Karaage Programmer Documentation

Release 5.0.19

Brian May

Jul 04, 2019

CONTENTS

This is documentation for Karaage 3.

Date Jul 01, 2019

Version 5.0

Contents:

CHAPTER

ONE

KARAAGE ARCHITECTURE

This document describes the basic Karaage architecture.

1.1 Karaage core

The core Karaage defines the following db models in the *karaage.models* module.

class karaage.models.**LogEntry**
Represents a log entry for any action or comment left on an object.

class karaage.models.**Institute**
Represents an *institute* for a *person*.

class karaage.models.**InstituteDelegate**
Represents an *institute delegate* for an *institute* with extra attributes.

class karaage.models.**Machine**
Represents an individual *machine* or cluster in a *machine category*.

class karaage.models.**Account**
Represents an *account* for a particular *person* on a particular *machine category*.

class karaage.models.**Person**
Represents a *person* who may have one or more *accounts*. A person is global across all *machine categories*.

class karaage.models.**Group**
Represents a *group* of *people*. A group is global accross all *machine categories*.

class karaage.models.**Project**
Represents a *project* for a set of *machine categories*. A project is considered global, although is only active on given machine categories.

1.2 Karaage Applications plugin

Karaage Applications is a plugin that defines additional functionality used for applications. It defines the following db models in the *karaage.plugins.kgapplications.models* module.

class karaage.plugins.kgapplications.models.**Application**
Abstract class that represents any application. Further classes should inherit from this class.

class karaage.plugins.kgapplications.models.**ProjectApplication**
Class that is derived from *Application* for project applications.

class karaage.plugins.kgapplications.models.**Applicant**
An applicant for an application who doesn't already have a *karaage.models.Person* entry.

1.3 Karaage Software plugin

Karaage Software is a plugin that defines additional functionality used for tracking software. It defines the following db models in the *karaage.plugins.kgsoftware.models* module.

class karaage.plugins.kgsoftware.models.**Software**
Represents a particular software package.

class karaage.plugins.kgsoftware.models.**SoftwareCategory**
Represents a category of software, for easy searching.

class karaage.plugins.kgsoftware.models.**SoftwareVersion**
Repesents a specific version of a software package.

class karaage.plugins.kgsoftware.models.**SoftwareLicense**
Represents a license for a software package. A software package may have zero or more licenses. If there are none, the user' won't be able to add the software. There there are more then one, the latest is used by default.

class karaage.plugins.kgsoftware.models.**SoftwareLicenseAgreement**
Represents the fact a person agreed to a particular *SoftwareLicense* at a particular point in time.

class karaage.plugins.kgsoftware.models.**SoftwareApplication**
Class that is derived from *karaage.plugins.kgapplications.models.Application* for applications to access restricted software.

1.4 Karaage Usage plugin

Karaage Usage is a plugin that defines additional functionality used for tracking cluster usage. It may get rewritten in the future, and you should not rely on anything remaining the same.

CHAPTER

TWO

SETTING UP DEVELOPMENT ENVIRONMENT

This section talks about setting up schroot, for releasing new Karaage versions and for testing Karaage with Karaage-test.

It is assumed the system is running Debian Jessie; other build systems may be possible, but will require variations.

These steps only need to be done once for a system.

1. Ensure required packages installed:

```
apt-get install dput-ng dpkg-dev schroot sbuild slapd
apt-get install python-django python3-django
apt-get install python-tldap python3-tldap
apt-get install python-schroot python3-schroot
```

   The above list is probably incomplete. Any omissions will cause errors when running Karaage tests.

2. Stop and disable slapd, it will prevent slapd running in a schroot:

```
service slapd stop
systemctl disable slapd
```

3. Add yourself to the sbuild group. Replace `brian` with your unix user id.

```
adduser brian sbuild
```

   Logout and login again for this to work.

4. Run the following commands:

```
cd tree
git clone https://github.com/brianmay/bampkgbuild.git
sudo ~/tree/bampkgbuild/create_schroot debian sid amd64
sudo ~/tree/bampkgbuild/create_schroot debian sid i386
sudo ~/tree/bampkgbuild/create_schroot debian jessie amd64
sudo ~/tree/bampkgbuild/create_schroot debian jessie i386
```

5. Test schroot is in working order. Changes should disappear after exiting the schroot.

```
schroot --chroot jessie-amd64
schroot --chroot jessie-amd64 --user root
```

6. To make changes to the underlying chroot (you shouldn't have to do this) use:

```
schroot --chroot source:jessie-amd64
```

7. If making releases you will need to have a GPG key to use to distribute the changes and this should have an established web of trust. If not, create a key and get other trusted people to sign it.

CHAPTER

THREE

CONTRIBUTING CODE

github pull requests should be used.

3.1 Getting Started

1. Checkout the latest version of Karaage:

   ```
   git clone https://github.com/Karaage-Cluster/karaage.git
   cd karaage
   ```

   You can test that you've setup the commit-msg script correctly by doing a commit and then looking at the log. You should see a "Change-Id: I[hex]" line show up in your commit message text.

2. Make changes, commit, and submit as github pull request.

3. After the pull request is created, travis will run a complete set of tests against the request to ensure it doesn't break Karaage.

CHAPTER

FOUR

TESTING KARAAGE

This section talks about the steps involved in creating a new official release of Karaage.

It is assumed the system is running Debian Jessie; other build systems may be possible, but will require variations.

4.1 Preparing system

1. Follow the instructions under *Setting up Development Environment*.
2. Install karaage-test.

```
cd tree/karaage
git clone https://github.com/Karaage-Cluster/karaage.git
git clone https://github.com/Karaage-Cluster/karaage-test.git
cd karaage-test
```

3. Edit `dotest.ini`, update pathes to reflect true location.

4.2 Automatically getting test data

1. Run a command like:

```
cd tree/karaage/karaage-test
./getdata -n vpac -s db1.vpac.org -l ldap1.vpac.org
```

   This will create the following large files:

   - `data/vpac/complete.ldif`
   - `data/vpac/complete.sql`
   - `data/vpac/nousage.sql`
   - `data/vpac/onlyusage.sql`

   The `data` directory can be a symlink if required.

2. Create additional LDAP ldif files by hand. Samples below are for openldap.
   - `data/vpac/complete-config.ldif` gets loaded first, so ensure that the LDAP configuration is appropriate for this data.

```
dn: olcDatabase={1}mdb, cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=vpac,dc=org
-
replace: olcRootDN
olcRootDN: cn=admin,dc=vpac,dc=org
-
replace: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange by anonymous auth by dn=
↪"cn=admin,dc=vpac,dc=org" write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to * by dn="cn=admin,dc=vpac,dc=org" write by * read
-

dn: cn=module,cn=config
changetype: add
objectClass: olcModuleList
cn: module
olcModulepath: /usr/lib/ldap
olcModuleload: ppolicy

dn: olcOverlay=ppolicy,olcDatabase={1}mdb,cn=config
changetype: add
objectClass: olcPPolicyConfig
olcPPolicyDefault: cn=default,ou=policies,dc=vpac,dc=org
```

- `data/vpac/settings.py` for telling Karaage the appropriate settings to use to access the LDAP data. Make sure that `_ldap_password` is correct.

```
_ldap_base = 'dc=vpac,dc=org'
_ldap_old_account_base = 'ou=people,%s' % _ldap_base
_ldap_old_group_base = 'ou=groups,%s' % _ldap_base

#_ldap_person_base = 'ou=people,%s' % _ldap_base
#_ldap_person_group_base = 'ou=people_groups,%s' % _ldap_base

_ldap_person_base = None
_ldap_person_group_base = None

_ldap_account_base = 'ou=people,%s' % _ldap_base
_ldap_account_group_base = 'ou=groups,%s' % _ldap_base

#_ldap_person_base = 'ou=people,%s' % _ldap_base
#_ldap_person_group_base = 'ou=people,%s' % _ldap_base
#_ldap_account_base = 'ou=accounts,%s' % _ldap_base
#_ldap_account_group_base = 'ou=accounts,%s' % _ldap_base

_ldap_user = 'cn=admin,%s' % _ldap_base
_ldap_password = 'XXXXX'
```

4.3 Testing Karaage in schroot

Examples for running tests in a schroot:

- Display help information:

```
./dotest --help
```

- Create Karaage from last release available at linuxpenguins.xyz, install with empty data, and create super user.

```
./dotest --distribution jessie --architecture amd64 --shell --create_superuser
```

The `--shell` option means that we open up a shell instead of immediately destroying the schroot when we finished.

- Same as above, but build packages from local git source.

```
./dotest --distribution jessie --architecture amd64 --shell --source=local
```

- Build test Karaage from copy of production data, and run full set of migrations.

```
./dotest --distribution jessie --architecture amd64 -k
data/vpac/settings.py -L data/vpac/complete.ldif  -S
data/vpac/nousage.sql --shell
```

4.4 Testing Karaage in Vagrant

Assumption: using virtualbox, and virtualbox already installed.

1. Load vagrant Jessie image:

```
vagrant box add jessie https://github.com/holms/vagrant-jessie-box/releases/
↪download/Jessie-v0.1/Debian-jessie-amd64-netboot.box
```

See http://www.vagrantbox.es/ for more available VMs.

2. Change to vagrant directory:

```
cd vagrant
```

3. Check the `Vagrantfile` and `bootstrap.sh` config files.
4. Bring VM up:

```
vagrant up
vagrant ssh
sudo -s
```

5. If you want to connect to VM without using vagrant's port forwarding, you may need to alter the `HTTP_HOST` setting in `/etc/karaage3/settings.py`.

CHAPTER

FIVE

CREATING NEW KARAAGE RELEASE

This section talks about the steps involved in creating a new official release of Karaage.

It is assumed the system is running Debian Jessie; other build systems may be possible, but will require variations.

5.1 Preparing system

These steps only need to be done once for a system.

Follow the instructions under *Setting up Development Environment*.

5.2 Make upstream release

This needs to happen first before building the Debian packages. You will need to have write access to the github repository for Karaage and PyPI.

1. Check all changes pushed to github and [travis tests](https://travis-ci.org/Karaage-Cluster/karaage/builds) for the appropriate branch pass.
2. Check `CHANGES.rst` has entry for new release.
3. Create a tag for the new release.

```
git tag --sign x.y.z
```

4. Check version is correct.

```
./setup.py --version
```

5. Push and upload.

```
python ./setup.py sdist upload -s -i 0xGPGKEY
git push
git push --tags
```

5.3 Make Debian release

This needs to happen after the upstream release. You will need to have write access to the github repository for Karaage Debian and somewhere to upload the changes to.

Warning: Current versions of Karaage use git-dpm for the git work flow. This is a good solution and is the solution used by the Debian Python Modules team, Unfortunately it is no longer actively developed and can be quirky at times. As such it is difficult to document all the quirks here.

1. Ensure schroot are up to date:

```
sudo ~/tree/bampkgbuild/update_schroot
```

2. Ensure we are in the karaage-debian tree on the master branch.

```
cd tree/karaage/karaage-debian
```

3. Ensure there are no git uncommited git changes or staged changes.

```
git status
```

4. Ensure all branches are up to date.

```
git pull --ff-only --all
```

5. Copy the new upstream source from the upstream repository.

```
cp ../karaage/dist/karaage-X.Y.Z.tar.gz ../karaage3_X.Y.Z.orig.tar.gz
```

6. Merge the new upstream source.

```
git checkout master
git-dpm import-new-upstream --ptc --rebase-patched ../karaage3_X.Y.Z.orig.tar.gz
```

7. It is possible conflicts may occur in the previous step, when it rebases the Debian changes. If so, fix them and complete the rebase before continuing.
8. Sometimes git-dpm will leave you in the patches directory, you need to be in the Master directory.

```
git-dpm update-patches
```

9. Update `debian/changelog` command.

```
dch -v "X.Y.Z-1" "New upstream version."
git commit debian/changelog -m "Version X.Y.Z-1"
git push --all
```

10. Check Debian package builds.
11. Make changelog for release.

```
dch --release
git commit debian/changelog -m "Release version X.Y.Z"
```

12. Build and upload package.
13. When sure everything is ok, push changes to github:

```
git-dpm tag
git push origin
git push origin --tags
```

14. Merge changes into `karaage4` branch:

```
git checkout karaage4
git merge origin
```

15. When sure everything is ok, push changes to github:

```
git push origin
git checkout master
```

CHAPTER

SIX

PLUGINS

A plugin is a Django app with extra Karaage specific features. It can defined extra settings, extra templates, extra URLS, and extra code.

For the purposes of this document, we assume the plugin is called kgplugin, and defines a Django app with a python module called `kgplugin`. You should change this name.

6.1 Settings

6.1.1 PLUGINS

Default: `[]` (Empty list)

A list of classes that define Karaage plugins.

6.2 Creating a plugin

A plugin needs to provide a urls.py file. This file can be empty if it doesn't provide any urls. It can optionally provide values for `urlpatterns` and `profile_urlpatterns`.

A plugin needs to provide a plugin class that is derived from the *`BasePlugin`* class. It is configured with the *`PLUGINS`* setting.

class karaage.plugins.**BasePlugin**

Base class used for defining Karaage specific settings used to define plugins in Karaaage.

BasePlugin is derived from `django.apps.AppConfig` if Django 1.7 is detected.

Here is an example, taken from the karaage-usage pugin:

```
from karaage.plugins import BasePlugin

 class plugin(BasePlugin):
     name = "karaage.plugins.kgusage"
     xmlrpc_methods = (
         ('karaage.plugins.kgusage.xmlrpc.parse_usage', 'parse_usage',),
     )
     settings = {
         'GRAPH_DEBUG': False,
         'GRAPH_DIR': 'kgusage/',
         'GRAPH_TMP': 'kgusage/',
```

(continues on next page)

(continued from previous page)

```
    }
    depends = ("karaage.plugins.kgsoftware.plugin",)
```

The `name` value is required, all other attributes are optional.

The following attributes can be set:

`BasePlugin.`**`name`**
: The python module for the Django app. This will be added to the `INSTALLED_APPS` Django setting.

  If Django 1.7 is detected, the plugin class is added to `INSTALLED_APPS`, not this value. This setting is used by Django to locate the module.

`BasePlugin.`**`django_apps`**
: A typle list of extra Django apps that are required for this plugin to work correctly. This will be added to the `INSTALLED_APPS` setting.

`BasePlugin.`**`xmlrpc_methods`**
: A tuple list of extra methods to add to the `XMLRPC_METHODS` setting.

`BasePlugin.`**`settings`**
: A dictionary of extra settings, and default values. These are added to the Django settings. If the setting is already defined, the value given here is ignored.

`BasePlugin.`**`depends`**
: A tuple list of plugins this plugin requires to be installed for it to operate correctly.

6.3 Templates

The python module directory, can contain the `templates` directory. This can have custom templates under the `kgplugin` directory. In addition, Karaage will see the following extra files.

- `kgplugin/index_top.html`: contains HTML code to add to the top of the top level Karaage page.
- `kgplugin/index_bottom.html`: contains HTML code to add to the bottom of the top level Karaage page.
- `kgplugin/main_admin.html`: Links to add to the admin menu.
- `kgplugin/main_profile.html`: Links to add to the profile menu.
- `kgplugin/misc.html`: Links to add to the misc menu.
- `emails/email_footer.txt`: Footer to add to every outgoing email.

6.4 URLS

Extra URLS can be defined in the `kgplugin.urls` module, and should be called `urlpatterns` or `profile_urlpatterns` for URLS that should appear under the profile directory.

CHAPTER

SEVEN

GLOSSARY

account A person may have one or more accounts. An account allows a person to access *machines* on a given *machine category*.

administrator A person who has unlimited access to Karaage.

data store A list of external databases that we should link to and update automatically. Supported databases include LDAP, MAM, and Slurm.

global data store A *data store* for storing global data. The global datastores are responsible for writing global data, such as *people* (not *accounts*) to external databases such as LDAP.

group A list of *people*. Usually maps directly to an LDAP Group, but this depends on the data stores used.

institute An entity that represents the organisation or group that every *person* and *project* belongs to.

institute delegate A person who manages an term:*institute*, and can allow new *project's* for the institute.

machine A single cluster or computer which is managed as a distinct unit.

machine category A group of *machines* that share the same authentication systems.

machine category data store A *data store* for storing *machine category* specific data The machine category datastores are specific to a given machine machine, and are responsible for writing machine category specific data, such as *accounts* (not *people*) to external databases such as LDAP.

person A person who has access to the Karaage system. A person could have one/more accounts, be an administrator, be a project leader, be an *institute delegate*. These are optional.

project A list of *people* who share a common goal.

project leader A person who manages a *project*, and can allow new user's to use the project.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

k

INDEX