_class', 'consensys_utils.gunicorn.workers.
↪SyncIteratingWorker')


if __name__ == "__main__":
    # Run iterator
    app = Application()
    app.run()
```

CHAPTER 2

Resources

2.1 ConsenSys-Utils Resources

2.1.1 Config

ConsenSys-Utils makes active use of cfg-loader for loading configuration. It is highly recommended you have some basic knowledge of it before using ConsenSys-Utils.

Schema

ConsenSys-Utils gathers a bench of useful `ConfigSchema` that can be reused in any project.

Logging

Logging schema

class consensys_utils.config.schema.logging.**LoggingConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Logging configuration schema

Describes and validates against

Key	Comment	Default value
`LOGGING_CONFIG_PATH`	Valid path to a .yml logging configuration file	logging.yml

Flask

Flask application configuration schemas

class consensys_utils.config.schema.flask.**FlaskConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Flask application configuration schema

Describes and validates against

Key	Comment	Default value
base	Required base configuration in *BaseConfigSchema* format	*BaseConfigSchema* default
session	Cookie session configuration in *SessionConfigSchema* format	*SessionConfigSchema* default
PERMANENT_SESSION_LIFETIME	Cookie's expiration in number of seconds	2678400
healthcheck	Healthcheck configuration in *HealthCheckConfigSchema* format	
swagger	Swagger configuration in *SwaggerConfigSchema* format	

Base

class consensys_utils.config.schema.flask.**BaseConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Base flask configuration schema

Describes and validates against

Key	Comment	Default value
APP_NAME	Required name of the application	Required

Session

class consensys_utils.config.schema.flask.**SessionConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Flask Session configuration

Describes and validates against

Key	Comment	Default value
cookie	Session cookie configuration in *CookieConfigSchema* format	*CookieConfigSchema* default
REFRESH_EACH_REQUEST	Control whether the cookie is sent with every response	True

class consensys_utils.config.schema.flask.**CookieConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Flask Session cookie configuration

Describes and validates against

Key	Comment	Default value
NAME	The name of the session cookie	'session'
DOMAIN	The domain match rule that the session cookie will be valid for	None
PATH	Path to the session cookie will be valid for	None
HTTPONLY	Browsers will not allow JavaScript access to cookies marked as "HTTP only" for security	True
SECURE	Browsers will only send cookies with requests over HTTPSd	False
SAMESITE	Restrict how cookies are sent with requests from external sites	None

Health Check

class consensys_utils.config.schema.flask.**HealthCheckConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Healthcheck configuration configuration schema

Describes and validates against

Key	Comment	Default value
ENDPOINT_URL	Endpoint URL for healthcheck	*/healthcheck*

Swagger

class consensys_utils.config.schema.flask.**SwaggerConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Swagger configuration

Key	Comment	Default value
specs	List of Swagger-UI specs in SwaggerSpecConfigSchema format	[{'ENDPOINT': 'apispec_1', 'ROUTE': '/apispec_1.json'}]
STATIC_URL_PATH	Endpoint for Swagger static files	'/flasgger_static'
SWAGGER_UI	Boolean indicating if Swagger UI should be activated	False
SPECS_ROUTE	Route to retrieve specifications	'/apidocs/'

WSGI

Schema for WSGI middlewares

Request ID

class consensys_utils.config.schema.wsgi.**WSGIConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Configuration relative to wsgi middlewares

Describes and validates against

Key	Comment	Default value
`request_id`	Request ID configuration in *RequestIDConfigSchema*	`None`

class consensys_utils.config.schema.wsgi.**RequestIDConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Request ID Middleware configuration

Describes and validates against

Key	Comment	Default value
`REQUEST_ID_HEADER`	Required header where to load/inject correlation ID	'X-Request-ID'

Gunicorn

Gunicorn configuration schemas

class consensys_utils.config.schema.gunicorn.**GunicornConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Gunicorn configuration

Please refer to http://docs.gunicorn.org/en/stable/settings.html for exhaustive listing of Gunicorn settings.

Describes and validates against

Key	Comment	Default value
`config`	Gunicorn config file path	
`debugging`	Debugging config in format `DebuggingConfigSchema`	`DebuggingConfigSchema` default
`logging`	Gunicorn logging config in format *`LoggingConfigSchema`*	*`LoggingConfigSchema`* default
`process-naming`	Process naming config in format *`ProcessNamingConfigSchema`*	*`ProcessNamingConfigSchema`* default
`ssl`	Debugging config in format *`SSLConfigSchema`*	*`SSLConfigSchema`* default
`security`	Security config in format *`SecurityConfigSchema`*	*`SecurityConfigSchema`* default
`server-mechanics`	Server mechanics config in format *`ServerMechanicsConfigSchema`*	*`ServerMechanicsConfigSchem`* default
`server-socket`	Server Socket config in format *`ServerSocketConfigSchema`*	*`ServerSocketConfigSchema`* default
`worker-processes`	Worker processes config in format *`WorkerProcessesConfigSchema`*	*`WorkerProcessesConfigSchem`* default

class consensys_utils.config.schema.gunicorn.**ServerSocketConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Server Socket configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#server-socket

class consensys_utils.config.schema.gunicorn.**WorkerProcessesConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Worker Processes configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#worker-processes

class consensys_utils.config.schema.gunicorn.**LoggingConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Logging configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#logging

class consensys_utils.config.schema.gunicorn.**ServerMechanicsConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Server Mechanics configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#server-mechanics

class consensys_utils.config.schema.gunicorn.**ProcessNamingConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Process Naming configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#process-naming

class consensys_utils.config.schema.gunicorn.**SSLConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

SSL configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#ssl

class consensys_utils.config.schema.gunicorn.**SecurityConfigSchema**(**args*, *substitution_mapping=None*, ***kwargs*)

Security configuration

c.f http://docs.gunicorn.org/en/stable/settings.html#security

Loader

consensys_utils.config.loader.**create_yaml_config_loader**(*config_schema*, *default_config_path='config.yml'*)

Create a configuration loader that can read configuration from .yml file

Parameters

- **config_schema** (subclass of cfg_loader.ConfigSchema) – Configuration schema
- **default_config_path** (*str*) – Default path where to load configuration from

2.1.2 Flask

ConsenSys-Utils defines many resources for working with Flask application. It is highly recommended you have some basic knowledge of Flask before using ConsenSys-Utils.

Application Factory

ConsenSys-Utils provides useful resources to implement the Flask application factory pattern

class consensys_utils.flask.**FlaskFactory**(*import_name=None*, *yaml_config_loader=<cfg_loader.loader.YamlConfigLoader object>*, *default_config=None*, *config_path=None*, *middlewares=None*, *extensions=None*, *hook_setters=None*, *blueprints=None*, ***flask_kwargs*)

ConsenSys Flask factory. It inherits from *BaseFlaskFactory()*

By default it applies

Middlewares

- *consensys_utils.flask.wsgi.apply_request_id_middleware()*: A middleware to inject a custom Request ID header

Extensions

- *consensys_utils.flask.extensions.initialize_health_extension()*: Init a Flask extension for health check
- *consensys_utils.flask.extensions.initialize_web3_extension()*: Init a FlaskWeb3 extension

Hooks

- *consensys_utils.flask.hooks.set_request_id_hook()*: Hook injecting Request ID header on‘‘flask.request‘‘

class consensys_utils.flask.**BaseFlaskFactory**(*import_name=None*, *yaml_config_loader=<cfg_loader.loader.YamlConfigLoader object>*, *default_config=None*, *config_path=None*, *middlewares=None*, *extensions=None*, *hook_setters=None*, *blueprints=None*, ***flask_kwargs*)

A factory to create Flask application

```
>>> app_factory = BaseFlaskFactory(__name__)
```

When creating an application a *FlaskFactory* accomplishes next steps

1. Initialize Flask application

   By default it creates a *consensys_utils.flask.Flask* application

2. Set application configuration by using a .yml configuration loader

   You can refer to *consensys_utils.flask.config.set_app_config()* for more information

3. Apply WSGI middlewares on the application

   You can refer to *consensys_utils.flask.wsgi.apply_middlewares()* for more information

4. Initialize extensions on the application

   You can refer to *consensys_utils.flask.extensions.initialize_extensions()* for more information

5. Set hooks on the application

   You can refer to *consensys_utils.flask.hooks.set_hooks()* for more information

6. Register blueprints on the application

   You can refer to *consensys_utils.flask.blueprints.register_blueprints()* for more information

It is possible to override default behavior by creating a new class that inherits from *FlaskFactory*

Example: Adding default hooks

```
>>> def set_foo_request_id_hook(app):
...     @app.before_request
...     def set_request_id():
...         request.id = 'foo'
```

(continues on next page)

(continued from previous page)

```
>>> class CustomFlaskFactory(BaseFlaskFactory):
...     default_hook_setters = [set_foo_request_id_hook]


>>> app_factory = CustomFlaskFactory(__name__)
```

Parameters

- **import_name** (*str*) – The name of the application package
- **yaml_config_loader** (cfg_loader.loader.YamlConfigLoader) – Optional config loader
- **middlewares** (*list*) – Middlewares to apply on the application (c.f *consensys_utils.flask.wsgi.apply_middlewares()*)
- **extensions** (*list*) – Extensions to initiate on the application (c.f. *consensys_utils.flask.extensions.initialize_extensions()*)
- **hook_setters** (*list*) – Hooks to set on the application (c.f. *consensys_utils.flask.hooks.set_hooks()*)
- **blueprints** (*list*) – Blueprints to register on the application (c.f. *consensys_utils.flask.blueprints.register_blueprints()*)

apply_middlewares()
Apply middlewares on application

create_app(*config_path=None*, *config=None*, ***kwargs*)
Create an application

Parameters

- **config_path** (*str*) – .yml configuration path
- **config** (*dict*) – Optional application config
- **kwargs** – Keyword arguments to provide to *flask_class* when instantiating the application object

flask_class
alias of *Flask*

init(***kwargs*)
Instantiate Flask application

Parameters **kwargs** (*dict*) – Keyword arguments to provide to the Flask application

initialize_extensions()
Initialize extensions on application

load_config(*config_path=None*)
Load configuration

Parameters **config_path** (*str*) – Configuration path

register_blueprints()
Register blueprints on application

set_config(*config=None*)
Set application config

Parameters **raw_config** (*dict*) – Optional application config

set_hooks()
Set hooks on application

class consensys_utils.flask.**Flask**(*import_name*, *static_url_path=None*, *static_folder='static'*, *static_host=None*, *host_matching=False*, *subdomain_matching=False*, *template_folder='templates'*, *instance_path=None*, *instance_relative_config=False*, *root_path=None*)
ConsenSys-Utils Flask class

It applies a light overriding on top of :class‘flask.Flask‘ to enable

- usage of a logger that can be configured from .yml file

WSGI

ConsenSys-Utils implements functions to facilitate Flask app decoration with WSGI middlewares

consensys_utils.flask.wsgi.**apply_middlewares**(*app*, *middlewares=None*)
Apply WSGI middlewares to a Flasks application

Example:

```
>>> app = Flask(__name__)

>>> class AuthMiddleware:
...     def __init__(self, wsgi):
...         self.wsgi = wsgi
...
...     @staticmethod
...     def is_authenticated(header):
...         if not header:
...             return False
...         _, encoded = header.split(None, 1)
...         decoded = base64.b64decode(encoded).decode('UTF-8')
...         username, password = decoded.split(':', 1)
...         return username == password
...
...     def __call__(self, environ, start_response):
...         if self.is_authenticated(environ.get('HTTP_AUTHORIZATION')):
...             return self.wsgi(environ, start_response)
...         start_response('401 Authentication Required',
...             [('Content-Type', 'text/html'),
...              ('WWW-Authenticate', 'Basic realm="Login"')])
...         return [b'Login']

>>> middlewares = [AuthMiddleware]

>>> apply_middlewares(app, middlewares)
```

Parameters

- **app** (flask.Flask) – Flask application
- **middlewares** (*list*) – WSGI middleware to apply on the application. Expects a list of elements which are either
  - A class taking a wsgi as an argument

  - A function that takes a `flask.Flask` as argument and even eventually apply a middleware on it

`consensys_utils.flask.wsgi.`**`apply_request_id_middleware`**(*app*)
Apply a `consensys_utils.wsgi.RequestIDMiddleware()` on a Flask application

Parameters **app** (`flask.Flask`) – Flask application

Extensions

ConsenSys-Utils implements functions to facilitate initialization of Flask extensions on an application

`consensys_utils.flask.extensions.`**`initialize_extensions`**(*app*, *extensions=None*)
Initialize extensions on a Flask application

Example: Adding an extension

```
>>> app = Flask(__name__)

>>> swag = Swagger(template={'version': '0.3.4-dev'})

>>> my_extensions = [swag]

>>> initialize_extensions(app, my_extensions)
```

Parameters

- **app** (`flask.Flask`) – Flask application
- **extensions** (*list*) – Extensions to initialize on the application. Expects a list of elements which are either
  - a Flask extension object (having a callable attribute `init_app`)
  - a function that takes a `flask.Flask` as argument and eventually initialize an extension on it

`consensys_utils.flask.extensions.`**`initialize_health_extension`**(*app*)
Initialize healthcheck extension

If `health` is missing in application configuration then this function has no effect

Parameters **app** (`flask.Flask`) – Flask application

`consensys_utils.flask.extensions.`**`initialize_web3_extension`**(*app*)
Initialize Web3 extension

If `web3` is missing in application configuration then this function has no effect

Parameters **app** (`flask.Flask`) – Flask application

ConsenSys-Utils defines a bench of *Flask* extensions that can be smoothly re-used.

Healthcheck

class consensys_utils.flask.extensions.health.**HealthCheck**(*app=None*, *path=None*, *success_status=200*, *success_headers=None*, *success_handler=<function json_success_handler>*, *success_ttl=27*, *failed_status=500*, *failed_headers=None*, *failed_handler=<function json_failed_handler>*, *failed_ttl=9*, *exception_handler=<function basic_exception_handler>*, *checkers=None*, *log_on_failure=True*, ***options*)

Healthcheck extension that declares an health check route

Healthcheck URL can be set at application initialization by reading app configuration

Swagger

class consensys_utils.flask.extensions.swagger.**Swagger**(**args*, *template=None*, *openapi='3.0'*, *version='0.1.0-dev'*, *title='Base App'*, *tags=None*, ***kwargs*)

Flask extension that allow integration with Swagger

Web3

class consensys_utils.flask.extensions.web3.**FlaskWeb3**(**args*, *app=None*, *create_provider=<function create_provider>*, ***kwargs*)

A Flask-Web3 class that supports initializing application with configuration in format `consensys_utils.config.schema.flask.ConfigSchema()`

You can customize this class the same you would do with `flask_web3.FlaskWeb3`

init_app(*app*)

Initialize application

Parameters **app** (`flask.Flask`) – Flask application or blueprint object to extend

Config

consensys_utils.flask.config.**set_app_config**(*app*, *config=None*)

Set application configuration

Parameters

- **app** (flask.Flask) – Flask application
- **config** (*dict*) – Optional Application configuration

Hooks

ConsenSys-Utils implements functions to facilitate setting Flask hooks on an application

consensys_utils.flask.hooks.**set_hooks**(*app*, *hook_setters=None*)
Set hooks on a Flask application

Example: Adding a hook

```
>>> app = Flask(__name__)

>>> def set_log_request_hook(app):
...     @app.before_request
...     def log_request():
...         current_app.logger.debug(request)

>>> my_hook_setters = [set_log_request_hook]

>>> set_hooks(app, my_hook_setters)
```

Parameters

- **app** (flask.Flask) – Flask application
- **hook_setters** (*list*) – Hooks to set on the application. Expects a list of functions that takes a flask.Flask as argument

consensys_utils.flask.hooks.**set_request_id_hook**(*app*)
Set a hook to inject request ID

It basis on application config to get the request header from which to retrieve request ID

Parameters **app** (flask.Flask) – Flask application

Blueprints

ConsenSys-Utils implements functions to facilitate registering blueprints on an application

consensys_utils.flask.blueprints.**register_blueprints**(*app*, *blueprints=None*)
Register blueprints on a Flask application

Example:

```
>>> app = Flask(__name__)

>>> my_bp1 = Blueprint('my-bp1', __name__)
>>> my_bp2 = Blueprint('my-bp2', __name__)

>>> blueprints = [
...     lambda app: app.register_blueprint(my_bp1),
...     my_bp2,
... ]
>>> register_blueprints(app, blueprints)
```

Parameters

- **app** (flask.Flask) – Flask application
- **blueprints** (*list*) – Blueprints to register on the application. Expects a list of elements which elements are either
  - a flask.Blueprint
  - a function that takes a flask.Flask as argument and eventually register a blueprint on it

Logging

ConsenSys-Utils implements resources to facilitate logging blueprints on an application

class consensys_utils.flask.logging.**RequestIDFilter**(*name=''*)
Logging filter that allows to enrich log with Flask request ID

filter(*record*)
Enrich log record with request ID

consensys_utils.flask.logging.**create_logger**(*app*, *logger='app'*)
Create logger for Flask app

Parameters

- **config** (*dict*) – Logging configuration
- **logger** (*str*) – Name of the logger

2.1.3 Gunicorn

ConsenSys-Utils slightly enhances gunicorn for better compatibility with its features

Application

class consensys_utils.gunicorn.app.**WSGIApplication**(*loader*, **args*, ***kwargs*)
An enhanced gunicorn WSGIApplication including ConsenSys-Utils features

load_config()
This method is used to load the configuration from one or several input(s). Custom Command line, configuration file. You have to override this method in your class.

Config

class consensys_utils.gunicorn.config.**Config**(*usage=None*, *prog=None*)
Gunicorn Configuration that ensures next settings are correctly discovered

- LoggingConfig()
- WSGIConfig()

Logging

class consensys_utils.gunicorn.logging.**Logger**(*cfg*)
Enrich Gunicorn logger class

In particular it overrides the following methods

- *setup* to load logging configuration from a .yml file

setup(*cfg*)
Setup the logger configuration from .yml file

class consensys_utils.gunicorn.logging.**RequestIDLogger**(**args*, ***kwargs*)
Gunicorn logger that handles Request ID header

access(*resp*, *req*, *environ*, *request_time*)
See http://httpd.apache.org/docs/2.0/logs.html#combined for format details

setup(*cfg*)
Setup the logger configuration from .yml file

Workers

class consensys_utils.gunicorn.workers.**SyncIteratingWorker**(*age*, *ppid*, *sockets*, *app*, *timeout*, *cfg*, *log*)
A Gunicorn synchronous worker that allows to run an iterable WSGI application.

It allows to run a loop process that iterates over a WSGI application object while allowing to process HTTP requests.

Since the worker is synchronous it is thread safe to modify the WSGI object either when iterating or when handling an HTTP request.

Remark

Such a worker should not be considered highly performing as HTTP server but for dealing with a few requests to control the iterable WSGI application it is well suited.

handle(*listener*, *client*, *address*)
Handle a request

Method is almost identical to `gunicorn.workers.sync.SyncWorker()` one.

We need to overide this method because we use non blocking socket connections thus we are more sensitive to `errno.EAGAIN()` errors.

iterate()
Iterate on WSGI object

run()
Run the main worker loop

At each step of the loop it

1. Handles entry socket request if available
2. Iterate on the WSGI iterable object

If a *consensys_utils.exceptions.PauseIteration()* is caught when iterating on the WSGI object then the loop waits by entering a stale state freeing CPU usage.

Receiving an HTTP request instantaneously gets the loop out of stale state.

2.1.4 Exceptions

class consensys_utils.exceptions.**PauseIteration**(*timeout=None*)
Error indicating to pause iteration

Useful when combined with *consensys_utils.gunicorn.workers.SyncIteratingWorker()*

Parameters **timeout** (*float*) – Maximum time to pause before re-starting iteration

CHAPTER 3

Contributing

If you are interested in contributing to the project please refer to *Contributing guidelines*

3.1 Contributing guidelines

3.1.1 Feature Requests, Bug Reports, and Feedback...

...should all be reported on the GitHub Issue Tracker .

Reporting issues

- Describe what you expected to happen.
- If possible, include a minimal, complete, and verifiable example to help
- Describe what actually happened. Include the full traceback if there was an exception.

3.1.2 Setting-Up environment

Requirements

1. Having the latest version of `git` installed locally
2. Having Python 3.6 installed locally
3. Having `virtualenv` installed locally

   To install `virtualenv` you can run the following command

   ```
   $ pip install virtualenv
   ```

4. Having `docker` and `docker-compose` installed locally

5. Having `pip` environment variables correctly configured

   Some of the package's dependencies of the project could be hosted on a custom PyPi server. In this case you need to set some environment variables in order to make `pip` inspect the custom pypi server when installing packages.

   To set `pip` environment variables on a permanent basis you can add the following lines at the end of your `\.bashrc` file (being careful to replace placeholders)

```
# ~/.bashrc

...

# Indicate to pip which pypi server to download from
export PIP_TIMEOUT=60
export PIP_INDEX_URL=<custom_pypi_protocol>://<user>:<password>@<custom_pypi_host>
export PIP_EXTRA_INDEX_URL=https://pypi.python.org/simple
```

First time setup

- Clone the project locally
- Create development environment using Docker or Make

```
$ make init
```

3.1.3 Project organisation

The project

```
.
├── consensys_utils/           # Main package source scripts (where all functional␣
↪python scripts are stored)
├── docs/                    # Docs module containing all scripts required by sphinx␣
↪to build the documentation
├── tests/                   # Tests folder where all test modules are stores
├── .coveragerc              # Configuration file for coverage
├── .gitignore               # List all files pattern excluded from git's tracking
├── .gitlab-ci.yml           # GitLab CI script
├── AUTHORS                  # List of authors of the project
├── CHANGES                  # Changelog listing every changes from a release to␣
↪another
├── CONTRIBUTING.rst         # Indicate the guidelines that should be respected when␣
↪contributing on this project
├── LICENSE                  # License of the project
├── Makefile                 # Script implement multiple commands to facilitate␣
↪developments
├── README.rst               # README.md of your project
├── setup.cfg                # Configuration of extra commands that will be installed␣
↪on package setup
├── setup.py                 # File used to setup the package
└── tox.ini                  # Configuration file of test suite (it runs test suite␣
↪in both Python 3.5 and 3.6 environments)
```

3.1.4 Coding

Development Workflow

Please follow the next workflow when developing

- Create a branch to identify the feature or issue you will work on (e.g. `feature/my-feature` or `hotfix/2287`)
- Using your favorite editor, make your changes, committing as you go and respecting the AngularJS Commit Message Conventions
- Follow PEP8 and limit script's line length to **120 characters**. See *testing-linting*
- Include tests that cover any code changes you make. See *running-test* and *running-coverage*
- Update `setup.py` script with all dependencies you introduce. See *adding-dependency* for precisions
- Write clear and exhaustive docstrings. Write docs to precise how to use the functionality you implement. See *writing-docs*
- Update changelog with the modifications you proceed to. See *updating-changelog*
- Your branch will soon be merged ! :-)

Testing

Running tests

Run test suite in by running

```
$ make test
```

Running coverage

Please ensure that all the lines of source code you are writing are covered in your test suite. To generate the coverage report, please run

```
$ make coverage
```

Read more about coverage.

Running the full test suite with `tox` will combine the coverage reports from all runs.

Testing linting

To test if your project is compliant with linting rules run

```
$ make test-lint
```

To automatically correct linting errors run

```
$ make lint
```

Running full test suite

Run test suite in multiple distinct python environment with following command

```
$ make tox
```

Writing documentation

Write clear and exhaustive docstrings in every functional scripts.

This project uses sphinx to build documentations, it requires docs file to be written in `.rst` format.

To build the documentation, please run

```
$ make docs
```

Precisions

Updating changelog

Every implemented modifications on the project from a release to another should be documented in the changelog `CHANGES.rst` file.

The format used for a release block is be the following

```
Version <NEW_VERSION>
---------------------

Released on <NEW_VERSION_RELEASED_DATE>, codename <NEW_VERSION_CODENAME>.

Features

- Feature 1
- Feature 2
- Feature 3

Fixes

- Hotfix 1 (``#134``)
- Hotfix 2 (``#139``)

.. _#134: https://github.com/ConsenSys/consensys-utils/issues/134
.. _#139: https://github.com/ConsenSys/consensys-utils/issues/139
```

Be careful to never touch the header line as well as the release's metadata sentence.

```
Version <NEW_VERSION>
---------------------

Released on <NEW_VERSION_RELEASED_DATE>, codename <NEW_VERSION_CODENAME>.
```

Adding a new dependency

When adding a new package dependency it should be added in `setup.py` file in the `install_requires` list

The format should be `dependency==1.3.2`.

When adding a dev dependency (e.g. a testing dependency) it should be added in

- `setup.py` file in the `extra_requires` `dev` list
- `tox.ini` file in the `[testenv]` `deps`

3.1.5 Makefile commands

`Makefile` implements multiple handful shell commands for development

make init

Initialize development environment including

- venv creation
- package installation in dev mode

make clean

Clean the package project by removing some files such as `.pyc`, `.pyo`, `*.egg-info`

make test-lint

Check if python scripts are compliant with PEP8 rules

make lint

Automatically correct PEP8 mistakes contained in the project.

make coverage

Run the test suite and computes test coverage. It creates an html report that is automatically open after the commands terminates

make tox

Run the test suites in multiple environments

make docs

Build documentation from the `docs` folder using sphinx. It generates a build of the documentation in html format located in `docs/_build/html`.

CHAPTER 4

Additional Notes

Legal information and changelog are here for the interested.

4.1 Changelog

Here you can see the full list of changes between each releases of ConsenSys-Utils.

4.1.1 Version 0.2.0

Unreleased

4.1.2 Version 0.2.0b3

Released on August 9th 2018

Chore

- Requirements: add requirements for doc (required by readthedocs)

4.1.3 Version 0.2.0b2

Released on August 9th 2018

Fix

- Flask: remove swagger extension from default extensions

4.1.4 Version 0.2.0b1

Released on August 6th 2018

Feat

- Config: schema for web3 provider
- Web3: implement create_provider function
- Flask: implement Web3 extension
- Flask: implement Flask-Iterable extension
- Gunicorn: implement SyncIteratingWorker

Chore

- Examples: implement an example for an iterating worker

4.1.5 Version 0.1.0

Released on July 30th 2018

Fix

- Flask: Enhance consensys_utils.flask.cli.FlaskGroup
- Flask: Improve Factory pattern

4.1.6 Version 0.1.0b4

Released on July 27th 2018

Refactor

- Config: update default values of Gunicorn configuration schema

4.1.7 Version 0.1.0b3

Released on July 27th 2018

Fix

- Gunicorn: fix gunicorn application to use `consensys_utils.gunicorn.config.Config`

Tests

- Gunicorn: add tests for `gunicorn.config.schema.GunicornConfigSchema`

4.1.8 Version 0.1.0b2

Released on July 26th 2018

Fix

- Flask: update FlaskFactory

4.1.9 Version 0.1.0b1

Released July 26th 2018

Features

- Config: implement config package
- Flask: implement WSGI middlewares helpers
- Flask: implement application hooks helpers
- Flask: implement config features to integrate with cfg-loader
- Flask: implement flask extensions helpers
- Flask: implement default extension for healthcheck
- Flask: implement default extension for Swagger
- Flask: implement logging features
- Flask: implement blueprints helpers
- Gunicorn: implement custom Gunicorn application
- Flask: implement CLI resources in particular FlaskGroup that allows to smoothly integrates with Gunicorn
- Config: Implement Gunicorn config schema

4.1.10 Version 0.0.0

Unreleased

Chore

- Project: Initialize project

4.2 License

4.2.1 Authors

ConsenSys-Utils is developed and maintained by the ConsenSys France team and community contributors. The core maintainers are:

- Nicolas Maurice (nmvalera)

4.2.2 General License Definitions

The following section contains the full license texts for ConsenSys-Utils and the documentation.

- "AUTHORS" hereby refers to all the authors listed in the *Authors* section.
- The "*License*" applies to all the source code shipped as part of ConsenSys-Utils (ConsenSys-Utils itself as well as the examples and the unit tests) as well as documentation.

4.2.3 License

Copyright (c) 2017 by ConsenSys France and contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

P

R

S

W