META-SHARE V2.9-beta
Installation Manual

Authors: Christian Federmann, Marc Schröder, Christian Spurk

Date: August 22, 2012

Table of Contents

1 Executive Summary

This document is a guide for installing META-SHARE V2.9-beta. It is intended for system administrators setting up META-SHARE nodes. It also contains a section on how to migrate an existing META-SHARE V2.1.x installation to V2.9-beta.

2 Migrating Data and User Accounts from a V2.1.x Installation

If you already have a META-SHARE V2.1.x installation and you would like to migrate your resource descriptions, uploaded resource data, user accounts and statistics to a new META-SHARE installation, then this section is the right place to start with.

2.1 Migrated Data and Non-Migrated Data

META-SHARE V2.9-beta provides tools for the semi-automatic migration of data from a META-SHARE V2.1.x installation. The following data can be migrated with these tools:

- *resource descriptions* (metadata) – there is no need for a manual export/import cycle!

- actual *resource data* which can be directly downloaded from the node

- *user accounts* and the corresponding user profiles

- resource *ownership* information

- collected node *statistics*

This should cater for most META-SHARE V2.1.x installations. Please note, however, that the following data is *not* migrated:

- *local settings:* you have to make a new node installation before you can migrate your data, see Section 0.

- *any custom Django groups:* as a superuser you could create Django groups, however, on a V2.1.x node, this information would not have been used by META-SHARE directly. Therefore we assume that no such groups have been created.

- *permissions:* as a superuser you could assign Django permissions to users/groups, however, on a V2.1.x node, this information would not have been used by META-SHARE directly. Therefore we assume that no such groups have been created. An exception to this might be META-SHARE membership permissions: while these

permissions influence the behaviour of the system, we assume that they have not been used as they were not documented.

Even if you should have created custom Django groups or if you should have assigned permissions manually, going through the migration steps in Section 2.2 should be worthwhile. While we recommend to use the newly introduced user rights management features (see the META-SHARE Provider Manual for more information), you can still always recreate your custom groups in the new installation again and/or reassign any permissions.

2.2 Step-by-Step Migration

Here are the steps you should follow for a successful migration:

1. Get META-SHARE V2.9-beta from https://github.com/metashare/META-SHARE/downloads and install it as explained in Sections 3–6. Note: during the installation you can skip the creation of a superuser account (cf. Section 4); we will migrate your old superuser account(s) instead.

2. Copy `misc/tools/migration/export_node_from_2_1_2_to_2_9.py` to the `metashare/` folder of your old META-SHARE V2.1.x installation.

3. In the `metashare/` folder of your old META-SHARE V2.1.x installation run:
   ```
   python2.7 ./export_node_from_2_1_2_to_2_9.py /tmp/MS21_EXPORT | tee
   /tmp/ms21_export.log
   ```
   Note: you can use any other temporary directory as an argument to the Python script; just make sure that there is enough space available for at least your actual resource data plus about 100 megabytes more.

4. You can now remove the `export_node_from_2_1_2_to_2_9.py` Python script again from your old installation.

5. Copy `misc/tools/migration/import_node_to_2_9_from_2_1_2.py` to the `metashare/` folder of your new META-SHARE V2.9-beta installation.

6. In the `metashare/` folder of your new META-SHARE V2.9-beta installation run:
   ```
   python2.7 ./import_node_to_2_9_from_2_1_2.py /tmp/MS21_EXPORT | tee
   /tmp/ms21_import.log
   ```

   – *Note:* make sure to use the same temporary directory as an argument to the Python script that you have used above.

   – *Also note:* as the data from your temporary export directory is copied to the new META-SHARE installation, make sure that there is again enough space available for at least a full copy of the directory!

7. If all went as expected, you should now have a new META-SHARE V2.9-beta installation containing your old data and user accounts.

8. You can now remove the `import_node_to_2_9_from_2_1_2.py` Python script again from your new installation.

9. You can also remove your temporary export directory and any created log files, for example:
   ```
   rm -rf /tmp/MS21_EXPORT /tmp/ms21_export.log /tmp/ms21_import.log
   ```

2.3 Post-Migration Recommendations

In META-SHARE V2.1.x, users could either be editors or not, and all editors could edit all available resources. The new META-SHARE version has a more fine-grained access rights system. After the migration, users will only be allowed to edit their own resources. We highly recommend using the new user access rights system which is described in the META-SHARE Provider Manual. As a superuser you should create one or more editor groups and assign the relevant user accounts to these groups. In addition, you should assign all resources to the relevant groups.

After a successful migration and the setup/assignment of new editor groups, you can also remove all former editor users from the legacy `globaleditors` group to get a clean new node installation.

3 Installation Requirements

This section lists all software packages that are required to install and run a META-SHARE node. We assume that all software is installed into a designated META-SHARE folder, e.g., `/path/to/local/MetaShareNode/`.

Note: if you just want to run META-SHARE in **development mode**, you can skip the web server/database setup.

3.1 Software Dependencies

3.1.1 Web Server

META-SHARE is a web application that builds on a web server. Deployment has been tested with *lighttpd 1.4.29*. Other web servers can be used, but you do so on your own risk.

3.1.2 Database Software

We currently use SQLite or PostgreSQL as our database software. SQLite comes built-in with Python 2.7. Since SQLite has a number of limitations, including missing transaction management and access permission management, the preferred database is PostgreSQL.

We have tested *PostgreSQL 9.0.5*. To connect PostgreSQL and your Django project, you need to install *psycopg2*. See section 10.9 for how to install it.

3.2 Python Module Dependencies

All Python-related dependencies are bundled with META-SHARE V2.9-beta. A Linux/Unix/Mac install script is provided as `/path/to/local/MetaShareNode/install-dependencies.sh`. Run this script once after unpacking the bundle.

If you don't have a local Python installed, or it does not have the right version, the install script will build[1] and install the required python version locally below `/path/to/local/MetaShareNode/opt/`. If you use this version of Python, make sure to prepend `/path/to/local/MetaShareNode/opt/bin` to your `PATH` variable.

The compatible Python module dependencies are compiled and installed into `/path/to/local/MetaShareNode/lib/python2.7/site-packages`.

This location is automatically added to the `PYTHONPATH` in the Django server script `/path/to/local/MetaShareNode/metashare/manage.py`.

For information, the dependencies and their versions are listed here:

1) *Python 2.7.2*: available from
   http://www.python.org/download/releases/2.7.2/

2) *Django 1.3.3*: available from
   https://www.djangoproject.com/download/

3) *setuptools 0.6c11*: available from
   http://pypi.python.org/packages/source/s/setuptools/setuptools-0.6c11.tar.gz

4) *django-countries 1.0.2*: available from
   http://pypi.python.org/pypi/django-countries/1.0.2

5) *httplib2 0.7.1*: available from
   http://httplib2.googlecode.com/files/httplib2-0.7.1.tar.gz

---

[1] In order to build Python with the modules required for META-SHARE, you probably need to install some development files on your local machine first. Section 10.8 details these requirements.

6) *pygeoip 0.2.2*: available from
   http://code.google.com/p/pygeoip/downloads/

7) *Django Kronos 0.3*: available from
   https://github.com/jgorset/django-kronos

8) *Django Analytical 0.12.1*: available from
   http://packages.python.org/django-analytical/

4 Development Server

To verify that you have installed all dependencies correctly, you should first set up a development server. Proceed as follows.

1) Install all required software as described in Chapter 3, "Installation Requirements".

2) Extract the metashare-v2.9-beta.tar.gz release package into a local folder MetaShareNode.

3) Create local_settings.py for your local META-SHARE node:

   $ cd MetaShareNode/metashare

   $ cp local_settings.sample local_settings.py

   Edit at least the following constants: `DJANGO_URL`, `DJANGO_BASE`, `STORAGE_PATH`, `DEBUG`, `ADMINS`, `DATABASES`, and `EMAIL_BACKEND`. More information is available in Chapter 4.1, "Local Settings for META-SHARE Nodes".

4) Initialise database contents using manage.py syncdb.

   $ python manage.py syncdb

   Answer "yes" when asked to create a superuser account and fill in details.

5) Start an Apache Solr server for the search index:

   $ cd metashare

   $ ./start-solr.sh

6) Run tests to check that Django can load and serve META-SHARE.

   $ python manage.py test repository storage accounts sync

   This should return "OK".

7) Run Django development server using `manage.py runserver`.

   $ python manage.py runserver

   Validating models…

0 errors found

Django version 1.3.x, using settings 'metashare.settings' Development server is running at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

…

Congratulations: you have successfully started a META-SHARE V2.9-beta node in development mode. This means that all required Python/Django dependencies are functioning correctly.

4.1 Local Settings for META-SHARE Nodes

Django projects usually store all their configuration settings in a file named `settings.py`. For META-SHARE, we have split up the set of configuration parameters into two groups: *local* and *global* settings. While *global* settings can be stored within the Git repository (as they are neither security-critical nor node-dependant) we cannot do so for the *local* parameters. Hence, these are stored in an own file named `local_settings.py`, which is automatically imported by `settings.py`. The repository contains a file named `local_settings.sample` that lists and explains all *local* settings available for META-SHARE nodes. The local settings are the following:

- `DJANGO_URL = 'http://www.example.com/path/to/metashare'`
  The URL for this META-SHARE node Django application. Do not use a trailing slash (/)!You can use http://127.0.0.1:8000 when running a development mode server.

- `DJANGO_BASE = 'path/to/metashare/'`
  The base path under which Django is deployed at `DJANGO_URL`. Use a trailing slash(/). Do not use a leading slash, though. Leave empty if META-SHARE is deployed directly under the given `DJANGO_URL`.

- `FORCE_SCRIPT_NAME = ""`
  This is required when the META-SHARE node is deployed using FastCGI and for example lighttpd. There is a known bug with FCGI hosted applications and lighttpd; it basically messes up the URL after HTTP submits. `FORCE_SCRIPT_NAME= ""` fixes the issue and hence is required for lighttpd use.

- `STORAGE_PATH = '/path/to/storage/path'`
  Absolute path to the local *storage base*, i.e., the folder in which local `StorageObject` instances can store their attachments. You need to supply an existing path here, even for development mode! This folder will contain data related to your language resources, so choose a suitable location that is accessible but safe.

- `DEBUG, TEMPLATE_DEBUG, DEBUG_JS`

Debug settings – setting `DEBUG=True` will give exception stacktraces on the website, for example. This may include sensitive information, so use with care, preferably only for local development servers.

- `ADMINS`

    Configure administrators for this Django project. If `DEBUG=False`, all errors will be reported as e-mails to these persons.

- `DATABASES`

    Configures the database settings for Django. For SQLite, use the following settings:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '{0}/testing.db'.format(ROOT_PATH)
    }
}
```

    For PostgreSQL, the following settings are required:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'metashare',
        'USER': 'db_user',
        'PASSWORD': 'db_password',
        'HOST': 'localhost',
        # Set to empty string for default.
        'PORT': '',
        # This is required to make import more robust.
        'OPTIONS': {
          'autocommit': True,
        }
    }
}
```

- `EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'`

    Settings for sending mail. Production servers should use the SMTP email backend.

- `TIME_ZONE = 'Europe/Berlin'`

    Local time zone for this installation.

5 Deployment Server

For deployment, we assume that you have downloaded and installed the *lighttpd web server* and a *PostgreSQL database*. You have to adapt `start_server.sh` and `stop_server.sh` with correct IP addresses and port numbers. The IP addresses should be identical to the one you added to your `lighttpd.conf`, the port number, of course, needs to be different from the web server's. You can test your PostgreSQL database by calling `manage.py syncdb`; this

will complain if it cannot properly access the database. Once both the web server and the database are ready, use `start_server.sh` to start the threaded production server via FastCGI; don't forget to set `DEBUG=False`! `stop_server.sh` of course stops the FastCGI server and the corresponding lighttpd process.

6 Solr Server for Browsing and Searching

The META-SHARE release comes with a pre-configured Solr server used to index the META-SHARE database for browsing and searching.

To start the preconfigured Solr server, go to the `metashare` folder and run:

```
./start-solr.sh
```

To stop a running Solr server, go to the `metashare` folder and run:

```
./stop-solr.sh
```

These commands must be run by hand for the development server; they are included in the `start-server.sh` and `stop-server.sh` scripts used for the deployment server.

This should be all you need for usual operation. The following subsections are required only for people who want to understand in depth how to operate and configure the Solr server.

6.1 Installing Solr

1. Make sure you have Java 1.6 or later (run `java -version` to check!).
2. Download the latest version of Solr from here.
3. Unzip into a folder, henceforth called `$SOLR_DIR`.
4. Go to `misc/solr-config-sample` in your local META-SHARE-Software repository and run:
   `./create_solr_config.sh "$SOLR_DIR"`
   This will configure your Solr server with a sample configuration. It will overwrite the default Solr configuration. After this step you will have a Solr server which is configured with two cores (→ indexes) `main` and `testing`.
5. Change directory to `$SOLR_DIR/example`.
6. Run
   `java -jar start.jar`
7. Open a web browser and go to http://localhost:8983/solr/main/admin/. You should be able to see Solr's admin interface for the `main` core.

For further help go to the Solr Tutorial page.

6.2 Keeping the Solr Configuration Up-to-Date

As development on the search functionality continues, you may have to occasionally recreate your Solr configuration. Before doing that you have to shut down your Solr server (Ctrl+C). Now you can either:

- Follow the steps in the previous section. This will erase all your index data. After that, run `python manage.py rebuild_index` to rebuild your index from the current database content.

- Or you manually update the Solr configuration by going through the following steps.

6.3 Manually Updating the Solr Configuration

1. Create Solr schema files automatically by running:
   `python manage.py build_solr_schema`
   The XML output of this command should go into both

   `$SOLR_DIR/example/solr/main/conf/schema.xml` and

   `$SOLR_DIR/example/solr/testing/conf/schema.xml.`

2. If there should be any changes in the files in `misc/solr-config-sample`, then copy these files to both `$SOLR_DIR/example/solr/main/conf` and `$SOLR_DIR/example/solr/testing/conf`

3. Restart the Solr server.

4. If you already have any data in your database, then manually build the search index once. Run:
   `python manage.py rebuild_index`

Any future changes and additions to your database should automatically be reflected in the search index. A manual rebuild should not be required anymore (except when working on the indexing itself).

7 Importing and Exporting Resources

Metadata descriptions of language resources can be imported into the META-SHARE software from XML files obeying the META-SHARE schema format. Likewise, the metadata descriptions in the META-SHARE database can be exported into XML files in the format defined by the META-SHARE XML schema.

7.1 Importing XML Files into META-SHARE

There are two possibilities of importing language resource XML descriptions which are outlined in the following sections.

In general, all files to import should be schema-valid according to the current META-SHARE XML schema file which is located in `misc/schema/v2.1/META-SHARE-Resource.xsd`. Please use an XML schema validator to verify that the import files are valid before trying to import them into META-SHARE. For example, you can use `xmllint` like so:

```
xmllint --schema META-SHARE-Resource.xsd data.xml
```

Schema validity is not strictly required by the importer; reasonable efforts are made to import partial or erroneous XML files. However, in order to avoid loosing data, please try to make your files schema valid.

7.1.1 Importing from the Command Line

META-SHARE comes with a tool called `import_xml.py` to import XML files describing language resources into the system. To import, run `import_xml.py` as follows:

```
python import_xml.py <file.xml|archive.zip> [<file.xml|archive.zip> …]
```

In other words, you can provide one or more individual XML files or zip files containing XML files. The script will print a summary count of successfully imported and erroneous files at the end.

7.1.2 Importing from the Editor

An alternative way of importing resources is provided by the "Upload" menu item of the editor. There you can also provide individual XML files or zip files containing XML files. Compared to the shell importer, the upload size is limited, though.

7.2 Exporting XML Files from META-SHARE

META-SHARE aims to be an open platform and therefore allows for the export of resources in the original XML format. As with the import, there are two possible ways for exporting, both of which are described in the following sections.

7.2.1 Exporting from the Command Line

The script `export_xml.py` will export *all* entries from the database into a zip archive containing one XML file per resource. The script requires a valid META-SHARE V2.9-beta database. It can be run as follows:

```
python export_xml.py <archive.zip>
```

The resulting archive is suitable for import in any META-SHARE V2.1 (or later) installation.

7.2.2 Exporting from the Editor

As an alternative to the shell exporter you may export resource descriptions from the editor.

- A single resource XML description can be exported from the main editor page of the resource using the "Export Resource Description to XML" button at the top of the page.

- A bundle of freely selectable resources may be exported as a zip archive from the "Editable Resources" page using the "Action" menu. The resulting archive is suitable for import in any META-SHARE V2.1 (or later) installation.

7.3 Copying Data between META-SHARE Nodes

Future versions of META-SHARE will support automatic synchronization of the data between a configurable set of META-SHARE nodes. In the meantime, the data provided by multiple nodes can be manually synchronized by means of XML export and import.

Assume a set of language resource descriptions have been authored on node A using the META-SHARE metadata editor. These descriptions are to be made available on node B. The sequence of steps is the following.

1. Export the data from node A using one of the possibilities described in Section 7.2.
2. Import the data into node B as described in Section 7.1.

7.4 Migrating Data from a Previous META-SHARE Installation

The procedure for upgrading a META-SHARE node from a previous version can be sketched as follows:

1. Export the metadata records from the previous version as XML files;

2. Upgrade the XML files to the latest version of the metadata schema;

3. Import the upgraded XML files into the latest META-SHARE software.

The following subsections provide details for each step.

7.4.1 Exporting the Metadata from META-SHARE

Section 7.2.1 explains how all language resource descriptions from a META-SHARE node can be exported.

A procedure to export data from META-SHARE V1.0 is described in the META-SHARE V1.1 installation manual. It is not repeated here because all known users have upgraded to V1.1 or newer.

7.4.2 Upgrade the XML Files to the Latest Metadata Format

META-SHARE V2.9-beta comes with a `metashare-resource-upgrader` tool which automates, to the extent possible, the conversion of previous versions of the META-SHARE resource XML format to the newest version. It processes all XML files in a source folder and saves into a target folder a version of each file converted to the newest format.

The tool uses a sequence of XSLT stylesheets[2] to convert metadata from previous versions. Supported data formats are META-SHARE V1.0, META-SHARE V1.1, and META-SHARE

---

[2] The actual stylesheets are located in `META-SHARE/misc/tools/metashare-resource-upgrader/src/src/main/resources/metashare/upgrader/conversion`. They can also be applied manually using any XSLT transformer.

V2.0. The stylesheets automate the conversion to the extent possible, but due to differing requirements, it may be that META-SHARE V1.x resources need manual work before they can be imported into META-SHARE V2.1 (or higher).

To verify if a file can be imported as is, or if manual work is required before it can be imported, `metashare-resource-upgrader` automatically performs an XML Schema validation of each target file after conversion, and prints out the result on the command line.

To use the `metashare-resource-upgrader`, proceed as follows:

1. Unpack the exported zip file into a folder, say `sourcedir`.

2. Run the conversion tool as follows:

   ```
   cd META-SHARE/misc/tools/metashare-resource-upgrader
   java -jar metashare-resource-upgrader-2.1.jar sourcedir targetdir
   ```

3. The target folder `targetdir` may exist but must not contain any files.

4. The tool will inform you for every converted resource whether it is schema-valid or not. For the files in `targetdir` that are not schema-valid, you need to manually identify the problems by means of a schema-validating parser and fix them. One option is `xmllint`:

   ```
   xmllint --schema META-SHARE/misc/schema/v2.1/META-SHARE-Resource.xsd
   targetdir/resource-1.xml
   ```

When all XML files in `targetdir` are schema-valid, you are ready to proceed to the import step into v2.1 (or higher).

7.4.3 Importing the Metadata into a New META-SHARE Installation

Once the converted files are schema-valid, the import step should be straightforward, as described in Section 7.1.1:

```
python import_xml.py targetdir/*.xml
```

8 Setting up Editor User Accounts

For information on how to set up and manage user accounts, please see the META-SHARE provider manual.

9 Search Engine Optimization

META-SHARE integrates the most common techniques for Search Engine Optimization (SEO). In order to check whether SEO works as it should, META-SHARE also integrates "django-analytical", a package for easily integrating analytics services like Google Analytics

or Clicky. If you would like to use any analytics service, then just add the corresponding configuration to your `local_settings.py` file. Valid configuration options for the supported analytics services can be found at http://packages.python.org/django-analytical/install.html#enabling-the-services.

10 Frequently Asked Questions

This section compiles a number of the most frequently asked questions.

10.1 I Want to use MySQL and/or Apache

A: It may be possible to get these to work, but we have not tested these configurations and therefore cannot provide any support for them. The recommended database and web server technologies are listed in section 3.1.

10.2 I Need Help Configuring lighttpd

A: The release includes a sample `lighttpd.conf` configuration file which you can use as the basis for your configuration.

Also, look at the scripts `start-server.sh` and `stop-server.sh` which should show you how to start up and shut down the production server.

10.3 I am Getting Storage Errors when Importing or Saving

```
File "/usr/local/MetaShareNode/metashare/../metashare/storage/models.py",=
 line 254, in save    mkdir(self._storage_folder()) OSError: [Errno 2] No
such file or directory: '/home/storage/b557040eff1d11=
e09075080027fee6a9b7ffe41433e94b19844c6038a825a145'  File
"/usr/local/MetaShareNode/metashare/../metashare/storage/models.py",= line
254, in save    mkdir(self._storage_folder())  OSError: [Errno 2] No such
file or directory:
'/home/storage/b557040eff1d11=e09075080027fee6a9b7ffe41433e94b19844c6038a82
5a145'
```

A: The first thing to verify is whether the `STORAGE_PATH` setting in `local_settings.py` points to a valid and existing folder – see section 4.1 for details.

10.4 Why Can Django not Serve the Static Files?

A: While in principle, Django could also serve those static files, this is not recommended for production use – it makes a lot more sense to have a dedicated, lightweight web server handle that task. Some more information on combining Django and lighttpd is available here: https://docs.djangoproject.com/en/dev/howto/deployment/fastcgi/#lighttpd-setup

10.5 PostgreSQL Error Message

```
--- File "/usr/lib/python2.7/site-
packages/django/db/backends/postgresql_psycopg2/base.py", line 24, in
<module>    raise ImproperlyConfigured("Error loading psycopg2 module: %s"
% e) django.core.exceptions.ImproperlyConfigured: Error loading psycopg2
module: No module named psycopg2 ---
```

A: Seems like you are trying to use PostgreSQL but you have not installed the *psycopg2* dependency. See Section 10.9 for how to install it.

10.6 Problems with Importing XML Files

Q: We are trying to use `import_xml.py` to import XML files into the database. We are using an XML file that validates against the schema, but we get the following error:

```
$ /usr2/MetaShareNode/software/bin/python import_xml.py
  ApertiumLMFBasqueDictionary.xml

Importing XML file: "ApertiumLMFBasqueDictionary.xml"
  Could not import XML file into database!
…
```

A: If you encounter this error, please first check that the XML file is indeed schema-valid with respect to the latest schema files. If so, there might be a bug – please send us the example file if possible so that we can reproduce and fix it: helpdesk-technical@meta-share.eu

10.7 Updating the GeoIP Database for Statistics Collection

Q: The country-based statistics do not seem to properly work anymore.

A: For statistical purposes, META-SHARE collects information about the country of origin of web site visitors. In this process, the IP address of the visiting user is converted to the country using the GeoLite Country database. We are shipping a fixed version of this database. As IP address to country mappings may change over time, you might also want to update the used database for better statistics results. In order to update the current version of the database, download a new version and move the file `GeoIP.dat` into the directory `/path/to/local/MetaShareNode/metashare/stats/resources/`. New versions of the database are available here:

http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz

10.8 How can I Correctly Build Python for META-SHARE?

A: Building and installing the Python version which comes with META-SHARE (see Section 3.2), has a few dependencies. You need to have *libsqlite3-dev*, *libssl-dev* und *zlib1g-dev* installed. Please note that these packages may have different names depending on your

Linux/Unix distribution. On an older Ubuntu without Python 2.7 you might also use the following command to get all required build dependencies: `apt-get build-dep python2.6`

10.9 How do I Install psycopg2 for using PostgreSQL?

A: We recommend installing *pscopg2* using the *pip* installer (`pip install psycopg2`). For this to work on Ubuntu, you have to first install the two packages *libpq-dev* and *python-dev* (`apt-get install libpq-dev python-dev`). On some versions of Ubuntu, you may also have to use an older *pscopg2* version which you can install like this: `pip install psycopg2==2.4.1`

Otherwise *pscopg2* is available on the following website:

http://pypi.python.org/pypi/psycopg2/