

Deconcentrator

let deconcentrator do the „hard“ work of checking multiple natural language understanding processing providers.

Table Of Content

1. [Deconcentrator](#)
2. [Table of content](#)
3. [Scope](#)
4. [How To](#)
5. [Authors](#)

scope

be a common endpoint for various nlu providers:

- [RASA](#)

others aren't implemented yet, but implementation should be trivial:

- [Microsoft LUIS](#)
- [Google Cloud NLU](#)
- [IBM Watson NLU](#)

principles

important frameworks/software pieces

- [nginx](#): reverse proxy (static files) and uwsgi gateway - [uWSGI](#): wsgi implementation - [Django REST framework](#): REST interfaces, viewsets, generic serialization

logic etc.

- [Django](#): „The web framework for perfectionists with deadlines.“ - [Celery](#): Distributed task queue for delegating I/O tasks (like doing web requests) - [RabbitMQ](#): async task queue itself - [redis](#): django cache, session cache, celery result backend - [memcached](#): django cache. - [PostgreSQL](#): database backend.

important models

- ``Method``: the abstraction of a function to retrieve an actual NLU processing result. - ``Provider``: the actual provider, which is doing some kind of NLU processing. - ``Strategy``: how to select a ``Provider``

for a specific `Objective` - `Objective`: kind of a task that has to be done. It's the main entry-point, user-supplied. It contains the actual

payload which has to be NLU processed and selects an strategy.

- `Job`: the `Strategy` creates jobs from an `Objective`. Each job then has a specific `Provider` to use for processing. - `Result`: the outcome of asking a `Provider`.

implementation details

- `Objective`, `Job` and `Result` make use of non-abc-dispatching (i. e. dispatching without a common abstract base

```
class). That means:
- they have a common method with equal signature called `execute()` and
are connected to the same `post_save`
  handler.
- once an object of one of these classes is `save`d, the `post_save` hook
will call that common method.
- that method, then, calls the `Strategy` model method for further
handling.
- to avoid infinite recursion, one has to avoid calling `save()` within
the `Strategy` method, instead using the
  `

```

- the `ViewSet`'s design:

1. `Method`s, `Strategy`s, `Job`s and `Result`s can only be read
2. `Provider`s can be CRUD on demand.
3. `Objective`s can be created and retrieved, but not updated or deleted.

how-to

The project contains multiple `docker-compose` files; therefore, basically you only have to `docker-compose up` the project.

- You can use the `management.sh` script to create the required secret files: `./management.sh prepare`; they are

required to start the containers.

- You will probably want to define a `docker-compose.override.yml` to export the nginx port to the public. - You will probably want to run the migrations; basically it's a `./manage.py migrate` call, which can be done via

```
./management.sh exec uwsgi /mnt/deconcentrator/manage.py migrate`.
```

- To access the admin interface, you'll have to create a superuser account first:

```
`./management.sh exec uwsgi /mnt/deconcentrator/manage.py  
createsuperuser`.
```

Authors

Kai Nessig - *Initial work* - [GitHub](#)

See also the list of <https://github.com/beuthbot/deconcentrator/graphs/contributors> who participated in this project.

Nutzungshinweis: Auf dieses vorliegende Schulungs- oder Beratungsdokument (ggf.) erlangt der Mandant vertragsgemäß ein nicht ausschließliches, dauerhaftes, unbeschränktes, unwiderrufliches und nicht übertragbares Nutzungsrecht. Eine hierüber hinausgehende, nicht zuvor durch *datenschutz-maximum* bewilligte Nutzung ist verboten und wird urheberrechtlich verfolgt.