Simphony-remote Documentation

Release 2.2.0

SimPhoNy Project

Jun 27, 2022

Contents

The Simphony-remote is web service that allows users to start and work with simphony enabled environments remotely.

Key provided features:

- Isolated working environments using docker containers.
- No install remote access through a web browser.
- Sharing of working sessions.
- Based on community supported open source initiatives (JupyterHub)

CHAPTER 1

Acknowledgments

This software is developed under the SimPhoNy project, an EU-project funded by the 7th Framework Programme (Project number 604005) under the call NMP.2013.1.4-1: “Development of an integrated multi-scale modelling environment for nanomaterials and systems by design”.

CHAPTER 2

Documentation

A quick setup guide is given below; please see the *documentation* for more detailed information.

CHAPTER 3

Deployment (Quick Start)

Basic instructions for a single-user deployment on a local (or virtual) machine are provided below. A more comprehensive deployment documentation, including use of a `nginx` reverse proxy and running as a service can be found *here*

3.1 Single Machine

If you would like to test a deployment of S-R using Docker for CI purposes, then please use the following instructions.

Note: The following instructions assume a clean up-to-date Ubuntu 18.04 or CentOS 7 system with `git` and `make` installed.

1. Retrieve the required repository:

```
git clone https://github.com/simphony/simphony-remote.git
```

2. Make sure that you have a recent version of Docker. This guide has been tested on version 19.03.5 (build 633a0ea838). `make deps` will install the latest version if you do not already have a version of docker available. Full instructions available at the Docker website for Ubuntu and for CentOS operating systems. Docker installed using Ubuntu's Snap package manager might not work as expected; see https://github.com/simphony/simphony-remote/issues/572 for details. A Makefile rule is provided for convenience. **NOTE: this overwrites the docker.list file you might have setup in your /etc/apt/sources.d/ directory**. You might be prompted for the root password to execute this:

```
make deps
```

3. Make sure your docker server is running, and your user is allowed to connect to the docker server (check accessibility of `/var/run/docker.sock`). You obtain this by running:

```
sudo service docker start
```

followed by either:

```
(Ubuntu) sudo addgroup your_username docker
(CentOS) sudo groupmod -aG docker your_username
```

and logging out and in again. Check if your docker server is operative by running:

```
docker info
```

4. Create and activate a virtual environment, then set the appropriate PATH for the node modules:

```
make venv
. venv/bin/activate
export PATH=`npm bin`:$PATH
```

5. Install the python dependencies:

```
make pythondeps
```

6. And install the package itself:

```
make install
```

7. Generate the SSL certificates if you do not already have them. The resulting certificates will have names test.* because they are self-signed and **are not supposed to be used for production**. A CA-signed certificate should be obtained instead. The certificates will be created in the jupyterhub directory:

```
make certs
```

8. Create the database. By default, this is a sqlite file:

```
make db
```

9. If you are using virtual users (users that are not present on the system) you need to create a temporary space where the virtual user homes are created:

```
mkdir /tmp/remoteapp
```

The installation is complete, you can now start the service.

3.2 Start JupyterHub

1. Change dir into `jupyterhub/`:

```
cd ./jupyterhub
```

2. Verify that `jupyterhub_config.py` is correct for your deployment machine setup (see *configuration* for more details). Some example configuration files are provided in the `example_configurations/` directory.
3. Start JupyterHub by invoking the start script:

```
bash start.sh
```

Note: If you want to keep the application running, use screen to start a detachable terminal.

Note: Running on OSX or with a separate docker machine requires that the appropriate environment variables are set before starting jupyterhub. refer to the command `docker-machine env` to setup the appropriate environment. In general, invoking:

```
eval `docker-machine env`
```

will enable the appropriate environment. On Linux, by default the host machine and the docker machine coincide, so this step is not needed.

4. JupyterHub is now running at https://127.0.0.1:8000

   For many browsers this must be typed exactly as shown - using http://127.0.0.1:8000 or localhost:8000 will not work. As mentioned above, the self-signed SSL certificates should cause your browser to raise a warning and require adding 127.0.0.1 to the list of security exceptions.

   Currently, the only fully supported browser is Google Chrome/Chromium. The latest version of Firefox has shown some issues with keyboard input when the vnc is running, however for the most part users will likley not suffer any issues.

3.3 Setting up Docker images

Next, we need to obtain a docker image to run on Simphony-Remote. This can be done by either pulling an existing image from a docker registry, or creating our own locally.

To create new images, please follow the documentation hosted at Horizon 2020 Simphony project repository.

3.4 Setup Database Accounting

A database is needed for managing the remote applications available for each user. Note that this database is in addition to the database created or used by JupyterHub.

Default sqlite database

> **remoteappmanager** by default uses a sqlite database *remoteappmanager.db* in the current work directory. The **remoteappdb** command-line tool is provided for setting up the database. Please refer to the *utilities* section for details on the use of this program.

3.5 Setting up Users

Whilst Simphony-Remote is running using the standard `jupyter_config.py` script, navigate to https://127.0.0.1:8000 in your browser and login with the username 'admin' and no password. Select the 'Users' tab on the left hand menu and click the 'Create New Entry' button in the top right. Make up a username and click submit.

Next, click the Applications tab in the left hand menu and click the 'Create New Entry' button in the top right. We can add the name of any docker image available to the Docker daemon.

Then go back to the 'Users' tab, select the 'Policies' button next to the username. Create a new entry and choose the added image name from the dropdown menu. Nothing else needs to be set, unless you want to mount a directory within the docker container.

Log out of SimphonyRemote (red 'admin' button in the top right) and log in using the username you specified for your new user and no password, you should now be able to start your application!

3.6 Contents

This documentation is for simphony-remote 2.2 (2.2.0)

3.6.1 Deployment

Single Machine

Deployment of the complete system in a single machine/VM.

Note: The following instructions assume a clean up-to-date Ubuntu 18.04 or CentOS 7 system with `git` and `make` installed.

---

1. Retrieve the required repository:

```
git clone https://github.com/simphony/simphony-remote.git
```

2. Make sure that you have a recent version of Docker. This guide has been tested on version 19.03.5 (build 633a0ea838). `make deps` will install the latest version if you do not already have a version of docker available. Full instructions available at the Docker website. A Makefile rule is provided for convenience. **NOTE: this overwrites the docker.list file you might have setup in your /etc/apt/sources.d/ directory**. You might be prompted for the root password to execute this:

```
make deps
```

3. Make sure your docker server is running, and your user is allowed to connect to the docker server (check accessibility of */var/run/docker.sock*). You obtain this by running:

```
sudo service docker start
sudo addgroup your_username docker
```

   and logging out and in again. Check if your docker server is operative by running:

```
docker info
```

4. Create and activate a virtual environment, then set the appropriate PATH for the node modules:

```
make venv
. venv/bin/activate
export PATH=`npm bin`:$PATH
```

5. Install the python dependencies:

```
make pythondeps
```

6. And install the package itself:

```
make install
```

7. Generate the SSL certificates if you do not already have them. The resulting certificates will have names test.* because they are self-signed and **are not supposed to be used for production**. A CA-signed certificate should be obtained instead. The certificates will be created in the jupyterhub directory:

```
make certs
```

8. Create the database. By default, this is a sqlite file:

```
make db
```

9. Change dir into jupyterhub:

```
cd ./jupyterhub
```

and verify that *jupyterhub_config.py* is correct for your deployment machine setup (see *Configuration*).

10. If you are using virtual users (users that are not present on the system) you need to create a temporary space where the virtual user homes are created:

```
mkdir /tmp/remoteapp
```

11. You can now start the service:

```
 bash start.sh

.. note::
  If you want to keep the application running, use screen to start
  a detachable terminal.
```

Note:

Running on OSX or with a separate docker machine requires that the appropriate environment variables are set before starting jupyterhub. refer to the command `docker-machine env` to setup the appropriate environment. In general, invoking:

```
eval `docker-machine env`
```

will enable the appropriate environment. On Linux, by default the host machine and the docker machine coincide, so this step is not needed.

Currently, the only fully supported browser is Google Chrome/Chromium. The latest version of Firefox has shown some issues with keyboard input when the vnc is running, however for the most part users will likley not suffer any issues.

---

12. Visit the site at:

```
https://127.0.0.1:8000
```

For many browsers this must be typed exactly as shown - using http://127.0.0.1:8000 or localhost:8000 will not work. As mentioned above, the self-signed SSL certificates should cause your browser to raise a warning and require adding 127.0.0.1 to the list of security exceptions.

Setup docker containers

To create new images, please follow the documentation hosted at Horizon 2020 Simphony project repository.

Setup Database Accounting

A database is needed for managing the remote applications available for each user. Note that this database is in addition to the database created or used by JupyterHub.

Various accounting sources are supported:

1. Default sqlite database

   remoteappmanager by default uses a sqlite database *remoteappmanager.db* in the current work directory. The **remoteappdb** command-line tool is provided for setting up the database. Please refer to the *Utilities* section for details on the use of this program.

2. Other DBAPI implementations and databases

   For database implementation supported by SQLAlchemy, you may configure **remoteappmanager** to use `remoteappmanager.db.orm.AppAccounting`. Please also refer to *Configure the remoteappmanager* for details on setting up the accounting class.

   Note: The use of databases other than sqlite3 is not tested

3. CSV file

   You may configurate **remoteappmanager** to use a CSV file as its database. Please refer to *Configure the remoteappmanager* for details on setting up the accounting class to use `remoteappmanager.db.csv_db.CSVAccounting`.

4. Others

   Any arbitrary database implementation may be used as long as an accounting class compliant to the API of `remoteappmanager.db.interfaces.ABCAccounting` is provided. Please also refer to *Configure the remoteappmanager* for details on setting up the accounting class.

Start JupyterHub

1. Start jupyterhub by invoking the start script:

   ```
   sh start.sh
   ```

   Note: If you want to keep the application running, use screen to start a detachable terminal.

   Note: Running on OSX or with a separate docker machine requires that the appropriate environment variables are set before starting jupyterhub. refer to the command *docker-machine env* to setup the appropriate environment. In general, invoking:

   ```
   eval `docker-machine env`
   ```

   will enable the appropriate environment. On Linux, by default the host machine and the docker machine coincide, so this step is not needed.

2. JupyterHub is now running at https://localhost:8000

3.6.2 Using Nginx Reverse Proxy

Although the SimPhoNy-Remote installation will include `nginx`, it is up to the developer to make sure that it is running correctly, with `http` and `https` firewall access also set up.

To begin with, please first ensure that the following directories exist in your file system (you will need root privileges in order to do so):

```
/etc/ssl/certs/
/etc/ssl/private/
/etc/nginx/conf.d/
```

If you want to use self-signed certificates, you can run the following `make` command in order to generate a more secure set of RSA certificates and Diffie-Helman parameters for the Nginx reverse proxy:

```
make certs CERT_TYPE='nginx'
sudo mv nginx/certs/nginx-selfsigned.key /etc/ssl/private/
sudo mv nginx/certs/nginx-selfsigned.crt /etc/ssl/certs/
sudo mv nginx/certs/dhparam.pem /etc/ssl/certs/
```

Then edit the `nginx/nginx.conf.template` file provided will in order to include a public IP address / server name in the sections marked `<external server name>`. If you prefer to use authenticated certificates this is also the time to edit the `ssl_certificate` and `ssl_certificate_key` sections with the updated locations of these files.

The Nginx template file will need to be copied into the following location (and given the `.conf` extension) in order to be discoverable by the `nginx` proxy:

```
sudo cp nginx/nginx.conf.template /etc/nginx/conf.d/sr-nginx.conf
```

After restarting the `nginx` service to make sure the new configuration is applied, run SimPhoNy-Remote as normal and it will be discoverable at https://<external_server_name>

3.6.3 Running as a Service

Instructions for how to run a general JupyterHub deployment as a service can be found here.

Instead of executing a `jupyterhub <config.py> <flags>` command upon starting the service, it is more advisable to call `bash <simphony-remote>/jupterhub/start.sh` instead.

3.6.4 Configuration

Configure the spawner

The jupyterhub configuration is documented in the jupyterhub documentation. The important difference is the spawner to use, which is configured as:

```
c.JupyterHub.spawner_class = 'remoteappmanager.jupyterhub.spawners.SystemUserSpawner'
# or
# c.JupyterHub.spawner_class = 'remoteappmanager.jupyterhub.spawners.
↪VirtualUserSpawner'
```

in the *jupyterhub_config.py* file.

Please refer to *remoteappmanager.jupyterhub.spawners* for the available spawners in this project.

Configure the authenticator and the admin user

Different authenticators can be plugged into jupyterhub. In the configuration file, the following entry will change the authenticator:

```
c.JupyterHub.authenticator_class = ('remoteappmanager.jupyterhub.auth.
↪WorldAuthenticator')
```

WorldAuthenticator will allow any user to pass authentication. Use this authenticator only for testing purposes.

Administration capabilities are decided by jupyterhub, not remoteappmanager. *jupyterhub_config.py* allows to setup admin users with the following entry:

```
c.Authenticator.admin_users = {"admin"}
```

Note that the entry must be a python set. Users in this set will, once logged in, be able to launch an administrative interface in addition to the standard docker application management.

Configure the remoteappmanager

Configuration of the remote application is performed from two sources.

- the command line, specified by the Spawner.
- a config file. The location of this file is specified as part of the command line options.

Their options are fully disjoint, and they configure different aspects of the application: Command line options are dynamically decided according to the user that requests the spawn; Config file options are general in nature, and allow the remoteappmanager to perform adequately against the current docker setup.

1. Command line options

   ```
   --base_urlpath                  The base url where the server resides
   --config_file                   The path of the configuration file
   --cookie_name                   The cookie name for authentication
   --hub_prefix                    The url prefix of the jupyterhub
   --ip                            The IP address to bind
   --login_service                 The name of the JupyterHub Authenticator class
   --logout_url                    The logout url of the jupyterhub
   --port                          Port at which to spawn
   --proxy_api_url                 The url of the reverse proxy API
   --user                          The user as specified at the jupyterhub login
   ```

   When **remoteappmanager** is started from jupyterhub using the spawner, all the command line options are filled in automatically.

2. Config file

   The **remoteappmanager** has a number of parameters configurable via a config file. The path of the config file should be specified in the spawner in *jupyterhub_config.py*:

   ```
   c.SystemUserSpawner.config_file_path = "/path/to/config.py"
   ```

   Please refer to `remoteappmanager.file_config.FileConfig` for the configurable parameters. Note that this config file will be used by all remoteappmanagers for any user.

   For example, to use CSV as the database, */path/to/config.py* would contain the followings:

```
database_class = 'remoteappmanager.db.csv_db.CSVDatabase'
database_kwargs = {'url': '/path/to/csv_file'}
```

3.6.5 Administration

As specified in the configuration section, the authenticator will grant additional administrative rights to users in the specified set.

Once logged in, an administrative user will have the option to spawn an "Admin" session, providing them with a different application, where they can add or remove users, applications, and authorize users to run specific applications. It is also possible to stop currently running containers

> **NOTE**: the existing "Admin" or "User" session must be shut down before the options form will be shown again. This is a JupyterHub-level operation and is not performed by default upon logging out. Typically it must be manually carried out by either navigating to `https://<simphony-remote>/hub/admin` or `https://<simphony-remote>/hub/home` whilst logged in and selecting the appropriate the "Stop My Sever" option. For convenience we provide a custom logout handler that automatically shuts down sessions upon an administrator sign out. This can be used with any `jupyterhub.auth.Authenticator` subclass via inheriting the `remoteappmanager.jupyterhub.auth.SimphonyRemoteAuthMixin` mixin.

It is important to note that the administrative interface works only with accounting backends supporting addition and removal. More specifically, it does not support the CSV backend. Read operations are supported, but write operations will be denied.

3.6.6 Utilities

Simphony remote comes with two utility scripts:

- *remoteappdb*: Allows to add new applications, create new users, and specify permissions between users and applications in a database from the command line. It is targeted at system administrators.
- *remoteapprest*: Allows to start, stop, inquire running containers from the command line.

Remoteappdb

Note: As of version 0.9.0 the management functionality of this utility is also covered by the administrative web interface.

The script is aimed at system administrators using the database (by default, a sqlite database) to perform accounting of users and applications.

The database must be first initialized with the *init* command:

```
remoteappdb ~/remoteappmanager.db init
```

Once initialized, the database content is ready to be configured. New applications are registered with *app create*. The image name must match the image name in docker:

```
remoteappdb ~/remoteappmanager.db app create myimage
```

The option *–verify* can be used to validate the image name against docker.

You can also create users with the *user create* command:

```
remoteappdb ~/remoteappmanager.db user create myuser
```

An application will not be visible not can be started by a user until permission is granted. To grant permission, use the *app grant* command:

```
remoteappdb ~/remoteappmanager.db app create myimage myuser
```

By default, this command will grant no special options. It is however possible to specify a different running policy, like for example mounting a common home directory, with the following options:

```
--allow-home    Enable mounting of home directory
--allow-view    Enable third-party visibility of the running container.
--volume TEXT   Application data volume, format=SOURCE:TARGET:MODE, where
                mode is 'ro' or 'rw'.
```

Note that you can grant access to the same application with multiple, different policies. Each application and policy will appear as a separate option in the user choice of runnable applications.

The script provides additional functionality to inquire the current state of the database, such as listing the current users, applications, revoke permissions, remove applications and so on.

Remoteapprest

This script is experimental and exploits the REST API provided by the server to allow inquiring, starting, and stopping containers from the command line.

Before using the CLI, you need to authenticate against the jupyterhub server with the *login* command:

```
remoteapprest login http://jupyterhubserver.example/
```

You will be inquired about username and password. Once sucessfully logged in, your credentials will be stored in a file *.remoteapprest* in your home directory. Note that your password will not be saved, only an authentication token.

Once logged in, you can inquire about the available applications by issuing:

```
remoteapprest app available
```

Note that you don't need to specify the endpoint. This command will show you a list of the available applications, preceded by a unique identifier:

```
6dbe8e166c94b0b4b36a2d961586acc0: myapplication
```

This identifier can be used to start a new container, using the following command:

```
remoteapprest app start 6dbe8e166c94b0b4b36a2d961586acc0
```

The application will run, and can be seen with:

```
remoteapprest app running

83c18fcd833595a571d556a5e6c253f8: myapplication
```

Which will show a different identifier for this running instance. Finally, the application can be stopped using the *stop* command:

```
remoteapprest app stop 83c18fcd833595a571d556a5e6c253f8
```

3.6.7 Design

Simphony remote is derived from the Jupyterhub design with a custom single user application that manages the available docker images for each user (Fig 1).

Components

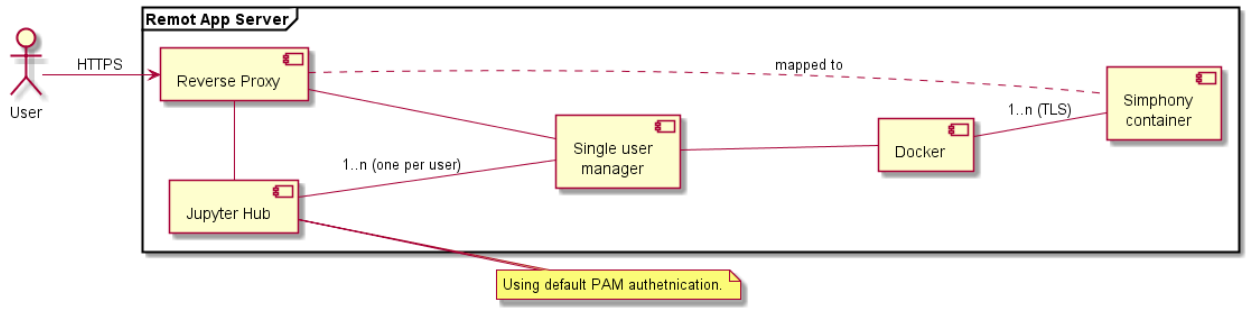

Fig. 1: **Figure. 1:** Component diagram a basic remote app server based on the Jupyter hub infrastructure.

- **Reverse proxy**: Proxy: the public facing part of the server that uses a dynamic proxy to route HTTP requests to the Hub and Single User Servers.
- **Jupyter Hub**: manages user accounts and authentication and coordinates Single Users Servers using a Spawner.
- **Single user manager**: A web server to manage the images and the active sessions. There is one such server for each authenticated user.
- **Docker**: The docker engine managing the docker containers.

Note: Docker containers are connected via HTTP. HTTPS encryption is only provided by the proxy.

Usecases

A Scientist should be able to:

- Login
- Inspect the available docker images
- Start a new session
- Stop a running session
- Share a session with another user

Furthermore an Administrator should be able to:

- Build compatible docker images
- Upload docker images.

The above design as of version 0.1 supports the usecases (see example in Fig 3):

3.6.8 Developer documentation

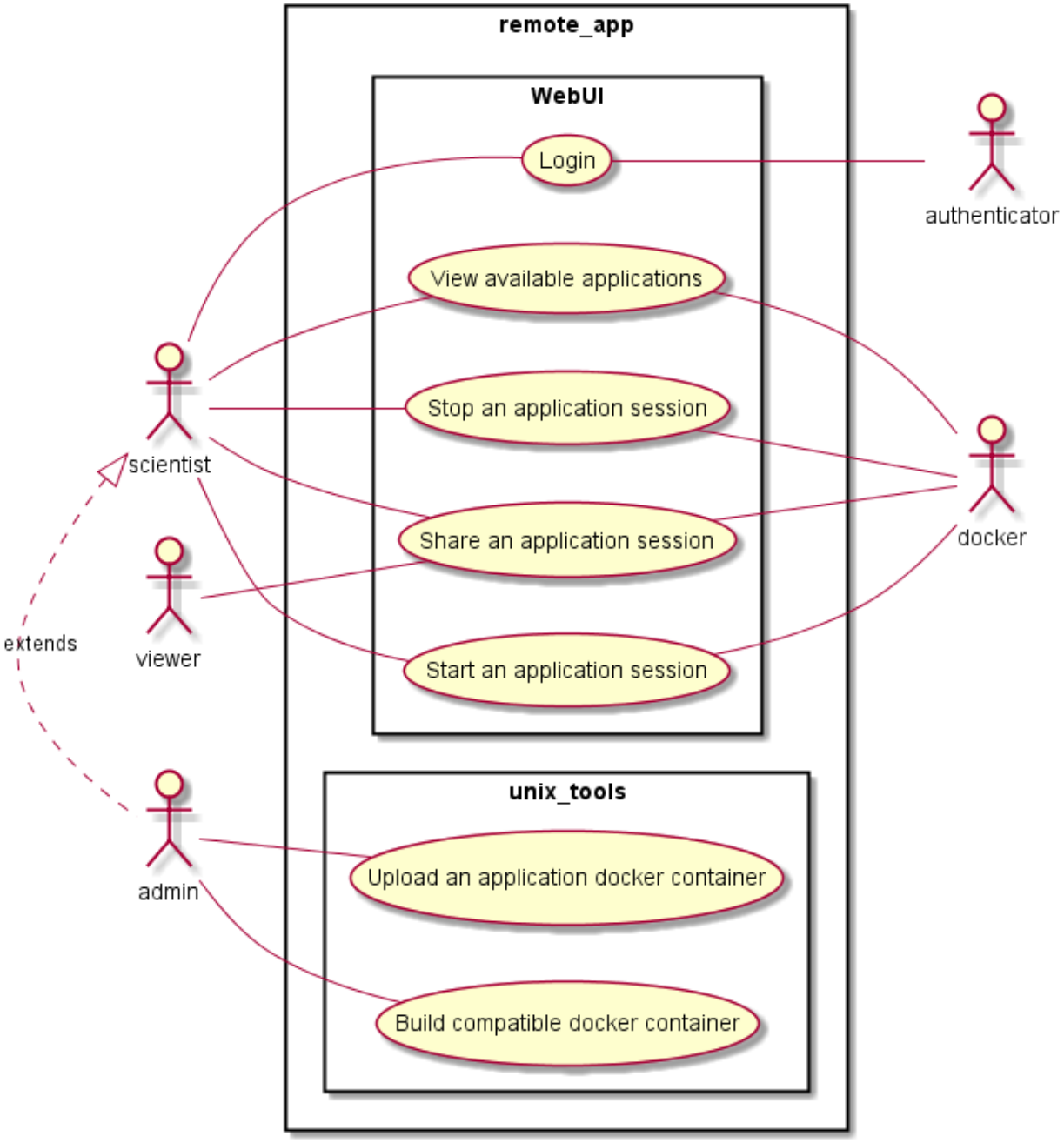

Fig. 2: **Figure 2**: Basic usecases

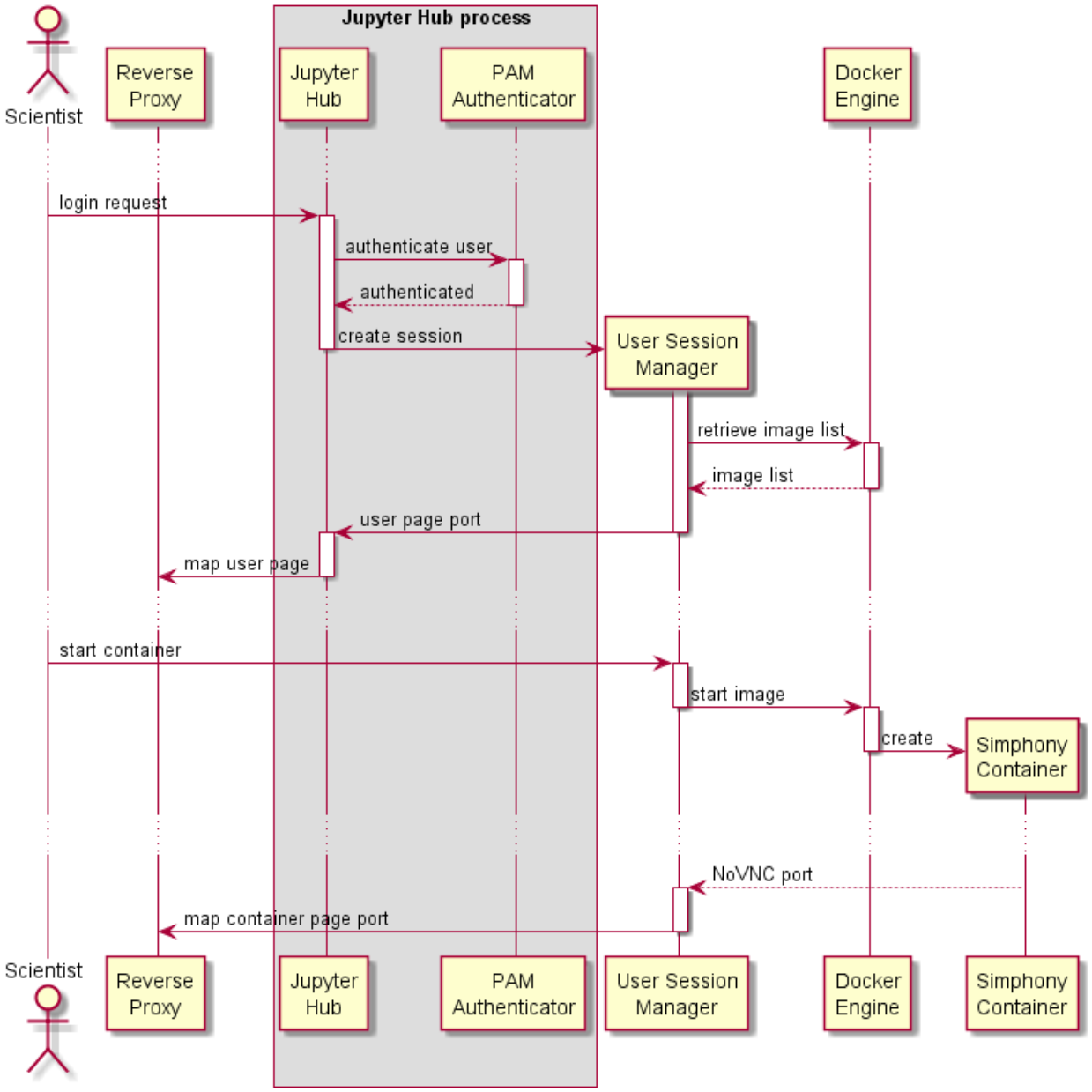

Fig. 3: **Figure 3**: Sequence diagram of an authorised user starting a remote session.

RemoteAppManager

The main tornado web application that manages the containers (docker applications) for each user.

Docker image specifications

Docker images compliant to the simphony-remote application define a protocol through docker LABEL and environment variables.

Labels

Labels are defined with the prefix namespace:

```
eu.simphony-project.docker
```

The following labels are currently defined. Their definition can be found in `remoteappmanager.docker.docker_labels`

- `ui_name`: the UI visible name of the image
- `icon_128`: a base64 encoded png image that will result as an icon
- `description`: a user-readable description of the image
- `type`: a string identifying the type of the container, depending on the original base image (vncapp or webapp)
- `env`: subnamespace for accepted environment variables. See below.

The env is a subnamespace defining the environment variables the image internals can understand. This does not mean that they are the only ones that will be passed to the image.

The naming strategy works around the docker label restrictions of having kebab case vs envvars that are traditionally MACRO_CASE. Additionally, it allows new variables to be added by layers without having to know the variables understood by the base layer.

The strategy is as follows: the name after the `env` will be converted to uppercase and dashes converted to underscores. For example:

> `env.x11-width` -> container accepts and understands envvar `X11_WIDTH`

the value of the label is currently unused, and should be left empty.

If your application uses variables with a different convention, or uses double underscores, you will have to define an auxilliary variable and transfer the value in the image startup scripts.

Currently reserved env keys:

- `x11-width`: for the VNC images, the X11 width
- `x11-height`: for the VNC images, the X11 height
- `x11-depth`: for the VNC images, the X11 depth (currently unused, fixed at 16)
- `startupdata`: this variable can be set to a file that will be loaded by an application upon startup

Container Labels

When a container is started, the following labels will be added:

- `url_id`: unique identifier that ends up in the URL when the user is redirected

- `mapping_id`: a unique key identifying the combination of image and policy used to start the container.
- `user`: the user that started the container

Environment variables

The following environment variables are passed at container startup:

- `JPY_USER`: the username used to login to the Jupyterhub frontend. Can be an email address, or anything else your authenticator accepts.
- `JPY_BASE_USER_URL`: The base URL _path_ where the user has its service.
- `USER`: A unix-likable username to create the container user.
- `URL_ID`: a unique key assigned to the container that will end up in the user-exposed URL to reach the container.

If the image accepts additional envvars (through the env labels mechanism outlined above) these variables will be passed through the configurables mechanism: special variables are recognized and exposed to the user as a configurable UI, then passed to the container at startup. See the reserved `env` labels for details.

3.6.9 Docker CI for SimPhoNy-Remote

This module contains Docker files that can be used to test installations for various platforms.

Please ensure that you are building from a clean repository, since all contents will be copied into the Docker image and may affect the build. For example, if a `venv` is already present then this will prevent the Python virtual environment inside the container from being set up correctly.

However, running S-R as a Docker container image is NOT fully supported for deployment, since image applications will not be able to be run.

Building SimPhoNy-Remote as a Docker Image

It is also possible to build SimPhoNy-Remote as a Docker container image using a *Dockerfile* script provided for either *ubuntu* or *centos* Linux OS:

```
docker build . --file Dockerfile-<linux os> -t simphony-remote
```

Alternatively, you can provide the following *docker-compose* command:

```
docker-compose build simphony-remote-<linux os>
```

Running SimPhoNy-Remote as a Docker Image

When running, S-R needs to be given a reference to the Docker daemon that contains the applications (also Docker container images) required by the Hub session. Running Docker inside Docker is not recommended, but there are a couple of approaches that can be used as a work around, such as sharing volumes or performing a bind mount.

We have successfully ran a S-R session using a Docker container image by sharing volumes with the following command:

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock -p 8000:8000 simphony-
↪remote-<linux os> bash
```

This will allow a jupyterhub session with user login / management services to be initiated as normal. However, as stated previously, any S-R applications will not be able to run.

API reference

failed to import remoteappmanager.application

failed to import remoteappmanager.webapi.application

failed to import remoteappmanager.webapi.container

failed to import remoteappmanager.webapi.admin.application

failed to import remoteappmanager.webapi.admin.container

toctree references unknown document 'api/remoteappmanager.application'

toctree references unknown document 'api/remoteappmanager.webapi.application'

toctree references unknown document 'api/remoteappmanager.webapi.container'

toctree references unknown document 'api/remoteappmanager.webapi.admin.application'

toctree references unknown document 'api/remoteappmanager.webapi.admin.container'

remoteappmanager.application	
remoteappmanager.command_line_config	
remoteappmanager.file_config	
remoteappmanager.netutils	
remoteappmanager.traitlets	Additional traitlets that we use in our application.
remoteappmanager.user	
remoteappmanager.utils	
remoteappmanager.cli.remoteappdb.__main__	Script to perform operations on the database of our application.
remoteappmanager.cli.remoteapprest.__main__	
remoteappmanager.db.csv_db	This module provides support for using CSV file as the database of the remoteappmanager.
remoteappmanager.db.interfaces	
remoteappmanager.db.orm	
remoteappmanager.docker.async_docker_client	
remoteappmanager.docker.container	
remoteappmanager.docker.container_manager	
remoteappmanager.docker.image	
remoteappmanager.jupyterhub.auth	
remoteappmanager.jupyterhub.spawners	
remoteappmanager.handlers.base_handler	
remoteappmanager.handlers.user_home_handler	
remoteappmanager.logging.logging_mixin	
remoteappmanager.webapi.application	
remoteappmanager.webapi.container	

Continued on next page

Table 1 – continued from previous page

remoteappmanager.webapi.admin.application	
remoteappmanager.webapi.admin.container	
remoteappmanager.services.hub	
remoteappmanager.services.reverse_proxy	

command_line_config

file_config

netutils

Functions

wait_for_http_server_2xx(url[, timeout])	Wait for an HTTP Server to respond at url and respond with a 2xx code.

remoteappmanager.netutils.**wait_for_http_server_2xx**(*url*, *timeout=10*)
Wait for an HTTP Server to respond at url and respond with a 2xx code.

traitlets

Functions

as_dict(traited_instance)	Returns a dictionary from the traited class, with keys equal to trait names and values the corresponding values.
set_traits_from_dict(traited_instance, d)	Given a class with traitlets and a dictionary with keys corresponding to the traitlet names, set the traitlets to the associated dict values.

remoteappmanager.traitlets.**as_dict**(*traited_instance*)
Returns a dictionary from the traited class, with keys equal to trait names and values the corresponding values.

remoteappmanager.traitlets.**set_traits_from_dict**(*traited_instance*, *d*)
Given a class with traitlets and a dictionary with keys corresponding to the traitlet names, set the traitlets to the associated dict values.

Note: if a set operation fails, the appropriate traitlet exception is raised. Traitlets that were already set won't be

rolled back.

user

utils

Functions

deprecated(func)	Decorator.
mergedoc(function, other)	Merge the docstring from the other function to the decorated function.
one(elements)	Returns True if only one element is not None, false otherwise
parse_volume_string(volume_string)	Parses a volume specification string SOURCE:TARGET:MODE into its components, or raises click.BadOptionUsage if not according to format.
remove_quotes(s)	Removes start/end quotes from a string, if needed.
url_path_join(*pieces)	Join components of url into a relative url path Use to prevent double slash when joining subpath.
with_end_slash(url)	Normalises a url to have an ending slash, and only one.
without_end_slash(url)	Makes sure there is no end slash at the end of a url.

remoteappmanager.utils.**deprecated**(*func*)
Decorator. Marks a function/method as deprecated.

remoteappmanager.utils.**mergedoc**(*function*, *other*)
Merge the docstring from the other function to the decorated function.

remoteappmanager.utils.**one**(*elements*)
Returns True if only one element is not None, false otherwise

remoteappmanager.utils.**parse_volume_string**(*volume_string*)
Parses a volume specification string SOURCE:TARGET:MODE into its components, or raises click.BadOptionUsage if not according to format.

remoteappmanager.utils.**remove_quotes**(*s*)
Removes start/end quotes from a string, if needed. If s is not a string, it is returned untouched.

remoteappmanager.utils.**url_path_join**(**pieces*)
Join components of url into a relative url path Use to prevent double slash when joining subpath. This will leave the initial and final / in place

Assume pieces do not contain protocol (e.g. http://)

remoteappmanager.utils.**with_end_slash**(*url*)
Normalises a url to have an ending slash, and only one.

remoteappmanager.utils.**without_end_slash**(*url*)
Makes sure there is no end slash at the end of a url.

__main__

Functions

database(db_url)	Retrieves the orm.Database object from the passed db url.
get_docker_client()	Returns docker.APIClient object using the local environment variables
is_sqlitedb_url(db_url)	Returns True if the url refers to a sqlite database
main()	
normalise_to_url(url_or_path)	Normalises a disk path to a sqlalchemy url
print_error(error)	Prints an error message to stderr
sqlite_url_to_path(url)	Converts a sqlalchemy sqlite url to the disk path.
sqlitedb_present(db_url)	Checks if the db url is present.

remoteappmanager.cli.remoteappdb.__main__.**database**(*db_url*)
Retrieves the orm.Database object from the passed db url.

Parameters **db_url** (*str*) – A string containing a db sqlalchemy url.

Returns

Return type orm.Database instance.

remoteappmanager.cli.remoteappdb.__main__.**get_docker_client**()
Returns docker.APIClient object using the local environment variables

remoteappmanager.cli.remoteappdb.__main__.**is_sqlitedb_url**(*db_url*)
Returns True if the url refers to a sqlite database

remoteappmanager.cli.remoteappdb.__main__.**main**()

remoteappmanager.cli.remoteappdb.__main__.**normalise_to_url**(*url_or_path*)
Normalises a disk path to a sqlalchemy url

Parameters **url_or_path** (*str*) – a sqlalchemy url or a disk path

Returns A sqlalchemy url

Return type str

remoteappmanager.cli.remoteappdb.__main__.**print_error**(*error*)
Prints an error message to stderr

remoteappmanager.cli.remoteappdb.__main__.**sqlite_url_to_path**(*url*)
Converts a sqlalchemy sqlite url to the disk path.

Parameters **url** (*str*) – A "sqlite:///" path

Returns The disk path.

Return type str

remoteappmanager.cli.remoteappdb.__main__.**sqlitedb_present**(*db_url*)
Checks if the db url is present. Remote urls are always assumed to be present, so this method concerns mostly sqlite databases.

__main__

Functions

main()	

remoteappmanager.cli.remoteapprest.__main__.**main**()

csv_db

interfaces

orm

Functions

accounting_for_user(session, user)	Returns a list of Accounting objects, each containing an application and the associated policy that the specified orm user is allowed to run.
detached_session(db)	Creates a session where at the end, the objects retrieved are detached from the session itself
transaction(session)	handles a transaction in a session.

remoteappmanager.db.orm.**accounting_for_user**(*session*, *user*)
Returns a list of Accounting objects, each containing an application and the associated policy that the specified orm user is allowed to run. If the user is None, the default is to return an empty list. The id is a unique string identifying the combination of application and policy. It is not unique per user.

Parameters

- **session** (*Session*) – The current session
- **user** (*User or None*) – the orm User, or None.

Returns

Return type A list of Accounting objects

remoteappmanager.db.orm.**detached_session**(*db*)
Creates a session where at the end, the objects retrieved are detached from the session itself

remoteappmanager.db.orm.**transaction**(*session*)
handles a transaction in a session.

async_docker_client

container

container_manager

Exceptions

MultipleResultsFound	Raised when we are asking for a specific container, but more than one result is found.
OperationInProgress	Exception raised when the operation for the requested image or container is already in progress.

class remoteappmanager.docker.container_manager.**MultipleResultsFound**
Bases: Exception

Raised when we are asking for a specific container, but more than one result is found.

class remoteappmanager.docker.container_manager.**OperationInProgress**
Bases: Exception

Exception raised when the operation for the requested image or container is already in progress.

image

auth

spawners

Functions

escape(s)	Trivial escaping wrapper for well established stuff.

remoteappmanager.jupyterhub.spawners.**escape**(*s*)
Trivial escaping wrapper for well established stuff. Works for containers, file names. Note that it is not destructive, so it won't generate collisions.

base_handler

user_home_handler

logging_mixin

Functions

issue(self, message[, exc])	Accepts a message that will be logged with an additional reference code for easy log lookup.

remoteappmanager.logging.logging_mixin.**issue**(*self*, *message*, *exc=None*)

Accepts a message that will be logged with an additional reference code for easy log lookup.

The identifier will be returned for inclusion in user-visible error messages.

hub

reverse_proxy

3.6.10 Troubleshoot

A new user does not see the applications I am adding

You have to restart the user server. As a jupyterhub administrator, go to

https://your.jupyterhub.url/hub/admin

and restart the user server.

This problem will appear under this circumstance: 1. User who is not in the remoteappmanager db yet performs a login. 2. Admin adds User to remoteappmanager db, and grants him some applications. 3. User will not see the applications.

The reason is the following: when the user first performs the login, the remoteappmanager subprocess is started. The authentication mechanism looks the user up in the remoteappmanager database, does not find it, and therefore sets account to None. This operation is never performed again, so the user remains None even if later on it is added to the database. Only by restarting remoteappmanager the lookup is performed again.

It is debatable if this behavior is a bug or not (after all, bash also won't alter your current enviroment if root changes /etc/bashrc, and you will have to logout to get the new environment). Issue #305 debates this point.

I use the GitHub authenticator. A GitHub user has capitalisation in its username, but I see it as lowercase.

This is by design in both GitHub and JupyterHub. GitHub usernames are case insensitive, and case preserving. JupyterHub authenticator always normalises the usernames to lowercase.

The database is not initalised properly

Each user's server requires a database setup and readable by the local process on which the `remoteappmanager` web application is started. The error message indicates that the database is not readable (e.g. it does not exist). Please refer to *further documentation* for details and options on setting up the database.

For more details on how the local process is managed, please refers to `remoteappmanager.spawner`.

Docker timeouts

If the application is unable to connect to docker and timeouts with the following message

> Error while fetching server API version: HTTPSConnectionPool(host='192.168.99.100', port=2376): Max retries exceeded with url: /version (Caused by ConnectTimeoutError(<requests.packages.urllib3.connection.VerifiedHTTPSConnection object at 0x106299518>, 'Connection to 192.168.99.100 timed out. (connect timeout=60)')).

The likely problem is that your docker machine is not reachable. The most likely cause is that you recently recreated your default docker machine, or the docker machine is no longer reachable. Make sure that your docker environment (DOCKER_HOST environment variable) is compatible with the docker machine current ip address (*docker-machine ip*). If not, reconfigure your docker machine environment with *eval $(docker-machine env)*.

Error when connecting to docker: Permission denied

Check if your */var/run/docker.sock* is accessible and readable. The likely cause is that your current user is not in the *docker* group. Fix this by running:

```
sudo addgroup your_username docker
```

and then logging out and in again.

3.7 License

Python Module Index

r

Index

S

T

U

W