WAVES CORE Documentation

Release master

Jan 08, 2020

Contents

> **Warning:** If you migrate from 1.6.X to 1.6.5, and if you have any stored password A security issue has been fixed, you need to update your passwords in databases with this command:
>
> ./manage.py updates_keys

[![Build Status](https://travis-ci.org/lirmm/waves-core.svg?branch=master){]}(https://travis-ci.org/lirmm/waves-core)

README

1.1 What is WAVES for ?

WAVES stands for "**W**eb **A**pplication for **V**ersatile and **E**asy online bioinformatic **S**ervices."

WAVES is a dedicated Django based web application to ease bioinformatic tools integration through web interfaces in order to provide the scientific community with bioinformatic services.

It is aimed to help you easily present, publish and give access on the web to any bioinformatic tool.

1.2 Features

- Create and manage your services execution over platform such as Galaxy, DRMAA clusters (SGE), Direct script execution, API calls to other services, remote calls to other platforms via ssh, etc.

- Easily presents these tools in a nice frontend based on Bootstrap3, Bootstrap4 and soon Material Design

- Follow and manage remote REST API access to your service platform

- Manage user's access to your services

Note:

- WAVES main component is WAVES-core.

- WAVES-Galaxy is another component. It is a WAVES adapter to interact with Galaxy instances.

- WAVES-demo is a custom version of WAVES-core for demo purpose only.

1.3 Side projects

- WAVES-galaxy adapter https://www.github.com/lirmm/waves-galaxy
- WAVES-demo project https://www.github.com/lirmm/waves-demo

1.4 Installation

- You can use WAVES-core as a stand alone application.
- WAVES-core comply to standard reusable project layout for Django. So you may include it as a dependency in your own django project
- Complete documentation available on readthedocs

1.5 Support

If you are having issues, (or just want to say hello): we have a mailing list located at: waves@lirmm.fr

1.6 – UPDATE from <=1.6.4 –

If migrating from a version 1.6.4 or earlier to a most recent, you must patch you database encrypted password to a more reliable encrypted technology (https://github.com/pyca/cryptography). This script must be run once and only once ! To migrate your data please do the following: - retrieve the latest version - stop any running service - save your database first - once the new version is installed, run *manage.py update_keys* - if you don't have any error message: your keys are now more secured !

CHAPTER 2

Authors

Marc Chakiachvili [1,2], Sylvain Milanesi [1], Anne-Muriel Arigon Chifolleau [1], Vincent Lefort [1]

[1] LIRMM, UMR 5506 - CNRS & Université de Montpellier, Montpellier, FRANCE

[2] European Molecular Biology Laboratory, European Bioinformatic Institute, Wellcome Genome Campus, Hinxton, Cambridge, CB101SD, United Kingdom

License (GPLv3)

WAVES-core is free software:

you can redistribute it and/or modify it under the terms of the GNU General Public License version 3 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more specific details see http://www.gnu.org/licenses, the Quick Guide to GPLv3. in the codebase.

The GNU operating system which is under the same license has an informative FAQ here.

3.1 Note to developers

We very much appreciate you using our code to do fun and interesting things with. We hope that while doing this you may find and fix bugs or make enhancements that could be useful for the greater community and will makes the developers aware of them by emailing to waves-webapp@googlegroups.com

WAVES Extensions

WAVES team provides some useful extensions or side project to WAVES-core:

4.1 Waves Demo

Side project to show examples for extending WAVES-core to fit your needs:

- Extends Services , Submission
- Override Service / submissions templates
- Use of REST API forms
- Use a different skin for the back-office interface
- Custom front-end interface
- Override the Authentication class

See demo github repo for more details

4.2 Waves Galaxy

Galaxy dedicated adaptor, allows import and execution of services on remote Galaxy instances

See galaxy-adaptors github repo for more details See Documentation

4.3 Just have a try

This is a Singularity image containing a functional WAVES installation including two pre-configured services ('Hello world' and 'PhyML'). It is a good way to test a fully operating WAVES-core instance. To be used with caution : all data will be lost when singularity instance is stopped.

Singularity installation : on Linux or Mac.

Download WAVES test Singularity image : wavetest.simg

Example for Linux Debian ditribution (Ubuntu 16.04 or later) :

Install Singularity :

```
sudo wget -O- http://neuro.debian.net/lists/xenial.us-ca.full | sudo tee /
↪etc/apt/sources.list.d/neurodebian.sources.list
sudo apt-key adv --recv-keys --keyserver hkp://pool.sks-keyservers.net:80␣
↪0xA5D32F012649A5A9
sudo apt-get update
sudo apt-get install -y singularity-container
```

Get and use wavestest.simg (caution, you need to be sudoer to start an instance) :

```
wget http://www.atgc-montpellier.fr/download/binaries/waves/wavestest.simg
sudo singularity instance.start wavestest.simg waves
sudo singularity run instance://waves
```

When the instance is launched, WAVES-core is running. Open localhost:8000 on your favorite browser. Login with "admin" and "motdepasse".

Power off :

```
sudo singularity instance.stop waves
```

CHANGELOG

5.1 Version 1.6.7 - 2020-01-08

- [Mails] - Ignore settings for admin email - force sending
- [Services Templates] - Integrate JSON-LD into services displayed pages.

5.2 Version 1.6.6 - 2019-09-12

- [Commands] - Fix Error in command to migrate to new encryption lib
- [Queue] - Added Celry/Redis option to process job queue

5.3 Version 1.6.5 - 2019-06-15

- [Encryption] - MAJOR update on cryptography library in use within WAVES - https://github.com/pyca/cryptography see README
- [Commands] - Removed temporarly wqueue processing - issues with remaining processes

5.4 Version 1.6.x - 2018-03-10

- [Layout] - Corrected missing directories
- [ignore] - Added ignored files
- [Bug] - Corrected display in BO for passwords
- **[USERS] - Added API USER class, unifying authentication, url redirection dedicated to REST Api users**

- Added multi-site association allow returning site url prefix for jobs urls

- Test if job.client user has a 'site' property, in such case, retrieve domain name to generate Job / JobOutputs link

- [LOGGING] JobLogging: degraded mode, output to 'waves.errors' logger

- [Service list] Corrected: list services fails when services are not all public (1.1.9.2).

- [Admin JobList] Corrected: remove wrong filter on jobAdmin queryset (1.1.9.3)

- [API ServiceList] Added: added service_app_name to returned json

- [API Submission] Changed: api_name => submission_app_name in returned json

- [API ServiceList] Changed: changed format for service submission list

5.5 Version 1.5.x - 2018-02-10

- [JS] Corrected: event association on add new input in submission form

- [Daemon] Corrected: log any fatal exception during job processing in job file

- [Admin Service] Corrected: key error when editing service on 'created_by'

- [Admin Service] Corrected: inline popup on add ExitCode

- [Settings] Added: check for WAVES directories access rights

- [File Permission] Changed to 775/664 job created dirs/files.

5.6 Version 1.3 - 2018-02-07

- [Updated] - shared logging behaviour in Jobs / Importer

- [Corrected] - service import method

5.7 Version 1.2 - 2018-01-31

- [waves.front] Removed: waves.front, merged back inside wcore

- [Admin] Added: admin urls directly in related Wcore ModelAdmins

- [Models] Added: fields authors and citation to Service model

- [API] Added: Add Job cancel and delete services

- [Models] Removed: removed fields api_on and web_on in Service model

- [Jobs] Changed: JobInconsistentState treatment in job workflow

- [Models] Added: 'topics' and 'operations' property for reading EDAMS ontology data in Service templates

- [Adapters] Moved: Saga common processes in dedicated class SagaAdaptor

- [FileInput] Added: Configuration for enabling/disabling copy/paste form element

- [Docs]: Introduce Administrator guide / Developer guide

- [Api]: Introduce standard token authentication for API form integration

- [Db]: Removed migrations files - causing fails migrate with overridden Submission model - added makemigration upon install

5.8 Version 1.1.5 - 2017-11-30

- Corrected: .type property on JobInput
- Added: filter_fields on job ViewSet
- Added: job instance attribute in JobSubmissionViewSet (create_job)
- Added: submission label in Job detail api results

5.9 Version 1.1.4 - 2017-10-24

- Corrected fixture loading conflict with signals on api_name duplicate check

5.10 Version 1.1.3 - 2017-10-18

- Corrected bugs in Galaxy tool import
- Changed importer API to resolve problem with included runner / adapter params
- added SRV_IMPORT_LOG_LEVEL to configuration

5.11 Version 1.1.2 - 2017-09-30

- Added pip package (waves-core) - updated 2017/10/03
- Added changeLog in documentation
- **Added BinaryFile upload:**
    - Association with Runners / Services / Submission for command lines
- **Added standard Django inclusion tags to display Submissions forms templates**
    - {% load waves_tags %} => {% submission_form %}, {% service_inc "css" %}, {% service_inc "js" %}
    - Templates directories structure in order to use all available crispy templates packs
    - Check crispy configuration on startup
    - Dynamic includes tags for related css and js from cdnjs
- Simplify overrides for front templates
- Cleaning code to PEP8 standards
- Updated install documentation
- Make Submission model swappable as well
- Added Popup Edit for Submission params Model Admin
- Added quick and dirty solution to override services templates for specific service

- **Corrected bugs**

  - JsPopupInlines for Django-jet admin layout

  - workflow Runner tests

  - daemon command failed with SQLite DB (depends now on daemons package *https://pypi.python.org/pypi/daemons/1.3.0*)

5.12 Version 1.1.1 - 2017-07-30

- Corrected many bugs from beta

- api v2

- decoupled front / core templates

- Make Service model swappable (for overriding capabilities in other apps)

- Removed un-mandatory dependencies (django-constance, grappelli, django-jet) - added compat files

5.13 Version 1.1.0 - 2017-03-30

Initial Beta version

Installation

> **Warning:** To run WAVES-core, it's strongly recommended that you setup a dedicated user, because WAVES-core run with saga-python, and this module need to create some directories you might not be able to create (.radical and .saga) with another user (such as www-data)

> **Warning:** WAVES-core was initially developed with Python 2.7 and Django 1.11 and is currently not tested on latest version (Python 3 and Django 2.0).

You can install WAVES-core either as a stand alone application, or inside any existing Django project

6.1 0. Prerequisites

> **Note:**
>
> **In order to install WAVES-core you will need:**
>
> - python 2.7.X (WAVES-core is not yet compatible with python 3.5)
> - pip package manager (required packages : python-pip python-dev build-essential)
> - A web server: Apache or NGINX
> - A database backend (Mysql or Postgres) but by default WAVES-core runs with sqlite

WAVES is developed with Django. You may need to know a little about it.

6.2 1. Install WAVES-core as stand alone application

1.1. Clone repository:

Current version is master (master)

```
user@host:$ git clone https://github.com/lirmm/waves-core.git [your_app]
user@host:$ cd [your_app]
```

Note: To checkout a particular version:

user@host:~your_app$ git checkout tags/[VERSION]

1.2. Install dependencies:

```
user@host:~your_app$ virtualenv .venv
user@host:~your_app$ source .venv/bin/activate
(.venv) user@host:~your_app$ pip install -r requirements.txt
```

You might need other dependencies if working with other DB layout than sqlite.

1.3. Install database:

```
(.venv) user@host:~your_app$ ./manage.py check
(.venv) user@host:~your_app$ ./manage.py makemigrations wcore (may only␣
→display "No changes detected in app 'wcore'")
(.venv) user@host:~your_app$ ./manage.py migrate
(.venv) user@host:~your_app$ ./manage.py createsuperuser (then follow␣
→instructions)
```

1.4. If everything is ok:

You can start your test server and job queue like this:

```
(.venv) user@host:~your_app$ ./manage.py runserver
(.venv) user@host:~your_app$ ./manage.py crontab add
```

Go to http://127.0.0.1:8000/admin to setup your services WAVES-core comes with default front pages visible at http://127.0.0.1:8000

See also:

Django crontab for other crontab setup

Note: From previous release, a known bug occured while using wqueue command. This bug block you from using this daemon queue. Please use "crontab" instead, and contact us if you experience issues.

1.5.(optional) Use celery and redis as job queue

Using the combination of waves job queue and crontab can have some limitations. For example, with this configuration, the minimum refresh interval is one minute. If you need smaller refresh interval you can use the combination of celery and redis as an asynchronous job queue.

1.5.1. Install the redis server

Redis is an in-memory database that allows to store the tasks.

```
user@host:~your_app$ sudo apt-get install redis-server
```

1.5.2. Install dependencies

```
(.venv) user@host:~your_app$ pip install django-celery-beat==1.5.0
(.venv) user@host:~your_app$ pip install celery==4.3.0
(.venv) user@host:~your_app$ pip install redis==3.2.1
```

1.5.3. Launch job queue

```
(.venv) user@host:~your_app$ celery -A waves_core worker -l INFO
(.venv) user@host:~your_app$ celery -A waves_core beat -l INFO --
→scheduler django_celery_beat.schedulers:DatabaseScheduler
```

If the command crontab add was launched before, it is necessary to remove the crontab task:

```
(.venv) user@host:~your_app$ ./manage.py crontab remove
```

1.5.4. Add "django celery beat" to the project

Thanks to "django celery beat" library, you can configure the recurrent tasks directly in the administrator area. According to add this application to the project, you need to add the application in the configuration file (~your_app/waves_core/settings.py).

```
INSTALLED_APPS = [
    'polymorphic',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'waves.wcore',
    'waves.authentication',
    'crispy_forms',
    'rest_framework',
    'corsheaders',
    'adminsortable2',
    'django_crontab',
    'django_celery_beat',
]
```

1.5.5. Add periodic tasks to the database

Apply Django database migrations so that the necessary tables are created. Then, add some entries in the database with default configuration.

```
(.venv) user@host:~your_app$ ./manage.py migrate
(.venv) user@host:~your_app$ ./manage.py loaddata fixtures_celery_
→beat.json
```

Visit the Django-Admin interface to set up some periodic tasks.

In the periodic tasks pannel, the available tasks are job_queue and purge_jobs, two functions present in the tasks.py file of wcore application. Configure each of them with the desired intervals and saved.

6.3 2. Install WAVES-core inside existing Django project

To create a Django project, have a look at Django tutorial

See also:

WAVES-core is a reusable app see: https://docs.djangoproject.com/en/1.11/intro/reusable-apps/#your-project-and-your-reusable-app

2.0. Setup a virtualenv for your project:

```
virtualenv ~/.venv/[waves_env]
```

2.1. Install waves package:

Use pip to install waves-core as third party package.

```
pip install waves-core
```

If you want to install the latest development version (at your own risk :-)) `pip install -e git+https://github.com/lirmm/waves-core.git#egg=waves-core`

2.2. Activate WAVES-core in settings:

WAVES-core application has minimum dependencies to:

- Django polymorphic
- Crispy forms
- Django Rest Framework

Optionally, WAVES-core can use:

- Django CkEditor
- Django Admin sortable 2
- Django Jet

Add required dependencies to your INSTALLED_APPS, you should at least find these in your project:

```
INSTALLED_APPS = [
    'polymorphic', # mandatory
    ...
    'waves.wcore', # mandatory
    'waves.authentication', # mandatory if API token access needed
    'crispy_forms', # mandatory
    'rest_framework', # mandatory
    ...
    'rest_framework.authtoken', # optional see http://www.django-rest-
↪framework.org/api-guide/authentication/#tokenauthentication
    'corsheaders', # optional see https://github.com/ottoyiu/django-cors-
↪headers
    'adminsortable2', # optional see https://django-admin-sortable2.
↪readthedocs.io
    ...
]
```

2.3. Include the services urls in your project urls.py:

```
url(r'^waves/', include('waves.wcore.urls', namespace='wcore'))
url(r'^waves/api/', include('waves.wcore.api.urls', namespace='wapi'))
```

2.4. Create your database:

```
python manage.py makemigrations wcore
python manage.py migrate wcore
python manage.py check
```

2.5. Extra configuration:

Depending on your needs, you might want to expose WAVES API to any registered user, if so have a look at: Corsheader to allow cross-origin Resource Sharing

Some WAVES-core API services requires authentication, see DRF authentication for authenticating methods API POST calls

---

Note: WAVES-core allows simple "api_key" authentication with standard Token Authentication processes, to use it simply add 'waves.authentication' in INSTALLED_APPS.

This then allow to call WAVES API services with a api_key:

- with Authorization token header

- with GET / POST parameter with api_key value.

Each authenticated api service need a valid Authorization header as explained here: http://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication

To use this service with apache in mod_wsgi: please mind to enable "WSGIPassAuthorization On" parameter in conf

---

6.4 3. Use other than SqlLite default DB layer

You may need to install the Python and MySQL development headers and libraries like so:

- sudo apt-get install python-dev default-libmysqlclient-dev # Debian / Ubuntu

- sudo yum install python-devel mysql-devel # Red Hat / CentOS

- brew install mysql-connector-c # macOS (Homebrew) (Currently, it has bug. See below)

On Windows, there are binary wheels you can install without MySQLConnector/C or MSVC.

Then install pip mysql package in your virtualenv:

```
pip install mysqlclient
```

See also:

https://docs.djangoproject.com/en/1.11/ref/databases/

Administrator Guide

Services are defined by administrators, then configured to allow users to gain access to submission web forms and REST api entry points.

Everything is made simple with help of dedicated back office entries. There, administrators firstly describe services in term of name, description, authors, citation links, edams ontology topics and operations, computing infrastructure configuration (see "Execution adaptor").

Once service main data are setup, administrators can configure precisely related submission to setup inputs, execution parameters, outputs, etc...

Upon service publication, it is automatically available in two ways: standard web pages including html forms, and a REST API service entry point to be remotely accessed from a distant application.

Some more configuration can be made to tune user's access rights, execution parameters etc. Those are further detailed in this document.

7.1 Computing infrastructures

Computing infrastructures shortly named "Runner" are the entry point where you can setup different configuration on where and how services' jobs may be run.

7.1.1 Environment list

List all currently registered environment available on your WAVES application, you may add / edit / remove (cautiously).

7.1.2 Environment details

Main panel

On detailed environment page, configure some descriptive parameters :
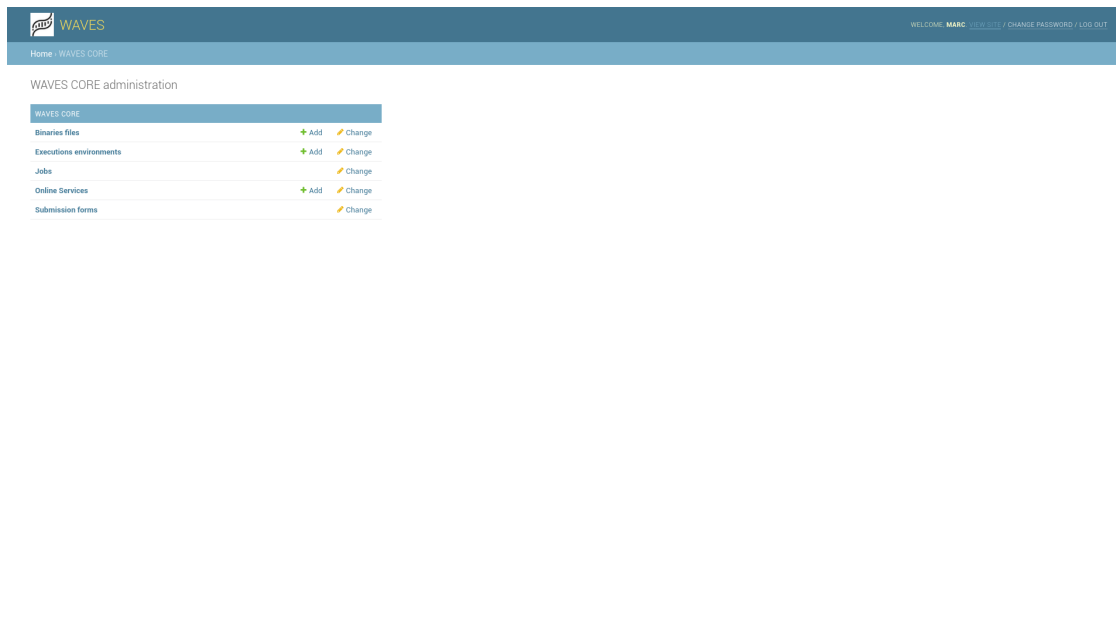
Fig. 1: Django classic back-office landing page for WAVES-core module

Fig. 2: List of environment already set

Fig. 3: Detail admin page

- **Label**: The displayed runner name used in front / back-office for reference (used some time in templates)

- **Run on**: Specify here which WAVES-core adapter is used for running jobs

- **Connexion string**: The used connexion string (readonly)

- **Reset related services**: When checked, upon save, all related services configuration is reset to defaults parameters

---

Caution: These services are now in stage 'Draft'

Environment setup

You can set 'run configuration' values such as login/password, destination host, etc... depending of the WAVES adapter you select in previous panel

---

Note: You can't set up your environment till you have saved your initial configuration once.

On the top left corner, once configured, a button allows you to test your parameters in order to verify if WAVES-core can actually connect to the Computing infrastructure.

Hint: You can prevent subsequent service(s) to override a value in their own configuration administration page, by checking 'Prevent override' related checkbox.

Fig. 4: Computing infrastructure init parameters

Running services

Down the page, there is a list of current services which use this Computing infrastructure.

7.2 Services administration

A service is a bioinformatic tool available online through the http protocol. It can be accessed from a web form or through REST API calls.

7.2.1 Service list

This is landing page when you click on 'Services' Links in Admin home page, you can see current list of services registered on your platform.

Click on **+ Add Service** to create a new service

7.2.2 Service details

General options

- **Service name**: Service name displayed on front or api.
- **Created by**: Only superuser can change this value, this is set by default to current user.
- **Version**: Current version for your service (no relation with actual software version).
- **Status**: Current online status for this service, upon creation, it's automatically set to 'Draft'.

Fig. 5: Main general options for a WAVES-core service.

- *DRAFT*: Service is under configuration, by now, it is not intended to be available to anyone except the service's creator.

- *STAFF*: Service configuration is finished (inputs / outputs / run configuration), it then can be open to the others team users, i.e back-office users.

- *REGISTERED*: Service is fully configured, tested, but restricted to registered users (those who have a Django activated account).

- *RESTRICTED*: Service is intended to be used by specific registered users. WAVES-core allows to set up these users by specifically selected them in service configuration back-office service page.

- *PUBLIC*: Service is open to any user who visit the website, still, access to REST API is subjected to user registration prior to use its capabilities.

- **App short code**: this value is used for generating urls and api entry points. For a service, this value must be unique.

- **Short description**: Short description text about what service is about (not displayed on front but only on api).

- **Computing infrastructure**: Execution configuration (see *Computing infrastructures* administration).

- **Binary file**: You can upload here the executable file which will be used for execution.

> **Caution:** Modify *app short code* attribute when service is online can break api clients
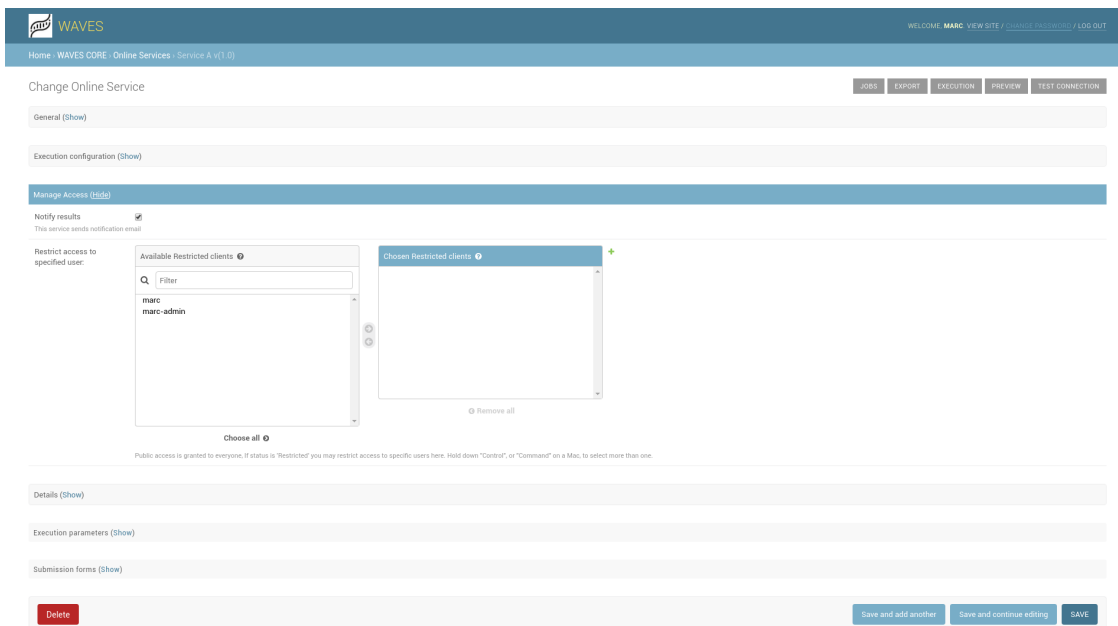
Access management

Fig. 6: Access panel presents granted given to Service.

- **Notify results**: Whether or not users are notified when job is terminated.

- **Access restriction**: When service's status is *'RESTRICTED'*, you may set up allowed users for this service.

Service details

Fig. 7: Optional detailed information for your service

- **Created on**: Creation date (automatic).

- **Last update**: Update date (automatic).

- **Description**: A longer description about your service, may include some HTML content (you may add CKEditor as a dependency for your project).

- **Edams topics**: A list of comma separated edam topics reference.

- **Edams operations**: A list of comma separated edam operation reference.

- **Remote service tool id**: Some remote computing platform may add a required id, once your service is deployed (automatic).

Service execution configuration

You can set 'run configuration' values for each expected parameters for service execution, one is always required: 'command'.

Hint: You can prevent subsequent submission(s) to override a value in their own configuration administration page, by checking related 'Prevent override' checkbox.

7.3 Submission administration

Many bioinformatic tools provide several distinct usages. For instance, a program can be run using the command-line interface or by providing a configuration file. Otherwise, the same tool can be run on different computing infrastructures.

A submission is therefore the combination of a usage and a computing infrastructure.

Thus, a service can rely on different submissions.

7.3.1 Access submission

WAVES-core provides submission administration available within the back-office.

From the Services list

Each service item from the services list provides direct link to their related submissions.

From the Service details page

Here you can manage some parameters directly, or create a new submission for service.

Note: When creating a new submission, you must first click *'Add another Submission method'*, fill the label field, and click on *'Save and continue editing'*. After this operation, you can access submission details page with the "change" link provided in list.

Note: You can still list all submissions, from the main menu for the WAVES-core app admin main page.

7.3.2 Submission details

For each service, you can setup multiple ways to submit a job. For example, your service may have a 'standard' inputs list, and some 'experts' one for another type of jobs. Inputs in each submission configuration are not necessarily correlated.

For each 'Submission configuration', you can add, remove, order possible inputs.

Note: Note on 'form' api entry point : When calling service for loading submission form, returned content is not JSON but standard HTML document. It allows to retrieve the content of the form usually displayed in a web browser, in order to integrate the form in another page issued from any other system. Form is then submitted normally on the 'Create a new job' entry point.

General information

Some information are displayed, more for information than for modification, but you can set up related service, label, availability (API related) and app short code (depending on your profile).

Warning:

- Changing related service can have strange side effect, but could be useful sometime. Use it carefully.

- Changing app short code change the final API uri for this submission. Once online, it could break your REST client.

Run configuration

General Use case

Most of the time, a 'service' needs only one 'submission method', defined to run 'jobs' on one single 'computing infrastructure'. Once configured, service jobs submitted by users are sent to the computing infrastructure with specified run parameters and user inputs. The job is then monitored during its execution to check for its different status, once job finish its execution, WAVES-core system retrieve results and store them on the platform to be made available online (HTML page or REST api entries).

Sometime, administrators want to run one submission on different calculation devices, or with different default parameters, WAVES-core allows then to create more than one submission for a single service with different configuration available.
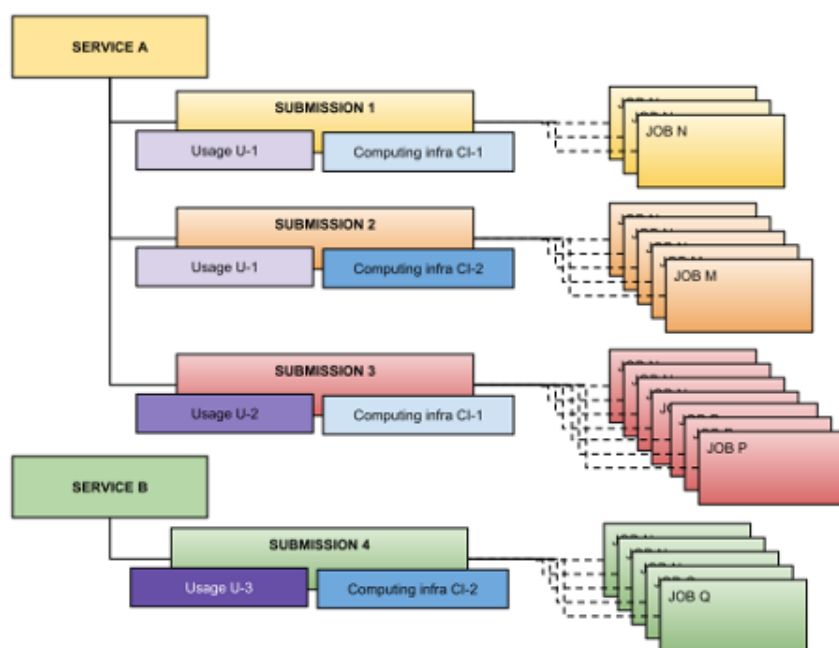
If you do so, after a save, you can access run configuration for this submission, exactly as seen in service *execution configuration*

7.3.3 Submission Inputs

The submission input panel shows all registered inputs.

You can modify some information on existing params directly in list. You can sort inputs by drag&drop on the left side.

Service input creation

Click on 'Add another input' to create a new input, it opens then a popup window to setup your new input.

In this first popup, you must choose type for your new submission input.

Add Input

Type:	◉	Text Input
	○	File input
	○	Boolean choice
	○	Decimal
	○	Integer
	○	List

SAVE

When type is selected, you enter the details information for your input. All inputs share the following information :

- **Label**: displayed label for your input.

- **Name**: the actual parameter name used in command line for job submission.

- **Command line format**: to generate expected command line, you set here the parameter type:

  - *Assigned named parameter*: [name]=value

  - *Named short parameter*: -[name] value

  - *Named assigned long parameter*: –[name]=value

  - *Named short option*: -[name]

  - *Named long option*: –[name]

  - *Positional parameter*: value

  - *Not used in command line*: to set up a condition for other parameters, but not used in job command line.

- **Required**: Set whether submission input is mandatory, optional, or not set by service user.

- **Help Text**: Displayed on form to help user to set values.

- **Multiple**: Set whether this input may hold multiple values (for example, multiple files inputs).

- **App short code**: Set input short code for api submission (set up automatically if not filled).

- **Default value**: The default value for this input.

Some other fields are displayed depending on input type.

Text input

- **Max Length**: Set up max length allowed for this text input

File input

- **Allowed copy paste**: Allow or not display in form, a Text field for copy/paste content added to upload file input
- **Allowed file size**: Max allowed size for file input
- **Edam format(s)**: Input file EDAM ontology format
- **Edam data(s)**: Input fil EDAM ontology data type
- **Validation Regexp**: For expert, set up a RegExp for validating file input names

File input adds another section where administrator can setup file sample that can be used in submission. Each Sample defines:

- **Label**: A displayed label
- **Sample file**: upload here sample file

Boolean input

- **True value**: Used value when boolean is set to True, default is "True"
- **False value**: Used value when boolean is set to False, default is "False"

Add File input

General (Hide)

Label:

Input displayed label

Parameter name:

Input runner's job param command line name

Command line format: Named short parameter: -[name] value ▾

Command line pattern

Required: Required ▾

Submitted and/or Required

Allow copy paste in forms   ☐
Set whether file input field should add a copy/paste text field

Allowed file size :   20480

in Ko

Filter by extensions: *

Comma separated list, * means no filter

More (Hide)

Help Text:

Multiple   ☐
Can hold multiple values

App short code:

App short code, used in url, leave blank for automatic setup

Default value:

Edam format(s):

comma separated list of supported edam-format

Edam data(s):

comma separated list of supported edam-data-param_type

Validation Regexp:

Dependencies (Show)

Input samples (Hide)

Input sample: #1   ⊗

Input Label:

Sample file: Choose File   No file chosen

＋ Add another input sample

Sample dependencies (Show)

SAVE

Add Boolean choice

General (Hide)

Label:

Input displayed label

Parameter name:

Input runner's job param command line name

Command line format: Named short parameter: -[name] value ▾

Command line pattern

Required: Required ▾

Submitted and/or Required

True value: True

False value: False

More (Hide)

Help Text:

Multiple   ☐
Can hold multiple values

App short code:

App short code, used in url, leave blank for automatic setup

Default value:

Dependencies (Show)

SAVE

Decimal parameter/Integer parameter

- **Min value**: Set up min value expected for this input

- **Max value**: Set up max value expected for this input

- **Step**: For HTML5 navigator, set up step when using number input controls up and down

List parameter

- **List display mode**: Set if list is displayed as a select box (default), check boxes, or radio buttons

- **Elements**: Field where to specify list labels and values. One element per line, separated with |
  special character

Inputs dependencies

Sometime, your services may setup dependencies between inputs, for example, if you setup a service which use DNA or Protein substitution models, you may want to change these models upon selection of type of data.

So, to help you, WAVES-core allows to add "Related input" to a service input (down Service Input form part), where you can set exactly the same values as for a normal input, **plus** :

- *When condition* : Activation value (from 'parent' Input), if parent is a list, correspond to selected value

> **Warning:** Related inputs can't be 'mandatory', because their submission is dependent on another one which is potentially not set

7.3.4 Submission Outputs

Along with your service inputs, you setup all expected outputs for each submission's job.

A service output is defined by:

- **Label** : The displayed name for your output
- **File Name or Pattern** : output file name, may contain a '%s' pattern referencing associated input value for creating file name
- **Extension** : Expected file output extension
- **App short code**: Set output short code for api output retrieval (automatically set if not filled)
- **Edam format**: Output EDAM ontology format
- **Edam data**: Output EDAM ontology data type
- **From Input** : script uses some inputs values to setup outputs file names, set corresponding input here
- **Help Text** : Associated text to output, may be displayed on job result page

7.3.5 Submission ExitCode

WAVES-core defines automatically the two exit codes "0" for normal process exit code, "1" for process error.

This is used in job run processing to declare a job as eventually successful or not

WAVES-core allows you to define more exit codes as needed by your script.

Here you can define:

- **Exit Code Value**: expected exit code, should be an int value
- **Message**: Exit code associated message, may be used on job result page.
- **Is and error**: Set whether job is marked as ERROR if exit code is meet

WAVES

WELCOME, **MARC**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › WAVES CORE › Submission forms › simple

✔ Param successfully saved

Change Submission form

ADD A NEW PARAM   EDIT SERVICE   PREVIEW   SERVICE'S LIST   HISTORY

General (Show)

Run (Show)

Inputs (Show)

Outputs (Hide)

Expected output: Copied file   ☐ Delete

General (Hide)

Label: Copied file
                        Label

File name or name pattern: %s
                        Pattern is used to match input value (%s to retrieve value from input)

Extension:
                        Leave blank for *, or set in file pattern

More (Hide)

App short code: copied_file
                        App short code, used in url. leave blank for automatic setup

Edam format:
                        Edam ontology format

Edam data:
                        Edam ontology data

From input: Output name (TextParam) ▼ ✎ ✚
                        Is valuated from an input

Help Text:

＋ Add another Expected output

Exit Codes (Show)

Delete   Save and continue editing   Save and back

Exit Codes (Hide)

EXIT CODE VALUE	EXIT CODE MESSAGE	IS AN ERROR	DELETE?
0 Process exit normally...			
0	Process exit normally	☐	☐
1 Process exit error...			
1	Process exit error	☑	☐

＋ Add another Exit Code

7.4 Job queue management

7.4.1 Job List

Jobs are listed in first screen when you click on **jobs** link in main backoffice view.

7.4.2 Job Details

On this view you may want to see job online, see associated service, or cancel the job (if possible).

General Information You can check general information about job here, such as :

- Title

- Associated Service

- Current Status

- Creation and last update date

- Associated client if job has been submitted by a registered user

- Email where notifications are sent

- Generated unique slug for this Job

- Current runner where job is actually run

- Generated command line where applicable for runner

Job History Retrieve here all logged events for this job, including administration message (may describe errors).

Job Inputs Designated inputs for this job

Job Outputs Current expected outputs

7.5 How to build a simple service in WAVES-core

Here you can follow a step by step process to create from scratch your first "hello world" service on your computer. We assume WAVES-core is installed (see *Installation* section)

7.5.1 Add a Computing infrastructure

In "WAVES CORE" menu, click on "Add", this will display the "Add Computing infrastructure" page. A second way is to click on "Computing infrastructure" and on the top right button "ADD COMPUTING INFRASTRUCTURE" of the listing page.

Click "Show" behind "Main"

Name the computing infrastructure to create. Caution, this label will be displayed later on services forms, choose an explicit name.

If left empty, it will be automatically named.

> **Run on:** Select "Local script", the *label* is automatically filled with "LocalShellAdaptor".

Click "Save".

You may check by clicking the "TEST CONNECTION" button on top right corner.

7.5.2 Add a Service

Go back to main menu, click "Add" on "Services" line or click "Services" and "ADD SERVICE" on the top right of the listing page.

Job outputs (Hide)		
NAME	OUTPUT VALUE	FILE PATH
Standard output - job.stdout View on site		
Standard output	job.stdout	/home/marc/work/waves_core/data/jobs/322ca3a9-0abf-49df-9754-7feda392a1ce/job.stdout
Standard error - job.stderr View on site		
Standard error	job.stderr	/home/marc/work/waves_core/data/jobs/322ca3a9-0abf-49df-9754-7feda392a1ce/job.stderr

Give a name to the service to create : **Service name:** SayHello

And select an infrastructure (the one we've just created) : **Computing infrastructure** LocalShel-lAdaptor

Click "Save and continue editing".

Click "Show" behind "Execution parameters". Add the command to be executed.

> **command** echo

Click "Save and continue editing".

Now, the service is configured to pass the command "echo" on a local shell.

This command requires an input (the string to echo). This is acheived to configuring the submission method.

7.5.3 Modify Submission methods

Click "Show" behind "Submission methods".

Click "Change" on top left of the default method line.

Click "Show" behind "Inputs"

Click "Add another input", a new window popup:

Choose "Text Input" and "SAVE" on the popup window

Fill the "Add Text Input" form :

> **Label:** Some words **Parameter name:** value **Command line format:** Positional parameter: value

Click "SAVE", the popup window closes

Click "Save and back"

The new service is created. We can try it right now !

7.5.4 Try your new service

Click "PREVIEW" on top right button

The popup window displays the automatically generated form.

> **Some words** Hello world

Remember the name of your analysis or the ID created when submitting the job.

Click "Submit a job", the message "Job successfully submitted xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" appears.

Click the bottom right button "Close".

Now, you may check if your job is well executed.

7.5.5 Follow the Job execution

Click "JOBS" on top right button.

This page list all the jobs, find yours by the ID or name previously noted.

Click "VIEW ON WEBSITE" on the top right button.

Look at job details, on "Standard output" you can note your "Hello world" was echoed successfully.

Contributing

You can contribute to WAVES project with following repositories:

- Git source code: https://github.com/lirmm/waves-core
- Issue tracker: https://github.com/lirmm/waves-core/issues
- Mailing list: waves-webapp@googlegroups.com

CHAPTER 9

Developer Guide

9.1 Definitions

9.1.1 Computing infrastructure

It is composed of computationally dedicated hardware and the software components required to operate it, including calculation management programs (distributed resource management systems, Galaxy,...).

9.1.2 Adaptor

It is a Django module allowing WAVES-core to communicate with a specified computing infrastructure. For each computing infrastructure, WAVES-core needs a dedicated adaptor.

9.1.3 Service

A service is a bioinformatic tool available online through the http protocol. It can be accessed from a web form or through REST API calls.

9.1.4 Submission

Many bioinformatic tools provide several distinct usages. For instance, a program can be run using the command-line interface or by providing a configuration file. Otherwise, the same tool can be run on different computing infrastructures. A submission is the combination of a usage and a computing infrastructure. Thus, a service can rely on different submissions.

9.1.5 Job

A job stands for a command with parameters. It is run on a dedicated computing infrastructure. It may require inputs such as files. It generates outputs: exit code, standard output and standard error, and possibly result files. A job is run each time a submission is invoked by a service.

9.1.6 User

A user is a client which accesses services. It can be a real person using a web browser or a software using the REST API.

9.1.7 Administrator

An administrator is a privileged user with granted access to the WAVES-core back-office. He manages configurations, services, submissions, adaptors and jobs.

9.2 Documentation

9.2.1 WAVES Deveveloper Guide

Job Workflow

Create a WAVES-core adaptor

The base abstract class "JobAdaptor" defines methods to manage a simple job execution workflow.

First of all, override class `__init__(self, *args, **kwargs)` method if you need more params to create your Adaptor instance. Then accordingly, override `init_params(self)` property function in order to return a dictionary for each entry expected in constructor. It allows WAVES-core administration module to automatically load these entries in computing infrastructure *configuration parameters panel*.

> **See also:**
>
> Look at source to find what to override and how it's already made in WAVES-core adaptors *Sources*

- `connect(self):` Process the connection to the calculation device

    - Override `_connect(self)` method to implement your own connection protocol implementation

- `disconnect(self):` Disconnect process from calculation device, may do some cleaning on device as well

    - Override `_disconnect(self)` method to realize cleaning on disconnect

- `prepare_job(self, job):` Job state must be "Created", this method is in charge of preparing job, job is now "Prepared"

    - Override `_prepare_job(self, job)` to prepare job for your needs

        * Create job expected output files

        * Possibly upload files to remote platform
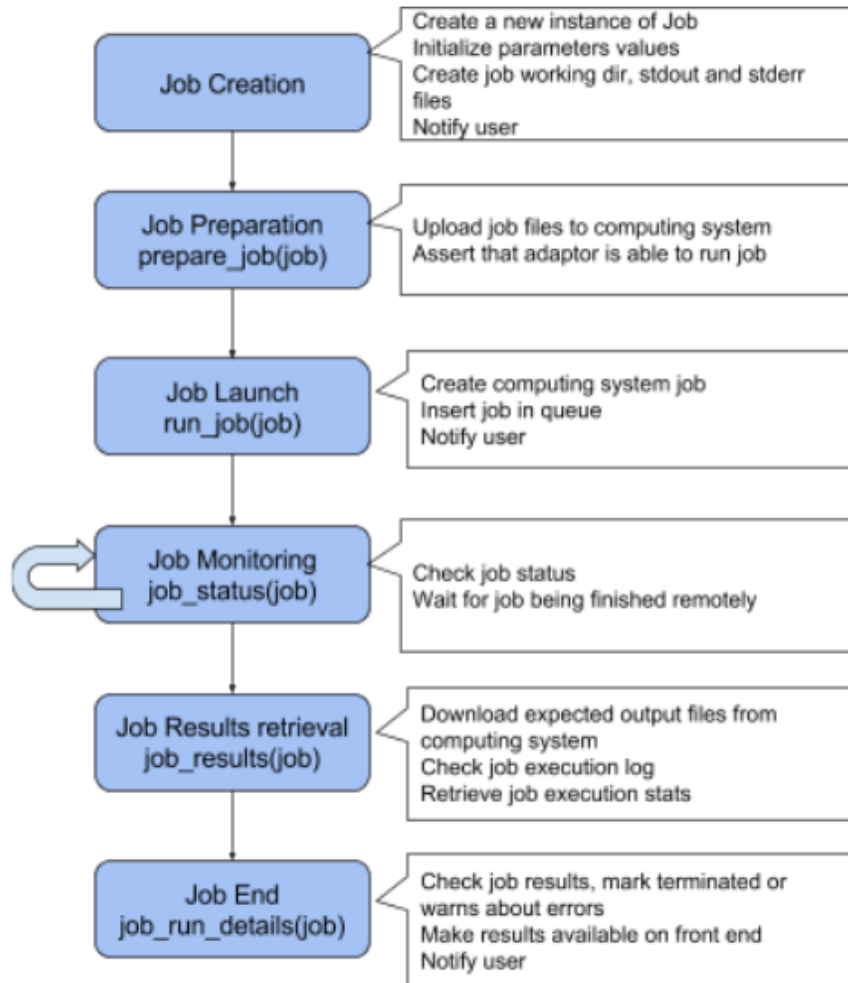
Fig. 1: Classic Job execution workflow in WAVES-core

- `run_job(self, job)`: Job state must be "Prepared", actually create job on computing infrastructure, queue it for execution Job is now "Queued".

    - Override `_run_job(self, job)` to launch job execution on your dedicated platform

---

Note: The job execution workflow is then relayed to computing infrastructure, WAVES does not intend to be a workflow manager, supervisor.

- `job_status(self, job)`: Job current status check, map WAVES-core status map to computing infra. Return current Job.

    - Override `_job_status(self, job)` to retrieve job status from your platform (should return an item mapped in _state_map

- `job_results(self, job)`: Once job is "remotely" finished, get (possibly download) the expected outputs from computing infra to job working dir.

    - Override `_job_results(self, job)` to retrieve job outputs and get them back to WAVES-core platform

- `job_run_details(self, job)`: Upon results retrieval, get job stats on computing infrastructure

    - Override `_job_run_details(self, job)` to create a JobRunDetail object with your job stats

- `cancel_job(self, job)`: Try to cancel job on computing infrastructure

    - Override `_cancel_job(self, job)` to perform job cancellation on your platform

Each of the preceding method definition calls an inner method prefixed by '_' (_connect, _disconnect, etc.) meant to be overridden in subclasses to actually process the action on computing infrastructure. Furthermore, an adaptor need to declare a simple dictionary mapping computing infrastructure job states code to WAVES-core ones : _states_map = {}.

WAVES-core uses constant for defining its jobs states as follows (available in waves.wcore.adaptors.const.py)

Job states constants

Python const	Int value
JOB_UNDEFINED	-1
JOB_CREATED	0
JOB_PREPARED	1
JOB_QUEUED	2
JOB_RUNNING	3
JOB_SUSPENDED	4
JOB_COMPLETED	5
JOB_TERMINATED	6
JOB_CANCELLED	7
JOB_WARNING	8
JOB_ERROR	9

Class diagram overview
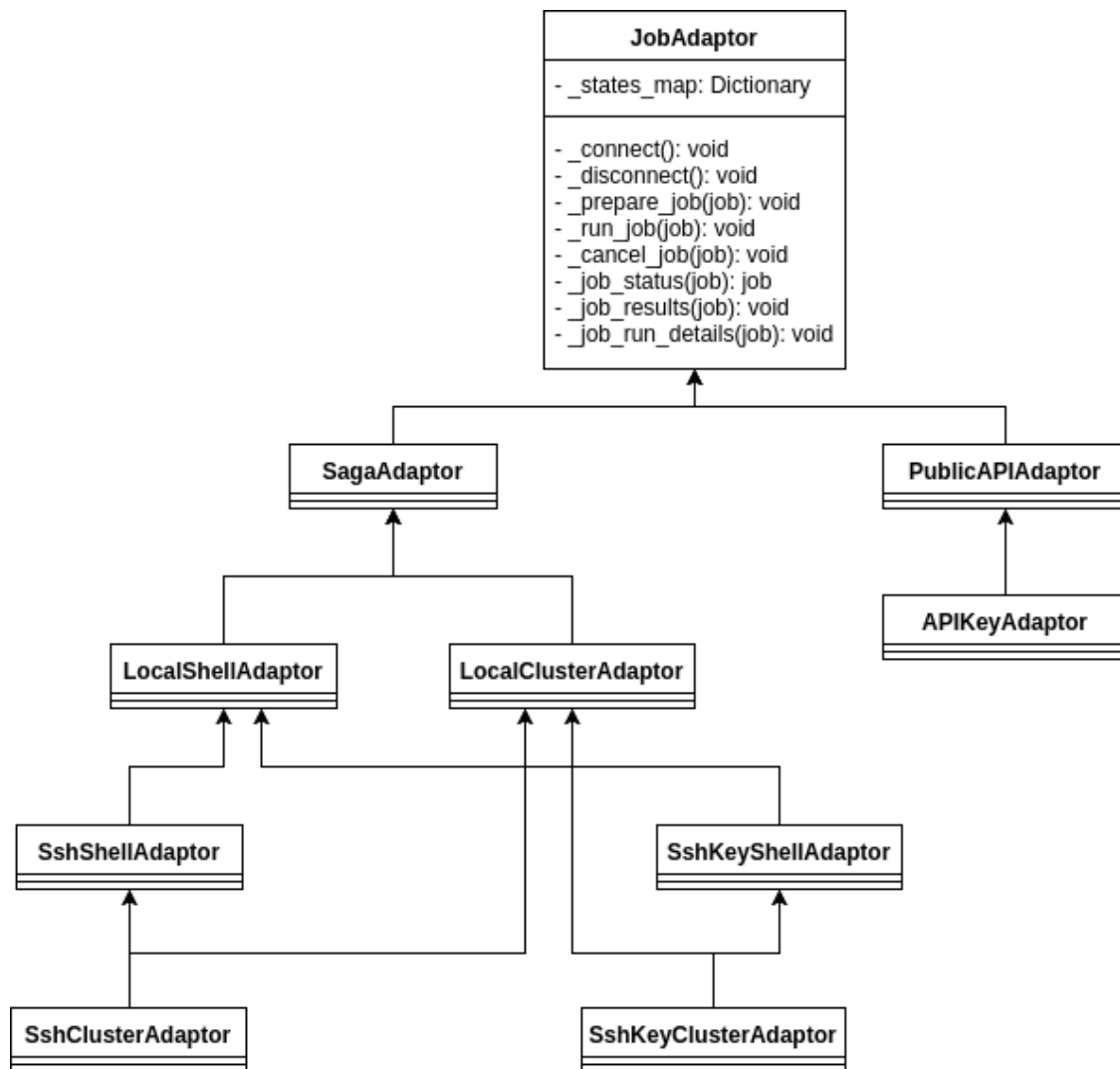
Fig. 2: Adaptor class diagram overview

Currently classes tree implemented in WAVES-core can communicate with a large number of calculation devices, locally or remotely over SSH:

- Sun Grid Engine - now Oracle Grid Engine
- SLURM
- PBS
- CONDOR
- PBS Pro
- LSF
- TORQUE

This is made possible thanks to SAGA Python that implements the GFD interface specification.

Note: A specific adaptor has been created in dedicated app to communicate with a galaxy server

---

Overriding Services and Submissions

Some WAVES-core models classes are easily extensible. WAVES-core offers the possibility to extends two main objects declared in application, in case these does not fit perfectly developers expectations:

- Service: waves.wcore.models.services.BaseService
- Submission: waves.wcore.models.services.BaseSubmission

To extend these models, simply declare your classes in your models, and then declare your classes as new "Service" and "Submission" models in your Django settings.py as follow:

WCORE_SERVICE_MODEL = 'yourapp.YourOverriddenServiceClass' WCORE_SUBMISSION_MODEL = 'yourapp.YourOverriddenSubmissionClass'

Remember to always use shortcut methods "get_service_model" and "get_submission_model" to gain access to model instances.

Note: An example of this capability is available in WAVES-demo project. This example override Service class to add classification with a standard category association, and adds some Meta information to services.

Overriding WAVES templates

Well, as WAVES-core complies to Django reusable app standard, it's pretty straigthforward to extend WAVES-core base templates following Django documentation, each submission configuration results in a dedicated form and a dedicated REST API entry. Thanks to Django framework, rendering forms inside other pages is made easy with standard templatetags dedicated to WAVES-core generated forms.

Related urls

URI	Description
/waves/services/	List all available services
/waves/services/{service_app_name}/	Display Service details
/waves/services/{service_app_name}/new	Create a job (access to available submission(s) form(s)
/waves/jobs/{slug}/	View job details
/waves/jobs/inputs/{slug}/[?export=1]	View Input file online / Download file
/waves/jobs/outputs/{slug}/[?export=1]	View Output file online / Download file

WAVES-core defines the following base templates

Services

Template path	Description
~/waves/services/base.html	Base template used for block definition
~/waves/services/service_list.html	List all available services defined in WAVES apps
~/waves/services/service_details.html	Main service page defined in WAVES apps
~/waves/services/service_form.html	Page to display service's related submissions forms
~/waves/services/file.html	Display a line for a file input / output for service

Hint: WAVES-core allows override for a single service / submission template, following naming convention for templates, simply create a new template in your templates subdir 'waves/override/' (service_app_name is the app_short_code defined in BO for the service):

- For service: service_[service_app_name]_detail.html
- For submission: submission_[service_app_name]_form.html

Jobs

Template path	Description
~/waves/jobs/job_list.html	Display a list of user's jobs
~/waves/jobs/parts/job_list_element.html	A list element template for a job in list
~/waves/jobs/job_detail.html	Job detail page, list submitted inputs parameters and expected outputs

See also:

https://docs.djangoproject.com/en/1.11/howto/overriding-templates/

Overriding API entries

WAVES-core heavily use Django Rest Framework to create api entries for service.

"GET" endpoints are by default accessible without login, POST method (create a job) needs a registered user. You can change this in DRF configuration.

Following standard url patterns definition you may override defaults defined hereafter:

Service endpoints

METHOD	URI	Description
GET	/waves/api/services	List all available services
GET	/waves/api/services/{service_app_name}	Retrieve service details
GET	/waves/api/services/{service_app_name}/form	Retrieve service forms (for all submissions)
GET	/waves/api/services/{service_app_name}/jobs	Retrieves services Jobs (only for logged in users)
GET	/waves/api/services/{service_app_name}/submissions	List all available submissions for this service
GET	/waves/api/services/{service_app_name}/submissions/{submission_app_name}	Get Service submission detailed information (inputs, parameters, expected outputs)
POST	/waves/api/services/{service_app_name}/submissions/{submission_app_name}/jobs	Create new job from submitted inputs
GET	/waves/api/services/{service_app_name}/submissions/{submission_app_name}/jobs	List submissions jobs for this submission
GET	/waves/api/services/{service_app_name}/submissions/{submission_app_name}/form	Service submission form as raw html

Jobs endpoints

METHOD	URI	Description
GET	/waves/api/jobs	List all available user's jobs
POST	/waves/api/jobs/{slug}/cancel	Try to cancel running job on remote calculation device if possible. Mark job as cancelled.
DELETE	/waves/api/jobs/{slug}	Try to cancel job on remote calculation device if possible. Delete Job from DB
GET	/waves/api/jobs/{slug}	Detailed job information
GET	/waves/api/jobs/{slug}/history	Job events history
GET	/waves/api/jobs/{slug}/status	Job current status
GET	/waves/api/jobs/{slug}/input	List job submitted inputs
GET	/waves/api/jobs/{slug}/outputs	List job outputs, associated with direct link to associated file

Overriding forms create template packs

Under construction

9.2.2 WAVES API python samples code

These examples use coreapi package functionalities see: http://www.coreapi.org/

Interact with services

You can interact with a WAVES API instance by using coreapi. These examples show how to get a list of services or a service details, a submission or inputs and outputs expected.

```python
from coreapi import Client, auth
from coreapi.codecs import CoreJSONCodec, JSONCodec

decoders = [JSONCodec(), CoreJSONCodec()]
# Create a client client codecs
client = Client(decoders=decoders)
document = client.get('http://waves.demo.atgc-montpellier.fr/waves/api/schema')
```

(continues on next page)

```python
# get service list - replace with actual waves-api urls
serviceList = client.action(document, ['services', 'list'])
print(serviceList)
# get service details
serviceDetails = client.action(document, ["services", "read"], params={'service_app_
→name': 'sample_service'})
print(serviceDetails['name'])
# get service submissions
submissions = client.action(document, ["services", "submissions_list"],
                            params={"service_app_name": 'sample_service'})
# get first submission details
sub_details = client.action(document, ["services", "submission"],
                            params={"service_app_name": "sample_service",
                                    "submission_app_name": "default"})
# get inputs / outputs
expected_inputs = sub_details['inputs']
print(expected_inputs)
expected_outputs = sub_details['outputs']
print(expected_outputs)
```

Authenticate with token

Some WAVES API entries require to be authenticated (jobs list, job details, job submission). Token are given by the administrator.

```python
client = Client(decoders=decoders,
           auth=auth.TokenAuthentication(
               token="6241961ef45e4bbe7bb01a05f938ed9f0f2a3926",
               scheme="Token"))
document = client.get('http://waves.demo.atgc-montpellier.fr/waves/api/schema')
# list jobs
# get job list
job_list = client.action(document, ['jobs', 'list'])
print("job_list", job_list)
if (len(job_list) > 0):
    job_details = client.action(document, ['jobs', 'read'], params={'unique_id': job_
→list[0]['slug']})
    print(job_details['title'])
    print(job_details['created'])
    print(job_details['updated'])
```

Integrate a WAVES service form

You've got a website and you want your visitors could submit jobs? The better way for this is to add in your website using API. Here, you're supposed to know there is a service named "sample_service" defined on demo WAVES instance (as you see in serviceList above).

```python
from coreapi import Client, auth
from coreapi.codecs import CoreJSONCodec, JSONCodec, TextCodec
decoders = [JSONCodec(), CoreJSONCodec(), TextCodec()]
client = Client(decoders=decoders)
document = client.get('http://waves.demo.atgc-montpellier.fr/waves/api/schema')
wavesform = client.action(document, ['services', 'form'], params={"service_app_name":
→'sample_service'}, validate=False, encoding='multipart/form-data')
```

Now, you just render this form into your template (ex. in a django tpl).

> **Warning:** Don't forget to add forms.css and services.js from your waves instance as in this sample.

```
{% block head %}
    {% addtoblock "waves_forms_css" %} <link rel="stylesheet" href="http://waves.demo.
↪atgc-montpellier.fr/static/waves/css/forms.css">{% endaddtoblock %}
{% endblock %}

{% block main %}
<!-- Import the web form as is -->
{{ wavesform|safe }}
{% endblock main %}

{% block footer %}
    {% addtoblock "js" %}
        <script src="http://waves.demo.atgc-montpellier.fr/static/waves/js/services.js
↪"></script>
    {% endaddtoblock %}
{% endblock footer %}
```

Create a job

It's also possible to create a job directly from your client interface. Here we see how to create a job called "Job Name" which use a "default" submission of "sample_service" service. Inputs are defined by expected inputs of the "sample_service". Be aware, "validate=false" is required to submit a file

```python
# submit a job
from coreapi.utils import File
from os.path import join, dirname

with open(join(dirname(__file__), "test.fasta"), 'r') as f:
    inputs = {
        "text_input": "This is text input",
        "input_file": File("test.fasta", f)
    }
    client.action(document, ["services", "submissions", "jobs", "create"],
                  params={
                      **inputs,
                      "title": "Job Name",
                      "service_app_name": "sample_service",
                      "submission_app_name": "default"
                  }, validate=False, encoding='multipart/form-data')
job_list = client.action(document, ['jobs', 'list'])
print(job_list)
```

9.2.3 WAVES API php samples code

You can interact with a WAVES API instance by using Curl library in php. These examples show how to get a list of services, get a service form and create a job. Some WAVES API entries require to be authenticated (jobs list, job details, job submission). Token are given by the administrator.

Create functions for GET and POST requests

First, you can create two function, one for GET requests and the other for POST requests.

```php
function callApiGET($url_api, $token='', $type='json')
    {
        $authorization = "Authorization: Token ".$token;
        $content_type=($type == "html")?'Content-Type: txt/html':'Content-Type:␣
↪application/json';
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url_api);
        curl_setopt($ch, CURLOPT_HTTPHEADER, array('Accept: application/json',
↪$content_type , $authorization));
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        curl_setopt($ch, CURLOPT_POST, 0);

        $curl_response = curl_exec($ch);
        if ($curl_response === false)
        {
            // throw new Exception('Curl error: ' . curl_error($crl));
            $rep = 'Curl error: ' . curl_error($ch);
            var_dump($rep);
        } else
        {
            // For debug :
            //print_r(curl_getinfo($ch));
            $rep = ($type=="html")?$curl_response:json_decode($curl_response);
        }
        curl_close($ch);
        return $rep;
    }

    // Appel l'API WAVES en requête http POST
function callApiPOST($url_api, $token='', $data_string)
    {
        $authorization = "Authorization: Token ".$token;
        $content_type='Content-Type: multipart/form-data';
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url_api);
        curl_setopt($ch, CURLOPT_HTTPHEADER, [$content_type , $authorization]);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
        curl_setopt($ch, CURLOPT_POSTFIELDS, $data_string);

        $curl_response = curl_exec($ch);
        if ($curl_response === false)
        {
            // throw new Exception('Curl error: ' . curl_error($crl));
            $rep = 'Curl error: ' . curl_error($ch);
            var_dump($rep);
        }
        else
        {
            // For debug :
            //print_r(curl_getinfo($ch));
            $rep = json_decode($curl_response);
        }
```

```
        curl_close($ch);
        return $rep;
    }
```

Display the list of services

```
$urlwaves = "http://waves.demo.atgc-montpellier.fr/waves";
$base_api = $urlwaves."/api/";
$api_key = "6241961ef45e4bbe7bb01a05f938ed9f0f2a3926";
$url_api = $base_api.'services';

$tabrep = callApiGET($url_api, $api_key);
$html = '<p>';
$html = 'Here is the list of services<br>name : service_app_name <br>';
foreach ($tabrep as $obj) {
    $html .= $obj->name." : ".$obj->service_app_name."<br>";
}
$html .= '</p>';
echo $html;
```

Integrate a WAVES service form

You can use the WAVES API to integrate a WAVES form to your website. Here, you're supposed to know there is a service named "sample_service" defined on demo WAVES instance.

```
$urlwaves = "http://waves.demo.atgc-montpellier.fr/waves";
$base_api = $urlwaves."/api/";
$api_key = "6241961ef45e4bbe7bb01a05f938ed9f0f2a3926";
$url_api = $base_api.'services';

$html='';
$tabrep = callApiGET($url_api, $api_key);

foreach ($tabrep as $obj) {
    if ($obj->service_app_name == 'sample_service') {
        $url_form = $obj->form;
    }
}

if($url_api!='') {
    $html .=callApiGET($url_form, $api_key, 'html');
} else {
    $html = 'pb with url_api';
}
echo $html;
```

Create a job

Here we see how to create a job called "Job Name" which use a "default" submission of "sample_service" service. We use CURLFile object to transfer the file needed for the analysis.

```
$submit_url = $url_api.'/sample_service/submissions/default/jobs';
$api_key = "6241961ef45e4bbe7bb01a05f938ed9f0f2a3926";

$data = [
    "title"=>"Job Name",
    "input_file"=> new CurlFile('test.fasta', 'text/plain')
    ];

$tabrep = callApiPOST($submit_url, $api_key, $data);

echo '<div id="reponse"><pre>';
print_r($tabrep);
echo '</pre></div>';

$status = $tabrep->status;

if ($status->code==0) {
    $api_html = '<p>You\'re job is submitted. Follow it on <a target="_new" href="'.
↪$tabrep->url.'">'.$tabrep->url."</a>. Warning you have to be logged on admin (due␣
↪to authentication)</p>";
} else {
    $api_html = "<p>An error occurred... call your admin ;-)</p>";
}

echo $api_html;
```

Application custom settings

WAVES-core application defines a waves_settings attributes generated from WAVES_CORE Django settings dictionary.

> **Note:** You may override these values in settings with a dict named WAVES_CORE

Here are the defaults values:

```python
DEFAULTS = {
    'VERSION': __version__,
    'DB_VERSION': __db_version__,
    'DATA_ROOT': join(getattr(settings, 'BASE_DIR', '/tmp'), 'data'),
    'JOB_BASE_DIR': join(getattr(settings, 'BASE_DIR', '/tmp'), 'data', 'jobs
↪'),
    'BINARIES_DIR': join(getattr(settings, 'BASE_DIR', '/tmp'), 'data', 'bin
↪'),
    'SAMPLE_DIR': join(getattr(settings, 'BASE_DIR', '/tmp'), 'data', 'sample
↪'),
    'UPLOAD_MAX_SIZE': 20 * 1024 * 1024,
    'HOST': HOSTNAME,
    'ADMIN_EMAIL': 'admin@your-site.com',
    'ALLOW_JOB_SUBMISSION': True,
    'APP_NAME': 'WAVES',
    'JOBS_MAX_RETRY': 5,
    'JOB_LOG_LEVEL': logging.INFO,
    'SRV_IMPORT_LOG_LEVEL': logging.INFO,
    'KEEP_ANONYMOUS_JOBS': 30,
    'KEEP_REGISTERED_JOBS': 120,
    'NOTIFY_RESULTS': True,
    'REGISTRATION_ALLOWED': True,
    'SERVICES_EMAIL': 'waves@atgc-montpellier.fr',
    'TEMPLATE_PACK': getattr(settings, 'CRISPY_TEMPLATE_PACK', 'bootstrap3'),
    'SECRET_KEY': getattr(settings, 'SECRET_KEY', '')[0:32],
    'ADAPTORS_CLASSES': (
```

(continues on next page)

```
        'waves.wcore.adaptors.shell.SshShellAdaptor',
        'waves.wcore.adaptors.cluster.LocalClusterAdaptor',
        'waves.wcore.adaptors.shell.SshKeyShellAdaptor',
        'waves.wcore.adaptors.shell.LocalShellAdaptor',
        'waves.wcore.adaptors.cluster.SshClusterAdaptor',
        'waves.wcore.adaptors.cluster.SshKeyClusterAdaptor',
    ),
    'PURGE_WAIT': 86400,
    'PERMISSION_CLASSES': (),
    'MAILER_CLASS': 'waves.wcore.mails.JobMailer',
}
```

Source Documentation

11.1 Services

Services are the main entry point for WAVEs application, managed by *Service Manager*.

11.2 Submissions

Services may be accessed from multiple 'submissions'

11.2.1 Submission Inputs

Classes for service's submission inputs information

Booleans

Decimals

Files

Integers

Text Inputs

11.2.2 Submission Outputs

Submission description defines expected outputs

11.2.3 Submission ExitCode

Submission description defines expected exitcode

11.2.4 Input Samples:

Services may provide sample data for their submissions

11.3 Jobs

Jobs models documentation

11.3.1 Job

Manager

11.3.2 Job Inputs

Job Inputs related to inputs defined in service configuration, see *Submission Inputs*.

Manager

11.3.3 Job Outputs

Job outputs related to outputs defined in service configuration

Manager

11.4 Computing infrastructures

Computing infrastructure models are in charge to store the configuration for expected adaptors to run jobs

11.4.1 Runner

11.4.2 Adaptors Parameters

11.5 Shared classes

WAVES-core defines some top level models classes shared among other resources

11.5.1 APIModel

11.5.2 Described

11.5.3 ExportAble

11.5.4 Ordered

11.5.5 Slugged

11.5.6 TimeStamped

11.5.7 UrlMixin

11.6 Managers

11.6.1 Service Manager

11.6.2 Job Manager

11.7 Adaptors

These modules execute job on dedicated platforms, remotely or locally, they are supposed to control job processing once submitted via services submissions

11.7.1 Constants

11.7.2 Exceptions

11.7.3 Adaptor Loader

11.7.4 Utilities classes

Adaptor base class

11.7.5 Shell script related adaptors class

11.7.6 Api related adaptor base class

11.7.7 Cluster related adaptors

Indices and tables

- genindex