Django Airavata Documentation

Release 0.9

Levit SCS

April 26, 2016

Contents

Python 2.7	Python 3.2	Python 3.4
Docs	Version	

Airavata ($\pi o \lambda \lambda$ - in Greek "lots of / multi") is a Django 1.8+ library that allows you to hosts multiple dynamic sites running on a single Django instance/db.

I have been using a customized version of dynamicsites for a while now. But with the new features from Django 1.7 and 1.8, there exists another simpler way to achieve the same results. Airavata is an attempt at that *other* way.

This project is licensed under the BSD 2-Clause License

Content:

CHAPTER 1

What is Airavata?

Ever wanted to run several sites using the same codebase and the same database without having to deploy (and maintain) several instances of your project? Than Airavata is for you! The new (Django 1.8+) implementation of `django.contrib.sites` already makes things easier but is still missing (probably by design) some usefull features.

Airavata is a tool providing those features.

1.1 Features

- `SiteAlias`: Airavata adds site aliases (other domain names) to the sites framework to allow having several domain names pointing to the same site (eg: http://john-doe.my-shiny-cms-platform.com and http://john-doe.com)
- `get_current_site`: Airavata leverages the changed behaviour of get_current_site in Django 1.8 and patches it to extend lookups to site aliases
- `setprimarydomain`: Airavata provides a management command to change the first domain name in the database and optionally create an alias for 'localhost'
- `SiteFilteredViewMixin`: Airavata provides a view mixin, to use with Django's generic class based views, which filters results based on the current site
- Unique domains names: Airavata patches the sites framework to ensure that domain names are unique across `Site` and `SiteAlias`.
- `AllowedSites` and `CachedAllowedSites`: Airavata provides 2 helper classes extended from django-allowedsites to use in your settings.py in order to fetch `ALLOWED_HOSTS` list from the database.

1.2 ToDo

- improve test coverage
- improve this doc
- Media file "finder" and upload_path builder
- provide a SiteFilteredModelAdmin

CHAPTER 2

Getting started

2.1 Install

```
pip install airavata
```

2.2 Configure

Add `airavata` and `django.contrib.sites` to your installed apps

```
## settings.py
INSTALLED_APPS = [
    ...
    'django.contrib.sites',
    'airavata',
]
```

Danger: Make sure SITE_ID in not set in `settings.py`

2.3 ALLOWED_HOSTS

Airavata provides two wrapper classes to fetch `ALLOWED_HOSTS` from the database instead of hard-coding them. These two classes are extended from kezabelle's django-allowedsites. Namely they are `airavata.utils.AllowedSites` and `airavata.utils.CachedAllowedSites`. Use either of those in your settings.py

```
## settings.py
from airavata.utils import AllowedSites
ALLOWED_HOSTS = AllowedSites()
```

2.4 Cache invalidation

If you are planning on using CachedAllowedSites, don't forget to register cache invalidation signals in your AppConfig.

```
## apps.py
from django.apps import AppConfig
from airavata.utils import register_signals

class MyAppConfig(AppConfig):
  name = 'my_app'
  verbose_name = "My app"

  def ready(self):
        from django.contrib.sites.models import Site
        from airavata.models import SiteAlias
        for model in [Site, SiteAlias]:
            register_signals(model)

## __init__.py
default_app_config = 'my_app.apps.MyAppConfig'
```

Note: Cache is supposed to be shared among Django instances in order for this process to work. Read more about cache

2.5 Set the domain for your primary site

Once `django.contrib.sites` is added to your `settings.py` Django won't let you access your website unless one of the following is true:

- `SITE_ID` is also set in your `settings.py` (which we don't want to do since this library is for hosting *multiple dynamic* sites)
- `DEBUG` is set to `True` which is ok for dev but not for live servers
- **Django finds a `Site` (or a `SiteAlias` with Airavata) corresponding to the `host` you are requesting**

In order to set the correct domain on the first site in your database, Airavata provides a management command. Simply run

```
python manage.py setprimarydomain
```

CHAPTER 3

Advanced usage

Danger: These advanced usages all require to resort to *local threads* to be able to access the current *requested domain name*. Some people have strong feeelings against local threads variables use in Django. Local threads in themselves (in our humble opinion) are not a security risk but may amplify some other security risks if you use them to store sensitive information.
Airavata uses local threads to store the *requested host name*. If you feel this is sensitive information, make sure you know what you are getting into.

3.1 Extra requirement

As said above `threadlocals` is an extra requirement for the advanced features to work, so go ahead and pip install it

```
pip install django-threadlocals
```

3.2 Common Settings

To use any of the following features, make sure you enable *LocalThreadMiddleware* (put it before `django.middleware.common.CommonMiddleware`.

```
##settings.py
MIDDLEWARE_CLASSES = (
    'airavata.middleware.ThreadLocalMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    ...
)
```

3.2.1 POLLA_SITES_DIR

Every site-specific feature (template, urls, static file) is hosted under a main directory (`BASE_DIR/sites` by default), to override it, provide `POLLA_SITES_DIR` in your `settings.py`

3.2.2 POLLA_REPLACE_DOTS_IN_DOMAINS

This setting is set to `False` by default. For people wanting to use Airavata as a drop-in replacement for `dynamicsites` or who would like to use the **Urls** feature, you shoud set it to `True`.

Setting `POLLA_REPLACE_DOTS_IN_DOMAINS` will change the default behaviour when it comes to looking for site specific features.

e.g: you are trying to load a template named `base.html` for the site `example.com`. having `POLLA_REPLACE_DOTS_IN_DOMAINS` set to `True` django will try looking for it under `sites/example_com/templates/base.html` instead of the default `sites/example.com/templates/base.html`

Note: In any case directory names are lower-case

3.3 TemplateLoader

Airavata provides a TemplateLoader allowing you to load different templates according to the requested host. Specific templates should be placed under the directory configured in `POLLA_SITES_DIR` under a sub-directory corresponding to the main domain name (the domain name in `Site`).

To enable Airavata's template loader, you have to make the following changes to your settings.py:

```
TEMPLATES = [
    {
        ...
        ## Make sure APP_DIRS is set to False
        'APP_DIRS': False,
        'OPTIONS': {
            ...
            ## add a loaders option
            'loaders': (
                'airavata.template_loader.Loader',
                ## Django uses the filesystem loader by default, I tend to try to avoid it
                ## but it's up to you
                # 'django.template.loaders.filesystem.Loader',
                'django.template.loaders.app_directories.Loader'
            ),
        },
    },
]
```

Now you can write `example.com` specific templates in `sites/example.com/templates/` (or `sites/example_com/templates/` depending on your settings)

Note: As with the other loaders, you will have to restart the web server in order for Django to find newly added templates.

3.4 StaticFile Finder

Airavata provides a StaticFile Finder to allow you to host site specific static files (js, css, img, etc).

Site specific should be located under `sites/<main domain name>/static/<file path>` and they will be served under `<STATIC_ROOT>/<main domain name>/<file path>`

To enable Airavata's StaticFile Finder, you have to make the following changes to your settings.py:

```
## Add the STATICFILES_FINDERS directive
STATICFILES_FINDERS = (
    "airavata.staticfiles_finder.SiteFinder",
    ## Django uses the filesystem finder by default, I tend to try to avoid it.
    ## This one is up to you too
    # "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder",
)
```

With this setting, `collectstatic` will collect files in `sites/<domain name>` for every domain listed in `Site`

Warning: Using this method will, by default, expose static files of **every** `Site` to **any** `Site` running under the same Django project. e.g: `css/site.css` sprcific to `site-a.com` wil be available on `http://site-a.com/static/site-a.com/css/site.css` as well as on `http://site-b.com/static/site-a.com/css/site.css` (provided `site-b.com` runs under the same django project). This side-effect might not be desirable and may be prevented using a clever configuration on your web server.

3.5 sitestatic templatetags library

To go hand-in-hand with the StaticFile finder, Airavata provides a replacement for `staticfiles` templatetags library. To use it, simply replace `{% load staticfiles %}` with `{% load sitestatic %}` in your templates.

The `static` templatetag from `sitestatic` will first try to find site-specific static files before defaulting to `staticfiles` behaviour.

```
{% load sitestatic %}
<html>
  <head>
    <link rel="stylesheet" href="{% static 'css/site.css' %}">
  </head>
  ...
```

3.6 UrlPatterns

Note: To use this feature, make sure you set `POLLA_REPLACE_DOTS_IN_DOMAINS` to `True` in your `settings.py` On Python 2 also make sure to include `__init__.py` in both `sites` and it's sub_directory

Airavata allows you to define different urlpatterns for specific domains. To use this feature, update your main `urls.py` to look like this

```
...
from airavata import urls


urlpatterns = urls.UrlPatterns([
```

```
    # Place your patterns here
    ...
    url(...),
])
```

Wrapping the `urlpatterns` list within `UrlPattern` will allow Airavata to check for a `urls.py` file in `sites/<your underscored domain name>/`. If it finds one, it will load it instead of the default provided `urlpatterns`.

If you need common urls feel free to extend the `UrlPattern` wrapper with a list of common urls like this

```
urlpatterns += [
    url(r'^' + settings.STATIC_URL[1:] + r'(?P<path>.*)$', serve, {'document_root': settings.STATIC_R
]
```

CHAPTER 4

Models

Airavata provides a `SiteAlias` model which allows you to create domain aliases. Even though serving the same content from different hostnames is not advisable it can be useful in at least 2 cases:

- local dev with live-ish data: simply create an alias to your existing site
- BitBucket-style case where main domain/address https://bitbucket.org/levit_scs/django-airavata has a convenience alias on http://bb.levit.be/django-airavata

CHAPTER 5

Views

Airavata provides a mixin for filtering views. Originally `SingleObjectMixin` and `MultipleObjectMixin` subclasses but feel free to use it on any View which provides a `get_queryset` method.

`SiteFilteredViewMixin` filters `get_queryset` by the current site. By default `SiteFilteredViewMixin` looks for a `site` field but you can override this with the `site_field` parameter.

```
from django.views.generic import DetailView, ListView
from airavata.views import SiteFilteredViewMixin

from .models import MyModel


class MyModelListView(SiteFilteredViewMixin, ListView):

  model = MyModel


class MyModelDetailView(SiteFilteredViewMixin, DetailView):

  model = MyModel
  site_field = 'base_site'
```

CHAPTER 6

FAQ

6.1 MonkeyPatching is bad. Why do you use it?

Yes MonkeyPatching is bad and if I had control over everything I would gladly extend or subclass whatever is being monkey-patched by Airavata. Unfortunately I don't.

Everything which is being monkey-patched by Airavata can be achieved in other ways by creating a replacement for `django.contrib.sites` which would be extending the existing code-base. And it would be a **much better practice**. But doing so would **break compatibility** with any third party library which already uses `django.contrib.sites`.

If you find a way around that, feel free to contribute.

CHAPTER 7

Contributing

Contributions, feature requests and bug reports are welcome on BitBucket

7.1 Guidelines

- Write tests for your pull requests
- Try to follow main PEP8 guidelines