amcat-book

2/7/23

Table of contents

Welcome

Welcome to the amcat manual.

Acknowledgments

This version of the book was built with the following packages and versions:

package	version
quarto	1.2.313
pandoc	2.19.2
R	4.2.2
Python	3.10
amcat4	
amcat4r	0.0.1.9000
amcat4apiclient	0.9
operating system	Linux 6.1.3-arch1-1

1 Why amcat

Tip

Say, there has been a media frenzy recently about a political scandal. You know that it started with a revelation by an anonymous poster on Reddit, spread through social media, and was investigated by respectable media outlets, which created public pressure until the topic was discussed in your country's parliament and a minister or two had to step down. You quickly realise that the case is a treasure trove for investigating your political system and testing several media theories in the process.

You spin up a new instance of the amcat suite on your research server or a new cloud instance and start to collect data through APIs of social media websites, media monitoring sites and scrapers and store everything in the amcat database. You share your data with collaborators, using fine-grained data access control since some of the scraped content is copyrighted. A new research assistant with some knowledge about the case but no technical training starts digging through the data using amcat's user interface and publicly available dashboard to search for potentially relevant terms and queries. One of your collaborators builds on this by producing time-series models and plots in `R`.

You decide to dig deeper into the content and start training coders and deploy coding tasks through amcat's annotation tool. Coders get through the task quickly as they can access the interface on their phones and code on their bus commutes or whenever else they feel like it. You use your amcat server to preprocess the data and train and validate a heap of advanced machine learning models that complete the coding task of all documents. Your analysis reveals new mechanisms and confirms some of the theories you worked with.

After writing up a paper, you submit your results to a journal. The editor asks for replication code and data, so you simply share your R or Python scripts and grant temporary access to your amcat server for a researcher tasked with replicating your results.

After publication, a newspaper picks up your results, leading some interested citizens to play with your dashboard. Even though users do not have access to the full text for copyright reasons, they can query different combinations of keywords, which makes your research transparent for a wider audience. Since the annotations and the preprocessed texts are also available, someone finds they get even better validation scores using a newly created algorithm.

1.1 What is amcat?

The `amcat`-suite consists of several packages for text analysis. It has two main goals: to **standardize** text analysis tasks with **easy to use** software, while offering quality-of-life features for power users. It consists of several different software packages, which are usually used together:

- `amcat4` takes care of **document storage**, provides fine-grained **data access control** (e.g., to restrict access to the parts of a dataset which are copyrighted, proprietary or (privacy-)sensitive) and supports fast queries using Elasticsearch
- `middlecat` provides **authentication methods** to support the fine-grained **data access control** built into `amcat4` (e.g., to make datasets available for which data owners have restricted full text access)
- `amcat4client` offers a user interface, which makes it easy to **query documents from `amcat4` via a web interface**, share data with collaborators or the public and present your corpora to stakeholders, the community or the public
- `amcat4apiclient` provides bindings to **manage and query corpora from the `Python` programming language** via `amcat4`'s REST API
- `amcat4r` provides bindings to **manage and query corpora from the `R` programming language** via `amcat4`'s REST API

These core packages can be extended by powerful addons which provide additional features:

- `annotinder` which let's you **manually annotate documents** with an appealing web interface (which also looks great on mobile!) and the possibility to deploy it to the web. There is also an R client!
- `nlpipe` can be used for **advanced document pre-processing and machine learning** tasks. You can't share your full text? How about letting `nlpipe` apply word embeddings on your corpus and share the embedding instead of the full text!

You can use many of these packages individually or you create a full setup, which would look something like this:

It might seem like this is overly complicated, given that all of the features are also available in other software packages, some of which you will also be familiar with already. However, the main reason that functions are split between different software modules is to make development easier and more transparent. **If you are only interested in amcat's capabilities, you can use it on our servers or conveniently install everything at once through docker.**

If you want to learn more about the project, have a look at the about chapter.

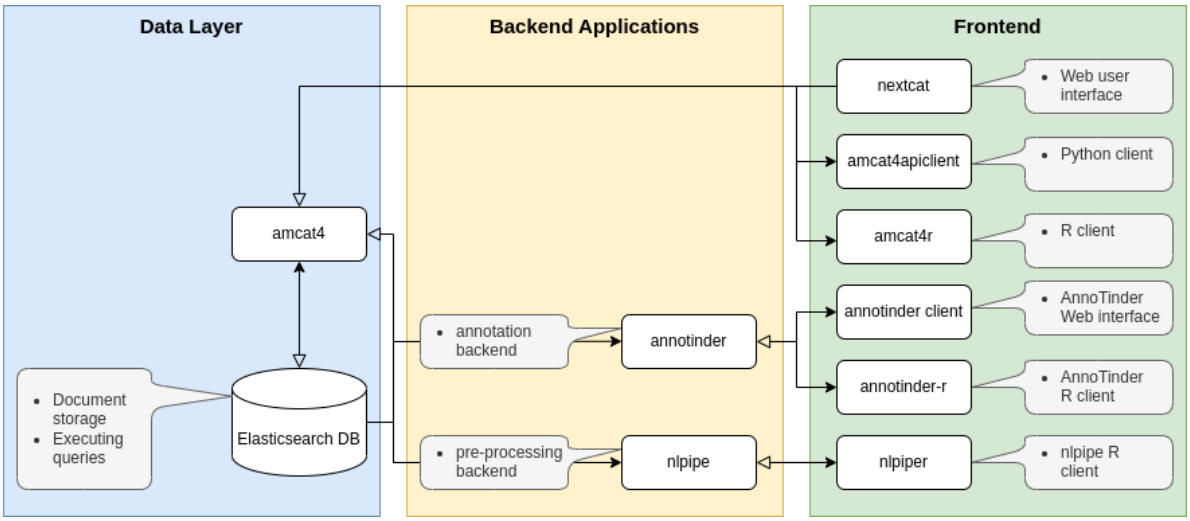


Figure 1.1: amcat-framework

2 Getting started

> **Important**
>
> **This is work in progress.** Passages highlighted in red are likely to change soon.

In this chapter, we show you how to set up the data layer and front-end of the amcat suite.

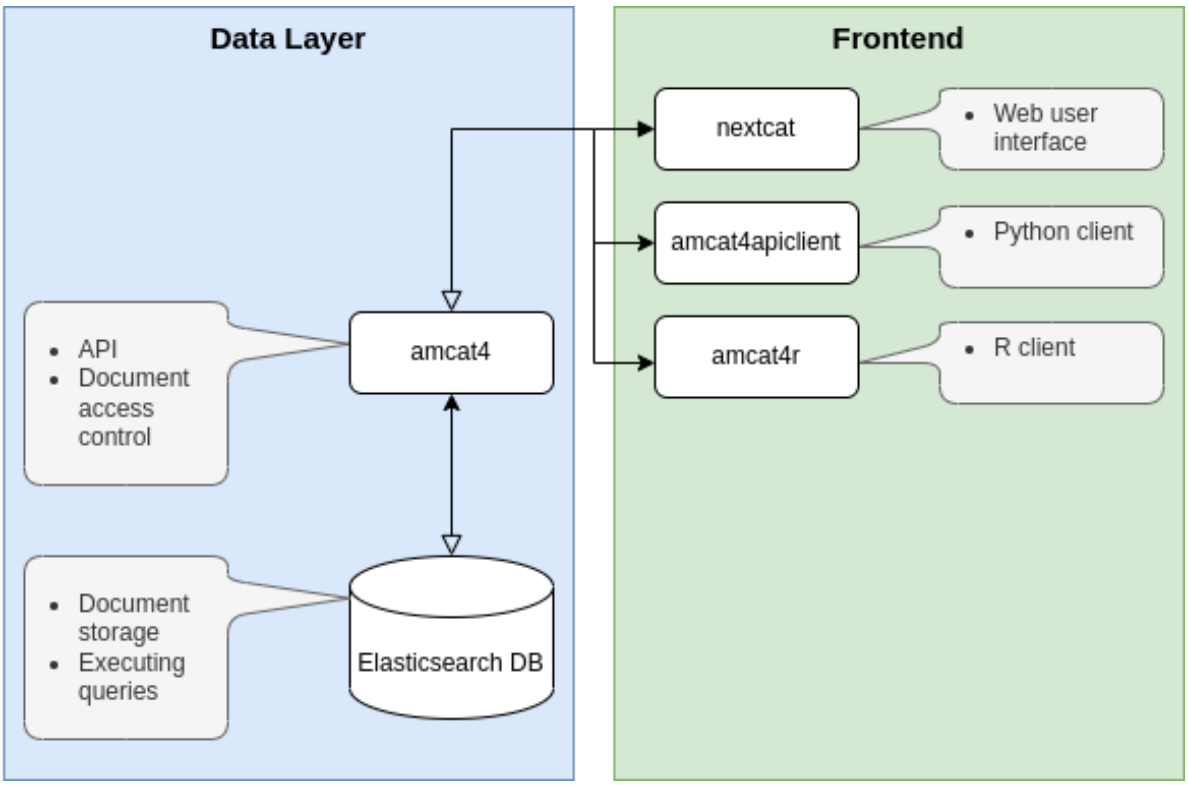


Figure 2.1: amcat instance after following this chapter

In the Data Layer section, it is sufficient if you choose one of the sub-sections to follow. We explain how you can run amcat

- on our servers, which we recommend for testing purposes;

- through a Docker image, which we recommend for most people who want to conduct a research project and/or share data online;
- or install amcat directly on your system, which we only recommend for advanced users, who want a customised setup.

In the Frontend section, it makes sense to cover the amcat4client, which provides a react web interface to query data. Then you can select to either install the R or Python client.

2.1 Data Layer

2.1.1 Run on our servers

! Important

3 Coming soon...

3.0.1 Setup through Docker

Tip

4 Why do we use Docker for installation?

Functionally, Docker containers are a cross-platform installation format that makes it trivially easy to install software packages on Linux, MacOS and Windows without needing users to deal with dependency issues or installation quirks on different systems.[a] A side effect is that we can easily develop amcat for all operating systems at once and you can be sure that we do not fall behind on developing amcat for your operating system of choice.

[a] Technically, it is a little more complicated, as Docker containers have many similarities to virtual machines. However, for most users that technical background is not really important. If you want to learn more, have a look here.

If you have never used Docker before, the first step is to install the infrastructure on your system. Head over to the Docker website to get Docker Desktop or the Docker Engine and Docker Compose to use Docker from the command line. We do not really need Docker Desktop, but it comes with both the Docker Engine and Docker Compose, which makes installation easier if you do not use a package manager (which many Windows and MacOS users do not).

To install the amcat data layer, you should use our Docker Compose file. You can get it from here (save the file as `docker-compose.yml`).

We tried to write the file to contain sensible defaults for testing it locally. If you plan to work with amcat for a research project and/or plan to update the images in the future, you should have a look at the customization option. Oterwise you can continue.

Customization

The current default docker-compose.yml looks like this:

```yaml
version: "3.8"
services:
  web_server:
    image: ccsamsterdam/ngincat:4.0.4
    build: ./ngincat
    container_name: ngincat
    restart: unless-stopped
    networks:
      - amcat-net
    environment:
      - amcat4_client=http://amcat4client:3000
      - amcat4_host=http://amcat4:5000/
    ports:
      - 80:80 # [local port]:[container port]
    depends_on:
      - "web_client"
      - "api"
  web_client:
    image: ccsamsterdam/amcat4client:4.0.4
    build: ./amcat4client
    container_name: amcat4client
    restart: unless-stopped
    networks:
      - amcat-net
    environment:
      # this can be changed later, it is just the suggested default
      - amcat4_host=http://localhost/amcat
    depends_on:
      - "api"
  api:
    image: ccsamsterdam/amcat4:4.0.4
    build: ./amcat4
    container_name: amcat4
    restart: unless-stopped
    networks:
      - amcat-net
    environment:
      # note that these take precedence over values set in `amcat4 config``
      - amcat4_elastic_host=elastic7:9200
      - amcat4_host=http://localhost/amcat
    depends_on:
      - "db"
  db:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.9
    container_name: elastic7
    restart: unless-stopped
    # for security reasons, the database is only exposed to the other containers in the
    # If you want to be able to access it locally, uncomment the following two lines
    # ports:
    # - 9200:9200
    networks:
      - amcat-net
```

Setting the containers up from `R` or `Python` uses the same settings. You can leave most lines as they are, but we want to draw your attention to a couple of settings you might want to change:

- In lines 13, 14, we set the port to `80` on the host machine. This means you will be able to access the amcat client without specifying a port (80 is the default port your browser uses to access an address). If the port is already in use, the container will crash. In this case, change `80`to a different port and access amcat through, for example, `localhost:5000`.
- In lines 54, 55, 56, we configured Elasticsearch to form a single-node cluster and use a maximum of 4GB memory.
- In lines 61, 62 we suggested a setting so Elasticsearch will store your data on a volume on the host machine. **If you do not use this, your data will be destroyed when you remove the Elasticsearch container!**. We recommend this to make it easier to back up your database and reuse it with a different installation of Elasticsearch (e.g., after an update) in the future. However, the container will not run if it does not have proper access to this folder. See the comment to solve this. Note that the suggested local path is just an example. Learn more about this in the chapter on backups and updates.

To download and start the amcat containers (Elasticsearch, amcat4, and amcat4client) use one of the approaches below:

4.0.0.1 Command Line

Start a terminal and navigate to the directory where you downloaded the docker-compose.yml file[1]:

```
docker-compose up --pull="missing" -d
```

Check if the containers are running with:

```
docker ps
#> CONTAINER ID   IMAGE                                                 COMMAND
#> 0628bc852c79   ccsamsterdam/amcat4client:0.0.1                       "nginx -g 'daemon
#> 8134bcd1cbe8   ccsamsterdam/amcat4:0.0.1                             "./wait-for-it.sh
#> 2d59e128e748   docker.elastic.co/elasticsearch/elasticsearch:7.17.7   "/bin/tini -- /us
```

[1]The yaml file is written for Docker Compose V2. If you are having trouble, check your version with `docker-compose --version`. Get the newest version as described here.

4.0.0.2 R

```
# install the required packages first:
# remotes::install_github("JBGruber/dockr")
# remotes::install_github("ccs-amsterdam/amcat4r")
amcat4r::run_amcat_docker()
```

This pulls downloads the default `docker-compose.yml` file from the same link as above. If you want to change the file first, just supply the path to the function afterwards:

```
amcat4r::run_amcat_docker("docker-compose.yml")
```

Check if the containers are running with:

```
amcat4r::docker_lc()
#> # A tibble: 3 × 5
#>   name          image                                                status        id
#>   <chr>         <chr>                                                <chr>         <chr
#> 1 /amcat4client ccsamsterdam/amcat4client:0.0.1                      Up 58 seconds 0628
#> 2 /amcat4       ccsamsterdam/amcat4:0.0.1                            Up 59 seconds 8134
#> 3 /elastic7     docker.elastic.co/elasticsearch/elasticsearch:7.17.7 Up 58 seconds 2d59
```

4.0.0.3 Python

```
# coming soon
```

If you are using Docker Desktop (which we recommend only for local installations on e.g., Windows), you can also monitor the containers there:

It might take a couple of seconds for Elasticsearch to start up. Then you can navigate to http://localhost/ in your browser to access the amcat client.

Before you access your newly created amcat suite, you should first check the settings and change to your needs. You can do this with `amcat4 config`:

4.0.0.4 Command Line

```
docker exec -it amcat4 amcat4 config
```

If you choose anything but `no_auth` for authentification options, you should also add a global admin user via:

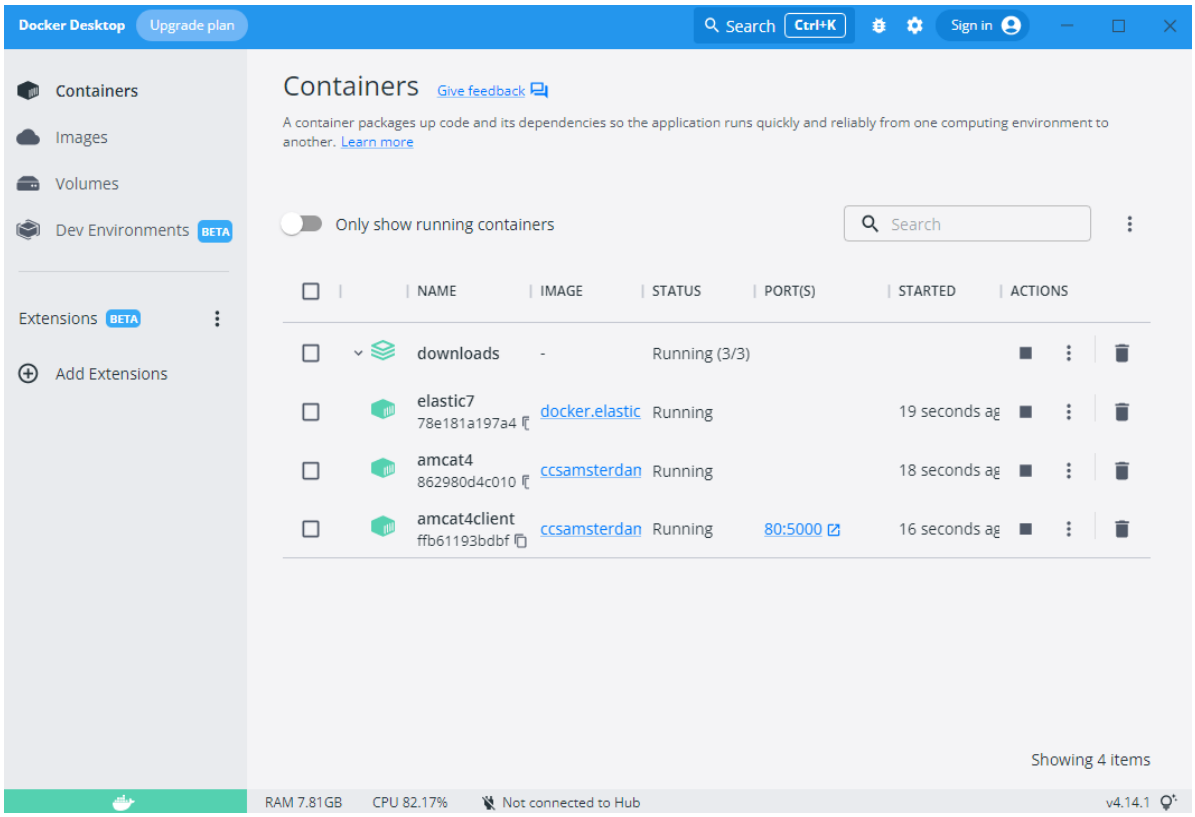


Figure 4.1: All containers are running

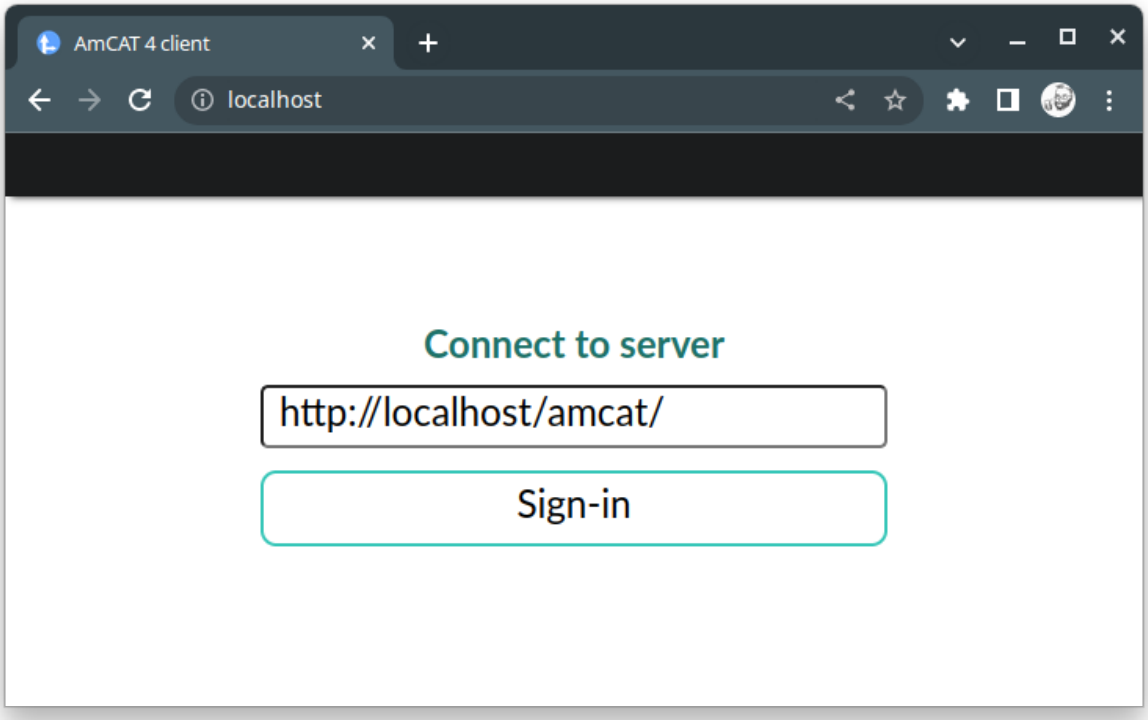


Figure 4.2: First view of the amcat react app in your browser

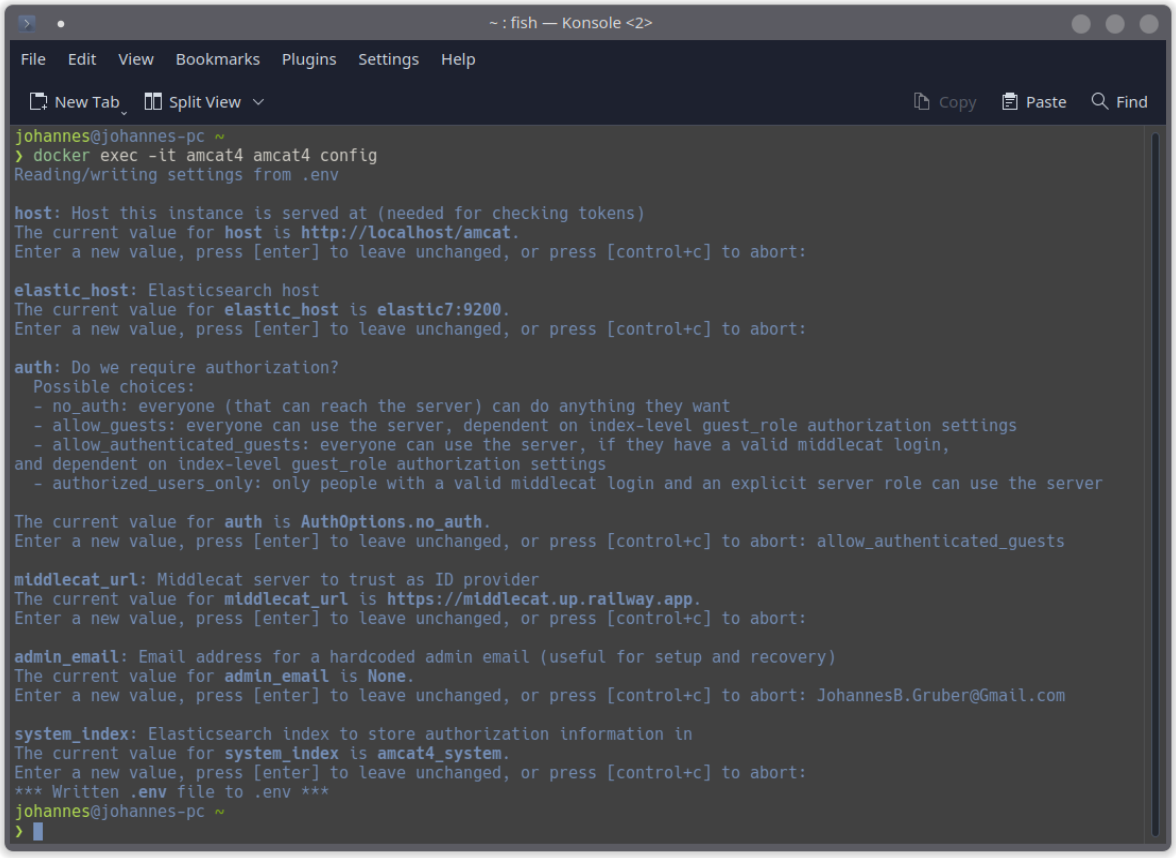


Figure 4.3: Configuring amcat4

```
docker exec -t amcat4 amcat4 add-admin admin@example.com
```

4.0.0.5 R

```
# coming soon
```

4.0.0.6 Python

```
# coming soon
```

You can also create an example data collection (which are called index in Elasticsearch):

4.0.0.7 Command Line

```
docker exec -t amcat4 amcat4 create-test-index
```

4.0.0.8 R

```
docker_exec(id = "amcat4", "amcat4 create-test-index")
```

4.0.0.9 Python

```
# coming soon
```

You can now use Middlecat to authenticate as the admin user:

Now you can access the test index at `http://localhost/`:

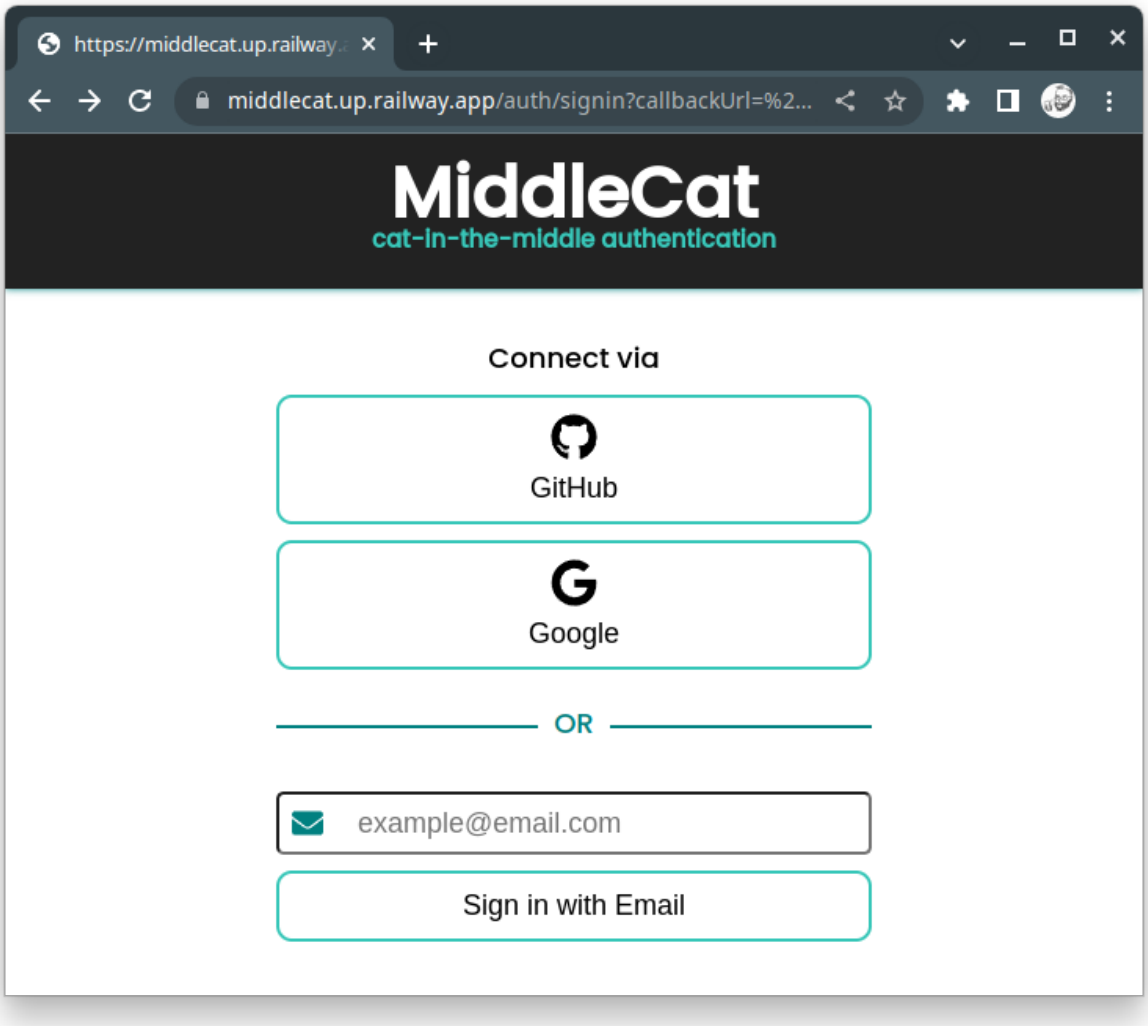


Figure 4.4: After logging into the amcat react app in your browser

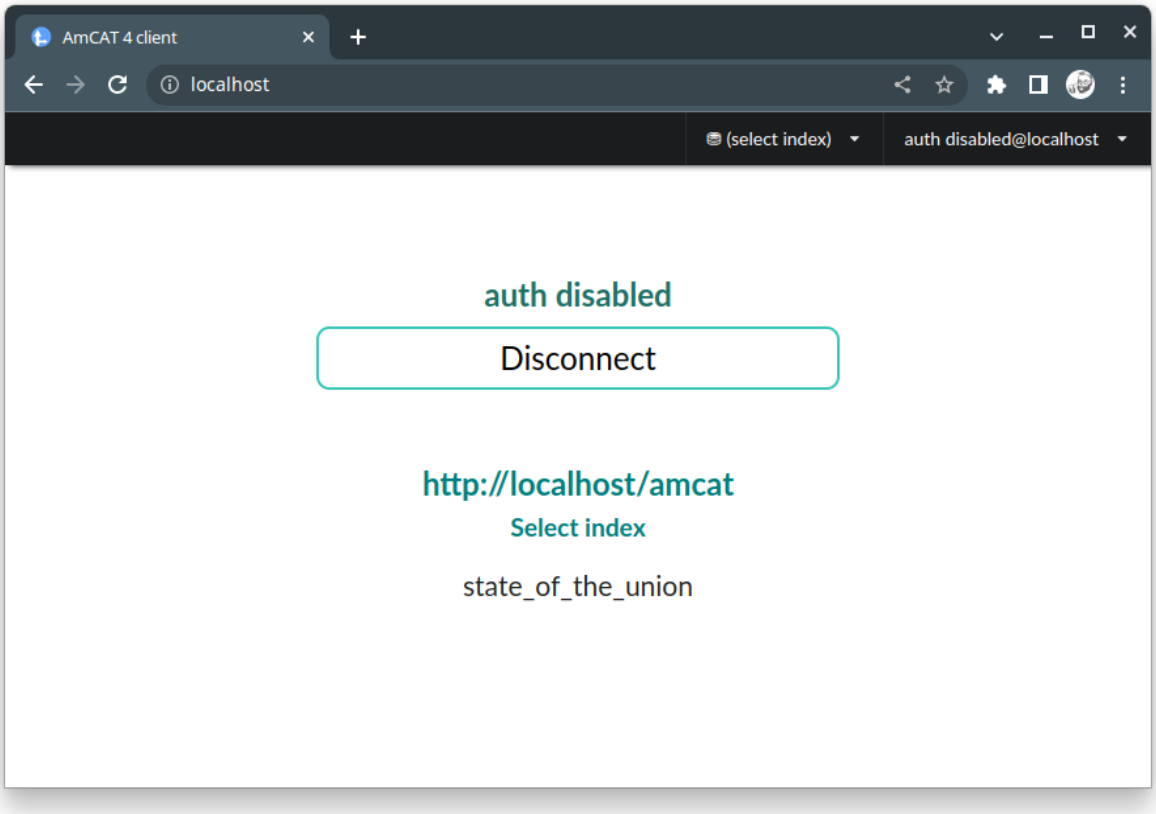


Figure 4.5: After logging into the amcat react app in your browser

4.0.1 Setup on your own server

If you decide not to go with Docker, for example, because you feel you need more control over what is happening, you can also run amcat on your system directly. We do not recommend this anymore and have, in fact, switched our own servers over to use the docker image. If there is something wrong with the images or you simply want to customise the setup, we suggest you head over to the GitHub repo and change the files as you like.

If you still want to go without docker, feel free to use the example configuration below. We assume that if you are going this route, you are running a Linux server. Below we show one example setup. Obviously feel free to replace the suggested Linux tools like *systemd* or *nginx* with your own choice.

4.0.1.1 amcat4 – aka amcat server

The first piece to set up is the amcat server and the Elasticsearch database it interacts with. To download and install Elasticsearch, refer to their website, or, preferably, install it through a package manager. For example, if you are running Debian or Ubuntu or another distro which uses `apt` you can install Elasticsearch 7.x (which we are currently working with) like this:

```
curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr
echo "deb [signed-by=/usr/share/keyrings/elastic.gpg] https://artifacts.elastic.co/package
sudo apt update
sudo apt install elasticsearch
```

In the next step, you need to configure Elasticsearch:

```
sudo nano /etc/elasticsearch/elasticsearch.yml
```

Configure the database to your own liking in terms of user management and exposure. Since we are controlling it through `amcat4`, the only two things that really matter is the address and port (and that `amcat4` still has access after you're done configuring Elasticsearch). So within `elasticsearch.yml`, we only look for two lines:

```
network.host: localhost
...
http.port: 9200
```

You can configure the memory usage of Elasticsearch

```
echo "-Xms4g" | sudo tee -a /etc/elasticsearch/jvm.options.d/memory.options
```

Leaving the values at their defaults here, we can enable the systemd service (skip this step if you've installed Elasticsearch through Docker):

```
sudo systemctl daemon-reload
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch
```

You can check if everything is working with:

```
curl -X GET 'http://localhost:9200'
#> {
#>   "name" : "amcat-opted-trekdrop0",
#>   "cluster_name" : "amcat-opted",
#>   "cluster_uuid" : "Sx-D89zmSx2zAcwl62u32A",
#>   "version" : {
#>     "number" : "7.17.6",
#>     "build_flavor" : "default",
#>     "build_type" : "deb",
#>     "build_hash" : "f65e9d338dc1d07b642e14a27f338990148ee5b6",
#>     "build_date" : "2022-08-23T11:08:48.893373482Z",
#>     "build_snapshot" : false,
#>     "lucene_version" : "8.11.1",
#>     "minimum_wire_compatibility_version" : "6.8.0",
#>     "minimum_index_compatibility_version" : "6.0.0-beta1"
#>   },
#>   "tagline" : "You Know, for Search"
#> }
```

Next, you want to setup the amcat server. You can do this wherever you like, but will will set things up at `/srv/amcat`:

```
sudo git clone https://github.com/ccs-amsterdam/amcat4 /srv/amcat
sudo chown -R $USER:$USER /srv/amcat
cd /srv/amcat
python3 -m venv env
env/bin/pip install -e .[dev]
```

To test if it runs as expected, you can use:

```
env/bin/python -m amcat4 run
#> /srv/amcat/env/lib/python3.9/site-packages/elasticsearch/connection/base.py:200: Elasti
#>   warnings.warn(message, category=ElasticsearchWarning)
```

```
#> [INFO   :root           ] Starting server at port 5000, debug=True
#> INFO:     Started server process [1001112]
#> [INFO   :uvicorn.error  ] Started server process [1001112]
#> INFO:     Waiting for application startup.
#> [INFO   :uvicorn.error  ] Waiting for application startup.
#> INFO:     Application startup complete.
#> [INFO   :uvicorn.error  ] Application startup complete.
#> INFO:     Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
#> [INFO   :uvicorn.error  ] Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
```

To check and adapt the settings of amcat use:

```
env/bin/python -m amcat4 config
```

Since you probably don't want to run amcat in an open ssh tab all the time, you should set it up as a service, for example with `systemd`. So head over to `/etc/systemd/system` and create a new file, for example, `amcat.service`.

Here is a small example to set things up:

```
[Unit]
Description=Amcat4 API
After=network.target
Requires=elasticsearch.service

[Service]
Type=simple
User=amcat
Group=amcat
WorkingDirectory=/srv/amcat/amcat4

Environment=AMCAT4_ELASTIC_HOST=http://localhost:9200
Environment=AMCAT4_DB_NAME=/srv/amcat/amcat4.db

ExecStart=/srv/amcat/env/bin/uvicorn \
        --proxy-headers \
        --forwarded-allow-ips='*' \
        --workers=2 \
        --no-access-log \
        --uds /tmp/amcat.socket \
        --root-path /api \
        amcat4.api:app
```

```
ExecReload=/bin/kill -HUP ${MAINPID}
RestartSec=1
Restart=always

[Install]
WantedBy=multi-user.target
```

In the above service, we run the amcat server as the user `amcat`. To create this user and hand over the ownership of the amcat server folder to it use:

```
sudo useradd amcat
sudo chown -R amcat:amcat /srv/amcat
```

Then you can start the service and enable it to run on startup:

```
sudo systemctl daemon-reload
sudo systemctl start amcat.service
sudo systemctl enable amcat.service
```

Now you can check if everything is working with

```
systemctl status amcat.service
 amcat.service - Amcat4 API
#>      Loaded: loaded (/etc/systemd/system/amcat.service; enabled; vendor preset: enabled
#>      Active: active (running) since Thu 2022-11-03 10:39:23 CET; 3min 29s ago
#>    Main PID: 197173 (uvicorn)
#>       Tasks: 4 (limit: 33532)
#>      Memory: 86.8M
#>         CPU: 1.770s
#>      CGroup: /system.slice/amcat.service
#>              197173 /srv/amcat/env/bin/python3 /srv/amcat/env/bin/uvicorn --proxy-head
#>              197174 /srv/amcat/env/bin/python3 -c from multiprocessing.resource_tracke
#>              197175 /srv/amcat/env/bin/python3 -c from multiprocessing.spawn import sp
#>              197176 /srv/amcat/env/bin/python3 -c from multiprocessing.spawn import sp
```

If something went wrong, you can troubleshoot with `sudo journalctl -eu amcat.service`.

4.1 Frontend

4.1.1 amcat4client

> **Note**
>
> If you are using amcat on our servers or through docker, you can skip this section and move on to install an API client to start managing amcat from either the R or Python .

If you have checked port 5000 of your new amcat server while testing it above (i.e., `http://0.0.0.0:5000`), you were probably disappointed by a simple `{"detail":"Not Found"}` message. This is because the client has been split from the main package to make it easier to develop. You can install the React client next to amcat in`/srv/amcat4client` using:

```
cd /srv/
sudo git clone https://github.com/ccs-amsterdam/amcat4client.git
sudo chown -R $USER: amcat4client
cd amcat4client
npm install
```

If you get error messages about outdated versions of dependencies (which is likely on Ubuntu and Debian) you should update `Node.js`. On Debian, you can do this likes so:

```
su
curl -fsSL https://deb.nodesource.com/setup_19.x | bash - &&\
apt-get install -y nodejs
exit
```

And the equivalent on Ubuntu:

```
curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash - &&\
sudo apt-get install -y nodejs
```

See this repository for instructions for other Linux flavours.

After that, you can build the React app:

```
npm run build
```

If your amcat instance will be publicly reachable, you can build the React app permanently attached to only your instance of amcat:

```
REACT_APP_FIXED_HOST=https://example.com/api npm run build
```

Once this has finished, you should hand over ownership of the React application to the previously created amcat user

```
sudo chown -R amcat:amcat .
```

Now we have an Elasticsearch and amcat4 running. But they are currently not accesible. To solve this, we can use, for example, `nginx` to provide users access to the React frontend and the amcat API. Create a new `nginx` config file with, for example, nano:

```
sudo nano /etc/nginx/sites-available/amcat.conf
```

Below is a minimal example of the `amcat.conf` file, which you can copy and paste: For more information, visit the uvicorn documentation website.

```
server {
    client_max_body_size 4G;

    listen 5000;

    location /api/ {
      rewrite  ^/api/(.*) /$1 break;
      proxy_set_header Host $http_host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection $connection_upgrade;
      proxy_redirect off;
      proxy_buffering off;
      proxy_pass http://amcat;
    }

    location / {
      root /srv/amcat4client/build;
      index index.html;
      try_files $uri $uri/ /index.html;
    }

}

map $http_upgrade $connection_upgrade {
```

```
    default upgrade;
    '' close;
}

upstream amcat {
    server unix:/tmp/amcat.socket;
}
```

Warning

5 warning about http

This setup assumes that your amcat sever will only be available in the local network. If it should be accesible via the internet, we **strongly** recommend to enable https. You can find more information about that on the `nginx` website or this guide.

To enable the site, use:

```
sudo ln -s /etc/nginx/sites-available/amcat.conf /etc/nginx/sites-enabled/amcat.conf
```

Then simply restart `nginx`, for example, through systemd:

```
sudo systemctl restart nginx.service
```

To test if the API is reachable, use this:

```
curl http://localhost:5000/api/
#> {"detail":"Not Found"}
```

If everything works, you can now access the client at http://localhost:5000, or the address of your server, if you installed amcat remotely:

The React app is always running locally in your browser, even if you've accessed it on another computer. So the appropriate host needs to be the route to the amcat server. In the example above, I set up an amcat instance in my local network on a computer with the IP address `192.168.2.180` and port `5000`. To access that host, you need to enter:

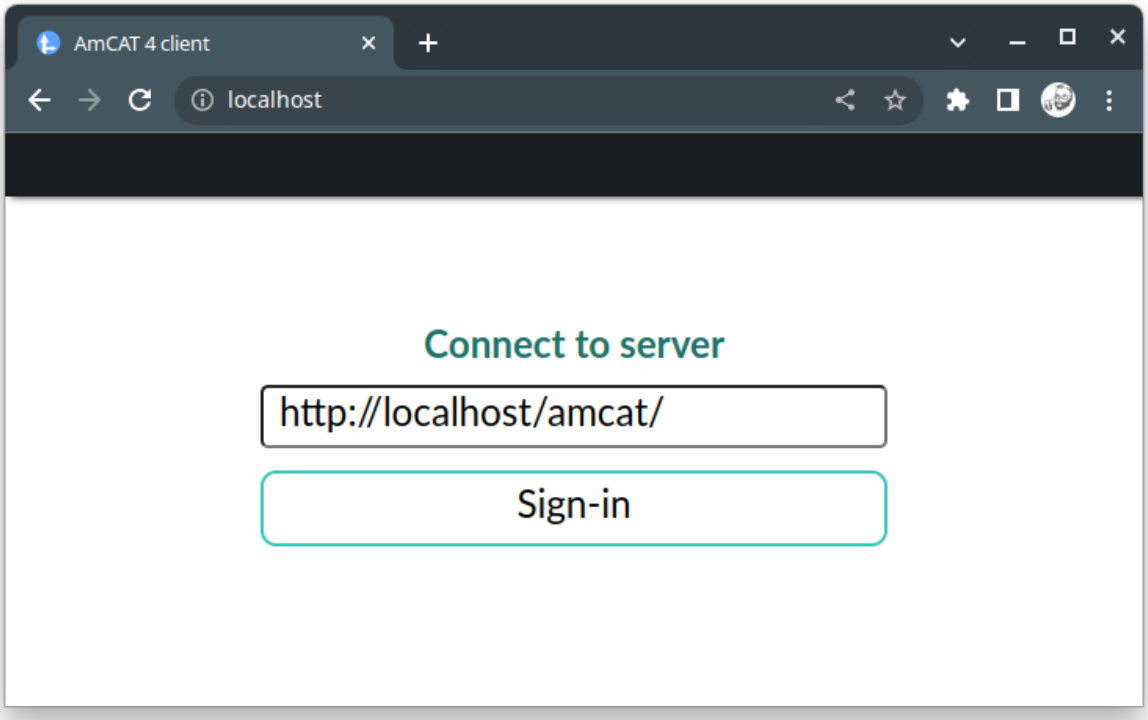


Figure 5.1: First view of the amcat react app in your browser

> **Note**
>
> Host: “http://192.168.2.180:5000/api”
> Email: “admin”
> Password: “admin”

Just replace `192.168.2.180` with the address of the machine you set up amcat on.

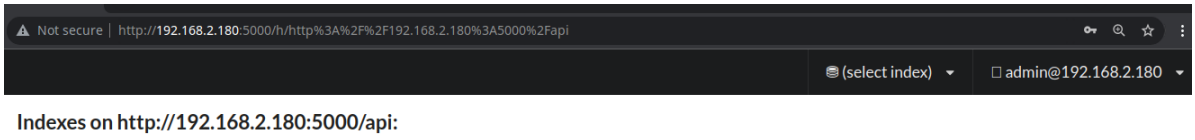


Figure 5.2: After logging into the amcat react app in your browser

Success! However, the interface doesn't show much at this point, since we added no data yet. We will do that in the storage chapter.

5.0.1 API Client

5.0.1.1 R

The `R` client is called `amcat4r` and can be installed via the following command in `R` (install `remotes` first if you don't have it yet):

```
remotes::install_github("ccs-amsterdam/amcat4r")
```

If you have set up the amcat suite as shown above, you should be able to log into the database:

```
library(amcat4r)
login("http://localhost/api",  username = "admin", password = "supergeheim")
```

If this does not throw an error, you have set everything up correctly.

5.0.1.2 Python

Install `amcat4py` from the command line through `pip`:

```
pip install amcat4py
```

The you can open Python and log in:

```
from amcat4py import AmcatClient

amcat = AmcatClient("http://localhost/amcat")
```

If this does not throw an error, you have set everything up correctly.

6 Document Storage

Note

You need at least the modules in the Data Layer and one Frontend Client to work with document storage:

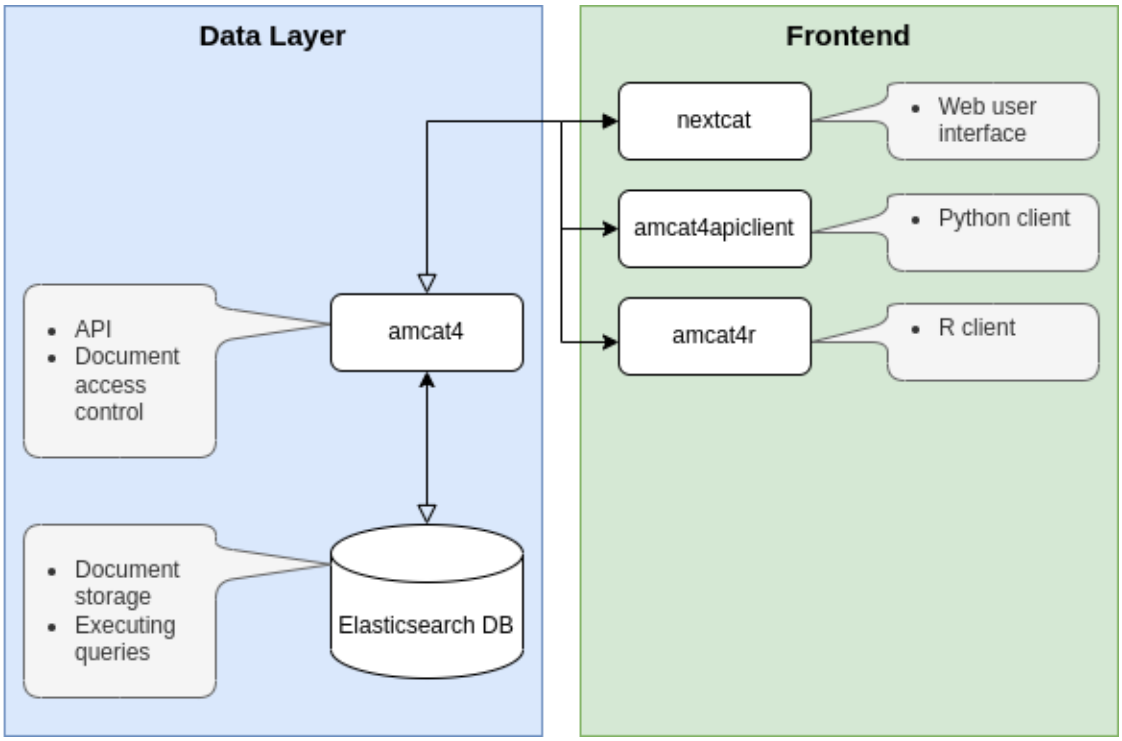


Figure 6.1: amcat instance after following this chapter

This chapter covers how you can upload, change, query and delete documents and indexes on the amcat server. Only a few tasks are implemented in the Web user interface, which means you will need to use one of the clients in `R` or `Python` or another means to call the API (e.g., through cURL as shown below). You can also use the API calls to build your own client. More information on the API can be found on every amcat4 instance at `/redoc` (e.g., http://localhost/amcat/redoc). Let us know about it and we will promote your new API wrapper package here.

! Important

7 Coming soon...

! Important

8 Will change soon

Currently, there is no way to upload, change, or delete documents and indexes through the web interface. Rather, you can add new datasets through calls to the amcat API.

8.1 Manage Documents With a Client

For this overview, we log into a local `amcat4` (i.e., `http://localhost/amcat`). Replace this with the address to the `amcat4` instance you are working with (e.g., `https://opted.amcat.nl/api`).

We first need to log in:

8.1.0.1 R

```
library(amcat4r)
amcat_login("http://localhost/amcat")
```

8.1.0.2 Python

```
from amcat4py import AmcatClient
amcat = AmcatClient("http://localhost/amcat")
```

8.1.0.3 cURL

There is no dedicated way at the moment to get a token via cURL. You can still use cURL with instances that do not require authentication or by copying the token from `Python` or `R`. In these cases, you can make requests with an extra header, for example:

```
AMCAT_TOKEN="YOUR_TOKEN"
curl -s http://localhost/amcat/index/ \
  -H "Authorization: Bearer ${AMCAT_TOKEN}"
```

We can first list all available indexes, as a document collection is called in Elasticsearch and thus in amcat:

8.1.0.4 R

```
list_indexes()
```

```
# A tibble: 1 x 1
  name
  <chr>
1 state_of_the_union
```

8.1.0.5 Python

```
amcat.list_indices()
```

```
[{'name': 'state_of_the_union'}]
```

8.1.0.6 cURL

```
curl -s http://localhost/amcat/index/
```

```
[{"name":"state_of_the_union"}]
```

You can see that the test index we added in the Data Layer section is here and that it is called "state_of_the_union". To see everyone who has been granted access to an index we can use:

8.1.0.7 R

```
list_index_users(index = "state_of_the_union")
```

```
# A tibble: 0 x 0
```

8.1.0.8 Python

```
amcat.list_index_users(index="state_of_the_union")
```

```
[]
```

8.1.0.9 cURL

```
curl -s http://localhost/amcat/index/state_of_the_union/users
```

```
[]
```

We will learn more about these roles in the chapter on access management. To see what an index looks like, we can query it leaving all fields blank to request all data at once:

8.1.0.10 R

```
sotu <- query_documents(index = "state_of_the_union", queries = NULL, fields = NULL)
```

```
Retrieved 234 results in 1 pages
```

```
str(sotu)
```

```
tibble [234 x 9] (S3: tbl_df/tbl/data.frame)
 $ .id      : chr [1:234] "tMmcK4YBzjTKB-2w5n6J" "tcmcK4YBzjTKB-2w5n6J" "tsmcK4YBzjTKB-2w5n6J
 $ title    : chr [1:234] "1790: George Washington" "1790: George Washington" "1791: George W
 $ text     : chr [1:234] "Fellow-Citizens of the Senate and House of Representatives:  \nI e
 $ date     : POSIXct[1:234], format: "1790-01-08" "1790-12-08" ...
 $ president: chr [1:234] "George Washington" "George Washington" "George Washington" "George
 $ year     : num [1:234] 1790 1790 1791 1792 1793 ...
 $ party    : chr [1:234] "N/A" "N/A" "N/A" "N/A" ...
 $ keyword  :List of 234
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
```

```
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
```

```
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  .. [list output truncated]
 $ url       : chr [1:234] NA NA NA NA ...
```

8.1.0.11 Python

```
sotu = list(amcat.query("state_of_the_union", fields=None))
print(len(sotu))
```

```
234
```

```
for k, v in sotu[1].items():
  print(k + "(" + str(type(v)) + "): " + str(v)[0:100] + "...")
```

```
_id(<class 'str'>): tcmcK4YBzjTKB-2w5n6J...
```

```
title(<class 'str'>): 1790: George Washington...
text(<class 'str'>): Fellow-Citizens of the Senate and House of Representatives:
In meeting you again I feel much satis...
date(<class 'datetime.datetime'>): 1790-12-08 00:00:00...
president(<class 'str'>): George Washington...
year(<class 'float'>): 1790.0...
party(<class 'str'>): N/A...
```

8.1.0.12 cURL

To not clog the output, we save it into file and display only the beginning:

```
curl -s http://localhost/amcat/index/state_of_the_union/documents > sotu.json
# show the first few characters only
head -c 150 sotu.json
```

```
{"results":[{"_id":"tMmcK4YBzjTKB-2w5n6J","title":"1790: George Washington","text":"Fellow-C
```

Knowing now what a document should look like in this index, we can upload a new document to get familiar with the process:

8.1.0.13 R

```
new_doc <- data.frame(
  title = "test",
  text = "test",
  date = as.Date("2022-01-01"),
  president = "test",
  year = "2022",
  party = "test",
  url = "test"
)
upload_documents(index = "state_of_the_union", new_doc)
```

8.1.0.14 Python

```
from datetime import datetime
new_doc = {
  "title": "test",
  "text": "test",
  "date": datetime.strptime("2022-01-01", '%Y-%m-%d'),
  "president": "test",
  "year": "2022",
  "party": "test",
  "url": "test"
}
amcat.upload_documents("state_of_the_union", [new_doc])
```

8.1.0.15 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/documents \
  -H "Content-Type: application/json" \
  -d '{
        "documents":[
           {
              "title":"test",
              "text":"test",
              "date":"2022-01-01",
              "president":"test",
              "year":"2022",
              "party":"test",
              "url":"test"
           }
        ]
     }'
```

Let's see if the the new document is in the index:

8.1.0.16 R

```
query_documents(index = "state_of_the_union", fields = NULL, filters = list(title = "test"
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 9
  .id          date                 year text  title keyword url   party presi~1
  <chr>        <dttm>              <dbl> <chr> <chr> <list>  <chr> <chr> <chr>
1 nMmfK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
2 1            2022-01-02 00:00:00  2022 A se~ test  <NULL>  <NA>  <NA>  test
3 nsmgK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <NULL>  test  test  test
# ... with abbreviated variable name 1: president
```

8.1.0.17 Python

```python
import pprint
pp = pprint.PrettyPrinter(depth=4)
res=list(amcat.query("state_of_the_union", fields=None, filters={"title": "test"}))
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': '1',
  'date': datetime.datetime(2022, 1, 2, 0, 0),
  'president': 'test',
  'text': 'A second test',
  'title': 'test',
  'year': 2022.0},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0}]
```

8.1.0.18 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```


```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "title": "test",
    "date": "2022-01-02",
    "text": "A second test",
    "president": "test",
    "year": 2022
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "title": "test",
    "date": "2022-01-01",
    "text": "test",
    "url": "test",
    "party": "test",
    "president": "test",
    "year": 2022
  }
]
```


We will learn more about queries later on in the Writing a Query chapter.

Instead of adding whole documents, you can also change fields in an index. Fields are similar to columns in a table in Excel. However, you need to define the type of a field upon its creation and make sure that you later only add data which adheres to the specifications of the type (otherwise you will get an error). To learn more about the fields in the test index, you can use:

8.1.0.19 R

```
get_fields(index = "state_of_the_union")
```

```
# A tibble: 8 x 2
  name      type
  <chr>     <chr>
1 date      date
2 keyword   keyword
3 party     keyword
4 president keyword
5 text      text
6 title     text
7 url       url
8 year      double
```

8.1.0.20 Python

```
amcat.get_fields("state_of_the_union")
```

```
{'date': {'name': 'date', 'type': 'date'}, 'keyword': {'name': 'keyword', 'type': 'keyword'}
```

8.1.0.21 cURL

```
curl -s http://localhost/amcat/index/state_of_the_union/fields
```

```
{"date":{"name":"date","type":"date"},"keyword":{"name":"keyword","type":"keyword"},"party":
```

You can see that there are five different types in this index: date, keyword, text, url and double. Keyword, text, url are all essentially the same type in `R`, namely character strings. The date needs to be a `POSIXct` class, which you can create with `as.Date`. Year should be a double, i.e., a numeric value or integer.

You can add new fields to this, for example, if you want to add a keyword to the documents:

8.1.0.22 R

```
set_fields(index = "state_of_the_union", list(keyword = "keyword"))
```

8.1.0.23 Python

```
amcat.set_fields("state_of_the_union", {"keyword":"keyword"})
```

8.1.0.24 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/fields \
  -H 'Content-Type: application/json' \
  -d '{"keyword":"keyword"}'
```

When you now query a document, however, you will not see this new field:

8.1.0.25 R

```
query_documents(index = "state_of_the_union", fields = NULL, filters = list(title = "test"
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 9
  .id          date                 year text  title keyword url   party presi~1
  <chr>        <dttm>              <dbl> <chr> <chr> <list>  <chr> <chr> <chr>
1 nMmfK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
2 1            2022-01-02 00:00:00  2022 A se~ test  <NULL>  <NA>  <NA>  test
3 nsmgK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <NULL>  test  test  test
# ... with abbreviated variable name 1: president
```

8.1.0.26 Python

```
res = list(amcat.query("state_of_the_union", fields=None, filters={"title": "test"}))
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': '1',
  'date': datetime.datetime(2022, 1, 2, 0, 0),
  'president': 'test',
  'text': 'A second test',
  'title': 'test',
  'year': 2022.0},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0}]
```

8.1.0.27 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```

```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
```

```
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "title": "test",
    "date": "2022-01-02",
    "text": "A second test",
    "president": "test",
    "year": 2022
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "title": "test",
    "date": "2022-01-01",
    "text": "test",
    "url": "test",
    "party": "test",
    "president": "test",
    "year": 2022
  }
]
```


This is because it is empty for this document, just as the url field, which is absent from all documents in this index. We can add something to the new field and see if it shows up:

8.1.0.28 R

```r
update_tags(index = "state_of_the_union",
            action = "add",
            field = "keyword",
            tag = "test",
            filters = list(title = "test"))
```

```r
query_documents(index = "state_of_the_union",
                fields = c("title", "keyword"),
                filters = list(title = "test"))
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 3
  .id                   title keyword
  <chr>                 <chr> <list>
1 nMmfK4YBzjTKB-2wn39U test  <chr [1]>
2 1                     test  <chr [1]>
3 nsmgK4YBzjTKB-2wGH8W test  <chr [1]>
```

8.1.0.29 Python

```python
test_doc = list(amcat.query("state_of_the_union", fields=["id"], filters={"title": "test"}
amcat.update_document("state_of_the_union", doc_id=test_doc["_id"], body={"keyword": "test
```

```python
res=list(amcat.query("state_of_the_union", fields=["title", "keyword"], filters={"title":
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U', 'keyword': ['test'], 'title': 'test'},
 {'_id': '1', 'keyword': ['test'], 'title': 'test'},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W', 'keyword': ['test'], 'title': 'test'}]
```

8.1.0.30 cURL

```
test_doc=$(curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results[]._id")
curl -s -X PUT http://localhost/amcat/index/state_of_the_union/documents/${test_doc} \
  -H 'Content-Type: application/json' \
  -d '{"keyword": "test"}'
```

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```

```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "date": "2022-01-02",
    "year": 2022,
    "text": "A second test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "president": "test"
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  }
]
```


Now that we have a better idea of what an index is and how it looks like, we can create a new one>

8.1.0.31 R

```
create_index(index = "new_index", guest_role = "admin")
list_indexes()
```

```
# A tibble: 2 x 1
  name
  <chr>
1 new_index
2 state_of_the_union
```

```
get_fields(index = "new_index")
```

```
# A tibble: 7 x 2
  name      type
  <chr>     <chr>
1 date      date
2 party     text
3 president text
4 text      text
5 title     text
6 url       url
7 year      text
```

8.1.0.32 Python

```
amcat.create_index(index="new_index", guest_role="admin")
```

```
amcat.list_indices()
```

```
[{'name': 'new_index'}, {'name': 'state_of_the_union'}]
```

```
amcat.get_fields("new_index")
```

```
{'date': {'name': 'date', 'type': 'date'}, 'party': {'name': 'party', 'type': 'text'}, 'presi
```

8.1.0.33 cURL

```
curl -s -X POST http://localhost/amcat/index/ \
  -H 'Content-Type: application/json' \
  -d '{
        "name": "new_index",
        "guest_role": "ADMIN"
      }'
```

```
curl -s http://localhost/amcat/index/
curl -s http://localhost/amcat/index/new_index/fields
```

```
[{"name":"new_index"},{"name":"state_of_the_union"}]{"date":{"name":"date","type":"date"},"pa
```

As you can see, the newly created index already contains fields. You could now manually define new fields to fit your data. Or you can simply start uploading data:

8.1.0.34 R

```
new_doc <- data.frame(
  title = "test",
  text = "test",
  date = as.Date("2022-01-01"),
  president = "test",
  year = "2022",
  party = "test",
  url = "test"
)
upload_documents(index = "new_index", new_doc)
```

```
get_fields(index = "new_index")
```

```
# A tibble: 7 x 2
  name      type
  <chr>     <chr>
1 date      date
2 party     text
3 president text
```

```
4 text      text
5 title     text
6 url       url
7 year      text
```

8.1.0.35 Python

```
new_doc = {
  "title": "test",
  "text": "test",
  "date": datetime.strptime("2022-01-01", '%Y-%m-%d'),
  "president": "test",
  "year": "2022",
  "party": "test",
  "url": "test"
}
amcat.upload_documents("new_index", [new_doc])
```

```
amcat.get_fields("new_index")
```

```
{'date': {'name': 'date', 'type': 'date'}, 'party': {'name': 'party', 'type': 'text'}, 'presi
```

8.1.0.36 cURL

```
curl -s -X POST http://localhost/amcat/index/new_index/documents \
  -H "Content-Type: application/json" \
  -d '{
        "documents":[
           {
              "title":"test",
              "text":"test",
              "date":"2022-01-01",
              "president":"test",
              "year":"2022",
              "party":"test",
              "url":"test"
           }
        ]
     }'
```

```
curl -s http://localhost/amcat/index/new_index/fields
```

```
{"date":{"name":"date","type":"date"},"party":{"name":"party","type":"text"},"president":{"na
```

amcat4 guesses the types of fields based on the data. You can see here that this might not be the best option if you care about data types: party and president have been created as text, when they should be keywords; year is now a long type instead of double or integer.

Finally, we can also delete an index:

8.1.0.37 R

```
delete_index(index = "new_index")
```

8.1.0.38 Python

```
amcat.delete_index("new_index")
```

8.1.0.39 cURL

```
curl -s -X DELETE http://localhost/amcat/index/new_index
```

8.1.1 A Note on the ID Field and Duplicated Documents

amcat indexes can have all kinds of fields, yet one special field must be present in every document of every index: a unique ID. This ID is usually not that noteworthy, since the user does not really need to take care of it. This changes, however, with the special case of duplicated documents, that is, a document with the exact same information in the same fields. amcat does not normally check if your documents are duplicated when you upload them. However, when no ID is present in an uploaded document, as in the example we uploaded above, amcat will construct a unique ID from the available data of a document. Let us have another look at that document we titled “test”:

8.1.1.1 R

```
query_documents(index = "state_of_the_union", fields = NULL, filters = list(title = "test"
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 9
  .id          date                 year text  title keyword url   party presi~1
  <chr>        <dttm>              <dbl> <chr> <chr> <list>  <chr> <chr> <chr>
1 nMmfK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
2 1            2022-01-02 00:00:00  2022 A se~ test  <chr>   <NA>  <NA>  test
3 nsmgK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
# ... with abbreviated variable name 1: president
```

8.1.1.2 Python

```
res=list(amcat.query("state_of_the_union", fields=None, filters={"title": "test"}))
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': '1',
  'date': datetime.datetime(2022, 1, 2, 0, 0),
  'keyword': ['test'],
  'president': 'test',
  'text': 'A second test',
  'title': 'test',
  'year': 2022.0},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
```

```
 'president': 'test',
 'text': 'test',
 'title': 'test',
 'url': 'test',
 'year': 2022.0}]
```

8.1.1.3 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```


```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "date": "2022-01-02",
    "year": 2022,
    "text": "A second test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "president": "test"
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "date": "2022-01-01",
    "year": 2022,
```

```
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  }
]
```

Note that amcat has automatically added an `_id` field (in `R` it is `.id` due to naming conventions) to the document. If we would upload the same document again, the algorithm that constructs the `_id` field would come up with the same value and the document would be replaced by the newly uploaded document. If we wanted to keep a duplicate for some reason, we could accomplish that by either changing at least one value in a field or by assigning an ID column manually:

8.1.1.4 R

```r
new_doc <- data.frame(
  .id = "1",
  title = "test",
  text = "test",
  date = as.Date("2022-01-01"),
  president = "test",
  year = "2022",
  party = "test",
  url = "test"
)
upload_documents(index = "state_of_the_union", new_doc)
```

```r
query_documents(index = "state_of_the_union", fields = NULL, filters = list(title = "test"
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 9
  .id          date                 year text  title keyword url   party presi~1
  <chr>        <dttm>              <dbl> <chr> <chr> <list>  <chr> <chr> <chr>
1 nMmfK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
```

```
2 nsmgK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
3 1            2022-01-01 00:00:00  2022 test  test  <NULL>  test  test  test
# ... with abbreviated variable name 1: president
```

8.1.1.5 Python

```
from datetime import datetime
new_doc = {
  "_id": "1",
  "title": "test",
  "text": "test",
  "date": datetime.strptime("2022-01-01", '%Y-%m-%d'),
  "president": "test",
  "year": "2022",
  "party": "test",
  "url": "test"
}
amcat.upload_documents("state_of_the_union", [new_doc])
```

```
res=list(amcat.query("state_of_the_union", fields=None, filters={"title": "test"}))
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
```

```
{'_id': '1',
 'date': datetime.datetime(2022, 1, 1, 0, 0),
 'party': 'test',
 'president': 'test',
 'text': 'test',
 'title': 'test',
 'url': 'test',
 'year': 2022.0}]
```

8.1.1.6 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/documents \
  -H "Content-Type: application/json" \
  -d '{
        "documents":[
          {
            "_id": "1",
            "title":"test",
            "text":"test",
            "date":"2022-01-01",
            "president":"test",
            "year":"2022",
            "party":"test",
            "url":"test"
          }
        ]
      }'
```

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```

```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
```

```
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "title": "test",
    "date": "2022-01-01",
    "text": "test",
    "url": "test",
    "party": "test",
    "president": "test",
    "year": 2022
  }
]
```


As you can see, we now have the example document in the index twice – although with different IDs. To simulate what would have happened without an ID, or rather if the ID had been constructed automatically, we can upload different data, but with the same ID to see what changes:

8.1.1.7 R

```r
new_doc <- data.frame(
  .id = "1",
  title = "test",
  text = "A second test",
  date = as.Date("2022-01-02"),
  president = "test",
  year = "2022"
)
upload_documents(index = "state_of_the_union", new_doc)
```

```r
query_documents(index = "state_of_the_union", fields = NULL, filters = list(title = "test"
```

```
Retrieved 3 results in 1 pages
```

```
# A tibble: 3 x 9
  .id          date                year text  title keyword url   party presi~1
  <chr>        <dttm>              <dbl> <chr> <chr> <list>  <chr> <chr> <chr>
1 nMmfK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
2 nsmgK4YBzjT~ 2022-01-01 00:00:00  2022 test  test  <chr>   test  test  test
3 1            2022-01-02 00:00:00  2022 A se~ test  <NULL>  <NA>  <NA>  test
# ... with abbreviated variable name 1: president
```

8.1.1.8 Python

```python
from datetime import datetime
new_doc = {
  "_id": "1",
  "title": "test",
  "text": "A second test",
  "date": datetime.strptime("2022-01-02", '%Y-%m-%d'),
  "president": "test",
  "year": "2022"
}
amcat.upload_documents("state_of_the_union", [new_doc])
```

```python
res=list(amcat.query("state_of_the_union", fields=None, filters={"title": "test"}))
pp.pprint(res)
```

```
[{'_id': 'nMmfK4YBzjTKB-2wn39U',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': 'nsmgK4YBzjTKB-2wGH8W',
  'date': datetime.datetime(2022, 1, 1, 0, 0),
  'keyword': ['test'],
  'party': 'test',
  'president': 'test',
  'text': 'test',
  'title': 'test',
  'url': 'test',
  'year': 2022.0},
 {'_id': '1',
  'date': datetime.datetime(2022, 1, 2, 0, 0),
  'president': 'test',
  'text': 'A second test',
  'title': 'test',
  'year': 2022.0}]
```

8.1.1.9 cURL

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/documents \
  -H "Content-Type: application/json" \
  -d '{
        "documents":[
          {
            "_id": "1",
            "title":"test",
            "text":"A second test",
            "date":"2022-01-02",
            "president":"test",
            "year":"2022"
          }
        ]
      }'
```

```
curl -s -X POST http://localhost/amcat/index/state_of_the_union/query \
  -H 'Content-Type: application/json' \
  -d '{"filters":{"title":["test"]}}' | jq -r ".results"
```


```
[
  {
    "_id": "nMmfK4YBzjTKB-2wn39U",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "nsmgK4YBzjTKB-2wGH8W",
    "date": "2022-01-01",
    "year": 2022,
    "text": "test",
    "title": "test",
    "keyword": [
      "test"
    ],
    "url": "test",
    "party": "test",
    "president": "test"
  },
  {
    "_id": "1",
    "title": "test",
    "date": "2022-01-02",
    "text": "A second test",
    "president": "test",
    "year": 2022
  }
]
```


The document with the ID 1 has been replaced with the new data. This is the normal behaviour

of amcat: when we tell it to add data to an already present document, identified by the ID, it will be replaced. If a field was present in the old document, but not in the data it is replaced with, this field will be empty afterwards.

9 Document Sharing and Access Management

> ❗ Important

9.1 Cat-in-the-middle authentication

9.2 Access Management

9.2.1 Creating and Deleting User Accounts

```
list_users()
```

```
create_user(
  email = "user@example.com",
  password = "test",
  global_role = "writer",
  index_access = "state_of_the_union"
)
```

While you are here, it probably makes sense to change the admin password to something secure:

```
modify_user(email = "admin", new_password = "a")
```

```
add_index_user()
```

```
delete_index_user()
```

```
modify_index_user()
```

9.2.2 Roles and Guest Roles

10 Public Document Presentation

Sharing documents with your collaborators and research assistants is not the only scenario in which you want to make documents available to others. Sometimes, you want to share a document collection with a wider public. Whether you want to make the full-text documents available or just let users discover frequencies of terms and word combinations in your text, a dashboard built on top of amcat can be the solution. This chapter introduces you to how you can build a dashboard on top of amcat.

> ! Important

11 Fast searches in amcat databses

> **!** Important

12 amcat for document annotation

> **!** Important

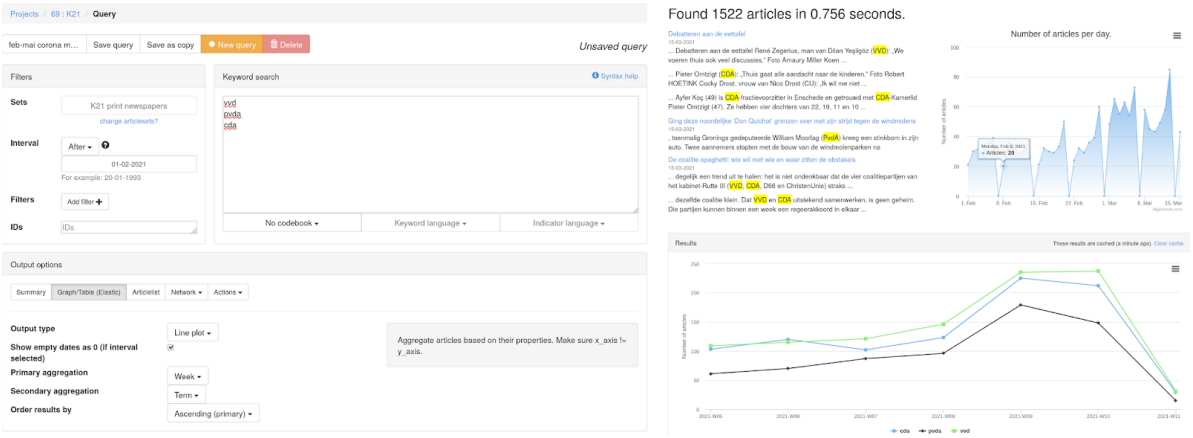
13 Preprocessing and machine learning

> ! Important

13.1 Why nlpipe?

14 About amcat

AmCAT has been in development in various guises since about 2001 as a text research / content analysis platform at the VU. From the start, the core has been a **database** of news articles, an **API** allows other programs and power users to connect to it directly from e.g. Python or R and an intuitive **query interface** for performing quantitative analysis. The screenshots below give some indication of the database and query interface of AmCAT 3.5.



In the last 20 years, the availability of user friendly tools for text analysis has greatly increased. Many proprietary platforms like LexisNexis or Coosto allow basic text analysis directly on the dashboard, while R and Python based toolkits like *quanteda* (Benoit et al. 2018) and *scikit-learn* (Pedregosa et al. 2011) have put very powerful analysis possibilities within the reach of a technically inclined social scientist. However, even with those advances we believe that a tool such as AmCAT can still play an important role for text analysis in the social sciences:

- An intuitive user interface gives a visual overview of the stored texts and **allows non-technical users to interact with the system**, without restricting its use to a single proprietary platform. This also makes it a **valuable resource in teaching** quantitative content analysis.
- A shared storage format, and API makes it easier to **share and reuse** tools and analysis scripts. It also aids **reproducible science** by allowing analyses and data to be shared and validated based on immutable data sets.
- Finally, as communication science data is often proprietary, fine-grained access control allows results to be shared and validated without giving access to the underlying data using **non-consumptive research**.

This is, why AmCAT was redesigned within work package 7 of the OPTED (Observatory for Political Texts in European Democracies: A European research infrastructure) project. The goal of the redesign is to create an infrastructure that is simple and flexible enough to be adapted for various projects at OPTED partner universities, other research insitutions and by basically everyone interested in text analysis.

References

Benoit, Kenneth, Kohei Watanabe, Haiyan Wang, Paul Nulty, Adam Obeng, Stefan Müller, and Akitaka Matsuo. 2018. "Quanteda: An r Package for the Quantitative Analysis of Textual Data." *Journal of Open Source Software* 3 (30): 774. https://doi.org/10.21105/joss.00774.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.