Saddle Documentation

Release 0.1

Colm O'Connor

April 22, 2015

Contents

Saddle is a black box testing framework built upon python's unittest that helps you write simple, easy to read functional tests for any software.

Contents:

CHAPTER 1

Saddle Tutorials

The following are tutorials for setting up different technology stacks for testing in Saddle.

Contents:

1.1 Getting started testing with Saddle and Django, Celery, Redis and Postgresql

This is a tutorial that will take you through the steps you need to take to create a test harness and write a simple test for a Django app.

It assumes a basic level of Django knowledge.

1.1.1 Preparation

You need several packages installed before you can begin here:

On Ubuntu:

```
$ sudo apt-get install python python-virtualenv python-pip postgresql redis
```

I also recommend that on ubuntu, you *stop* redis and postgresql (they are auto-started) and prevent them from being started on boot up (why?_).

This can be done with:

```
$ sudo /etc/init.d/redis-server stop
$ sudo update-rc.d -f redis-server disable
```

To stop redis, and prevent it from being started at startup. And:

```
$ sudo /etc/init.d/postgresql stop
$ sudo update-rc.d -f postgresql disable
```

To stop Postgresql. You can always re-enable them later.

1.1.2 Check out the code

Find a new directory and check out the code. You can do that by:

```
$ git clone git@github.com:crdoconnor/django-remindme.git
```

Or:

```
$ git clone https://github.com/crdoconnor/django-remindme.git
```

1.1.3 Create your Service Engine

Currently we have a django/celery app, but we need to make it run before we can test it.

The first thing that we must do is to create a harness and a basic test case that uses it.

Create a directory called 'tests', and in the directory, put this in a file called "testcase.py":

```
import saddle
import os


class RemindMeServiceEngine(saddle.Harness):
    def configure(self, fixtures):
        """Set up
        self.directory = self.relative_to_this_file("..")
        self.timeout = 15.0
```

This contains a very basic skeleton of a service engine that states two things:

- If all of the services are not ready within 15 seconds it times out.
- The default directory to run tests in is the one above the file (testcase.py) this class is in - the project's root directory in other words.

CHAPTER 2

Saddle Glossary

These are terms that have a specific meaning for the Saddle project.

Contents:

2.1 DRY

See: Don't Repeat Yourself Wikipedia Page

2.2 FIRST

FIRST is a set of principles for good tests.

These principles come from chapter 9 of Clean Code.

Occasionally these principles will conflict with one another and with other good software developement practises (e.g. loose coupling or DRY).

2.2.1 Fast

Tests should complete in as little time as possible.

2.2.2 Isolated

Test isolation should be maximized as far as possible.

Isolation means that tests set themselves up and clean up after themselves.

2.2.3 Repeatable

Tests must be able to be run repeatedly without intervention.

2.2.4 Self-verifying

Tests should be pass-fail and not require human input for verification.

2.2.5 Timely

Tests should be written before the code that makes them pass.

2.3 Artefacts Directory

Not yet implemented.

2.4 Event oriented testing

Event oriented testing is a functional testing pattern whereby events drive the test, and all waiting is done explicitly.

See also *Sleep oriented testing*.

2.5 faketime

See *libfaketime*.

2.6 Fixture

The word fixture has several potential meanings.

2.6.1 Saddle meaning

Under saddle, a fixture is a type of object which is passed to a service which then uses it to set itself up in a desired state.

They are defined in a variable called fixtures in each test case class and passed to the ServiceEngine's configure method, which passes them to the correct service.

Each fixture is associated with a service. Examples of fixture types:

- `PostgresUser`
- `PostgresDatabase`
- `DjangoFixture`
- `DjangoSettings`

2.6.2 The wikipedia meaning

"Something used to consistently test some item, device or piece of software."

See: Test Fixture Wikipedia Page

2.6.3 Service specific meanings

Django also has the concept of a 'fixture', which refers to a JSON/YAML file representing database data.

DjangoFixture is a type of saddle fixture, representing a Django fixture.

2.7 Functional test

See the functional testing wikipedia page.

2.8 Kill

A kill message sends a SIGKILL signal to a service.

2.9 libfaketime

libfaketime intercepts various system calls which *services* use to retrieve the current date and time. It can then report faked dates and times (specified by you, the user) to these services.

It is currently *included* in saddle and compiled for your architecture upon installation (e.g. from pypi).

It works on Linux and Mac OS.

This can be used to ascertain that the services you run behave correctly when they are moved forward to specific points in time.

Libfaketime has the following known issues:

- Java – Java apps do not pick up the faked time correctly.
- NodeJS – currently breaks all node.js apps in recent linux kernels.
- Windows – does not work at all on Windows.

See: Libfaketime Project Page

2.10 Loaded

A *Service* is loaded but not *Ready* if the *Setup* method has completed, the service has been started and the service has issued a log message indicating its readiness.

The *Poststart* method is run when a service is loaded.

2.11 Logs

Saddle has several different types of logs.

It has the following types of logs for *each* service:

- Setup stdout
- Setup stderr

- Process stdout
- Process stderr
- Poststart stdout
- Poststart stderr

Additionally, it has:

- Test stdout
- Test stderr

for the test itself (e.g. if you use a print statement during the test).

All of the logs are aggregated and presented to the user by the *Service engine*.

2.12 Loose coupling

In computing and systems design a loosely coupled system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of other separate components. Sub-areas include the coupling of classes, interfaces, data, and services.

See: Loose coupling wikipedia page.

2.12.1 Applied to tests

Tightly coupled tests are tests that require more rewriting when refactoring code.

Loosely coupled tests can be left the same when refactoring code.

See (TODO):

- Unit testing of integration code: antipattern.
- Integration testing of algorithmic code: antipattern

2.13 Needs

By default, all services are started simultaneously.

If you have one service which requires another to be set up before it is run (e.g. django requires postgres and redis), you can add a list of services that the service 'needs', and the service engine will not start it until those services are ready.

Once all services are started and ready, the engine is declared ready.

The `service engine` determines service readiness and starts services accordingly.

2.14 Pgid

A pgid is a process group id. This is an id given to groups of processes in UNIX.

In saddle, each service has its own PGID so that SIGINT messages received by the harness (e.g. when the user hits ctrl-C) are not passed directly on to services.

2.15 Pipe

A pipe in saddle is a representation of a stdout or stderr handle from a service and the stdout/sterr handle of the test itself.

All of the output fed through these pipes is aggregated along with saddle log messages and displayed to the user, together, through the test's stdout pipe.

2.16 Poststart

Services in Saddle have a *Setup* method, which is called before the service is started and a *Poststart* method that is called after the service is *Loaded* (i.e. it has issued a log message indicating that it is ready).

See also:

- *Setup*
- *Loaded*
- *Ready*

2.17 Preconditions

See `fixtures`.

2.18 Ready

A *Service* in Saddle can be said to be ready when:

- The *Setup* method completes without errors.
- The process has started.
- The process has logged the `ready_line`.
- The *Poststart* method has completed without errors.

The harness is ready when all services are ready.

See also *Loaded*.

2.19 Saddle directory

$ .saddle

The saddle directory contains all of the runtime files required by saddle to run the tests, including:

- *Saddle State Directory*
- *Snapshot Directory*
- *Artefacts Directory*
- `running.json`

- `test.log`

2.20 SaddleREST

2.21 SaddleSMTP

SaddleSMTP is a mock SMTP server.

Every SMTP message sent to this server is logged as JSON to stdout, where it can be parsed by the harness to verify that it was sent correctly.

It can also mock failure scenarios (e.g. rejected email) by sending to specific email addresses.

It is a separate application and does not have to be used with Saddle exclusively.

2.22 Saddle State Directory

Not yet implemented.

2.23 Service

A service in Saddle is a process which must be run for the duration of the test.

It is either:

- A process that is running as part of the software under test.
- A process that is required for the software under test to function

(e.g. a database). * A mock service - a service imitating a real external service required by the software under test (e.g. *SaddleSMTP*).

All services have a *Setup* and *Poststart* method that runs just before the process starts and just after it is *Ready*.

For example, the `postgres_service` runs the initdb command in setup to create the database data directory and creates databases and users specified in fixtures in poststart.

See also Service (systems architecture).

2.24 Service engine

The service engine is the beating heart of saddle. It runs all *Setup* and *Poststart*, code, it starts all the services and aggregates all of their output into the output of the test itself.

The service engine also passes messages between the test and the services - e.g. the *Stop* message or exceptions.

2.25 Setup

Setup has two meanings in Saddle:

2.25.1 *Service* setup

Services in Saddle have a setup method, which is called before the service is started.

See also *Poststart*.

2.25.2 Testcase setUp

The harness TestCase class that you create to test your application inherits from python's unittest TestCase class.

The setUp method is run before each test to create a running environment that can be tested. It starts all services, installs all *fixtures* and does any other miscellaneous setup required.

2.26 Shutdown timeout

Shutdown timeout is the duration which saddle waits after sending a SIGINT/SIGQUIT before sending a SIGKILL.

Unless directly specified, it is assumed that the shutdown timeout is the same as the global timeout.

See also:

- *Stop*
- *Timeout*

2.27 Sleep

See `Event oriented testing` and `Sleep oriented testing`.

2.28 Sleep oriented testing

Sleep oriented testing is a common anti-pattern in functional testing, whereby waiting steps are added that sleep for a specified duration (e.g. one second) in order to wait for an event to take place that will allow the test to continue.

Examples of such events:

- Waiting for a UI element to appear in a page before clicking.
- Waiting or an email to arrive.
- Waiting for row to appear in a database.
- Waiting for file to appear or to change.

Sleep oriented testing causes the following problems (in order of importance):

- It reduces repeatability - the test can behave differently and fail differently on different computers owing to the different speeds in which they take place.
- It reduces speed - waiting 30 seconds for an event that may take between 10 and 20 seconds is slower than simply waiting for the event and continuing.

See also *Event oriented testing*.

2.29 Snapshot Directory

Not yet implemented.

2.30 stdlog

Stdlog is a directory in the *Saddle State Directory* that contains the redirected stdout and stderr from *Poststart*, *Setup* and the test itself (e.g. when you write a test and use the print command).

2.31 Stop

In saddle, the a 'stop' message is sent under the following circumstances:

- At the end of a successfully completed test with no errors.
- Once an error is detected during a test.
- If a user hits ctrl-C.

The *Service engine* reacts to the stop message by sending a SIGINT and SIGQUIT signal to every service near-simultaneously.

The service engine waits until all of the services have closed or until the *Shutdown timeout* expires, whichever comes first.

If the shutdown timeout expires, a SIGKILL signal is sent to all remaining services.

2.32 Time Travel

Time travel in saddle is a process by which the system time is faked for the services run by the service engine.

You can skip forward in time using the following commands:

```
>>> import datetime
>>> self.time_travel(datetime.timedelta(days=30))
```

2.33 Timeout

Timeout is a specified, configurable period of time which the *Service engine* waits before aborting a test.

See also *Shutdown timeout*.