Superdesk Documentation

Release 1.0b1

Sourcefabric

October 20, 2016

Contents

This documentation is technical, by developers for developers. It focuses on Superdesk backend (server part), without much info about Superdesk client.

Contents:

CHAPTER 1

Contributing

1.1 Documentation

Superdesk documentation is written using rst format and generated via Sphinx. It's organized by topic, using autodoc as much as possible to include docstrings from python code.

When working on docs, you can use live preview. In docs folder with virualenv enabled run:

```
$ make livehtml
```

This will build docs and run a server on `localhost:8000`. It will refresh as you modify documentation, but not when you modify python docstrings, in order to see some changes done there you still have to make some changes in rst files.

Once you make a PR and it gets merged, you will see updated docs on superdesk.readthedocs.io.

1.1.1 Updating docs

Documentation should be added/updated together with code changes in a single PR to superdesk-core repository. There can be also PRs with only documentation.

1.1.2 New topic/module

To add a new topic or module docs, you create a file eg. `foo.rst` in `docs` folder and then you have to add it to `index.rst` toctree. This will make it appear in table of contents in both sidebar and on homepage.

1.1.3 Docs conventions

Again - we should use autodocs as much as possible so that documentation is close to code and thus should get updated with it. Thus to document a class or function, use autoclass and autofunction:

```
.. autoclass:: apps.publish.content.common.BasePublishService
    :members:

.. autofunction:: superdesk.publish.transmit
```

If you want to document multiple classes/functions from same module, you should use automodule or module first:

```
.. module:: superdesk.io.feed_parsers

.. autoclass:: ANPAFeedParser
.. autoclass:: IPTC7901FeedParser
```

If you want to document whole module with all its members, you can just use automodule:

```
.. automodule:: superdesk.io.feed_parsers
    :members:
```

This will document all public members from the module which have a docstring.

CHAPTER 2

Architecture

Here there is info about main components in Superdesk and how these interact. To run superdesk we use honcho to define processes for each of components:

```
rest: gunicorn -c gunicorn_config.py wsgi
wamp: python3 -u ws.py
work: celery -A worker worker
beat: celery -A worker beat --pid=
```

2.1 REST API Server

The entry point is Superdesk REST API. This is a python application built on top of eve and flask frameworks. Clients communicate with this api to authenticate, fetch and modify data, upload new content etc.

There is an app factory which you can use to create apps for production/testing:

`superdesk.factory.`**`get_app`**(*config=None*, *media_storage=None*, *config_object=None*)

App factory.

Parameters

- **`config`** – configuration that can override config from `default_settings.py`
- **`media_storage`** – media storage class to use
- **`config_object`** – config object to load (can be module name, module or an object)

Returns a new SuperdeskEve app instance

It can use different wsgi servers, we use Gunicorn.

2.2 Notifications

There is also websockets server where both API server and celery workers can push notifications to clients, which use that information to refresh views or otherwise keep in sync. In the background it's using celery queue and from there it sends everything to clients. There is no communication from client to server, all changes are done via API server.

There is also a factory to create notification server:

`superdesk.ws.`**`create_server`**(*config*)

Create websocket server and run it until it gets Ctrl+C or SIGTERM.

Parameters **`config`** – config dictionary

2.3 Celery Workers

Tasks that involve communication with external services (ingest update, publishing), do some binary files manipulation (image cropping, file metadata extraction) or happen periodically (content expiry) are executed using celery.

It uses same app factory like API server.

2.4 Data Layer

In short - main data storage is mongoDB, content items are also indexed using elastic. This logic is implemented via custom eve data layer, superdesk service layer and data backend.

class `superdesk.datalayer.`**`SuperdeskDataLayer`**(*app*)
Superdesk Data Layer.

Implements eve data layer interface, is used to make eve work with superdesk service layer. It handles app initialization and later it forwards eve calls to respective service.

`init_elastic`(*app*)
Init elastic index.

It will create index and put mapping. It should run only once so locks are in place. Thus mongo must be already setup on an up before running this.

class `superdesk.services.`**`BaseService`**(*datasource=None*, *backend=None*)
Base service for all endpoints, defines the basic implementation for CRUD datalayer functionality.

`find`(*where*, ***kwargs*)
Find items in service collection using mongo query.

Parameters **`where`** (*dict*) –

`is_authorized`(***kwargs*)
Subclass should override if the resource handled by the service has intrinsic privileges.

Parameters **`kwargs`** – should have properties which help in authorizing the request

Returns `False` if unauthorized and True if authorized

`remove_from_search`(*_id*)
Remove item from search by its id.

Parameters **`_id`** – item id

`search`(*source*)
Search using search backend.

Parameters **`source`** – query source param

class `superdesk.eve_backend.`**`EveBackend`**
Superdesk data backend, handles mongodb/elastic data storage.

`create`(*endpoint_name*, *docs*, ***kwargs*)
Insert documents into given collection.

Parameters

- **`endpoint_name`** – api resource name
- **`docs`** – list of docs to be inserted

create_in_mongo(*endpoint_name*, *docs*, ***kwargs*)
Create items in mongo.

Parameters

- **endpoint_name** – resource name
- **docs** – list of docs to create

create_in_search(*endpoint_name*, *docs*, ***kwargs*)
Create items in elastic.

Parameters

- **endpoint_name** – resource name
- **docs** – list of docs

delete(*endpoint_name*, *lookup*)
Delete method to delete by using mongo query syntax.

Parameters

- **endpoint_name** – Name of the endpoint
- **lookup** – User mongo query syntax. example 1. `{'_id':123}`, 2. `{'item_id': {'$in': [123, 234]}}`

Returns Returns the mongo remove command response. {'n': 12, 'ok': 1}

find(*endpoint_name*, *where*, *max_results=0*)
Find items for given endpoint using mongo query in python dict object.

It handles request creation here so no need to do this in service.

:param string endpoint_name :param dict where :param int max_results

find_and_modify(*endpoint_name*, ***kwargs*)
Find and modify in mongo.

Parameters

- **endpoint_name** – resource name
- **kwargs** – kwargs for pymongo `find_and_modify`

find_one(*endpoint_name*, *req*, ***lookup*)
Find single item.

Parameters

- **endpoint_name** – resource name
- **req** – parsed request
- **lookup** – additional filter

get(*endpoint_name*, *req*, *lookup*)
Get list of items.

Parameters

- **endpoint_name** – resource name
- **req** – parsed request
- **lookup** – additional filter

get_from_mongo(*endpoint_name*, *req*, *lookup*)
Get list of items from mongo.

No matter if there is elastic configured, this will use mongo.

Parameters

- **endpoint_name** – resource name
- **req** – parsed request
- **lookup** – additional filter

remove_from_search(*endpoint_name*, *_id*)
Remove document from search backend.

:param endpoint_name :param _id

replace(*endpoint_name*, *id*, *document*, *original*)
Replace an item.

Parameters

- **endpoint_name** – resource name
- **id** – item id
- **document** – next version of item
- **original** – current version of document

replace_in_mongo(*endpoint_name*, *id*, *document*, *original*)
Replace item in mongo.

Parameters

- **endpoint_name** – resource name
- **id** – item id
- **document** – next version of item
- **original** – current version of item

replace_in_search(*endpoint_name*, *id*, *document*, *original*)
Replace item in elastic.

Parameters

- **endpoint_name** – resource name
- **id** – item id
- **document** – next version of item
- **original** – current version of item

search(*endpoint_name*, *source*)
Search for items using search backend

:param string endpoint_name :param dict source

set_default_dates(*doc*)
Helper to populate `_created` and `_updated` timestamps.

system_update(*endpoint_name*, *id*, *updates*, *original*)
Only update what is provided, without affecting etag.

This is useful when you want to make some changes without affecting users.

Parameters

- **endpoint_name** – api resource name
- **id** – document id
- **updates** – changes made to document
- **original** – original document

update(*endpoint_name*, *id*, *updates*, *original*)
Update document with given id.

Parameters

- **endpoint_name** – api resource name
- **id** – document id
- **updates** – changes made to document
- **original** – original document

update_in_mongo(*endpoint_name*, *id*, *updates*, *original*)
Update item in mongo.

Modifies `_updated` timestamp and `_etag`.

Parameters

- **endpoint_name** – resource name
- **id** – item id
- **updates** – updates to item to be saved
- **original** – current version of the item

2.5 Media Storage

By default uploaded/ingested files are stored in mongoDB GridFS.

class superdesk.storage.desk_media_storage.**SuperdeskGridFSMediaStorage**(*app=None*)

media_id(*filename*, *content_type=None*, *version=True*)
Get media id for given filename.

It can be used by async task to first generate id upload file later.

Parameters **filename** – unique file name

put(*content*, *filename=None*, *content_type=None*, *metadata=None*, *resource=None*, ***kwargs*)
Store content in gridfs.

Parameters

- **content** – binary stream
- **filename** – unique filename
- **content_type** – mime type
- **metadata** – file metadata
- **resource** – type of resource

remove_unreferenced_files(*existing_files*)
Get the files from Grid FS and compars agains existing files and deletes the orphans.

url_for_media(*media_id*, *content_type=None*)
Return url for givne media id.

Parameters **media_id** – media id from media_id method

There is also Amazon S3 implementation, which is used when Amazon is configured via settings.

class superdesk.storage.amazon.amazon_media_storage.**AmazonMediaStorage**(*app=None*)

delete_objects(*ids*)
Delete the objects with given list of ids.

exists(*id_or_filename*, *resource=None*)
Test if given name or unique id already exists in storage system.

get(*id_or_filename*, *resource=None*)
Open the file given by name or unique id.

Note that although the returned file is guaranteed to be a File object, it might actually be some subclass. Returns None if no file was found.

get_all_keys()
Return the list of all keys from the bucket.

media_id(*filename*, *content_type=None*, *version=True*)
Get the media_id path for the given filename.

if filename doesn't have an extension one is guessed, and additional *version* option to have automatic version or not to have, or to send a *string* one.

AMAZON_S3_SUBFOLDER configuration is used for easement deploying multiple instance on the same bucket.

put(*content*, *filename=None*, *content_type=None*, *resource=None*, *metadata=None*, *_id=None*, *version=True*)
Save a new file using the storage system, preferably with the name specified.

If there already exists a file with this name name, the storage system may modify the filename as necessary to get a unique name. Depending on the storage system, a unique id or the actual name of the stored file will be returned. The content type argument is used to appropriately identify the file when it is retrieved.

remove_unreferenced_files(*existing_files*)
Get the files from S3 and compare against existing and delete the orphans.

CHAPTER 3

Publishing

3.1 Publish types

There are multiple types of publishing, which correcponds with item life cycle:

- *publish*
- *correct*
- *kill*

For each there is specific resource and service:

class apps.publish.content.publish.**ArchivePublishService**

class apps.publish.content.correct.**CorrectPublishService**

class apps.publish.content.kill.**KillPublishService**

all inheriting from base publish service:

class apps.publish.content.common.**BasePublishService**

These in general handle validation and update item metadata.

3.2 Validation

When publishing starts, it first validates the item based on its content profile definition or in case content profile is missing it will get validators from db. There are different validators for different content types (text, package, picture, etc) and publish type.

Items in packages are also validated if were not published before. Package is considered not valid if any of its item is not valid.

3.2.1 Schema definition

When using content profiles or validators, you specify a schema for each field like:

```
"headline": {
    "type": "string",
    "required": true,
    "maxlength": 140,
```

```
    "minlength": 10
}
```

More info about validation rules in Eve docs.

3.3 Published

When item is valid, it gets some metadata updates:

- `state` is set based on action
- `_current_version` is incremented
- `version_creator` is set to current user

These changes are saved to `archive` collection and `published` collection. On client those items are not visible anymore in monitoring, only in desk output.

3.4 Publish queue

New items from `published` collection are further processed via async task:

`apps.publish.enqueue.`**`enqueue_published`**`()`
Pick new items from `published` collection and enqueu it.

Enqueueing is done via:

class `apps.publish.enqueue.enqueue_service.`**`EnqueueService`**
Creates the corresponding entries in the publish queue for items marked for publishing

`enqueue_item`(*item*)
Creates the corresponding entries in the publish queue for the given item

Return bool True if item is queued else false.

There it finds all subscribers that should recieve the item and if any it will format the item and queue transmission.

3.5 Output Formats

class `superdesk.publish.formatters.`**`NINJSFormatter`**
The schema we use for the ninjs format is an extension of the standard ninjs schema.

Changes from ninjs schema:

- **`uri` was replaced by `guid`: `uri` should be the resource identifier on the web** but since the item was not published yet it can't be determined at this point
- added `priority` field
- added `service` field
- added `slugline` field
- added `keywords` field

Associations dictionary may contain entire items like in ninjs example or just the item `guid` and `type`. In the latest case the items are sent separately before the package item.

Superdesk NINJS Schema in JSON.

class superdesk.publish.formatters.**NITFFormatter**
NITF Formatter for Superdesk

Format items to NITF version *3.6*.

class superdesk.publish.formatters.**NewsML12Formatter**
NewsML 1.2 Formatter

class superdesk.publish.formatters.**NewsMLG2Formatter**
NewsML G2 Formatter

class superdesk.publish.formatters.**EmailFormatter**
Superdesk Email formatter.

- Does not support any media output, it's for text items only.

It uses templates to render items, those can be overriden to customize the output:

- **email_article_subject.txt** email subject
- **email_article_body.txt** email text content
- **email_article_body.html** email html content

It gets article with item data, can be used in templates like:

```
<strong>{{ article.headline }}</strong>
```

3.6 Transmission

Last task is to send items to subscribers, that's handled via another async task:

superdesk.publish.**transmit**()
Transmit items from publish_queue collection.

This task runs every 10s.

3.7 Content Transmitters

class superdesk.publish.transmitters.**HTTPPushService**
HTTP Publish Service.

The HTTP push service publishes items to the resource service via POST request. For media items it first publishes the media files to the assets service.

For text items the publish sequence is like this:

- POST to resource service the text item

For media items the publish sequence is like this:

- Publish media files: for each file from renditions perform the following steps:
  - Verify if the rendition media file exists in the assets service (GET assets/{media_id})
  - If not, upload the rendition media file to the assets service via POST request
- Publish the item

For package items with embedded items config on there is only one publish request to the resource service.

For package items without embedded items the publish sequence is like this:

•Publish package items

•Publish the package item

Publishing assets

The `POST` request to the assets `URL` has the `multipart/data-form` content type and should contain the following fields:

`media_id` URI string identifing the rendition.

`media` `base64` encoded file content. See Eve documentation.

`mime_type` mime type, eg. `image/jpeg`.

`filemeta` metadata extracted from binary. Differs based on binary type, eg. could be exif for pictures.

The response status code is checked - on success it should be `201 Created`.

class `superdesk.publish.transmitters.`**`FTPPublishService`**
FTP Publish Service.

It creates files on configured FTP server.

Parameters

- **`username`** (*`string`*) – auth username
- **`password`** (*`string`*) – auth password
- **`path`** – server path
- **`passive`** – use passive mode (on by default)

class `superdesk.publish.transmitters.`**`FilePublishService`**
Superdesk file transmitter.

It creates files on superdesk server in configured folder.

class `superdesk.publish.transmitters.`**`EmailPublishService`**
Email Transmitter

Works only with email formatter.

Parameters **`recipients`** – email addresses separated by `;`

class `superdesk.publish.transmitters.`**`ODBCPublishService`**
Superdesk ODBC transmitter.

Calls a stored procedure with item data.

Parameters

- **`connection_string`** (*`string`*) –
- **`stored_procedure`** (*`string`*) –

CHAPTER 4

Ingest

With ingest you can import content into Superdesk. It supports multiple formats and ways of delivery.

Ingest is running using celery, an update is triggered every 30s.

superdesk.io.**update_ingest**()
Check ingest providers and trigger an update when appropriate.

It iterates over all providers and check if provider is not closed, and then checks `last_updated` time and schedule to realise if provider should be updated now or later. If now it runs another celery task for each so it can execute multiple updates in parallel.

superdesk.io.**update_provider**()
Fetch items from ingest provider, ingest them into Superdesk and update the provider.

Parameters

- **provider** – Ingest Provider data
- **rule_set** – Translation Rule Set if one is associated with Ingest Provider.
- **routing_scheme** – Routing Scheme if one is associated with Ingest Provider.

Once provider is updated, `last_updated` time is updated and it will ignore that provider for some time according to `schedule`.

4.1 Ingest Provider

Ingest provider specifies configuration for single ingest channel.

class superdesk.io.**IngestProviderResource**(*endpoint_name*, *app*, *service*, *endpoint_schema=None*)
Ingest provider model

Parameters

- **name** – provider name
- **source** – populates item source field
- **feeding_service** – feeding service name
- **feed_parser** – feed parser name
- **content_types** – list of content types of items to ingest from provider
- **allow_remove_ingested** – allow deleting of items from ingest

- **content_expiry** – ttl for ingested items in minutes
- **config** – provider specific config
- **ingested_count** – number of items ingested so far
- **tokens** – auth tokens used by provider
- **is_closed** – provider closed status
- **update_schedule** – update schedule, will run every x hours x minutes x seconds
- **idle_time** – usual idle time for provider, if there is no item after that it will warn
- **last_updated** – last update timestamp
- **rule_set** – rule sets used when ingesting
- **routing_scheme** – routing scheme used when ingesting
- **notifications** – set when notification should be sent for this provider
- **last_closed** – info when and by whom provider was closed last time
- **last_opened** – info when and by whom provider was opened last time
- **critical_errors** – error codes which are considered critical and should close provider

4.2 Feeding Services

Handle transport protocols when ingesting.

class superdesk.io.feeding_services.**EmailFeedingService**
Feeding Service class which can read the article(s) from a configured mail box.

class superdesk.io.feeding_services.**FileFeedingService**
Feeding Service class which can read the configured local file system for article(s).

class superdesk.io.feeding_services.**FTPFeedingService**
Feeding Service class which can read article(s) which exist in a file system and accessible using FTP.

class superdesk.io.feeding_services.**HTTPFeedingService**
Feeding Service class which can read article(s) using HTTP.

class superdesk.io.feeding_services.**RSSFeedingService**
Feeding service for providing feeds received in RSS 2.0 format.

(NOTE: it should also work with other syndicated feeds formats, too, since the underlying parser supports them, but for our needs RSS 2.0 is assumed)

4.2.1 Add new Service

superdesk.io.**register_feeding_service**(*service_name*, *service_class*, *errors*)
Registers the Feeding Service with the application.

Class *superdesk.io.feeding_services.RegisterFeedingService* uses this function to register the feeding service.

Parameters

- **service_name** – unique name to identify the Feeding Service class
- **service_class** – Feeding Service class

- **errors** – list of tuples, where each tuple represents an error that can be raised by a Feeding Service class. Tuple syntax: (error_code, error_message)

Raises AlreadyExistsError if a feeding service with same name already been registered

4.3 Feed Parsers

Parse items from services.

class superdesk.io.feed_parsers.**ANPAFeedParser**
Feed Parser which can parse if the feed is in ANPA 1312 format.

class superdesk.io.feed_parsers.**IPTC7901FeedParser**
Feed Parser which can parse if the feed is in IPTC 7901 format.

class superdesk.io.feed_parsers.**NewsMLOneFeedParser**
Feed Parser which can parse if the feed is in NewsML 1.2 format.

class superdesk.io.feed_parsers.**NewsMLTwoFeedParser**
Feed Parser which can parse if the feed is in NewsML 2 format.

class superdesk.io.feed_parsers.**NITFFeedParser**
Feed Parser which can parse if the feed is in NITF format.

class superdesk.io.feed_parsers.**EMailRFC822FeedParser**
Feed Parser which can parse if the feed is in RFC 822 format.

class superdesk.io.feed_parsers.**WENNFeedParser**
Feed Parser for parsing the XML supplied by WENN

class superdesk.io.feed_parsers.**DPAIPTC7901FeedParser**

class superdesk.io.feed_parsers.**AFPNewsMLOneFeedParser**
AFP specific NewsML parser.

Feed Parser which can parse the AFP feed basicaly it is in NewsML 1.2 format, but the firstcreated and versioncreated times are localised.

class superdesk.io.feed_parsers.**ScoopNewsMLTwoFeedParser**

class superdesk.io.feed_parsers.**AP_ANPAFeedParser**
Feed parser for AP supplied ANPA, maps category codes and maps the prefix on some sluglines to subject codes

class superdesk.io.feed_parsers.**PAFeedParser**
NITF Parser extension for Press Association, it maps the category meta tag to an anpa category

4.3.1 Add new Parser

superdesk.io.**register_feed_parser**(*parser_name*, *parser_class*)
Registers the Feed Parser with the application.

Class *superdesk.io.feed_parsers.RegisterFeedParser* uses this function to register the feed parser.

Parameters

- **parser_name** – unique name to identify the Feed Parser class
- **parser_class** – Feed Parser class

Raises AlreadyExistsError if a feed parser with same name already been registered

CHAPTER 5

Configuration

We use `flask.app.config`, so to use it do:

```
from flask import current_app as app

print(app.config['SERVER_NAME'])
```

Configuration is combination of default settings module and settings module in application repo.

5.1 Default settings

5.1.1 APPLICATION_NAME

Default: `'Superdesk'`

5.1.2 SERVER_NAME

Default: `'localhost:5000'`

5.1.3 CLIENT_URL

Default: `'http://localhost:9000'`

5.1.4 DEFAULT_TIMEZONE

Default: `None`

Superdesk will try to guess the value from system if not set.

5.1.5 FTP_TIMEOUT

Default: `300`

This is used for all ftp operations. Increase if you get ftp timeout errors.

5.1.6 INSTALLED_APPS

Default: `[]`

You can install additional modules by adding their names here.

5.1.7 CONTENT_EXPIRY_MINUTES

Default: `43200` (30 days)

5.1.8 INGEST_EXPIRY_MINUTES

Default: `2880` (2 days)

5.1.9 SPIKE_EXPIRY_MINUTES

Default: `None`

Will use value from `CONTENT_EXPIRY_MINUTES` when empty.

5.1.10 MAX_VALUE_OF_INGEST_SEQUENCE

Default: `9999`

5.1.11 MAX_VALUE_OF_PUBLISH_SEQUENCE

Default: `9999`

5.1.12 DEFAULT_SOURCE_VALUE_FOR_MANUAL_ARTICLES

Default: `'AAP'`

5.1.13 DEFAULT_PRIORITY_VALUE_FOR_MANUAL_ARTICLES

Default: `6`

5.1.14 DEFAULT_URGENCY_VALUE_FOR_MANUAL_ARTICLES

Default: `3`

5.1.15 DEFAULT_GENRE_VALUE_FOR_MANUAL_ARTICLES

Default: `[{'qcode': 'Article', 'name': 'Article (news)'}]`

5.1.16 RESET_PRIORITY_VALUE_FOR_UPDATE_ARTICLES

Default: `False`

5.1.17 NEWSML_PROVIDER_ID

Default: `'sourcefabric.org'`

5.1.18 ORGANIZATION_NAME

Default: `'Australian Associated Press'`

5.1.19 ORGANIZATION_NAME_ABBREVIATION

Default: `'AAP'`

5.1.20 NO_TAKES

Default: `False`

Disable creation of takes packages.

5.1.21 MAX_TRANSMIT_RETRY_ATTEMPT

Default: `10`

Max retries attemps when transmitting an item.

5.1.22 TRANSMIT_RETRY_ATTEMPT_DELAY_MINUTES

Default: `3`

Delay between retry attempts.

5.1.23 MAX_TRANSMIT_QUERY_LIMIT

Default: `500`

Max transmit items to be fetched from mongo at once.

5.1.24 ODBC_PUBLISH

Default: `None`

Determines if the ODBC publishing mechanism will be used, If enabled then pyodbc must be installed along with it's dependencies.

5.2 Mongo connections

There are multiple connections by default, so that it can use different instances for legal archive and production content.

For each there is `_DBNAME` and `_URI` setting, if `_URI` is set it will be used as is, if not it will use `localhost` as server and `_DBNAME` as db.

5.2.1 MONGO_DBNAME

Default: `'superdesk'`

5.2.2 MONGO_URI

Default: `'mongodb://localhost/superdesk'`

5.2.3 LEGAL_ARCHIVE_DBNAME

Default: `'legal_archive'`

5.2.4 LEGAL_ARCHIVE_URI

Default: `'mongodb://localhost/legal_archive'`

5.2.5 ARCHIVED_DBNAME

Default: `'archived'`

5.2.6 ARCHIVED_URI

Default: `mongodb://localhost/archived'`

5.3 Elastic settings

5.3.1 ELASTICSEARCH_URL

Default: `'http://localhost:9200'`

5.3.2 ELASTICSEARCH_INDEX

Default: `'superdesk'`

5.4 Redis settings

5.4.1 REDIS_URL

Default: `'redis://localhost:6379'`

5.5 Cache settings

5.5.1 CACHE_URL

Default: `'redis://localhost:6379'`

New in version 1.3.

5.6 Celery settings

5.6.1 BROKER_URL

Default: `'redis://localhost:6379'`

5.7 Monitoring settings

5.7.1 SENTRY_DSN

Default: `None`

5.8 LDAP settings

Used for *LDAP* based authentication, if not configured it will use mongodb for credentials.

5.8.1 LDAP_SERVER

Default: `''`

5.8.2 LDAP_SERVER_PORT

Default: `389`

5.8.3 LDAP_FQDN

Default: `''`

5.8.4 LDAP_BASE_FILTER

Default: `''`

5.8.5 LDAP_USER_FILTER

Default: `'(&(objectCategory=user)(objectClass=user)(sAMAccountName={}))'`

5.8.6 LDAP_USER_ATTRIBUTES

Default:

```
{
    'givenName': 'first_name',
    'sn': 'last_name',
    'ipPhone': 'phone',
    'mail': 'email',
    'displayName':
    'display_name'
}
```

5.9 Amazon S3 settings

5.9.1 AMAZON_CONTAINER_NAME

Default: `''`

5.9.2 AMAZON_ACCESS_KEY_ID

Default: `''`

5.9.3 AMAZON_SECRET_ACCESS_KEY

Default: `''`

5.9.4 AMAZON_REGION

Default: `'us-east-1'`

5.9.5 AMAZON_SERVE_DIRECT_LINKS

Default: `False`

5.9.6 AMAZON_S3_USE_HTTPS

Default: `False`

5.9.7 AMAZON_SERVER

Default: `'amazonaws.com'`

5.9.8 AMAZON_PROXY_SERVER

Default: `None`

5.10 Security settings

5.10.1 SESSION_EXPIRY_MINUTES

Default: `240`

The number of minutes since the last update of the Mongo auth object after which it will be deleted.

5.10.2 RESET_PASSWORD_TOKEN_TIME_TO_LIVE

Default: `1`

The number of days a token is valid, env `RESET_PASS_TTL`.

5.10.3 ACTIVATE_ACCOUNT_TOKEN_TIME_TO_LIVE

Default: `7`

The number of days an activation token is valid, env `ACTIVATE_TTL`.

5.11 Email settings

5.11.1 MAIL_SERVER

Default: `'localhost'`

5.11.2 MAIL_PORT

Default: `25`

5.11.3 MAIL_USE_TLS

Default: `False`

5.11.4 MAIL_USE_SSL

Default: `False`

5.11.5 MAIL_USERNAME

Default: `''`

5.11.6 MAIL_PASSWORD

Default: `''`

5.11.7 MAIL_DEFAULT_SENDER

Default: `'superdesk@localhost'`

5.11.8 ADMINS

Default: `['']`

CHAPTER 6

Item Schema

Superdesk uses internally item schema that is an extension of ninjs, so on ingest everything is converted to this schema, and on publishing it's converted to different formats.

Basic Schema as defined in `metadata_schema` dict:

6.1 Identifiers

`guid` *string*

Globally unique id. Using external id for ingested content.

`unique_id` *integer*

Internally unique id.

`unique_name` *string*

Internally unique name. By default same as `unique_id`.

`family_id` *string*

Id for all items derived from single item via fetch or copy actions. For ingested items equals to `ingest_id`.

`related_to` *string*

Original item id when doing associate metadata action.

6.2 Content metadata

`headline` *string*

Item headline.

`slugline` *string*

Item slugline.

`byline` *string*

Item byline.

`abstract` *string*

Perex or lead.

`keywords` *list*

List of keywords.

`word_count` *integer*

Word count in `body_html` field.

`priority` *integer*

Item priority.

`urgency` *integer*

Item urgency.

`description_text` *string*

Text description of the item. Used for media types.

`body_html` *string*

Main content field for text items.

`body_text` *string*

Text content of the item. Used for preformatted text.

`body_footer` *string*

Content footer, used for additional information.

`dateline` *dict*

Info about when/where story happens.

`groups` *dict*

Package contents in *NewsML* like format.

`media` *string*

Binary file reference for media type items.

`mimetype` *string*

Binary file mime type.

`poi` *dict*

Point of Interest on a picture.

param x horizontal offset

param y vertical offset

`renditions` *dict*

Renditions of a media type item:

```
    'renditions': {
        'original': {
            'href': '...',
            'width': 1280,
            'height': 800,
        },
        ...
    }
```

`filemeta` *dict*

Extracted file metadata. Mimetype specific.

Deprecated since version 1.2.

`filemeta_json` *string*

JSON encoded filemeta. Avoids storage issues.

New in version 1.2.

`associations` *dict*

Embedded items within body text or predefined relations:

```
'associations': {
    'featured_image': {
        'type': 'picture',
        'guid': 'urn:localhost:123',
        ...
    }
}
```

`alt_text` *string*

Alternate text for picture type items.

`sms_message` *string*

Short summary of an item, can be used for sms/twitter subscribers.

6.3 Item metadata

`type` *string*

Item type. One of `text|composite|picture|audio|video`.

`package_type` *string*

Package type. Either `takes` for takes packages or empty.

`language` *string*

Item language code.

`anpa_take_key` *string*

Take id.

`profile` *string*

Content profile id.

`state` *string*

Workflow state.

`superdesk.metadata.item.`**`content_state`** **= ['draft', 'ingested', 'routed', 'fetched', 'submitted', 'in_progress', '**
item internal states

`revert_state` *string*

Previous item state, is updated on every state change.

`pubstatus` *string*

Publication state.

superdesk.metadata.item.**pub_status** = **['usable', 'withhold', 'canceled']**
item public states

`signal` *string*

Signal information sent to subscriber.

`ednote` *string*

Editorial comment.

`flags` *dict*

Various flags.

`expiry` *datetime*

When this item will expire. After that time it will be archived. It updates on every save/send action.

6.3.1 Copyright information

`usageterms` *string*

`copyrightnotice` *string*

`copyrightholder` *string*

`creditline` *string*

6.4 CVs

These attributes are populated using values from controlled vocabularies.

`anpa_category` *list*

Values from category cv.

`subject` *list*

Values from IPTC subjectcodes plus from custom cvs.

`genre` *list*

Values from genre cv.

`company_codes` *list*

Values from company codes cv.

`place` *list*

Place where story happened.

6.5 System

Set/updated by system mostly.

`_current_version` *integer*

Version of an item, gets incremented on save or publish.

`version` *integer*

Set by client - used to create items with version `0` which are used as drafts.

`firstcreated` *datetime*

When the item was created.

`versioncreated` *datetime*

When current version was created.

`original_creator` *id*

User who created/fetched item.

`version_creator` *id*

User who created current version.

`lock_user` *id*

User who has lock.

`lock_time` *datetime*

When item was locked.

`lock_session` *id*

Session id where item was locked. This way it can detect items locked by same user but in different sessions.

`template` *id*

Template id if item was created using a template.

6.6 Ingest

Set on ingest, might be empty for items created in house.

`ingest_id` *string*

Ingest item id from which item was fetched. For ingested items same as `family_id`.

`ingest_provider` *id*

Ingest provider id.

`source` *string*

Ingest provider source value. Using `DEFAULT_SOURCE_VALUE_FOR_MANUAL_ARTICLES` config for items created locally.

`original_source` *string*

Source value from ingested item.

`ingest_provider_sequence` *integer*

Counter for ingest items.

CHAPTER 7

Caching

New in version 1.3.

There is a Superdesk wrapper for Hermes cache.

Basic usage is:

```
from superdesk.cache import cache

@cache(ttl=30)
def some_func(foo):
     return foo * 5


 class Service():

     @cache(ttl=50)
     def foo(self):
         return
```

7.1 Cache providers

7.1.1 Redis

This one is enabled by default, using `REDIS_URL` for config.

7.1.2 Memcached

In order to use memcached:

- install `python3-memcached` library
- set `SUPERDESK_CACHE_URL` env var to your memcached instance, or set it via `CACHE_URL` in settings.

App Context

It requires `flask.app` for the config, you can still anotate methods/functions before the app is created, but you can not call these methods before there is an app context in place.

Serialization

Cache handles superdesk data encoding/decoding so you will get `datetime` and `ObjectId` instances. However, it doesn't handle yet more complex types - like `Cursor` objects returned after mongo/elastic queries:

```
@cache(ttl=50)
def return_from_db():
    return get_resource_service('foo').get()
```

Will raise an error. What you can do instead:

```
@cache(ttl=50)
def return_from_db():
    return [doc for doc in get_resource_service('foo').get()]
```

a

s

A

B

C

D

E

F

G

W