django-storages-s3upload Documentation

Release 0.1.6

Matt Austin

September 21, 2014

Contents

django-storages-s3upload provides forms and views for direct HTTP post to S3. It is an addition for the Amazon S3 boto storage backend provided by django-storages.

Please be *VERY* careful with security considerations, and check you know exactly what is happening. S3 HTTP POST upload will overwrite existing files if the key matches. You will want to limit uploads to trusted users, and use unique key prefixes. You will want to make sure that any processing of uploaded files is safe.

See also: https://docs.djangoproject.com/en/dev/topics/security/#user-uploaded-content

The goals of this Django app are to:

- Create signed forms for posting files directly in to Amazon S3 buckets.
- Provide views so that it is possible to process files which have been successfully uploaded.
- Provide an extended form class which uses dropzone.js for handling multiple uploads with thumbnails and progress bars.

CHAPTER 1

Quick Start

If you already have an Amazon S3 Boto backend configured as your project's `DEFAULT_FILE_STORAGE`, then it's easy to get an example going by extending `S3UploadFormView` or `DropzoneS3UploadFormView`:

```
class MyFormView(DropzoneS3UploadFormView):

    upload_to = 'test-uploads/'
```

CHAPTER 2

Contents

2.1 Change log

2.1.1 0.1.6

- New methods `get_validate_upload_form_class` and `get_validate_upload_form_kwargs` on S3UploadFormView, to make the validation form easier to extend/override. Thanks Josh Crompton.

2.1.2 0.1.5

- Pass-through (the upload/redirect) and validate CSRF token when processing uploads.
- Prevent Dropzone 'success' styling until processing has also finished.
- A custom `processed_key_generator` can be passed to ValidateS3UploadForm instances.

2.1.3 0.1.4

- `ValidateS3UploadForm` now renames/relocates files during the processing of uploads.
- Support for setting `Cache-Control` header.
- Ensure other http header data is retained when updating key metadata.

2.1.4 0.1.3

- Ensure use of UTC when constructing expiration time for policy.
- Fixed bug when generating the form action url if the storage location was empty.

2.1.5 0.1.2

- Fix for typo in settigs.
- Better file handling.

2.1.6 0.1.1

- Added missing `python-magic` to setup.
- Fixed import typo in sphinx configuration.

2.1.7 0.1.0

- Initial alpha/development release.

2.2 API

Contents:

2.2.1 Forms

S3UploadForm

class `s3upload.forms.`**`S3UploadForm`**(*success_action_redirect=None*, ***kwargs*)

Bases: `s3upload.forms.ContentTypePrefixMixin`, `s3upload.forms.KeyPrefixMixin`, `s3upload.forms.StorageMixin`, `django.forms.forms.Form`

Form for uploading a file directly to an S3 bucket.

`__init__`(*success_action_redirect=None*, ***kwargs*)

`_base64_encode`(*string*)

`add_prefix`(*field_name*)

`base_fields` = OrderedDict([('access_key', <django.forms.fields.CharField object at 0x7f937b558110>), ('acl', <django

`declared_fields` = OrderedDict([('access_key', <django.forms.fields.CharField object at 0x7f937b558110>), ('acl', <d

`expiration_timedelta` = datetime.timedelta(0, 1800)

`field_name_overrides` = {u'access_key': u'AWSAccessKeyId', u'cache_control': u'Cache-Control', u'content_type'

`get_access_key`()

`get_acl`()

Return the acl to be set on the uploaded file.

By default this sets a 'private' acl, and should probably be kept that way. You can set the final acl when the upload is validated/processed.

`get_action`()

`get_cache_control`()

`get_conditions`()

`get_connection`()

`get_expiration_time`(*refresh=False*)

`get_key`()

`get_policy`()

get_secret_key()

get_signature()

get_success_action_redirect()

get_success_action_status_code()

media

success_action_status_code = 204

DropzoneS3UploadForm

class s3upload.forms.**DropzoneS3UploadForm**(*success_action_redirect=None*, ***kwargs*)
Bases: s3upload.forms.S3UploadForm

Form for uploading a file directly to an S3 bucket using dropzone.js.

__init__(*success_action_redirect=None*, ***kwargs*)

class Media
Bases: object

css = {u'all': [u's3upload/css/dropzone.css']}

js = [u's3upload/dropzone.js', u's3upload/dropzone-options.js']

DropzoneS3UploadForm.**base_fields = OrderedDict([('access_key', <django.forms.fields.CharField object at 0x7f**

DropzoneS3UploadForm.**declared_fields = OrderedDict([('access_key', <django.forms.fields.CharField object a**

DropzoneS3UploadForm.**media**

DropzoneS3UploadForm.**success_action_status_code = 201**

ValidateS3UploadForm

class s3upload.forms.**ValidateS3UploadForm**(*process_to=None*, *processed_key_generator=None*, ***kwargs*)
Bases: s3upload.forms.ContentTypePrefixMixin, s3upload.forms.KeyPrefixMixin, s3upload.forms.StorageMixin, django.forms.forms.Form

Form used to validate returned data from S3.

Not for use in templates - we're only processing/validating the provided data.

__init__(*process_to=None*, *processed_key_generator=None*, ***kwargs*)

static _generate_processed_key_name(*process_to*, *upload_name*)
Returns a key name to use after processing based on timestamp and upload key name.

base_fields = OrderedDict([('bucket_name', <django.forms.fields.CharField object at 0x7f937b558950>), ('etag', <dj

clean()

clean_bucket_name()
Validates that the bucket name in the provided data matches the bucket name from the storage backend.

clean_key_name()
Validates that the key in the provided data starts with the required prefix, and that it exists in the bucket.

declared_fields = OrderedDict([('bucket_name', <django.forms.fields.CharField object at 0x7f937b558950>), ('etag

get_processed_acl()
Return the acl to be set on the processed file.

get_processed_key_name()
Return the full path to use for the processed file.

get_processed_path()
Returns the processed file path from the storage backend.

Returns File path from the storage backend.

Return type unicode

get_upload_content_type()
Determine the actual content type of the upload.

get_upload_key()
Get the *Key* from the S3 bucket for the uploaded file.

Returns Key (object) of the uploaded file.

Return type boto.s3.key.Key

get_upload_key_metadata()
Generate metadata dictionary from a bucket key.

get_upload_path()
Returns the uploaded file path from the storage backend.

Returns File path from the storage backend.

Return type unicode

media

process_to = u'processed/'

process_upload(*set_content_type=True*)
Process the uploaded file.

2.2.2 Views

S3UploadFormView

class s3upload.views.**S3UploadFormView**(***kwargs*)
Bases: django.views.generic.edit.FormMixin, django.views.generic.base.TemplateResponseMixi
django.views.generic.base.View

_get_bucket_name()

_get_etag()

_get_key_name()

content_type_prefix = u''

form_class
alias of S3UploadForm

form_invalid(*form*)

form_valid(*form*, **args*, ***kwargs*)

get(**args*, ***kwargs*)

get_content_type_prefix()

get_form_kwargs(**args*, ***kwargs*)

get_process_to()

get_processed_key_generator()

get_storage()

get_success_action_redirect()

get_upload_to()

get_validate_upload_form()
Return an instance of the form to use to validate the upload.

get_validate_upload_form_class()
Return the class of the form to use to validate the upload.

get_validate_upload_form_kwargs()
Return the keyword arguments for instantiating the form for validating the upload.

post(**args*, ***kwargs*)

process_to = None

processed_key_generator = None

set_content_type = True

storage = <django.core.files.storage.DefaultStorage object at 0x7f937b54c990>

template_name = u's3upload/form.html'

upload_to = None

validate_upload()

validate_upload_form_class
alias of `ValidateS3UploadForm`

DropzoneS3UploadFormView

class `s3upload.views.`**DropzoneS3UploadFormView**(***kwargs*)
Bases: `s3upload.views.S3UploadFormView`

form_class
alias of `DropzoneS3UploadForm`

get_success_action_redirect()

template_name = u's3upload/dropzone_form.html'

S