Modoboa Documentation

Release 1.2.0

Antoine Nguyen

December 31, 2014

Contents

CHAPTER 1

Getting started

1.1 Installation

1.1.1 Requirements

- Python version 2.6+
- Django version 1.6+
- south version 0.7+
- lxml python module
- pycrypto python module
- rrdtool python binding
- sievelib python module
- chardet python module
- argparse python module
- reversion python module

1.1.2 Get Modoboa

You can choose between two options:

- Use the Python package available on the PyPI
- Download the sources tarball

The easiest one is to install it from PyPI. Just run the following command and you're done:

```
$ pip install modoboa
```

If you prefer to use the tarball, download the latest one and run the following procedure:

```
$ tar xzf modoboa-<version>.tar.gz
$ cd modoboa-<version>
$ python setup.py install
```

All dependencies will be installed regardless the way you chose. The only exception concerns the RRDtool binding because there isn't any python package available, it is directly provided with the official tarball.

Fortunately, all major distributions include a ready-to-use package. On Debian/Ubuntu:

```
$ apt-get install libcairo2-dev libpango1.0-dev librrd-dev
$ apt-get install python-rrdtool
```

virtualenv users

When you deploy an application using virtualenv, you may have to compile some dependencies. For example, modoboa relies on lxml, which is a C python module. In order to install it, you will need to install the following requirements:

- python development files
- libxslt development files
- libxml2 development files
- libz development files

On a Debian like system, just run the following command:

```
$ apt-get install python-dev libxml2-dev libxslt-dev zlib1g-dev
```

1.1.3 Database

Thanks to Django, Modoboa supports several databases. Depending on the one you will use, you must install the appropriate python package:

- mysqldb for MySQL
- psycopg2 for PostgreSQL

Then, create a user and a database that will be used by Modoboa. Make sure your database is using UTF8 as a default charset.

1.1.4 Deployment

modoboa-admin.py, a command line tool, lets you deploy a *ready-to-use* Modoboa site using only one instruction:

```
$ modoboa-admin.py deploy modoboa_example --dbaction install --collectstatic [--with-amavis] [--dburl
```

Just answer the few questions and you're done. You can now go to the *First use* section.

Note: The *–with-amavis* option must be set only if you intend to use the *Amavisd-new frontend*.

In case you need a **silent installation** (e.g. if you're using Salt-Stack, Ansible or whatever), it's possible to supply the database credentials as commandline arguments.

You can see the complete option list by running the following command:

```
$ modoboa-admin.py help deploy
```

Note: *–dburl database-url* for the modoboa database credentials *–amavis_dburl database-url* for the amavis database credentials

Your database-url should meet the following syntax: `scheme://[user:pass@][host:port]/dbname`

or

`sqlite:////full/path/to/your/database/file.sqlite`

Available *schemes* are: * postgres * postgresql * postgis * mysql * mysql2 * sqlite

---

Note: If you plan to serve Modoboa using a URL prefix, you must change the value of the `LOGIN_URL` parameter to `LOGIN_URL = '/<prefix>/accounts/login/'`.

1.1.5 First use

Your installation should now have a default super administrator:

- Username: `admin`
- Password: `password`

It is **strongly** recommended to change this password the first time you log into Modoboa.

To check if your installation works, just launch the embedded HTTP server:

```
$ python manage.py runserver
```

You should be able to access Modoboa at http://localhost:8000/.

For a fully working interface using the embedded HTTP server, you need to set the `DEBUG` parameter in settings.py to `True`.

For a production environment, we recommend using a stable webserver like *Apache2* or *Nginx*. Don't forget to set `DEBUG` back to `False`.

1.2 Upgrading an existing installation

This section contains all the upgrade procedures required to use newest versions of Modoboa.

Note: Before running a migration, we recommend that you make a copy of your existing database.

1.2.1 Latest version

Fetch the latest version (see *Get Modoboa*) and install it. `pip` users, just run the following command:

```
$ pip install modoboa==<VERSION>
```

Replace `<VERSION>` by the appropriate value.

As for a fresh installation, `modoboa-admin.py` can be used to upgrade your local configuration. To do so, remove the directory where your instance was first deployed:

```
$ rm -rf <modoboa_instance_dir>
```

Warning: If you customized your configuration file (`settings.py`) with non-standard settings, you'll have to re-apply them.

Then, run `modoboa-admin.py deploy`:

```
$ modoboa-admin.py deploy <modoboa_instance_dir> --dbaction upgrade --collectstatic [--with-amavis]
```

If you prefer the manual way, check if *Specific upgrade instructions* are required according to the version you're installing.

To finish, restart the web server process according to the environment you did choose. See *Web servers* for more details.

1.2.2 Specific upgrade instructions

1.2.0

A new notification service let administrators know about new Modoboa versions. To activate it, you need to update the `TEMPLATE_CONTEXT_PROCESSORS` variable like this:

```
from django.conf import global_settings

TEMPLATE_CONTEXT_PROCESSORS = global_settings.TEMPLATE_CONTEXT_PROCESSORS + (
  'modoboa.core.context_processors.top_notifications',
)
```

and to define the new `MODOBOA_API_URL` variable:

```
MODOBOA_API_URL = 'http://api.modoboa.org/1/'
```

The location of external static files has changed. To use them, add a new path to the `STATICFILES_DIRS`:

```
# Additional locations of static files
STATICFILES_DIRS = (
  # Put strings here, like "/home/html/static" or "C:/www/django/static".
  # Always use forward slashes, even on Windows.
  # Don't forget to use absolute paths, not relative paths.
  "<path/to/modoboa/install/dir>/bower_components",
)
```

Run the following commands to define the hostname of your instance:

```
$ cd <modoboa_instance_dir>
$ python manage.py set_default_site <hostname>
```

If you plan to use the Radicale extension:

1. Add `'modoboa.extensions.radicale'` to the `MODOBOA_APPS` variable
2. Run the following commands:

   ```
   $ cd <modoboa_instance_dir>
   $ python manage.py syncdb
   ```

1.1.7: manual learning for SpamAssassin

A new feature allows administrators and users to manually train SpamAssassin in order to customize its behaviour.

Check *Manual SpamAssassin learning* to know more about this feature.

1.1.6: Few bugfixes

Catchall aliases were not really functional until this version as they were eating all domain traffic.

To fix them, a postfix map file (`sql-mailboxes-self-aliases.cf`) has been re-introduced and must be listed into the `virtual_alias_maps` setting. See *Configuration* for the order.

1.1.2: Audit trail issues

Update the `settings.py` file as follows:

1. Remove the `'reversion.middleware.RevisionMiddleware'` middleware from the `MIDDLEWARE_CLASSES` variable
2. Add the new `'modoboa.lib.middleware.RequestCatcherMiddleware'` middleware at the end of the `MIDDLEWARE_CLASSES` variable

1.1.1: Few bugfixes

For those who installed Dovecot in a non-standard location, it is now possible to tell Modoboa where to find it. Just define a variable named `DOVECOT_LOOKUP_PATH` in the `settings.py` file and include the appropriate lookup path inside:

```
DOVECOT_LOOKUP_PATH = ("/usr/sbin/dovecot", "/usr/local/sbin/dovecot")
```

1.1.0: relay domains and better passwords encryption

Due to code refactoring, some modifications need to be done into `settings.py`:

1. `MODOBOA_APPS` must contain the following applications:

   ```
   MODOBOA_APPS = (
     'modoboa',
     'modoboa.core',
     'modoboa.lib',

     'modoboa.extensions.admin',
     'modoboa.extensions.limits',
     'modoboa.extensions.postfix_autoreply',
     'modoboa.extensions.webmail',
     'modoboa.extensions.stats',
     'modoboa.extensions.amavis',
     'modoboa.extensions.sievefilters',
   )
   ```

2. Add `'modoboa.extensions.postfix_relay_domains'` to `MODOBOA_APPS`, just before `'modoboa.extensions.limits'`
3. `AUTH_USER_MODEL` must be set to `core.User`
4. Into `LOGGING`, replace `modoboa.lib.logutils.SQLHandler` by `modoboa.core.loggers.SQLHandler`

Then, run the following commands to migrate your installation:

```
$ python manage.py syncdb
$ python manage.py migrate core 0001 --fake
$ python manage.py migrate
$ python manage.py collectstatic
```

Finally, update both *Dovecot* and *Postfix* queries.

1.0.1: operations on mailboxes

The way Modoboa handles **rename** and **delete** operations on mailboxes has been improved. Make sure to consult *Operations on the file system* and *Postfix configuration*. Look at the `smtpd_recipient_restrictions` setting.

Run `modoboa-admin.py postfix_maps --dbtype <mysql|postgres|sqlite> <tempdir>` and compare the files with those that postfix currently use. Make necessary updates in light of the differences

1.0.0: production ready, at last

Configuration file update

Several modifications need to be done into `settings.py`.

1. Add the following import statement:

   ```
   from logging.handlers import SysLogHandler
   ```

2. Set the `ALLOWER_HOSTS` variable:

   ```
   ALLOWED_HOSTS = [
       '<your server fqdn>',
   ]
   ```

3. Activate the `django.middleware.csrf.CsrfViewMiddleware` middleware and add the `reversion.middleware.RevisionMiddleware` middleware to `MIDDLEWARE_CLASSES` like this:

   ```
   MIDDLEWARE_CLASSES = (
       'django.middleware.common.CommonMiddleware',
       'django.contrib.sessions.middleware.SessionMiddleware',
       'django.middleware.csrf.CsrfViewMiddleware',
       'django.contrib.auth.middleware.AuthenticationMiddleware',
       'django.contrib.messages.middleware.MessageMiddleware',
       'django.middleware.locale.LocaleMiddleware',
       # Uncomment the next line for simple clickjacking protection:
       # 'django.middleware.clickjacking.XFrameOptionsMiddleware',
       'reversion.middleware.RevisionMiddleware',

       'modoboa.lib.middleware.AjaxLoginRedirect',
       'modoboa.lib.middleware.CommonExceptionCatcher',
       'modoboa.lib.middleware.ExtControlMiddleware',
   )
   ```

4. Add the `reversion` application to `INSTALLED_APPS`

5. Remove all modoboa's application from `INSTALLED_APPS` and put them into the new `MODOBOA_APPS` variable like this:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'south',
    'reversion',
)

# A dedicated place to register Modoboa applications
# Do not delete it.
# Do not change the order.
MODOBOA_APPS = (
    'modoboa',
    'modoboa.auth',
    'modoboa.admin',
    'modoboa.lib',
    'modoboa.userprefs',

    'modoboa.extensions.limits',
    'modoboa.extensions.postfix_autoreply',
    'modoboa.extensions.webmail',
    'modoboa.extensions.stats',
    'modoboa.extensions.amavis',
    'modoboa.extensions.sievefilters',
)

INSTALLED_APPS += MODOBOA_APPS
```

6. Set the `AUTH_USER_MODEL` variable like this:

```
AUTH_USER_MODEL = 'admin.User'
```

7. Modify the logging configuration as follows:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'formatters': {
        'syslog': {
            'format': '%(name)s: %(levelname)s %(message)s'
        },
    },
    'handlers': {
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler'
        },
        'console': {
            # logging handler that outputs log messages to terminal
            'class': 'logging.StreamHandler',
```

```
            #'level': 'DEBUG', # message level to be written to console
        },
        'syslog-auth': {
            'class': 'logging.handlers.SysLogHandler',
            'facility': SysLogHandler.LOG_AUTH,
            'formatter': 'syslog'
        },
        'modoboa': {
            'class': 'modoboa.lib.logutils.SQLHandler',
        }
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins'],
            'level': 'ERROR',
            'propagate': True,
        },
        'modoboa.auth': {
            'handlers': ['syslog-auth', 'modoboa'],
            'level': 'INFO',
            'propagate': False
        },
        'modoboa.admin': {
            'handlers': ['modoboa'],
            'level': 'INFO',
            'propagate': False
        }
    }
}
```

Postfix and Dovecot configuration update

It is necessary to update the queries used to retrieve users and mailboxes:

1. Run `modoboa-admin.py postfix_maps --dbtype <mysql|postgres> <tempdir>` and compare the files with those that postfix currently use. Make necessary updates in light of the differences
2. Into `dovecot-sql.conf`, update the `user_query` query, refer to *MySQL users* or *PostgreSQL users*
3. Update dovecot's configuration to activate the new *quota related features*

Migration issues

When running the `python manage.py syncdb --migrate` command, you may encounter the following issues:

1. Remove useless content types

   If the script asks you this question, just reply **no**.

2. South fails to migrate `reversion`

   Due to the admin user model change, the script `0001_initial.py` may fail. Just deactivate `reversion` from `INSTALLED_APPS` and run the command again. Once done, reactivate `reversion` and run the command one last time.

1.3 Configuration

1.3.1 Online parameters

Modoboa provides online panels to modify internal parameters. There are two available levels:

- Application level: global parameters, define how the application behaves. Available at *Modoboa* > *Parameters*
- User level: per user customization. Available at *User* > *Settings* > *Preferences*

Regardless level, parameters are displayed using tabs, each tab corresponding to one application.

General parameters

The *admin* application exposes several parameters, they are presented below:

Name	Description	Default value
Authentication type	The backend used for authentication	Local
Default password scheme	Scheme used to crypt mailbox passwords	crypt
Secret key	A key used to encrypt users' password in sessions	random value
Handle mailboxes on filesystem	Rename or remove mailboxes on the filesystem when they get renamed or removed within Modoboa	no
Mailboxes owner	The UNIX account who owns mailboxes on the filesystem	vmail
Automatic account removal	When a mailbox is removed, also remove the associated account	no
Maximum log record age	The maximum age in days of a log record	365
Items per page	Number of displayed items per page	30
Default top redirection	The default redirection used when no application is specified	userprefs

Note: If you are not familiar with virtual domain hosting, you should take a look at postfix's documentation. This How to also contains useful information.

Note: A random secret key will be generated each time the *Parameters* page is refreshed and until you save parameters at least once.

Note: Specific LDAP parameters are also available, see *LDAP authentication*.

1.3.2 Media files

Modoboa uses a specific directory to upload files (ie. when the webmail is in use) or to create ones (ex: graphical statistics). This directory is named `media` and is located inside modoboa's installation directory (called `modoboa_site` in this documentation).

To work properly, the system user which runs modoboa (`www-data`, `apache`, whatever) must have write access to this directory.

1.3.3 Customization

Custom logo

You have the possibility to use a custom logo instead of the default one on the login page.

To do so, open the `settings.py` file and add a `MODOBOA_CUSTOM_LOGO` variable. This variable must contain the relative URL of your logo under `MEDIA_URL`. For example:

```
MODOBOA_CUSTOM_LOGO = os.path.join(MEDIA_URL, "custom_logo.png")
```

Then copy your logo file into the directory indicated by `MEDIA_ROOT`.

1.3.4 Host configuration

Note: This section is only relevant when Modoboa handles mailboxes renaming and removal from the filesystem.

To manipulate mailboxes on the filesystem, you must allow the user who runs Modoboa to execute commands as the user who owns mailboxes.

To do so, edit the `/etc/sudoers` file and add the following inside:

```
<user_that_runs_modoboa> ALL=(<mailboxes owner>) NOPASSWD: ALL
```

Replace values between <> by the ones you use.

1.3.5 Time zone and language

Modoboa is available in many languages.

To specify the default language to use, edit the `settings.py` file and modify the `LANGUAGE_CODE` variable:

```
LANGUAGE_CODE = 'fr' # or 'en' for english, etc.
```

Note: Each user has the possibility to define the language he prefers.

In the same configuration file, specify the timezone to use by modifying the `TIME_ZONE` variable. For example:

```
TIME_ZONE = 'Europe/Paris'
```

1.3.6 Sessions management

Modoboa uses Django's session framework to store per-user information.

Few parameters need to be set in the `settings.py` configuration file to make Modoboa behave as expected:

```
SESSION_EXPIRE_AT_BROWSER_CLOSE = False # Default value
```

This parameter is optional but you must ensure it is set to `False` (the default value).

The default configuration file provided by the `modoboa-admin.py` command is properly configured.

1.3.7 LDAP

Authentication

Modoboa supports external LDAP authentication using the following extra components:

- Python LDAP client
- Django LDAP authentication backend

If you want to use this feature, you must first install those components:

```
$ pip install python-ldap django-auth-ldap
```

Then, all you have to do is to modify the `settings.py` file. Add a new authentication backend to the *AUTHENTICATION_BACKENDS* variable, like this:

```
AUTHENTICATION_BACKENDS = (
  'modoboa.lib.authbackends.LDAPBackend',
  'modoboa.lib.authbackends.SimpleBackend',
)
```

Finally, go to *Modoboa* > *Parameters* > *General* and set *Authentication type* to LDAP.

From there, new parameters will appear to let you configure the way Modoboa should connect to your LDAP server. They are described just below:

Name	Description	Default value
Server address	The IP address of the DNS name of the LDAP server	localhost
Server port	The TCP port number used by the LDAP server	389
Use a secure connection	Use an SSL/TLS connection to access the LDAP server	no
Authentication method	Choose the authentication method to use	Direct bind
User DN template (direct bind mode)	The template used to construct a user's DN. It should contain one placeholder (ie. `%(user)s`)	
Bind BN	The distinguished name to use when binding to the LDAP server. Leave empty for an anonymous bind	
Bind password	The password to use when binding to the LDAP server (with 'Bind DN')	
Search base	The distinguished name of the search base	
Search filter	An optional filter string (e.g. '(objectClass=person)'). In order to be valid, it must be enclosed in parentheses.	(mail=%(user)s)
Password attribute	The attribute used to store user passwords	userPassword
Active Directory	Tell if the LDAP server is an Active Directory one	no
Administrator groups	Members of those LDAP Posix groups will be created ad domain administrators. Use ';' characters to separate groups.	
Groups search base	The distinguished name of the search base used to find groups	
Domain/mailbox creation	Automatically create a domain and a mailbox when a new user is created just after the first successful authentication. You will generally want to disable this feature when the relay domains extension is in use	yes

If you need additional parameters, you will find a detailled documentation here.

Once the authentication is properly configured, the users defined in your LDAP directory will be able to connect to Modoboa, the associated domain and mailboxes will be automatically created if needed.

The first time a user connects to Modoboa, a local account is created if the LDAP username is a valid email address. By default, this account belongs to the *SimpleUsers* group and it has a mailbox.

To automatically create domain administrators, you can use the **Administrator groups** setting. If a LDAP user belongs to one the listed groups, its local account will belong to the *DomainAdmins* group. In this case, the username is not necessarily an email address.

Users will also be able to update their LDAP password directly from Modoboa.

Note: Modoboa doesn't provide any synchronization mechanism once a user is registered into the database. Any modification done from the directory to a user account will not be reported to Modoboa (an email address change for example). Currently, the only solution is to manually delete the Modoboa record, it will be recreated on the next user login.

1.3.8 Database maintenance

Cleaning the logs table

Modoboa logs administrator specific actions into the database. A clean-up script is provided to automatically remove oldest records. The maximum log record age can be configured through the online panel.

To use it, you can setup a cron job to run every night:

```
0 0 * * * <modoboa_site>/manage.py cleanlogs
#
# Or like this if you use a virtual environment:
# 0 0 * * * <virtualenv path/bin/python> <modoboa_site>/manage.py cleanlogs
```

Cleaning the session table

Django does not provide automatic purging. Therefore, it's your job to purge expired sessions on a regular basis.

Django provides a sample clean-up script: `django-admin.py cleanup`. That script deletes any session in the session table whose `expire_date` is in the past.

For example, you could setup a cron job to run this script every night:

```
0 0 * * * <modoboa_site>/manage.py cleanup
#
# Or like this if you use a virtual environment:
# 0 0 * * * <virtualenv path/bin/python> <modoboa_site>/manage.py cleanup
```

CHAPTER 2

Plugins

2.1 Overview

2.1.1 Enable or disable a plugin

Modoboa provides an online panel to control plugins activation. You will find it at *Modoboa* > *Extensions*.

To activate a plugin, check the corresponding box and click on the *Apply* button.

To deactivate a plugin, uncheck the corresponding box and click on the *Apply* button.

2.2 Per-admin limits

This plugin offers a way to define limits about how many objects (aliases, mailboxes) a domain administrator can create.

It also brings a new administrative role: `Reseller`. A reseller is a domain administrator that can also manipulate domains and assign permissions to domain administrators.

If you don't want to limit a particular object type, just set the associated value to -1.

Default limits applied to new administrators can be changed through the *Modoboa* > *Parameters* > *Limits* page.

2.3 Postfix relay domains support

This plugin adds the support for relay domains using postfix. You can use it when the MTA managed by Modoboa is not the final destination of one or several domains.

If activated, two new objects will be available from the *Domains* listing page: *relay domain* and *relay domain alias*.

The extension is compatible with the *amavis* and *limits* ones. Resellers will be able to create both new objects.

Replace <driver> by the name of the database you use.To tell Postfix this feature exists, you must generate two new map files and then update your configuration.

To generate the map files, run the following command:

```
$ modoboa-admin.py postfix_maps --categories relaydomains --dbtype <the database you use> <path>
```

Replace values between <> by yours.

Edit the `/etc/postfix/main.cf` file and copy the following lines inside:

```
relay_domains = <driver>:/etc/postfix/sql-relaydomains.cf
transport_maps =
    <driver>:/etc/postfix/sql-relaydomains-transport.cf
    <driver>:/etc/postfix/sql-relaydomain-aliases-transport.cf

smtpd_recipient_restrictions =
    permit_mynetworks
    reject_unauth_destination
    check_recipient_access
        <driver>:/etc/postfix/sql-relay-recipient-verification.cf
```

Replace `<driver>` by the name of the database you use.

Reload postfix.

2.4 Amavisd-new frontend

This plugin provides a simple management frontend for amavisd-new. The supported features are:

- SQL quarantine management : available to administrators or users, possibility to delete or release messages
- Per domain customization (using policies): specify how amavisd-new will handle traffic

---

Note: The per-domain policies feature only works for new installations. Currently, you can't use modoboa with an existing database (ie. with data in `users` and `policies` tables).

Note: This plugin requires amavisd-new version **2.7.0** or higher. If you're planning to use the *Self-service mode*, you'll need version **2.8.0**.

2.4.1 Quick Amavis setup

By default, amavis doesn't use a database. To configure this behaviour, you first need to create a dedicated database. This step is a bit manual since no *ready-to-use* SQL schema is provided by amavis. The information is located inside README files, one for MySQL and one for PostgreSQL.

Then, you must tell amavis to use this database for lookups and quarantined messages storing. Here is a working configuration sample:

```
@lookup_sql_dsn =
  (['DBI:<driver>:database=<database>;host=<dbhost>;port=<dbport>', '<dbuser>', '<password>]']);

@storage_sql_dsn =
  (['DBI:<driver>:database=<database>;host=<dbhost>;port=<dbport>', '<dbuser>', '<password>]']);

# PostgreSQL users NEED this parameter!
# MySQL users only need this parameter is email addresses are stored
# using the VARBINARY type.
$sql_allow_8bit_address = 1;

$virus_quarantine_method = 'sql:';
$spam_quarantine_method = 'sql:';
```

```
$banned_files_quarantine_method = 'sql:';
$bad_header_quarantine_method = 'sql:';

$virus_quarantine_to = 'virus-quarantine';
$banned_quarantine_to = 'banned-quarantine';
$bad_header_quarantine_to = 'bad-header-quarantine';
$spam_quarantine_to = 'spam-quarantine';
```

Replace values between <> by yours. To know how to configure amavis to allow quarantined messages release, read this *section*.

Note: Amavis configuration allows for separate lookup and storage databases but Modoboa doesn't support it yet.

2.4.2 Connect Modoboa and Amavis

You must tell to Modoboa where it can find the amavis database. Inside `settings.py`, add a new connection to the `DATABASES` variable like this:

```
DATABASES = {
  # Stuff before
  #
  "amavis": {
    "ENGINE" : "<your value>",
    "HOST" : "<your value>",
    "NAME" : "<your value>",
    "USER" : "<your value>",
    "PASSWORD" : "<your value>"
  }
}
```

Replace values between <> with yours.

Cleanup

Storing quarantined messages to a database can quickly become a perfomance killer. Modoboa provides a simple script to periodically purge the quarantine database. To use it, add the following line inside root's crontab:

```
0 0 * * * <modoboa_site>/manage.py qcleanup
#
# Or like this if you use a virtual environment:
# 0 0 * * * <virtualenv path/bin/python> <modoboa_site>/manage.py qcleanup
```

Replace `modoboa_site` with the path of your Modoboa instance.

By default, messages older than 14 days are automatically purged. You can modify this value by changing the `MAX_MESSAGES_AGE` parameter in the online panel.

2.4.3 Release messages

To release messages, first take a look at this page. It explains how to configure amavisd-new to listen somewhere for the AM.PDP protocol. This protocol is used to send requests.

Below is an example of a working configuration:

```
$interface_policy{'SOCK'} = 'AM.PDP-SOCK';
$interface_policy{'9998'} = 'AM.PDP-INET';

$policy_bank{'AM.PDP-SOCK'} = {
  protocol => 'AM.PDP',
  auth_required_release => 0,
};
$policy_bank{'AM.PDP-INET'} = {
  protocol => 'AM.PDP',
  inet_acl => [qw( 127.0.0.1 [::1] )],
};
```

Don't forget to update the `inet_acl` list if you plan to release from the network.

Once amavisd-new is configured, just tell Modoboa where it can find the *release server* by modifying the following parameters in the online panel:

Name	Description	Default value
Amavis connection mode	Mode used to access the PDP server	unix
PDP server address	PDP server address (if inet mode)	localhost
PDP server port	PDP server port (if inet mode) 9998	
PDP server socket	Path to the PDP server socket (if unix mode)	/var/amavis/amavisd.sock

Deferred release

By default, simple users are not allowed to release messages themselves. They are only allowed to send release requests to administrators.

As administrators are not always available or logged into Modoboa, a notification tool is available. It sends reminder e-mails to every administrators or domain administrators. To use it, add the following example line to root's crontab:

```
0 12 * * * <modoboa_site>/manage.py amnotify --baseurl='<modoboa_url>'
#
# Or like this if you use a virtual environment:
# 0 12 * * * <virtualenv path/bin/python> <modoboa_site>/manage.py amnotify --baseurl='<modoboa_url>'
```

You are free to change the frequency.

Note: If you want to let users release their messages alone (not recommended), go to the admin panel.

The following parameters are available to let you customize this feature:

Name	Description	Default value
Check requests interval	Interval between two release requests checks	30
Allow direct release	Allow users to directly release their messages	no
Notifications sender	The e-mail address used to send notitications	notification@modoboa.org

2.4.4 Self-service mode

The *self-service* mode lets users act on quarantined messages without beeing authenticated. They can:

- View messages
- Remove messages
- Release messages (or send release requests)

To access a specific message, they only need the following information:

- Message's unique identifier
- Message's secret identifier

This information is controlled by *amavis*, which is in charge of notifying users when new messages are put into quarantine. Each notification (one per message) must embark a direct link containing the required identifiers.

To activate this feature, go the administration panel and set the **Enable self-service mode** parameter to yes.

The last step is to customize the notification messages amavis sends. The most important is to embark a direct link. Take a look at the README.customize file to learn what you're allowed to do.

Here is a link example:

```
http://<modoboa_url>/quarantine/%i/?rcpt=%R&secret_id=[:secret_id]
```

2.4.5 Manual SpamAssassin learning

It is possible to manually train SpamAssassin using the quarantine's content. By train, we mean:

- Mark message(s) as spam (false negative(s))
- Mark message(s) as non-spam (false positive(s))

This feature is available to all users (from super administrators to simple users) but not enabled by default.

SpamAssassin configuration

For better performance and to enable the per-user level, SpamAssassin must store bayes information into a SQL database.

Create a new database and a new user/password (using your favorite database server) and edit the default configuration file (`/etc/spamassassin/local.cf`) to add the following lines inside:

```
bayes_store_module    Mail::SpamAssassin::BayesStore::<Driver>
bayes_sql_dsn         <DSN>
bayes_sql_username    <db username>
bayes_sql_password    <db password>
```

Replace values between <> by yours. Possible values for `Driver` are `PgSQL` or `MySQL` (non exhaustive list). The syntax for `DSN` depends on the driver you choose. Please consult the official documentation.

Enable the feature through Modoboa

Manual learning is disabled by default. You can activate it through the administration panel (*Modoboa* > *Paremeters* > *Amavis*). There two learning levels:

1. Global: available to administrators only. A single (global) bayes database is shared between everyone.
2. Per domain: available to administrators and domain administrators. Each domain can have a dedicated database.
3. Per user: each user can create its own database to customize the way SpamAssassin will detect spam.

The domain and user levels are not activated by default, dedicated parameters are available through the panel.

Note: Domain and user databases are only created the first time someone calls the learning feature through the quarantine.

Warning: A bayes database needs to reach pre-defined thresholds before it can be used by SpamAssassin. The default values are **200** spams and **200** hams.

You will find other paramaters related to this feature. You won't need to change them most of the time, unless SpamAssassin is hosted on a different machine than Modoboa. (in this case, `spamc` will be used instead of `sa-learn`).

2.5 Graphical statistics

This plugin collects various statistics about emails traffic on your server. It parses a log file to collect information, store it into RRD files (see rrdtool) and then generates graphics in PNG format.

To use it, go to the online parameters panel and adapt the following ones to your environment:

Name	Description	Default value
Path to the log file	Path to log file used to collect statistics	/var/log/mail.log
Directory to store RRD files	Path to directory where RRD files are stored	/tmp/modoboa
Directory to store PNG files	Path to directory where PNG files are stored	<modoboa_site>/media/stats

Make sure the directory that will contain RRD files exists. If not, create it before going further. For example (according to the previous parameters):

```
$ mkdir /tmp/modoboa
```

To finish, you need to collect information periodically in order to feed the RRD files. Add the following line into root's crontab:

```
*/5 * * * * <modoboa_site>/manage.py logparser &> /dev/null
#
# Or like this if you use a virtual environment:
# 0/5 * * * * <virtualenv path/bin/python> <modoboa_site>/manage.py logparser &> /dev/null
```

Replace `<modoboa_site>` with the path of your Modoboa instance.

Graphics will be automatically created after each parsing.

2.6 Radicale frontend

This plugin provides a simple management frontend for Radicale, a complete CalDAV (calendar) and CardDAV (contact) server solution.

Features currently supported are:

- Private calendar creation (available for simple users)
- Rights management for private calendars
- Shared calendar creation by domain (available for domain administrators)

Note: This plugin requires Radicale 0.9 or higher.

2.6.1 Setup

Once Radicale is installed on your server, you must tell Modoboa where it can find the file used to store rules.

Go to the *Modoboa* > *Parameters* > *Radicale* panel and fill the **Radicale rights file path** setting (an absolute path is required).

When the configuration is done, Modoboa will completly handles the file's content. It means every manual modification you could made on this file would be overriden.

To do so, a new cron job must be created. You can use the following example:

```
*/2 * * * * <modoboa_site>/manage.py generate_rights
#
# Or like this if you use a virtual environment:
# */2 * * * * <virtualenv path/bin/python> <modoboa_site>/manage.py generate_rights
```

Note: In some cases, the modifications you make on Modoboa may not be applied to the rights file. In order to force the generation, manually run the `generate_rights` command using the `--force` option.

2.7 Postifx auto-reply messages

This plugin let users define an auto-reply message (*vacation*). It is based on Postfix capabilities.

The user that executes the autoreply script needs to access `settings.py`. You must apply proper permissions on this file. For example, if `settings.py` belongs to `www-data:www-data`, you can add the `vmail` user to the `www-data` group and set the read permission for the group.

To make Postfix use this feature, you need to update your configuration files as follows:

`/etc/postfix/main.cf`:

```
transport_maps = <driver>:/etc/postfix/sql-autoreplies-transport.cf
virtual_alias_maps = <driver>:/etc/postfix/sql-aliases.cf
        <driver>:/etc/postfix/sql-domain-aliases-mailboxes.cf,
        <driver>:/etc/postfix/sql-autoreplies.cf,
        <driver>:/etc/postfix/sql-catchall-aliases.cf
```

Note: The order used to define alias maps is important, please respect it

`/etc/postfix/master.cf`:

```
autoreply unix        -       n       n       -       -       pipe
          flags= user=vmail:<group> argv=python <modoboa_site>/manage.py autoreply $sender $mailbox
```

Replace `<driver>` by the name of the database you use. `<modoboa_site>` is the path of your Modoboa instance.

Then, create the requested map files:

```
$ modoboa-admin.py postfix_maps mapfiles --categories autoreply
```

mapfiles is the directory where the files will be stored. Answer the few questions and you're done.

Note: Auto-reply messages are just sent one time per sender for a pre-defined time period. By default, this period is equal to 1 day (86400s), you can adjust this value by modifying the **Automatic reply timeout** parameter available in the online panel.

2.8 Sieve filters

This plugin let users create server-side message filters, using the sievelib module (which provides Sieve and ManageSieve clients).

Two working modes are available:

- A raw mode: you create filters using the Sieve language directly (advanced users)
- An assisted mode: you create filters using an intuitive form

To use this plugin, your hosting setup must include a *ManageSieve* server and your local delivery agent must understand the *Sieve* language. Don't panic, Dovecot supports both :-) (refer to *Dovecot* to know how to enable those features).

Note: The sieve filters plugin requires that the *Webmail* plugin is activated and configured.

Go the online panel and modify the following parameters in order to communicate with the *ManageSieve* server:

Name	Description	Default value
Server address	Address of your MANAGESIEVE server	127.0.0.1
Server port	Listening port of your MANAGESIEVE server	4190
Connect using STARTTLS	Use the STARTTLS extension	no
Authentication mechanism	Prefered authentication mechanism	auto

2.9 Webmail

Modoboa provides a simple webmail:

- Browse, read and compose messages, attachments are supported
- HTML messages are supported
- CKeditor integration
- Manipulate mailboxes (create, move, remove)
- Quota display

To use it, go to the online panel and modify the following parameters to communicate with your *IMAP* server (under *IMAP settings*):

Name	Description	Default value
Server address	Address of your IMAP server	127.0.0.1
Use a secured connection	Use a secured connection to access IMAP server	no
Server port	Listening port of your IMAP server	143

Do the same to communicate with your SMTP server (under *SMTP settings*):

Name	Description	Default value
Server address	Address of your SMTP server	127.0.0.1
Secured connection mode	Use a secured connection to access SMTP server	None
Server port	Listening port of your SMTP server	25
Authentication required	Server needs authentication	no

Note: The size of each attachment sent with messages is limited. You can change the default value by modifying the **Maximum attachment size** parameter.

2.9.1 Using CKeditor

Modoboa supports CKeditor to compose HTML messages. To use it, first download it from the official website, then extract the tarball:

```
$ cd <modoboa_site_dir>
$ tar xzf /path/to/ckeditor/tarball.tag.gz -C sitestatic/js/
```

And you're done!

Now, each user has the possibility to choose between CKeditor and the raw text editor to compose their messages. (see *User* > *Settings* > *Preferences* > *Webmail*)

CHAPTER 3

Integration with other softwares

3.1 Dovecot and Postfix

3.1.1 Dovecot

Modoboa works better with Dovecot 2.0 so the following documentation is suitable for this combination.

In this section, we assume dovecot's configuration resides in `/etc/dovecot`, all pathes will be relative to this directory.

Mailboxes

First, edit the `conf.d/10-mail.conf` and set the `mail_location` variable:

```
# maildir
mail_location = maildir:~/.maildir
```

Then, edit the `inbox` namespace and add the following lines:

```
inbox = yes

mailbox Drafts {
  auto = subscribe
  special_use = \Drafts
}
mailbox Junk {
  auto = subscribe
  special_use = \Junk
}
mailbox Sent {
  auto = subscribe
  special_use = \Sent
}
mailbox Trash {
  auto = subscribe
  special_use = \Trash
}
```

With dovecot 2.1+, it ensures all the special mailboxes will be automaticaly created for new accounts.

For dovecot 2.0 and older, use the autocreate plugin.

Operations on the file system

Warning: Modoboa needs to access the `dovecot` binary to check its version. To find the binary path, we use the `which` command first and then try known locations (`/usr/sbin/dovecot` and `/usr/local/sbin/dovecot`). If you installed dovecot in a custom location, please tell us where the binary is by using the `DOVECOT_LOOKUP_PATH` setting (see `settings.py`).

Three operation types are considered:

1. Mailbox creation
2. Mailbox renaming
3. Mailbox deletion

The first one is managed by Dovecot. The last two ones may be managed by Modoboa if it can access the file system where the mailboxes are stored (see *General parameters* to activate this feature).

Those operations are treated asynchronously by a cron script. For example, when you rename an e-mail address through the web UI, the associated mailbox on the file system is not modified directly. Instead of that, a *rename* order is created for this mailbox. The mailbox will be considered unavailable until the order is not executed (see *Postfix configuration*).

Edit the crontab of the user who owns the mailboxes on the file system:

```
$ crontab -u <user> -e
```

And add the following line inside:

```
* * * * * python <modoboa_site>/manage.py handle_mailbox_operations
```

Warning: The cron script must be executed by the system user owning the mailboxes.

Warning: The user running the cron script must have access to the `settings.py` file of the modoboa instance.

The result of each order is recorded into Modoboa's log. Go to *Modoboa* > *Logs* to consult them.

Authentication

To make the authentication work, edit the `conf.d/10-auth.conf` and uncomment the following line at the end:

```
#!include auth-system.conf.ext
!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

Then, edit the `conf.d/auth-sql.conf.ext` file and add the following content inside:

```
passdb sql {
  driver = sql
  # Path for SQL configuration file, see example-config/dovecot-sql.conf.ext
  args = /etc/dovecot/dovecot-sql.conf.ext
}
```

```
userdb sql {
  driver = sql
  args = /etc/dovecot/dovecot-sql.conf.ext
}
```

Make sure to activate only one backend (per type) inside your configuration (just comment the other ones).

Edit the `dovecot-sql.conf.ext` and modify the configuration according to your database engine.

MySQL users

```
driver = mysql

connect = host=<mysqld socket> dbname=<database> user=<user> password=<password>

default_pass_scheme = CRYPT

password_query = SELECT email AS user, password FROM core_user WHERE email='%u' and is_active=1

user_query = SELECT '<mailboxes storage directory>/%d/%n' AS home, <uid> as uid, <gid> as gid, concat

iterate_query = SELECT email AS username FROM core_user WHERE email<>''
```

PostgreSQL users

```
driver = pgsql

connect = host=<postgres socket> dbname=<database> user=<user> password=<password>

default_pass_scheme = CRYPT

password_query = SELECT email AS user, password FROM core_user WHERE email='%u' and is_active

user_query = SELECT '<mailboxes storage directory>/%d/%n' AS home, <uid> as uid, <gid> as gid, '*:byt

iterate_query = SELECT email AS username FROM core_user WHERE email<>''
```

SQLite users

```
driver = sqlite

connect = <path to the sqlite db file>

default_pass_scheme = CRYPT

password_query = SELECT email AS user, password FROM core_user WHERE email='%u' and is_active=1

user_query = SELECT '<mailboxes storage directory>/%d/%n' AS home, <uid> as uid, <gid> as gid, ('*:by

iterate_query = SELECT email AS username FROM core_user WHERE email<>''
```

Note: Replace values between <> with yours.

LMTP

Local Mail Transport Protocol is used to let Postfix deliver messages to Dovecot.

First, make sure the protocol is activated by looking at the `protocols` setting (generally inside `dovecot.conf`). It should be similar to the following example:

```
protocols = imap pop3 lmtp
```

Then, open the `conf.d/10-master.conf`, look for `lmtp` service definition and add the following content inside:

```
service lmtp {
  # stuff before
  unix_listener /var/spool/postfix/private/dovecot-lmtp {
    mode = 0600
    user = postfix
    group = postfix
  }
  # stuff after
}
```

We assume here that Postfix is *chrooted* within `/var/spool/postfix`.

Finally, open the `conf.d/20-lmtp.conf` and modify it as follows:

```
protocol lmtp {
  postmaster_address = postmaster@<domain>
  mail_plugins = $mail_plugins quota sieve
}
```

Replace `<domain>` by the appropriate value.

Note: If you don't plan to apply quota or to use filters, just adapt the content of the `mail_plugins` setting.

Quota

Modoboa lets adminstrators define per-domain and/or per-account limits (quota). It also lists the current quota usage of each account. Those features require Dovecot to be configured in a specific way.

Inside `conf.d/10-mail.conf`, add the `quota` plugin to the default activated ones:

```
mail_plugins = quota
```

Inside `conf.d/10-master.conf`, update the `dict` service to set proper permissions:

```
service dict {
  # If dict proxy is used, mail processes should have access to its socket.
  # For example: mode=0660, group=vmail and global mail_access_groups=vmail
  unix_listener dict {
    mode = 0600
    user = <user owning mailboxes>
    #group =
  }
}
```

Inside `conf.d/20-imap.conf`, activate the `imap_quota` plugin:

```
protocol imap {
  # ...

  mail_plugins = $mail_plugins imap_quota

  # ...
}
```

Inside `dovecot.conf`, activate the quota SQL dictionary backend:

```
dict {
  quota = <driver>:/etc/dovecot/dovecot-dict-sql.conf.ext
}
```

Inside `conf.d/90-quota.conf`, activate the *quota dictionary* backend:

```
plugin {
  quota = dict:User quota::proxy::quota
}
```

It will tell Dovecot to keep quota usage in the SQL dictionary.

Finally, edit the `dovecot-dict-sql.conf.ext` file and put the following content inside:

```
connect = host=<db host> dbname=<db name> user=<db user> password=<password>
# SQLite users
# connect = /path/to/the/database.db

map {
  pattern = priv/quota/storage
  table = admin_quota
  username_field = username
  value_field = bytes
}
map {
  pattern = priv/quota/messages
  table = admin_quota
  username_field = username
  value_field = messages
}
```

PostgreSQL users

Database schema update The `admin_quota` table is created by Django but unfortunately it doesn't support `DEFAULT` constraints (it only simulates them when the ORM is used). As PostgreSQL is a bit strict about constraint violations, you must execute the following query manually:

```
db=> ALTER TABLE admin_quota ALTER COLUMN bytes SET DEFAULT 0;
db=> ALTER TABLE admin_quota ALTER COLUMN messages SET DEFAULT 0;
```

Trigger As indicated on Dovecot's wiki, you need a trigger to properly update the quota.

A working copy of this trigger is available on Modoboa's website.

Download this file and copy it on the server running postgres. Then, execute the following commands:

```
$ su - postgres
$ psql [modoboa database] < /path/to/modoboa_postgres_trigger.sql
$ exit
```

Replace `[modoboa database]` by the appropriate value.

Forcing recalculation

For existing installations, *Dovecot* (> 2) offers a command to recalculate the current quota usages. For example, if you want to update all usages, run the following command:

```
$ doveadm quota recalc -A
```

Be carefull, it can take a while to execute.

ManageSieve/Sieve

Modoboa lets users define filtering rules from the web interface. To do so, it requires *ManageSieve* to be activated on your server.

Inside `conf.d/20-managesieve.conf`, make sure the following lines are uncommented:

```
protocols = $protocols sieve

service managesieve-login {
  # ...
}

service managesieve {
  # ...
}

protocol sieve {
  # ...
}
```

Messages filtering using Sieve is done by the LDA.

Inside `conf.d/15-lda.conf`, activate the `sieve` plugin like this:

```
protocol lda {
  # Space separated list of plugins to load (default is global mail_plugins).
  mail_plugins = $mail_plugins sieve
}
```

Finally, configure the `sieve` plugin by editing the `conf.d/90-sieve.conf` file. Put the follwing caontent inside:

```
plugin {
  # Location of the active script. When ManageSieve is used this is actually
  # a symlink pointing to the active script in the sieve storage directory.
  sieve = ~/.dovecot.sieve

  #
  # The path to the directory where the personal Sieve scripts are stored. For
  # ManageSieve this is where the uploaded scripts are stored.
  sieve_dir = ~/sieve
}
```

Restart Dovecot.

3.1.2 Postfix

This section gives an example about building a simple virtual hosting configuration with *Postfix*. Refer to the official documentation for more explanation.

Map files

You first need to create configuration files (or map files) that will be used by Postfix to lookup into Modoboa tables.

To automaticaly generate the requested map files and store them in a directory, run the following command:

```
$ modoboa-admin.py postfix_maps --dbtype <mysql|postgres|sqlite> mapfiles
```

`mapfiles` is the directory where the files will be stored. Answer the few questions and you're done.

Configuration

Use the following configuration in the `/etc/postfix/main.cf` file (this is just one possible configuration):

```
# Stuff before
virtual_transport = lmtp:unix:private/dovecot-lmtp

relay_domains =
virtual_mailbox_domains = <driver>:/etc/postfix/sql-domains.cf
virtual_alias_domains = <driver>:/etc/postfix/sql-domain-aliases.cf
virtual_alias_maps = <driver>:/etc/postfix/sql-aliases.cf,
      <driver>:/etc/postfix/sql-domain-aliases-mailboxes.cf,
      <driver>:/etc/postfix/sql-mailboxes-self-aliases.cf,
      <driver>:/etc/postfix/sql-catchall-aliases.cf

smtpd_recipient_restrictions =
      ...
      check_recipient_access <driver>:/etc/postfix/sql-maintain.cf
      permit_mynetworks
      reject_unverified_recipient
      ...

# Stuff after
```

Replace `<driver>` by the name of the database you use.

Restart Postfix.

3.2 Web servers

3.2.1 Apache2

Note: The following instructions are meant to help you get your site up and running quickly. However it is not possible for the people contributing documentation to Modoboa to test every single combination of web server, wsgi server, distribution, etc. So it is possible that **your** installation of uwsgi or nginx or Apache or what-have-you works differently. Keep this in mind.

mod_wsgi

First, make sure that mod_wsgi is installed on your server.

Create a new virtualhost in your Apache configuration and put the following content inside:

```
<VirtualHost *:80>
  ServerName <your value>
  DocumentRoot <path to your site's dir>

  Alias /media/ <path to your site's dir>/media/
  <Directory <path to your site's dir>/media>
    Order deny,allow
    Allow from all
  </Directory>

  Alias /sitestatic/ <path to your site's dir>/sitestatic/
  <Directory <path to your site's dir>/sitestatic>
    Order deny,allow
    Allow from all
  </Directory>

  WSGIScriptAlias / <path to your site's dir>/wsgi.py
</VirtualHost>
```

This is just one possible configuration.

To use mod_wsgi daemon mode, add the two following directives just under `WSGIScriptAlias`:

```
WSGIDaemonProcess example.com python-path=<path to your site's dir>:<virtualenv path>/lib/python2.7/s
WSGIProcessGroup example.com
```

Replace values between <> with yours. If you don't use a virtualenv, just remove the last part of the `WSGIDaemonProcess` directive.

Note: You will certainly need more configuration in order to launch Apache.

3.2.2 Nginx

Note: The following instructions are meant to help you get your site up and running quickly. However it is not possible for the people contributing documentation to Modoboa to test every single combination of web server, wsgi server, distribution, etc. So it is possible that **your** installation of uwsgi or nginx or Apache or what-have-you works differently. Keep this in mind.

This section covers two different ways of running Modoboa behind Nginx using a WSGI application server. Choose the one you prefer between Green Unicorn or uWSGI.

In both cases, you'll need to download and install nginx.

Green Unicorn

Firstly, Download and install gunicorn. Then, use the following sample gunicorn configuration (create a new file named `gunicorn.conf.py` inside Modoboa's root dir):

```
backlog = 2048
bind = "unix:/var/run/gunicorn/modoboa.sock"
pidfile = "/var/run/gunicorn/modoboa.pid"
daemon = True
debug = False
workers = 2
logfile = "/var/log/gunicorn/modoboa.log"
loglevel = "info"
```

To start gunicorn, execute the following commands:

```
$ cd <modoboa dir>
$ gunicorn -c gunicorn.conf.py <modoboa dir>.wsgi:application
```

Now the nginx part. Just create a new virtual host and use the following configuration:

```
upstream modoboa {
      server      unix:/var/run/gunicorn/modoboa.sock fail_timeout=0;
}

server {
      listen 443 ssl;
      ssl on;
      keepalive_timeout 70;

      server_name <host fqdn>;
      root <modoboa's root dir>;

      access_log  /var/log/nginx/<host fqdn>.access.log;
      error_log /var/log/nginx/<host fqdn>.error.log;

      ssl_certificate     <ssl certificate for your site>;
      ssl_certificate_key <ssl certificate key for your site>;

      location /sitestatic/ {
              autoindex on;
      }

      location /media/ {
              autoindex on;
      }

      location / {
              proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
              proxy_set_header Host $http_host;
              proxy_redirect off;
              proxy_set_header X-Forwarded-Protocol ssl;
              proxy_pass http://modoboa;
      }
}
```

If you do not plan to use SSL then change the listen directive to `listen 80;` and delete each of the following directives:

```
ssl on;
keepalive_timeout 70;
ssl_certificate     <ssl certificate for your site>;
ssl_certificate_key <ssl certificate key for your site>;
proxy_set_header X-Forwarded-Protocol ssl;
```

If you do plan to use SSL, you'll have to generate a certificate and a key. This article contains information about how to do it.

Paste this content to your configuration (replace values between <> with yours), restart nginx and enjoy a really fast application!

uWSGI

The following setup is meant to get you started quickly. You should read the documentation of both nginx and uwsgi to understand how to optimize their configuration for your site.

The Django documentation includes the following warning regarding uwsgi:

> **Warning:** Some distributions, including Debian and Ubuntu, ship an outdated version of uWSGI that does not conform to the WSGI specification. Versions prior to 1.2.6 do not call close on the response object after handling a request. In those cases the request_finished signal isn't sent. This can result in idle connections to database and memcache servers.

Use uwsgi 1.2.6 or newer. If you do not, you *will* run into problems. Modoboa will fail in obscure ways.

To use this setup, first download and install uwsgi.

Here is a sample nginx configuration:

```
server {
    listen 443 ssl;
    ssl on;
    keepalive_timeout 70;

    server_name <host fqdn>;
    root <modoboa's settings dir>;

    ssl_certificate     <ssl certificate for your site>;
    ssl_certificate_key <ssl certificate key for your site>;

    access_log  /var/log/nginx/<host fqdn>.access.log;
    error_log /var/log/nginx/<host fqdn>.error.log;

    location <modoboa's root url>/sitestatic/ {
            autoindex on;
            alias <location of sitestatic on your file system>;
    }

    # Whether or not Modoboa uses a media directory depends on how
    # you configured Modoboa. It does not hurt to have this.
    location <modoboa's root url>/media/ {
            autoindex on;
            alias <location of media on your file system>;
    }

    # This denies access to any file that begins with
    # ".ht". Apache's .htaccess and .htpasswd are such files. A
    # Modoboa installed from scratch would not contain any such
    # files, but you never know what the future holds.
    location ~ /\.ht {
        deny all;
    }
```

```
    location <modoba's root url>/ {
        include uwsgi_params;
        uwsgi_pass <uwsgi port>;
        uwsgi_param UWSGI_SCRIPT <modoboa instance name>.wsgi:application;
        uwsgi_param UWSGI_SCHEME https;
    }
}
```

<modoboa instance name> must be replaced by the value you used when *you deployed your instance*.

If you do not plan to use SSL then change the listen directive to `listen 80;` and delete each of the following directives:

```
ssl on;
keepalive_timeout 70;
ssl_certificate     <ssl certificate for your site>;
ssl_certificate_key <ssl certificate key for your site>;
uwsgi_param UWSGI_SCHEME https;
```

If you do plan to use SSL, you'll have to generate a certificate and a key. This article contains information about how to do it.

Make sure to replace the <...> in the sample configuration with appropriate values. Here are some explanations for the cases that may not be completely self-explanatory:

<modoboa's settings dir> Where Modoboa's `settings.py` resides. This is also where the `sitestatic` and `media` directories reside.

<modoboa's root url> This is the URL which will be the root of your Modoboa site at your domain. For instance, if your Modoboa installation is reachable at at `https://foo/modoboa` then <modoboa's root url> is `/modoboa`. In this case you probably also have to set the `alias` directives to point to where Modoboa's sitestatic and media directories are because otherwise nginx won't be able to find them.

If Modoboa is at the root of your domain, then <modoboa root url> is an empty string and can be deleted from the configuration above. In this case, you probably do not need the `alias` directives.

<uwsgi port> The location where uwsig is listening. It could be a unix domain socket or an address:port combination. Ubuntu configures uwsgi so that the port is:

```
unix:/run/uwsgi/app/<app name>/socket
```

where <app name> is the name of the application.

Your uwsgi configuration should be:

```
[uwsgi]
# Not needed when using uwsgi from pip
# plugins = python
chdir = <modoboa's top dir>
module = <name>.wsgi:application
master = true
harakiri = 60
processes = 4
vhost = true
no-default-app = true
```

The plugins directive should be turned on if you use a uwsgi installation that requires it. If uwsgi was installed from pip, it does not require it. In the configuration above:

<modoboa's top dir> The directory where `manage.py` resides. This directory is the parent of <modoboa's settings dir>

<name> The name that you passed to `modoboa-admin.py deploy` when you created your Modoboa instance.

CHAPTER 4

Extending Modoboa

4.1 Development recipes for Modoboa

You would like to work on Modoboa but you don't know where to start? You're at the right place! Browse this page to learn useful tips.

4.1.1 Prepare a virtual environment

A virtual environment is a good way to setup a development environment on your machine.

Note: `virtualenv` is available on all major distributions, just install it using your favorite packages manager.

To do so, run the following commands:

```
$ virtualenv <path>
$ source <path>/bin/activate
$ git clone https://github.com/tonioo/modoboa.git
$ cd modoboa
$ python setup.py develop
```

The `develop` command creates a symbolic link to your local copy so any modification you make will be automatically available in your environment, no need to copy them.

4.1.2 Deploy an instance for development

Warning: Make sure to *create a database* before running this step.

Now that you have a *running environment*, you're ready to deploy a test instance:

```
$ modoboa-admin.py deploy --dbaction install --devel --dburl <database url> <path>
$ cd <path>
$ python manage.py runserver
```

You're ready to go!

4.1.3 Manage static files

Modoboa uses bower (thanks to django-bower) to manage its CSS and javascript dependencies.

Those dependencies are listed in a file called `dev_settings.py` located inside the `<path_to_local_copy>/modoboa/core` directory.

If you want to add a new dependency, just complete the `BOWER_INSTALLED_APPS` parameter and run the following command:

```
$ python manage.py bower install
```

It will download and store the required files into the `<path_to_local_copy>/modoboa/bower_components` directory.

4.2 Adding a new plugin

4.2.1 Introduction

Modoboa offers a plugin API to expand its capabilities. The current implementation provides the following possibilities:

- Expand navigation by adding entry points to your plugin inside the GUI
- Access and modify administrative objects (domains, mailboxes, etc.)
- Register callback actions for specific events

Plugins are nothing more than Django applications with an extra piece of code that integrates them into Modoboa. Usually, the *__init__.py* file will contain a complete description of the plugin:

- Admin and user parameters
- Observed events
- Custom menu entries

The communication between both applications is provided by *Available events*. Modoboa offers some kind of hooks to let plugin add custom actions.

The following subsections describe plugin architecture and explain how you can create your own plugin.

4.2.2 The required glue

To create a new plugin, just start a new django application like this (into Modoboa's directory):

```
$ python manage.py startapp
```

Then, you need to register this application using the provided API. Just copy/paste the following example into the `__init__.py` file of the future extension:

```
from modoboa.core.extensions import ModoExtension, exts_pool

class MyExtension(ModoExtension):
    name = "myext"
    label = "My Extension"
    version = "0.1"
    description = "A description"
```

```
    url = "myext_root_location" # optional, name is used if not defined

    def init(self):
        """This method is called when the extension is activated.
        """
        pass

    def load(self):
        """This method is called when Modoboa loads available and activated plugins.

        Declare parameters and register events here.
        """
        pass

    def destroy(self):
        """This function is called when a plugin is disabled from the interface.

        Unregister parameters and events here.
        """
        pass

exts_pool.register_extension(MyExtension)
```

Once done, simply add your plugin's module name to the `INSTALLED_APPS` variable located inside `settings.py`. Optionaly, run `python manage.py syncdb` if your plugin provides custom tables and `python manage.py collectstatic` to update static files.

4.2.3 Parameters

A plugin can declare its own parameters. There are two levels available:

- 'Administration' parameters : used to configure the plugin, editable inside the *Admin* > *Settings* > *Parameters* page,
- 'User' parameters : per-user parameters (or preferences), editable inside the *Options* > *Preferences* page.

Playing with parameters

To declare a new administration parameter, use the following function:

```
from modoboa.lib import parameters

parameters.register_admin(name, **kwargs)
```

To declare a new user parameter, use the following function:

```
parameter.register_user(name, **kwargs)
```

Both functions accept extra arguments listed here:

- `type` : parameter's type, possible values are : `int`, `string`, `list`, `list_yesno`,
- `deflt` : default value,
- `help` : help text,
- `values` : list of possible values if `type` is `list`.

To undeclare parameters (for example when a plugin is disabled is disabled from the interface), use the following function:

```
parameters.unregister_app(appname)
```

`appname` corresponds to your plugin's name, ie. the name of the directory containing the source code.

4.2.4 Custom administrative roles

Modoboa uses Django's internal permission system. Administrative roles are nothing more than groups (`Group` instances).

If an extension needs to add new roles, it needs to follow those steps:

1. Create a new `Group` instance. It can be done by providing fixtures or by creating it into the extension `init` function
2. A a new listener for the *GetExtraRoles* event that will return the group's name

4.2.5 Extending admin forms

the forms used to edit objects (account, domain, etc.) through the admin panel are composed of tabs. You can extend those forms (ie. add new tabs) in a pretty easy way by defining events.

Account

To add a new tab to the account edition form, define new listeners (handlers) for the following events:

- *ExtraAccountForm*
- *FillAccountInstances*
- *CheckExtraAccountForm* (optional)

Example:

```
from modoboa.lib import events

@events.observe("ExtraAccountForm")
def extra_account_form(user, account=None):
    return [
        {"id": "tabid", "title": "Title", "cls": MyFormClass}
    ]

@events.observe("FillAccountInstances")
def fill_my_tab(user, account, instances):
    instances["id"] = my_instance
```

Domain

To add a new tab to the domain edition form, define new listeners (handlers) for the following events:

- *ExtraDomainForm*
- *FillDomainInstances*

Example:

```
from modoboa.lib import events

@events.observe("ExtraDomainForm")
def extra_domain_form(user, domain):
    return [
        {"id": "tabid", "title": "Title", "cls": MyFormClass}
    ]

@events.observe("FillDomainInstances")
def fill_my_tab(user, domain, instances):
    instances["id"] = my_instance
```

4.3 Available events

4.3.1 Introduction

Modoboa provides a simple API to interact with events. It understands two kinds of events:

- Those returning a value
- Those returning nothing

Listening to a specific event is achieved as follows:

```
from modoboa.lib import events

def callback(*args):
  pass

events.register('event', callback)
```

You can also use the custom decorator `events.observe`:

```
@events.observe('event')
def callback(*args):
  pass
```

`event` is the event's name you want to listen to, `callback` is the function that will be called each time this event is raised. Each event impose to callbacks a specific prototype to respect. See below for a detailled list.

To stop listening to as specific event, you must use the `unregister` function:

```
events.unregister('event', callback)
```

The parameters are the same than those used with `register`.

To unregister all events declared by a specific extension, use the `unregister_extension` function:

```
events.unregister_extension([name])
```

`name` is the extension's name but it is optional. Leave it empty to let the function guess the name.

Read further to get a complete list and description of all available events.

4.3.2 Supported events

AccountAutoCreated

Raised when a new account is automatically created (example: LDAP synchronization).

Callback prototype:

```
def callback(account): pass
```

- `account` is the newly created account (`User` instance)

AccountCreated

Raised when a new account is created.

Callback prototype:

```
def callback(account): pass
```

- `account` is the newly created account (`User` instance)

AccountDeleted

Raised when an existing account is deleted.

Callback prototype:

```
def callback(account, byuser, **options): pass
```

- `account` is the account that is going to be deleted
- `byuser` is the adminstrator deleting `account`

AccountExported

Raised when an account is exported to CSV.

Callback prototype:

```
def callback(account): pass
```

- `account` is the account being exported

Must return a list of values to include in the export.

AccountImported

Raised when an account is imported from CSV.

Callback prototype:

```
def callback(user, account, row): pass
```

- `user` is the user importing the account
- `account` is the account being imported
- `row` is a list containing what remains from the CSV definition

AccountModified

Raised when an existing account is modified.

Callback prototype:

```
def callback(oldaccount, newaccount): pass
```

- `oldaccount` is the account before it is modified
- `newaccount` is the account after the modification

AdminMenuDisplay

Raised when an admin menu is about to be displayed.

Callback prototype:

```
def callback(target, user): pass
```

The `target` argument indicates which menu is being displayed. Possible values are:

- `admin_menu_box` : corresponds to the menu bar available inside administration pages
- `top_menu` : corresponds to the top black bar

See *UserMenuDisplay* for a description of what callbacks that listen to this event must return.

CanCreate

Raised just before a user tries to create a new object.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance

Return `True` or `False` to indicate if this user can respectively create or not create a new `Domain` object.

CheckDomainName

Raised before the unicity of a domain name is checked. By default, modoboa prevents duplicate names between domains and domain aliases but extensions have the possibility to extend this rule using this event.

Callback prototype:

```
def callback(): pass
```

Must return a list of 2uple, each one containing a model class and an associated label.

CheckExtraAccountForm

When an account is being modified, this event lets extensions check if this account is concerned by a specific form.

Callback prototype:

```
def callback(account, form): pass
```

- `account` is the `User` instance beeing modified
- `form` is a dictionnary (same content as for `ExtraAccountForm`)

Callbacks listening to this event must return a list containing one Boolean.

DomainAliasCreated

Raised when a new domain alias is created.

Callback prototype:

```
def callback(user, domain_alias): pass
```

- `user` is the new domain alias owner (`User` instance)
- `domain_alias` is the new domain alias (`DomainAlias` instance)

DomainAliasDeleted

Raised when an existing domain alias is about to be deleted.

Callback prototype:

```
def callback(domain_alias): pass
```

- `domain_alias` is a `DomainAlias` instance

DomainCreated

Raised when a new domain is created.

Callback prototype:

```
def callback(user, domain): pass
```

- `user` corresponds to the `User` object creating the domain (its owner)
- `domain` is a `Domain` instance

DomainDeleted

Raised when an existing domain is about to be deleted.

Callback prototype:

```
def callback(domain): pass
```

- `domain` is a `Domain` instance

DomainModified

Raised when a domain has been modified.

Callback prototype:

```
def callback(domain): pass
```

- `domain` is the modified `Domain` instance, it contains an extra `oldname` field which contains the old name

DomainOwnershipRemoved

Raised before the ownership of a domain is removed from its original creator.

Callback prototype:

```
def callback(owner, domain): pass
```

- `owner` is the original creator
- `domain` is the `Domain` instance being modified

ExtDisabled

Raised just after an extension has been disabled.

Callback prototype:

```
def callback(extension): pass
```

- `extension` is an `Extension` instance

ExtEnabled

Raised just after an extension has been activated.

Callback prototype:

```
def callback(extension): pass
```

- `extension` is an `Extension` instance

ExtraAccountActions

Raised when the account list is displayed. Let developers define new actions to act on a specific user.

Callback prototype:

```
def callback(account): pass
```

- `account` is the account being listed

ExtraAccountForm

Let extensions add new forms to the account edition form (the one with tabs).

Callback prototype:

```
def callback(user, account): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `account` is the account beeing modified (`User` instance)

Callbacks listening to the event must return a list of dictionnaries, each one must contain at least three keys:

```
{"id" : "<the form's id>",
 "title" : "<the title used to present the form>",
 "cls" : TheFormClassName}
```

ExtraAdminContent

Let extensions add extra content into the admin panel.

Callback prototype:

```
def callback(user, target, currentpage): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `target` is a string indicating the place where the content will be displayed. Possible values are : `leftcol`
- `currentpage` is a string indicating which page (or section) is displayed. Possible values are : `domains`, `identities`

Callbacks listening to this event must return a list of string.

ExtraDomainEntries

Raised to request extra entries to display inside the *domains* listing.

Callback prototype:

```
def callback(user, domfilter, searchquery, **extrafilters): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `domfilter` is a string indicating which domain type the user needs
- `searchquery` is a string containing a search query
- `extrafilters` is a set of keyword arguments that may contain additional filters

Must return a valid `QuerySet`.

ExtraDomainFilters

Raised to request extra filters for the *domains* listing page. For example, the *postfix_relay_domains* extension let users filter entries based on service types.

Callback prototype:

```
def callback(): pass
```

Must return a list of valid filter names (string).

ExtraDomainForm

Let extensions add new forms to the domain edition form (the one with tabs).

Callback prototype:

```
def callback(user, domain): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `domain` is the domain beeing modified (`Domain` instance)

Callbacks listening to the event must return a list of dictionnaries, each one must contain at least three keys:

```
{"id" : "<the form's id>",
 "title" : "<the title used to present the form>",
 "cls" : TheFormClassName}
```

ExtraDomainImportHelp

Raised to request extra help text to display inside the domain import form.

Callback prototype:

```
def callback(): pass
```

Must return a list a string.

ExtraDomainMenuEntries

Raised to request extra entries to include in the left menu of the *domains* listing page.

Callback prototype:

```
def callback(user): pass
```

- `user` is the `User` instance corresponding to the currently logged in user

Must return a list of dictionaries. Each dictionary must contain at least three keys:

```
{"name": "<menu name>",
 "label": "<menu label>",
 "url": "<menu url>"}
```

ExtraFormFields

Raised to request extra fields to include in a django form.

Callback prototype:

```
def callback(form_name, instance=None): pass
```

- `form_name` is a string used to distinguish a specific form
- `instance` is a django model instance related to `form_name`

Must return a list of 2uple, each one containing the following information:

```
('field name', <Django form field instance>)
```

ExtraRelayDomainForm

Let extensions add new forms to the relay domain edition form (the one with tabs).

Callback prototype:

```
def callback(user, rdomain): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `rdomain` is the relay domain being modified (`RelayDomain` instance)

Callbacks listening to the event must return a list of dictionnaries, each one must contain at least three keys:

```
{"id" : "<the form's id>",
 "title" : "<the title used to present the form>",
 "cls" : TheFormClassName}
```

FillAccountInstances

When an account is beeing modified, this event is raised to fill extra forms.

Callback prototype:

```
def callback(user, account, instances): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `account` is the `User` instance beeing modified
- `instances` is a dictionnary where the callback will add information needed to fill a specific form

FillDomainInstances

When a domain is beeing modified, this event is raised to fill extra forms.

Callback prototype:

```
def callback(user, domain, instances): pass
```

- `user` is a `User` instance corresponding to the currently logged in user
- `domain` is the `Domain` instance beeing modified
- `instances` is a dictionnary where the callback will add information needed to fill a specific form

FillRelayDomainInstances

When a relay domain is being modified, this event is raised to fill extra forms.

Callback prototype:

```
def callback(user, rdomain, instances): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `rdomain` is the `RelayDomain` instance being modified
- `instances` is a dictionnary where the callback will add information needed to fill a specific form

GetAnnouncement

Some places in the interface let plugins add their own announcement (ie. message).

Callback prototype:

```
def callback(target): pass
```

- `target` is a string indicating the place where the announcement will appear:
- `loginpage` : corresponds to the login page

Callbacks listening to this event must return a list of string.

GetDomainActions

Raised to request the list of actions available for the *domains* listing entry being displayed.

Callback prototype:

```
def callback(user, rdomain): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `rdomain` is the `RelayDomain` instance being displayed

Must return a list of dictionaries, each dictionary containing at least the following entries:

```
{"name": "<action name>",
 "url": "<action url>",
 "title": "<action title>",
 "img": "<action icon>"}
```

GetDomainModifyLink

Raised to request the modification url of the *domains* listing entry being displayed.

Callback prototype:

```
def callback(domain): pass
```

- `domain` is a model instance (`RelayDomain` for example)

Must return a dictionary containing at least the following entry:

```
{'url': '<modification url>'}
```

GetExtraLimitTemplates

Raised to request extra limit templates. For example, the *postfix_relay_domains* extension define a template to limit the number of relay domains an administrator can create.

Callback prototype:

```
def callback(): pass
```

Must return a list of set. Each set must contain at least three entries:

```
[('<limit_name>', '<limit label>', '<limit help text>')]
```

GetExtraParameters

Raised to request extra parameters for a given parameters form.

Callback prototype:

```
def callback(application, level): pass
```

- `application` is the name of the form's application (ie. admin, amavis, etc.)

- `level` is the form's level: `A` for admin or `U` for user

Must return a dictionary. Each entry must be a valid Django form field.

GetExtraRoles

Let extensions define new administrative roles.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance corresponding to the currently logged in user

Callbacks listening to this event must return a list of 2uple (two strings) which respect the following format: `(value, label)`.

GetStaticContent

Let extensions add static content (ie. CSS or javascript) to default pages. It is pretty useful for functionalities that don't need a template but need javascript stuff.

Callback prototype:

```
def callback(caller, st_type, user): pass
```

- `caller` is name of the application (or the location) responsible for the call
- `st_type` is the expected static content type (`css` or `js`)
- `user` is a `User` instance corresponding to the currently logged in user

Callbacks listening to this event must return a list of string.

ImportObject

Raised to request the function handling an object being imported from CSV.

Callback prototype:

```
def callback(objtype): pass
```

`objtype` is the type of object being imported

Must return a list of function. A valid import function must respect the following prototype:

```
def import_function(user, row, formopts): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `row` is a string containing the object's definition (CSV format)
- `formopts` is a dictionary that may contain options

MailboxAliasCreated

Raised when a new mailbox alias is created.

Callback prototype:

```
def callback(user, mailbox_alias): pass
```

- `user` is the new domain alias owner (`User` instance)
- `mailbox_alias` is the new mailbox alias (`Alias` instance)

MailboxAliasDeleted

Raised when an existing mailbox alias is about to be deleted.

Callback prototype:

```
def callback(mailbox_alias): pass
```

- `mailbox_alias` is an `Alias` instance

MailboxCreated

Raised when a new mailbox is created.

Callback prototype:

```
def callback(user, mailbox): pass
```

- `user` is the new mailbox's owner (`User` instance)
- `mailbox` is the new mailbox (`Mailbox` instance)

MailboxDeleted

Raised when an existing mailbox is about to be deleted.

Callback prototype:

```
def callback(mailbox): pass
```

- `mailbox` is a `Mailbox` instance

MailboxModified

Raised when an existing mailbox is modified.

Callback prototype:

```
def callback(mailbox): pass
```

- `mailbox` is the `Mailbox` modified instance. It contains a `old_full_address` extra field to check if the address was modified.

PasswordChange

Raised just before a *password change* action.

Callback prototype:

```
def callback(user): pass
```

- `user` is a `User` instance

Callbacks listening to this event must return a list containing either `True` or `False`. If at least one `True` is returned, the *password change* will be cancelled (ie. changing the password for this user is disabled).

TopNotifications

Lets extensions subscribe to the global notification service (located inside the top bar).

Callback prototype:

```
def callback(user, include_all): pass
```

- `request` is a `Request` instance
- `include_all` is a boolean indicating if empty notifications must be included into the result or not

Callbacks listening to this event must return a list of dictionary, each dictionary containing at least the following entries:

```
{"id": "<notification entry ID>",
 "url": "<associated URL>",
 "text": "<text to display>"}
```

If your notification needs a counter, you can specify it by adding the two following entries in the dictionary:

{"counter": <associated counter>, "level": "<info|success|warning|error>"}

UserLogin

Raised when a user logs in.

Callback prototype:

```
def callback(request, username, password): pass
```

UserLogout

Raised when a user logs out.

Callback prototype:

```
def callback(request): pass
```

UserMenuDisplay

Raised when a user menu is about to be displayed.

Callback prototype:

```
def callback(target, user): pass
```

The `target` argument indicates which menu is being displayed. Possible values are:

- `options_menu`: corresponds to the top-right user menu
- `uprefs_menu`: corresponds to the menu bar available inside the *User preferences* page
- `top_menu`: corresponds to the top black bar

All the callbacks that listen to this event must return a list of dictionnaries (corresponding to menu entries). Each dictionnary must contain at least the following keys:

```
{"name" : "a_name_without_spaces",
 "label" : _("The menu label"),
 "url" : reverse("your_view")}
```

RelayDomainAliasCreated

Raised when a new relay domain alias is created.

Callback prototype:

```
def callback(user, rdomain_alias): pass
```

- `user` is the new relay domain alias owner (`User` instance)
- `rdomain_alias` is the new relay domain alias (`DomainAlias` instance)

RelayDomainAliasDeleted

Raised when an existing relay domain alias is about to be deleted.

Callback prototype:

```
def callback(rdomain_alias): pass
```

- `rdomain_alias` is a `RelayDomainAlias` instance

RelayDomainCreated

Raised when a new relay domain is created.

Callback prototype:

```
def callback(user, rdomain): pass
```

- `user` corresponds to the `User` object creating the relay domain (its owner)
- `rdomain` is a `RelayDomain` instance

RelayDomainDeleted

Raised when an existing relay domain is about to be deleted.

Callback prototype:

```
def callback(rdomain): pass
```

- `rdomain` is a `RelayDomain` instance

RelayDomainModified

Raised when a relay domain has been modified.

Callback prototype:

```
def callback(rdomain): pass
```

- **`rdomain` is the modified `RelayDomain` instance, it contains an** extra `oldname` field which contains the old name

RoleChanged

Raised when the role of an account is about to be changed.

Callback prototype:

```
def callback(account, role): pass
```

- `account` is the account being modified
- `role` is the new role (string)

SaveExtraFormFields

Raised to save extra fields declared using *ExtraFormFields*.

Callback prototype:

```
def callback(form_name, instance, values): pass
```

- `form_name` is a string used to distinguish a specific form
- `instance` is a django model instance related to `form_name`
- `values` is a dictionary containing the form's values

UserCanSetRole

Raised to check if a user is allowed to set a given role to an account.

Callback prototype:

```
def callback(account, role): pass
```

- `user` is the `User` instance corresponding to the currently logged in user
- `role` is the role `user` tries to set

Must return a list containing `True` or `False` to indicate if this user can is allowed to set `role`.

CHAPTER 5

Additional resource

5.1 Migrating from other software

5.1.1 PostfixAdmin

Modoboa provides a simple script to migrate an existing PostfixAdmin (version 2.3.3+) database to a Modoboa one.

Note: This script is only suitable for a new installation.

First, you must follow the *Installation* step to create a fresh Modoboa database.

Once done, edit the `settings.py` file. First, add a new database connection named `pfxadmin` into the `DATABASES` variable corresponding to your PostfixAdmin setup:

```
DATABASES = {
    "default" : {
        # default connection definition
    },
    "pfxadmin" : {
        "ENGINE" : "<engine>",
        "NAME" : "<database name>",
        "USER" : "<database user>",
        "PASSWORD" : "<user password>",
    }
}
```

This connection should correspond to the one defined in PostfixAdmin's configuration file.

Then, uncomment the line containing `'modoboa.tools.pfxadmin_migrate'` inside the `MODOBOA_APPS` variable and save your changes.

You are now ready to start the migration so run the following commands:

```
$ cd <modoboa_site>
$ python manage.py migrate_from_postfixadmin -s <password scheme>
```

`<password scheme>` must be replaced by the scheme used within postfixadmin (`crypt` most of the time).

Depending on how many domains/mailboxes your existing setup contains, the migration can be long. Just wait for the script's ending.

The procedure is over, edit the `settings.py` file and:

- remove the `pfxadmin` database connection from the `DATABASES` variable

- remove the `'modoboa.tools.pfxadmin_migrate',` from the `MODOBOA_APPS` variable

You should be able to connect to Modoboa using the same credentials you were using to connect to PostfixAdmin.

5.2 Using the virtual machine

5.2.1 Introduction

A virtual machine with a ready-to-use Modoboa setup is available here. It is composed of the following components:

- Debian 6.0 (squeeze)
- Modoboa and its prerequisites
- MySQL
- Postfix
- Dovecot
- nginx and gunicorn

Actually, it is the result you obtain if you follow the official documentation.

The disk image is using the VMDK format and is compressed using bzip2. To decompress it, just run the following command:

```
$ bunzip2 modoboa.vmdk.bz2
```

If you can't use the vmdk format, you can use qemu to convert it to another one. For example:

```
$ qemu-img convert modoboa.vmdk -O qcow2 modoboa.qcow2
```

Then, just use your prefered virtualization software (qemu, kvm, virtualbox, etc.) to start the machine. You'll need to configure at least one bridged network interface if you want to be able to play with Modoboa, ie. your machine must be visible from your network.

The default network interface of the machine (`eth0`) is configured to use the DHCP protocol.

5.2.2 Connect to the machine

The following UNIX users are available if you want to connect to the system:

Login	Password	Description
root	demo	the root user
demo	demo	an unpriviliged user

To connect to Modoboa, first connect to the system and retrieve its current network address like this:

```
$ /sbin/ifconfig eth0
```

Once you know its address, open a web browser and go to this url:

```
http://<ip_address>/admin/
```

You should see the login page. Here are the users available by default:

Login	Password	Capabitilies
admin	password	Default super administrator. Can do anything on the admin but can't access applications
admin@demo.local	admin	Administrator of the domain *demo.local*. Can administrater its domain and access to applications.
user@demo.local	user	Simple user. Can access to applications.