Simphony-remote Documentation

Release 1.0.0

SimPhoNy Project

Nov 15, 2016

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

Contents

This documentation is for simphony-remote 1.0 (1.0.0)

2.1 Deployment

2.1.1 Single Machine

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

Note: The following instructions assume a clean up-to-date ubuntu 14.04 system.

1. Retrieve the single user session manager:

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

2. Make sure that you are obtaining a recent version of Docker, at least 1.12. 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 dockerengine
```

3. Install dependencies. You might be prompted for the root password to execute this:

```
make deps
```

4. 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
```

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

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

6. Install the python dependencies:

```
make pythondeps
```

7. And install the package itself:

```
make install
```

8. 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
```

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

```
make db
```

10. Change dir into jupyterhub:

```
cd ./jupyterhub
```

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

11. 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
```

12. You can now start the service:

```
sh start.sh
```

13. Visit the site at:

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

2.1.2 Setup docker containers

Compatible docker containers can be found in DockerHub. Refer to the documentation of simphony-remote-docker repository to deploy the images.

2.1.3 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.

2.1.4 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

2.2 Configuration

2.2.1 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.

2.2.2 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, reach an administrative interface, instead of the docker application management.

2.2.3 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_api_url                   The url of the jupyterhub REST API
--hub_host                      The url of the jupyterhub server
--hub_prefix                    The url prefix of the jupyterhub
--ip                            The IP address to bind
--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:

```
accounting_class = 'remoteappmanager.db.csv_db.CSVAccounting'
accounting_kwargs = {'url': '/path/to/csv_file'}
```

2.3 Administration

As specified in the deployment section, the authenticator will grant administrative rights to users in the specified set. Once logged in, an administrative user will be served by a different application, where it can add or remove users, applications, and authorize users to run specific applications. It is also possible to stop currently running containers.

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.

2.4 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.

2.4.1 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.

2.4.2 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
```

2.5 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).

2.5.1 Components

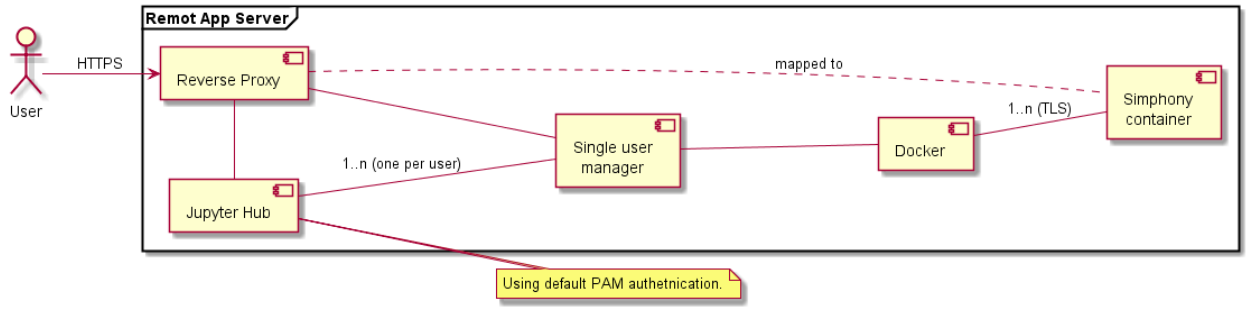

Fig. 2.1: **Figure. 1:** Component diagram a basic remote app server based on the Jupyter hub infrastructure.

- **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.

2.5.2 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):

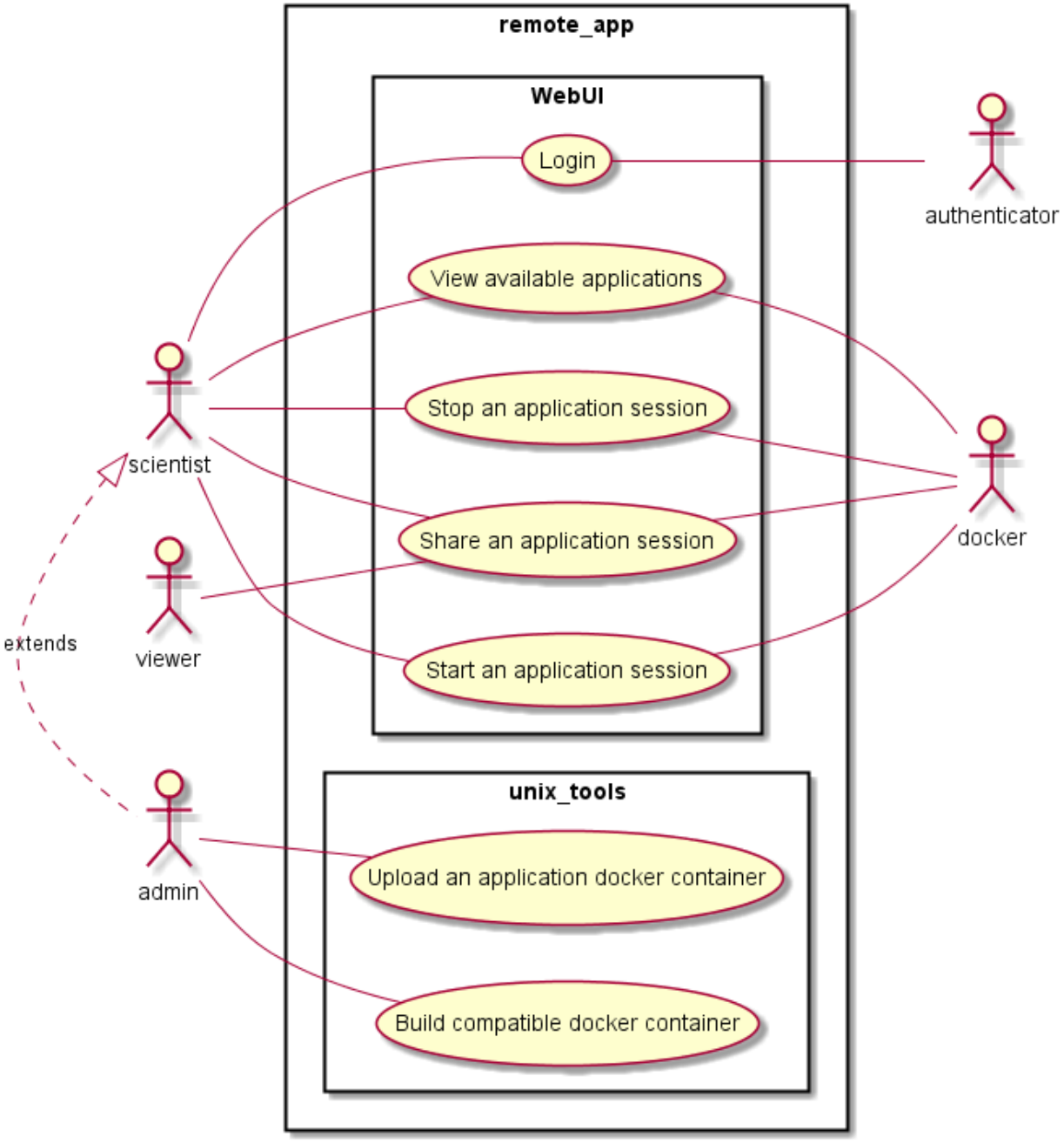

Fig. 2.2: **Figure 2**: Basic usecases

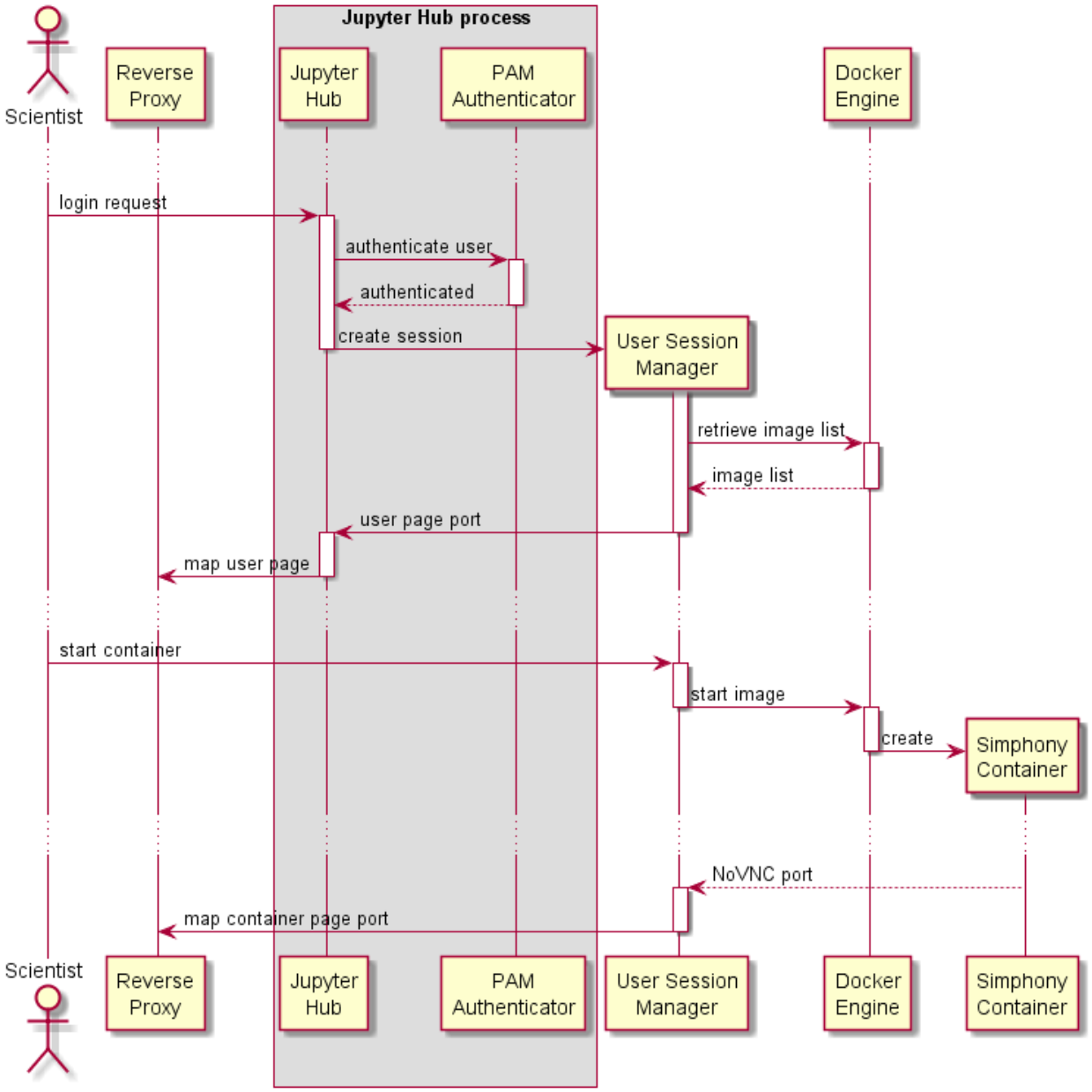

Fig. 2.3: **Figure 3**: Sequence diagram of an authorised user starting a remote session.

2.6 Developer documentation

2.6.1 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)

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.

2.6.2 API reference

remoteappmanager.application	
remoteappmanager.command_line_config	
remoteappmanager.file_config	
remoteappmanager.jinja2_adapters	
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	

Continued on next page

Table 2.1 – continued from previous page

remoteappmanager.handlers.base_handler
remoteappmanager.handlers.home_handler
remoteappmanager.logging.logging_mixin
remoteappmanager.webapi.application
remoteappmanager.webapi.container
remoteappmanager.webapi.admin.application
remoteappmanager.webapi.admin.container
remoteappmanager.services.hub
remoteappmanager.services.reverse_proxy

application

Classes

Application (command_line_config, ...)	Tornado main application

class remoteappmanager.application. **Application** (*command_line_config*, *file_config*, *environment_config*)

Bases: remoteappmanager.base_application.BaseApplication

Tornado main application

Initializes the application

config: ApplicationConfiguration The application configuration object

command_line_config

Classes

CommandLineConfig (*args, **kwargs)	Configuration options for the application server

class remoteappmanager.command_line_config. **CommandLineConfig** (**args*, ***kwargs*)

Bases: traitlets.traitlets.HasTraits

Configuration options for the application server

base_urlpath = Unicode

The base url where the server resides

command_line_options_inited = False

config_file = Unicode
The path of the configuration file

cookie_name = Unicode
The cookie name for authentication

hub_api_url = Unicode
The url of the jupyterhub REST API

hub_host = Unicode
The url of the jupyterhub server

hub_prefix = Unicode
The url prefix of the jupyterhub

ip = Unicode
The IP address to bind

parse_config ( )
Parses the command line arguments, and assign their values to our local traits.

port = Int
Port at which to spawn

proxy_api_url = Unicode
The url of the reverse proxy API

user = Unicode
The user as specified at the jupyterhub login

file_config

Classes

FileConfig (*args, **kwargs)	Configuration options for the application server.

class remoteappmanager.file_config. **FileConfig** (**args*, ***kwargs*)
Bases: traitlets.traitlets.HasTraits

Configuration options for the application server. They are sourced from the configuration file.

accounting_class = Unicode
The import path to a subclass of ABCAccounting

Default: 'remoteappmanager.db.orm.AppAccounting'

accounting_kwargs = Dict
The keyword arguments for initialising the Accounting instance

docker_config ( )
Extracts the docker configuration as a dictionary suitable to be passed as keywords to the docker client.

docker_host = Unicode
The docker host to connect to

Default: ''

ga_tracking_id = Unicode
The google analytics tracking id

login_url = Unicode
The url to be redirected to if the user is not authenticated for pages that require authentication

Default: '/hub'

network_timeout = Int
The timeout (seconds) for network operations

Default: 30

parse_config (*config_file*)
Parses the config file, and assign their values to our local traits.

static_path = Unicode
The path where to search for static files

template_path = Unicode
The path where to search for jinja templates

tls = Bool
If True, connect to docker with tls

Default: False

tls_ca = Unicode
Path to CA certificate for docker TLS

Default: ''

tls_cert = Unicode
Path to client certificate for docker TLS

Default: ''

tls_key = Unicode
Path to client key for docker TLS

Default: ''

tls_verify = Bool
If True, verify the CA certificate against a known or self-signed CA certificate

Default: True

jinja2_adapters

Classes

Jinja2LoaderAdapter (env)	Adapts the Jinja2 environment to act as a loader as desired by tornado.
Jinja2TemplateAdapter (template)	Adapts the Jinja template interface to act as a tornado template.

class remoteappmanager.jinja2_adapters. **Jinja2LoaderAdapter** (*env*)
Bases: object

Adapts the Jinja2 environment to act as a loader as desired by tornado.

The class uses duck typing to implement the interface of tornado.BaseLoader and relies on jinja caching to hold the premade templates.

Initializes the adapter.

Parameters **env** (*Environment*) – the jinja2 environment

load (*name*, *parent_path=None*)
Loads the template with a given name.

Parameters

- **name** (*str*) – the simple name of the template.
- **parent_path** (*str*) – The parent path (unused)

Returns

- *Jinja2TemplateAdapter object, adapting a jinja template into a tornado*
- *template interface*

reset ( )
Resets the LRU cache in jinja template environment. The method is already thread safe.

resolve_path (*name*, *parent_path=None*)
Returns the absolute name of the template according to the loader.

Parameters

- **name** (*str*) – the simple name of the template.
- **parent_path** (*str*) – The parent path (unused)

Returns

Return type The absolute path of the template

class remoteappmanager.jinja2_adapters. **Jinja2TemplateAdapter** (*template*)
Bases: tornado.template.Template

Adapts the Jinja template interface to act as a tornado template. It reimplements the base class, but it uses no functionality of it.

generate (***kwargs*)
Generate this template with the given arguments.

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.

Classes

UnicodeOrFalse ([default_value, allow_none, ...])	Declare a traitlet.

class remoteappmanager.traitlets. **UnicodeOrFalse** (*default_value=traitlets.Undefined*, *allow_none=False*, *read_only=None*, *help=None*, ***kwargs*)

Bases: traitlets.traitlets.Unicode

Declare a traitlet.

If *allow_none* is True, None is a valid value in addition to any values that are normally valid. The default is up to the subclass. For most trait types, the default value for allow_none is False.

Extra metadata can be associated with the traitlet using the .tag() convenience method or by using the traitlet instance's .metadata dictionary.

info_text = 'a unicode string or False'

validate (*obj*, *value*)

user

Classes

User (*args, **kwargs)	Represents the user.

class remoteappmanager.user. **User** (**args*, ***kwargs*)

Bases: traitlets.traitlets.HasTraits

Represents the user. It holds a reference to the ORM user, if available.

account = Any

Can be none if the username cannot be found in the database.

any value

name = Unicode

a unicode string

utils

Functions

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.
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. **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. **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.

Classes

mergedocs (other)	Merge the docstrings of other class to the decorated.

class remoteappmanager.utils. **mergedocs** (*other*)
Bases: object

Merge the docstrings of other class to the decorated.

__main__

Functions

database (db_url)	Retrieves the orm.Database object from the passed db url.
get_docker_client ()	Returns docker.client 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.client 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.

Classes

RemoteAppDBContext (db_url)

class remoteappmanager.cli.remoteappdb.__main__. **RemoteAppDBContext** (*db_url*)
Bases: object

__main__

Functions

main ()

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

Classes

Credentials (url, username, cookies)	Data class to hold the credentials extracted from the credential file.
RemoteAppRestContext	The click context passed around.

class remoteappmanager.cli.remoteapprest.__main__. **Credentials** (*url*, *username*, *cookies*)

Bases: object

Data class to hold the credentials extracted from the credential file.

classmethod from_file (*credentials_file*)

Extracts the authorization info from the credentials file. Returns a tuple with url, username, and a dict of credentials cookies

write (*credentials_file*)

Stores the credentials in a credentials file.

class remoteappmanager.cli.remoteapprest.__main__. **RemoteAppRestContext**

Bases: object

The click context passed around.

credentials = None

credentials_file = None

csv_db

Classes

CSVAccounting (csv_file_path, **kwargs)	Accounting class that reads a CSV file and is used by the remoteappmanager.
CSVApplication (id, image)	
CSVApplicationPolicy ([allow_home, ...])	
CSVUser (id, name)	

class remoteappmanager.db.csv_db. **CSVAccounting** (*csv_file_path*, ***kwargs*)

Bases: *remoteappmanager.db.interfaces.ABCAccounting*

Accounting class that reads a CSV file and is used by the remoteappmanager. Currently only accepts one csv file.

Initialiser

Parameters

- **csv_file_path** (*str*) – File path for the CSV file
- ****kwargs** – optional keyword arguments for open(csv_file_path)

create_application (*app_name*)
Creates a new application with the specified name. Raises if an application with the same name already exists

Parameters **app_name** (*str*) – The name of the application

Returns **id** – The id of the created application

Return type int

Raises `exceptions.Exists` – If the application already exists.

create_user (*user_name*)
Creates a user with the specified username, if the backend allows it.

Parameters **user_name** (*str*) – The user name

Returns **id** – The unique id of the user

Return type int

Raises `exceptions.Exists` – If the user with that name already exists.

get_apps_for_user (*user*)
Return an iterable of ApplicationConfig for a given user

Parameters **user** (*opaque-type*) – Same type as the result of *get_user*

Returns each item of the tuple should be a tuple of (id, ABCApplication, ABCApplicationPolicy) where id is a string used for identifying (ABCApplication, ABCApplicationPolicy)

Return type tuple

get_user (*, *user_name=None*, *id=None*)
Return a User for a given user_name or id, or return None if the User is not found. Only one argument is allowed.

Parameters

- **user_name** (*str*) – The user name
- **id** (*int*) – An id

Returns **user** – an user object that the database understands

Return type opaque-type

grant_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Grant access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

Returns **id** – A 32 characters id (mapping_id)

Return type str

list_applications ( )
List all available applications

Returns **applications** – A list of the available apps.

Return type list

list_users ( )
Returns a list of all available users.

Returns **users** – A list of users.

Return type list

remove_application (*, *app_name=None*, *id=None*)
Remove an existing application by name or id, depending what is provided. Only one argument is allowed. If the application is not present, does nothing.

Parameters

- **app_name** (*str*) – The name of the application
- **id** (*int*) – The id of the application

Raises exception.NotFound – If the application is not found.

remove_user (*, *user_name=None*, *id=None*)
Removes a user by name or id, if the backend allows it. Only one argument is allowed. If the user is not present, does nothing.

Parameters **user_name** (*str*) – The user name

revoke_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Revoke access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

revoke_access_by_id (*mapping_id*)
Like revoke_access, but uses the mapping id instead.

class remoteappmanager.db.csv_db. **CSVApplication** (*id*, *image*)
Bases: *remoteappmanager.db.interfaces.ABCApplication*

class remoteappmanager.db.csv_db. **CSVApplicationPolicy** (*allow_home=False*, *allow_view=False*, *allow_common=False*, *volume_source=None*, *volume_target=None*, *volume_mode=None*)

Bases: *remoteappmanager.db.interfaces.ABCApplicationPolicy*

class remoteappmanager.db.csv_db. **CSVUser** (*id*, *name*)

Bases: object

interfaces

Classes

ABCAccounting	Main accounting interface required by the single user application.
ABCApplication (id, image)	Description of an application
ABCApplicationPolicy ([allow_home, ...])	Policy for an application

class remoteappmanager.db.interfaces. **ABCAccounting**

Bases: object

Main accounting interface required by the single user application.

create_application (*app_name*)

Creates a new application with the specified name. Raises if an application with the same name already exists

Parameters **app_name** (*str*) – The name of the application

Returns **id** – The id of the created application

Return type int

Raises exceptions.Exists – If the application already exists.

create_user (*user_name*)

Creates a user with the specified username, if the backend allows it.

Parameters **user_name** (*str*) – The user name

Returns **id** – The unique id of the user

Return type int

Raises exceptions.Exists – If the user with that name already exists.

get_apps_for_user (*user*)

Return an iterable of ApplicationConfig for a given user

Parameters **user** (*opaque-type*) – Same type as the result of *get_user*

Returns each item of the tuple should be a tuple of (id, ABCApplication, ABCApplicationPolicy) where id is a string used for identifying (ABCApplication, ABCApplicationPolicy)

Return type tuple

get_user (*, *user_name=None*, *id=None*)
Return a User for a given user_name or id, or return None if the User is not found. Only one argument is allowed.

Parameters

- **user_name** (*str*) – The user name
- **id** (*int*) – An id

Returns **user** – an user object that the database understands

Return type opaque-type

grant_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Grant access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

Returns **id** – A 32 characters id (mapping_id)

Return type str

list_applications ( )
List all available applications

Returns **applications** – A list of the available apps.

Return type list

list_users ( )
Returns a list of all available users.

Returns **users** – A list of users.

Return type list

remove_application (*, *app_name=None*, *id=None*)
Remove an existing application by name or id, depending what is provided. Only one argument is allowed. If the application is not present, does nothing.

Parameters

- **app_name** (*str*) – The name of the application
- **id** (*int*) – The id of the application

Raises `exception.NotFound` – If the application is not found.

remove_user (*, *user_name=None*, *id=None*)
Removes a user by name or id, if the backend allows it. Only one argument is allowed. If the user is not present, does nothing.

Parameters **user_name** (*str*) – The user name

revoke_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Revoke access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

revoke_access_by_id (*mapping_id*)
Like revoke_access, but uses the mapping id instead.

class remoteappmanager.db.interfaces. **ABCApplication** (*id*, *image*)
Bases: object

Description of an application

id = None
Numerical id

image = None
Name of the image

class remoteappmanager.db.interfaces. **ABCApplicationPolicy** (*allow_home=False*, *allow_view=False*, *allow_common=False*, *volume_source=None*, *volume_target=None*, *volume_mode=None*)
Bases: object

Policy for an application

allow_common = None
Is the common data volume for the application mounted

allow_home = None
Is the home directory mounted

allow_view = None
Is the application viewable by others

volume_mode = None
Mode for read-write access (ro = Read-only. rw = Read-write)

volume_source = None
Source path for the common data volume on the host machine

volume_target = None
Target mount point of the common data volume in the application

orm

Functions

apps_for_user (session, user)	Returns a tuple of tuples, 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
transaction (session)	handles a transaction in a session.

remoteappmanager.db.orm. **apps_for_user** (*session*, *user*)
Returns a tuple of tuples, 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 mapping_id is a unique string identifying the combination of application and policy. It is not unique per user. :param session: The current session :type session: Session :param user: the orm User, or None. :type user: User or None

Returns

Return type A tuple of tuples (mapping_id, orm.Application, orm.ApplicationPolicy)

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.

Classes

Accounting (**kwargs)	Holds the information about who is allowed to run what.
AppAccounting (url, **kwargs)	Initialiser
Application (**kwargs)	Describes an application that should be available for startup
ApplicationPolicy (**kwargs)	A simple constructor that allows initialization from kwargs.
Database (url, **kwargs)	Initialises a database connection to a given database url.
IdMixin	Base class to provide an id
User (**kwargs)	Table for users.

class remoteappmanager.db.orm. **Accounting** (***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Holds the information about who is allowed to run what.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

application

application_id

application_policy

application_policy_id

id

user

user_id

class remoteappmanager.db.orm. **AppAccounting** (*url*, ***kwargs*)
Bases: *remoteappmanager.db.interfaces.ABCAccounting*

Initialiser

Parameters

- **url** (*str*) – the url for connecting to a database
- ****kwargs** – optional keyword arguments for *Database*

See also:

SQLAlchemy Database Urls

check_database_readable ( )
Raise IOError if the database url points to a sqlite database that is not readable

TODO: may extend for validating databases in other dialects?

create_application (*app_name*)
Creates a new application with the specified name. Raises if an application with the same name already exists

Parameters **app_name** (*str*) – The name of the application

Returns **id** – The id of the created application

Return type int

Raises exceptions.Exists – If the application already exists.

create_user (*user_name*)
Creates a user with the specified username, if the backend allows it.

Parameters **user_name** (*str*) – The user name

Returns **id** – The unique id of the user

Return type int

Raises exceptions.Exists – If the user with that name already exists.

get_apps_for_user (*user*)
Return an iterable of ApplicationConfig for a given user

Parameters **user** (*opaque-type*) – Same type as the result of *get_user*

Returns each item of the tuple should be a tuple of (id, ABCApplication, ABCApplicationPolicy) where id is a string used for identifying (ABCApplication, ABCApplicationPolicy)

Return type tuple

get_user (*, *user_name=None*, *id=None*)
Return a User for a given user_name or id, or return None if the User is not found. Only one argument is allowed.

Parameters

- **user_name** (*str*) – The user name
- **id** (*int*) – An id

Returns **user** – an user object that the database understands

Return type opaque-type

grant_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Grant access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

Returns **id** – A 32 characters id (mapping_id)

Return type str

list_applications ( )
List all available applications

Returns **applications** – A list of the available apps.

Return type list

list_users ( )
Returns a list of all available users.

Returns **users** – A list of users.

Return type list

remove_application (*, *app_name=None*, *id=None*)
Remove an existing application by name or id, depending what is provided. Only one argument is allowed. If the application is not present, does nothing.

Parameters

- **app_name** (*str*) – The name of the application
- **id** (*int*) – The id of the application

Raises `exception.NotFound` – If the application is not found.

remove_user (*, *user_name=None*, *id=None*)
Removes a user by name or id, if the backend allows it. Only one argument is allowed. If the user is not present, does nothing.

Parameters **user_name** (*str*) – The user name

revoke_access (*app_name*, *user_name*, *allow_home*, *allow_view*, *volume*)
Revoke access for user to application.

Parameters

- **app_name** (*str*) – The name of the application
- **user_name** (*str*) – The name of the user
- **allow_home** (*bool*) – If the home workspace should be mounted.
- **allow_view** (*bool*) – If the session should be visible by others.
- **volume** (*str*) – A volume to mount in the format source_path:target_path:mode mode being "ro" or "rw". (e.g. "/host/path:/container/path:ro").

Raises

- exception.NotFound: – if the app or user are not found.
- ValueError: – if the volume string is invalid.

revoke_access_by_id (*mapping_id*)
Like revoke_access, but uses the mapping id instead.

class remoteappmanager.db.orm. **Application** (***kwargs*)
Bases: *remoteappmanager.db.orm.IdMixin* , sqlalchemy.ext.declarative.api.Base

Describes an application that should be available for startup

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs` .

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

static from_image_name (*session*, *image_name*)

id

image
The docker image name where the application can be found

class remoteappmanager.db.orm. **ApplicationPolicy** (***kwargs*)
Bases: *remoteappmanager.db.orm.IdMixin* , sqlalchemy.ext.declarative.api.Base

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs` .

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

allow_common
If a common workarea should be mounted in the container

allow_home
If the home directory should be mounted in the container

allow_view
If the container should be accessible by other people

id

volume_mode

volume_source

volume_target

class remoteappmanager.db.orm. **Database** (*url*, ***kwargs*)
Bases: *remoteappmanager.logging.logging_mixin.LoggingMixin*

Initialises a database connection to a given database url.

Parameters

- **url** (*url*) – A sqlalchemy url to connect to a specified database.
- **kwargs** (*dict*) – Additional keys will be passed at create_engine.

create_session ( )
Create a new session class at the database url with the current engine.

reset ( )
Completely resets the content of the database, removing and reinitializing the tables. Should be used only if the database does not already exist, or if its contents are irrelevant or obsolete.

class remoteappmanager.db.orm. **IdMixin**
Bases: object

Base class to provide an id

classmethod from_id (*session*, *id*)

id = Column(None, Integer(), table=None, primary_key=True, nullable=False)

class remoteappmanager.db.orm. **User** (***kwargs*)
Bases: *remoteappmanager.db.orm.IdMixin*, sqlalchemy.ext.declarative.api.Base

Table for users.

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs .

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

id

name
The name of the user as specified by jupyterhub. This entry must be unique and is, for all practical purposes, a primary key.

async_docker_client

Classes

AsyncDockerClient (*args, **kwargs)	Provides an asynchronous interface to dockerpy.

class remoteappmanager.docker.async_docker_client. **AsyncDockerClient** (**args*, ***kwargs*)

Bases: object

Provides an asynchronous interface to dockerpy. All Client interface is available as methods returning a future instead of the actual result. The resulting future can be yielded.

This class is thread safe. Note that all instances use the same executor.

Initialises the docker async client.

The client uses a single, module level executor to submit requests and obtain futures. The futures must be yielded according to the tornado asynchronous interface.

The exported methods are the same as from the docker-py synchronous client, with the exception of their async nature.

Note that the executor is a ThreadPoolExecutor with a single thread.

container

Classes

Container (*args, **kwargs)	Class representing a container.

class remoteappmanager.docker.container. **Container** (**args*, ***kwargs*)

Bases: traitlets.traitlets.HasTraits

Class representing a container. Note that its existence just describes a container. It does not imply that the associated container is still running, registered, or anything

docker_id = Unicode

The docker id of the container

a unicode string

classmethod from_docker_dict (*docker_dict*)

Returns a Container object with the info given by a docker Client.

Parameters **docker_dict** (*dict*) – One item from the result of docker.Client.containers

Returns container

Return type *remoteappmanager.docker.container.Container*

Examples

```
>>> # containers is a list of dict
>>> containers = docker.Client().containers()
```

```
>>> Container.from_docker_dict(containers[0])
```

host_url
Returns the docker host where this server can be reached, in url form.

image_id = Unicode
And the image docker id

a unicode string

image_name = Unicode
The image name

a unicode string

ip = Unicode
The ip address...

a unicode string

mapping_id = Unicode
Mapping identifier

a unicode string

name = Unicode
The practical name of the container

a unicode string

port = Int
...and port where the container service will be listening

an int

Default: 80

url_id = Unicode
The id that will go in the URL of the container. This is a de-facto replacement for the container docker id. The reason why we don't use that instead is because the container id is difficult to obtain reliably from inside the container, and because we want more flexibility in the form of the user-exposed id. Important: must be globally unique, not just per-user unique.

a unicode string

urlpath = Unicode
The url path of the container as it is exported to the user. e.g. "/home/test/containers/12345" Must not have an end slash.

a unicode string

user = Unicode
The user currently running the container

a unicode string

container_manager

Classes

Continued on next page

Table 2.22 – continued from previous page

ContainerManager (docker_config, *args, **kwargs)	Initializes the Container manager.

class remoteappmanager.docker.container_manager. **ContainerManager** (*docker_config*, **args*, ***kwargs*)

Bases: *remoteappmanager.logging.logging_mixin.LoggingMixin*

Initializes the Container manager.

Parameters **docker_config** (*Dict*) – A dictionary containing the keywords for the configuration of the docker client in agreement to docker py documentation.

container_from_url_id (*url_id*)

Retrieves and returns the container by its url_id, if present. If not present, returns None.

container_port = Int

The container (not host) port. We decided it's 8888 by default. It will be mapped to a random port on the host, so that our reverse proxy can refer to it.

an int

Default: 8888

containers_from_filters (*filters*)

Returns the currently running containers for a given filter

Parameters **filters** (*dict*) – A dictionary of filters as in dockerpy

Returns

Return type A list of Container objects, or an empty list if nothing is found.

containers_from_mapping_id (*user_name*, *mapping_id*)

Returns the currently running containers for a given user and mapping_id.

Parameters

- **user_name** (*str*) – The username
- **mapping_id** (*str*) – The unique id to identify the container

Returns

Return type A list of Container objects, or an empty list if nothing is found.

docker_client = Instance

The asynchronous docker client.

an AsyncDockerClient

docker_config = Dict

The docker client configuration

a dict

image (*image_id_or_name*)

Returns the Image object associated to a given id

running_containers ( )
Returns all the running containers

start_container (*user_name*, *image_name*, *mapping_id*, *base_urlpath*, *volumes*, *environment=None*)
Starts a container using the given image name.

Parameters

- **user_name** (*string*) – The name of the user
- **image_name** (*string*) – A string identifying the image name.
- **mapping_id** (*str*) – A generic id used to recognize the container. it is expected to be unique (and persistent) for a specific combination of docker image (i.e. application) and setup (i.e. configuration).
- **base_urlpath** (*str*) – The base urlpath for the current user.
- **volumes** (*dict or None*) – {volume_source: {'bind': volume_target, 'mode': volume_mode}
- **environment** (*dict or None*) – Contains additional keyvalue pairs that will be exported as environment variables inside the container.

Returns

Return type A container object containing information about the started container.

Raises OperationInProgres: – if the requested mapping id is already scheduled for addition

stop_and_remove_container (*container_id*)
Idempotent removal of a container by id. If the container is there, it will be removed. If it's not there, the unexpected conditions will be logged.

Parameters **container_id** (*str*) – A string containing the container identifier.

Raises OperationInProgres: – if the requested container id is already scheduled for removal.

Exceptions

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

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

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

image

Classes

Image (*args, **kwargs)	Wrap class for the docker client images dict result.

class remoteappmanager.docker.image. **Image** (**args*, ***kwargs*)

Bases: traitlets.traitlets.HasTraits

Wrap class for the docker client images dict result. Extracts the relevant information in a convenient interface

configurables = List

a list

description = Unicode

A long description of the image.

a unicode string

docker_id = Unicode

The docker id of the image

a unicode string

env = Dict

a dict

classmethod from_docker_dict (*docker_dict*)

Converts the dict response from the dockerpy library into an instance of this class, extracting the relevant information.

Parameters **docker_dict** (*dict*) – Results of *docker.client.inspect_image* or an item of the result of *docker.client.images*

icon_128 = Unicode

A visual icon to associate to the image.

a unicode string

name = Unicode

The name of the image.

a unicode string

type = Unicode

The type of the image.

a unicode string

ui_name = Unicode

The user interface (web) name of the image.

a unicode string

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

Classes

BaseHandler (application, request, **kwargs)	Base class for the request handler.

class remoteappmanager.handlers.base_handler. **BaseHandler** (*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler, remoteappmanager.logging.logging_mixin.LoggingMixin

Base class for the request handler.

authenticator
The authenticator that is used to recognize the user.

alias of HubAuthenticator

prepare ( )
Runs before any specific handler.

render (*template_name*, ***kwargs*)
Reimplements render to pass well known information to the rendering context.

write_error (*status_code*, ***kwargs*)
Render error page for uncaught errors

home_handler

Classes

HomeHandler (application, request, **kwargs)	Render the user's home page

class remoteappmanager.handlers.home_handler. **HomeHandler** (*application*, *request*, ***kwargs*)

Bases: *remoteappmanager.handlers.base_handler.BaseHandler*

Render the user's home page

get ( )

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.

Classes

LoggingMixin (*args, **kwargs)	A HasTrait class that provides logging.

class remoteappmanager.logging.logging_mixin. **LoggingMixin** (**args*, ***kwargs*)

Bases: traitlets.traitlets.HasTraits

A HasTrait class that provides logging. Used as a mixin.

log = Instance

a logging.Logger

application

container

application

container

hub

Classes

Hub (*args, **kwargs)	Provides access to JupyterHub authenticator services.

class remoteappmanager.services.hub. **Hub** (**args*, ***kwargs*)
Bases: *remoteappmanager.logging.logging_mixin.LoggingMixin* , traitlets.traitlets.HasTraits

Provides access to JupyterHub authenticator services.

Initializes the hub connection object.

api_token = Unicode
The api token to authenticate the request

a unicode string

endpoint_url = Unicode
The url at which the Hub can be reached

a unicode string

verify_token (*cookie_name*, *encrypted_cookie*)
Verify the authentication token and grants access to the user if verified.

Parameters

- **cookie_name** (*str*) – A string containing the conventional name of the cookie.
- **encrypted_cookie** (*str*) – the cookie content, as received by JupyterHub (encrypted)

Returns **user_data** – If authentication is successful, user_data contains the user's information from jupyterhub associated with the given encrypted cookie. Otherwise the dictionary is empty.

Return type dict

reverse_proxy

Classes

ReverseProxy (*args, **kwargs)	Represents the remote reverse proxy.

class remoteappmanager.services.reverse_proxy. **ReverseProxy** (**args*, ***kwargs*)
Bases: *remoteappmanager.logging.logging_mixin.LoggingMixin* , traitlets.traitlets.HasTraits

Represents the remote reverse proxy. It is meant to have a high level API.

Initializes the reverse proxy connection object.

api_token = Unicode
The authorization API token to authenticate the request

a unicode string

endpoint_url = Unicode
The endpoint url at which the reverse proxy has its api

a unicode string

register (*urlpath*, *target_host_url*)
Register a given urlpath to redirect to a different target host. The operation is idempotent.

Parameters

- **urlpath** (*str*) – The absolute path of the url (e.g. /my/internal/service/)"
- **target_host_url** – The host to redirect to, e.g. http://127.0.0.1:31233/service/

unregister (*urlpath*)
Unregisters a previously registered urlpath. If the urlpath is not found in the reverse proxy, it will not raise an error, but it will log the unexpected circumstance.

Parameters **urlpath** (*str*) – The absolute path of the url (e.g. /my/internal/service/"

2.7 Troubleshoot

2.7.1 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 *Setup Database Accounting* for details and options on setting up the database.

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

2.7.2 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)*.

2.7.3 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.

CHAPTER 3

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the SimPhoNy Consortium nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

r

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W