DCOR

Paul Müller

Apr 22, 2024

CONTENTS:

This is the official documentation of the Deformability Cytometry Open Repository (DCOR), a public repository for deformability cytometry datasets hosted by the Max-Planck-Gesellschaft.

CHAPTER

ONE

INTRODUCTION

1.1 Background

RT-DC is a microfluidics-based imaging technique that provides a high-throughput, high-dimensional, single-cell analysis. Measurement rates reach 1000 cells per second. An image is recorded for each cell, enabling cell charactererization based on its phenotype. Due to the moderate microfluidic forces in the imaging channel, cells are deformed which makes it possible to infer mechanical properties. In addition, fluorescence information can be recorded, allowing a direct comparison to flow cytometry measurements.

Since RT-DC measurements are comparatively large (hundreds of MB to several GB), the handling and/or backup of these data can become a problem, especially for small research and diagnostics labs. The deformability cytometry open repository (DCOR) offers a solution to this problem. Users can upload their RT-DC data, create collections, share with other users, and cite their data in scientific publications. Furthermore, DCOR is designed to integrate with the open-source analysis software Shape-Out; with DCOR, data analysis only requires a network connection, the actual data remain on the server.

1.2 DCOR is open and free

The official DCOR service at https://dcor.mpl.mpg.de is free of charge. If you are not permitted (e.g. by data protection laws) to store your data there, you can always set up your own DCOR instance. This process is described in the *self-hosting section* and should probably (depending on your storage and backup strategy) involve your IT department. Please let us know if you are planning to set up your own DCOR instance so we can advertise this. Also, please don't hesitate to get into contact with us (e.g. issues and pull requests on GitHub) if you feel like you are missing a specific feature or configuration option. DCOR should be robust and user-friendly - let's improve it together!

1.3 Technology

DCOR is based on CKAN, an online data managing and publishing system. We provide a set of extensions and tools designed to make the work with RT-DC data easier. For instance, this includes a RESTful service that allows Shape-Out to directly access RT-DC resources without downloading entire measurements (ckanext-dc_serve) or previews of RT-DC data on CKAN web interface (ckanext-dc_view). You can find all extensions and tools at the DCOR-dev GitHub organization.

CHAPTER

TWO

USING DCOR

2.1 Accessing Data on DCOR

2.1.1 General remarks

There are two ways of interacting with data on a DCOR instance, via the web interface or via the API. With the web interface (not covererd here), you can browse and search data in a convenient way with your webbrowser. The API allows you to write custom scripts or libraries (DCOR-Aid uses the API).

Note that there are two main DCOR instances. One for development and testing (❖ DCOR-dev) and one for production use (❖ DCOR). If you are new to DCOR, please use the DCOR-dev instance to get to know the system. If you are ready to get serious, move on to the production instance.

2.1.2 Access via DCOR-Aid GUI

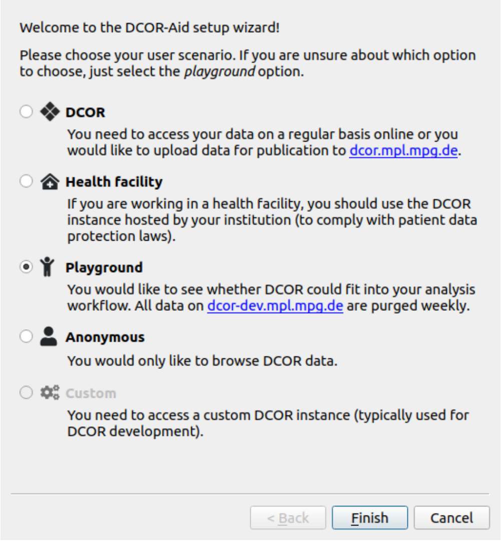
It is possible to access all data on DCOR via your browser by visiting https://dcor.mpl.mpg.de. However, you might want to consider using DCOR-Aid instead, because:

- You can more easily browse circles and collection in the DCOR-Aid GUI.
- You can drag and drop resources from DCOR-Aid into Shape-Out (no need to copy and paste resource IDs).
- DCOR-Aid comes with a resource download manager.

If you installed DCOR-Aid for the first time, the setup wizard will ask you to choose how you would like to use DCOR-Aid. If you are only interested in public data, then choose the *Anonymous* option.

When DCOR-Aid starts, you will then see several tabs. The tab on the right *Find Data* allows you to search the DCOR database for datasets and resources. If you previously entered an API token, then you can also browse all your datasets in the *My Data* tab.

To search for a particular dataset, simply type your search term in the search field. If you are interested in more elaborate search options, please create an issue at the DCOR-Aid issue page.



Welcome to the DCOR-Aid setup wizard!

Please choose your user scenario. If you are unsure about which option to choose, just select the *playground* option.

○ ❖ **DCOR**
You need to access your data on a regular basis online or you would like to upload data for publication to dcor.mpl.mpg.de.

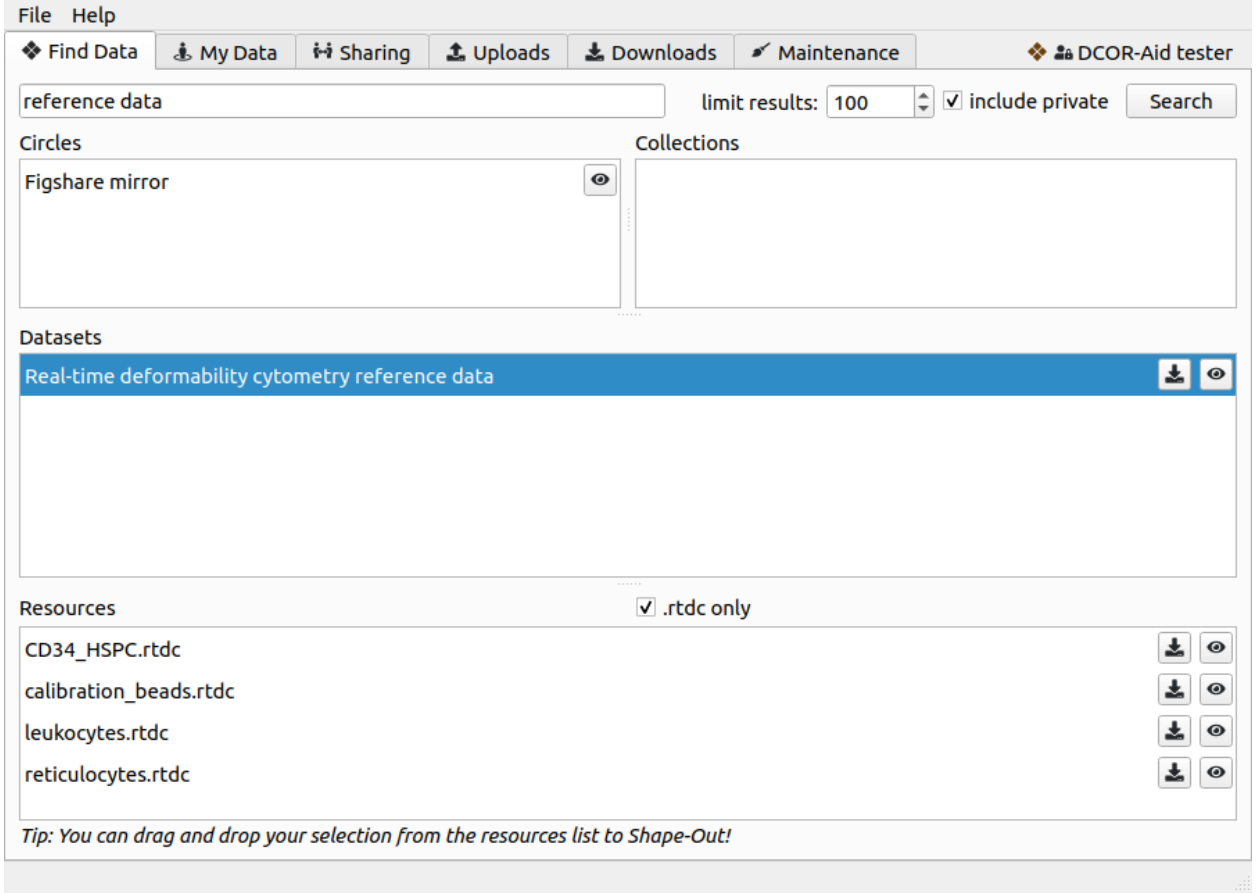
○ **Health facility**
If you are working in a health facility, you should use the DCOR instance hosted by your institution (to comply with patient data protection laws).

● **Playground**
You would like to see whether DCOR could fit into your analysis workflow. All data on dcor-dev.mpl.mpg.de are purged weekly.

○ **Anonymous**
You would only like to browse DCOR data.

○ Custom
You need to access a custom DCOR instance (typically used for DCOR development).

Fig. 2.1: The DCOR-Aid setup wizard guides you through the initial setup.



reference data

limit results: 100 ☑ include private

Circles: Figshare mirror

Datasets: Real-time deformability cytometry reference data

Resources ☑ .rtdc only

CD34_HSPC.rtdc

calibration_beads.rtdc

leukocytes.rtdc

reticulocytes.rtdc

Tip: You can drag and drop your selection from the resources list to Shape-Out!

Fig. 2.2: The search results in the *Find Data* tab can be filtered by circle and collection. The tool buttons allow you to download datasets and resources and to view them online.

2.1.3 Access via DCOR-Aid Python library

The DCOR-Aid Python library provides you with a convenient interface to the API. In principle, you are not limited to Python or DCOR-Aid, as DCOR is basically CKAN and thus uses the same API.

To initiate a connection with DCOR, run:

```
In [1]: import dcoraid

In [2]: api␣
↪= dcoraid.CKANAPI(server="dcor-dev.mpl.mpg.de",
␣
↪  ...:                     ␣
↪   api_key="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJqdGkiOiItNUVsLVBTZVdfZ3hMM2tKNnZXS0hWZUdsN011SnpMRlFRMHluNzdUanZqRnhLX3VNLTQyUHhsbVQwRl9yOGlZbkl0a
↪VfHEPXdEZKjCZOP4bO8cl0OiIxsvZZksWyQLl80UGbI")
   ...:

# check that everything works
In [3]: assert api.is_available()
```

Here, `server` is the DCOR instance you are connecting to and `api_key` is your personal access token that you need if you would like to access private data. You can omit `api_key` if you are only interested in public data (or if you don't have an account).

The `dcoraid.CKANAPI` class gives you full access to the underlying API. For instance, you could list all details of this dataset with:

```
In [4]: dataset_dict = api.get("package_show", id="figshare-7771184-v2")

# the first ten entries of the dataset dictionary
In [5]: for key in list(dataset_dict.keys())[:10]:
   ...:     print(f"{key:18s}: {dataset_dict[key]}")
   ...:
authors           : Philipp Rosendahl, Christoph Herold, Paul Müller, Jochen Guck
creator_user_id   : 60a214ed-a079-4334-b277-7b64c40ae675
doi               : 10.6084/m9.figshare.7771184.v2
id                : 89bf2177-ffeb-9893-83cc-b619fc2f6663
isopen            : True
license_id        : CC0-1.0
license_title     : Creative Commons Public Domain Dedication
license_url       : https://creativecommons.org/publicdomain/zero/1.0/
metadata_created  : 2024-04-20T22:45:39.708858
metadata_modified : 2024-04-20T22:48:02.914773

# all resource names in the dataset
In [6]: print([r["name"] for r in dataset_dict["resources"]])
['CD34_HSPC.rtdc', 'calibration_beads.rtdc', 'README.txt', 'leukocytes.rtdc',
↪'reticulocytes.rtdc']

# the first ten metadata entries of the first resource
In [7]: for key in list(dataset_dict["resources"][0].keys())[:10]:
   ...:     print(f"{key:31s}: {dataset_dict['resources'][0][key]}")
   ...:
```

(continues on next page)

(continued from previous page)

```
cache_last_updated             : None
cache_url                      : None
created                        : 2024-04-20T22:45:40.368694
dc:experiment:date             : 2017-02-09
dc:experiment:event count      : 112000
dc:experiment:run index        : 1
dc:experiment:sample           : HSC_apher_raw_APC
dc:experiment:time             : 15:13:04
dc:fluorescence:bit depth      : 16
dc:fluorescence:channel 3 name : 700/75
```

Note: Beware of the *dataset ambiguity*: On DCOR, a dataset (or package) contains a number of resources. You would call one of those resources a dataset in dclab. In other words, on DCOR a dataset consists of multiple RT-DC files while with `dclab.new_dataset()` you always ever only open one resource.

Another very useful tool in DCOR-Aid is the `APIInterrogator` class which sits on top of `CKANAPI` and, amongst other things, simplifies searching for datasets:

```
# instantiate APIInterrogator
In [8]: air = dcoraid.APIInterrogator(api)

# search for a dataset in a DCOR circle
In [9]: dbe = air.search_dataset(query="reference data",
   ...:                          circles=["figshare-import"])
   ...:

# the returned database extract (one hit)...
In [10]: len(dbe)
Out[10]: 1

# ...contains all metadata of the datasets matching the search query
In [11]: dbe[0]["name"]
Out[11]: 'figshare-7771184-v2'
```

Example: List all RT-DC resources for a DCOR circle

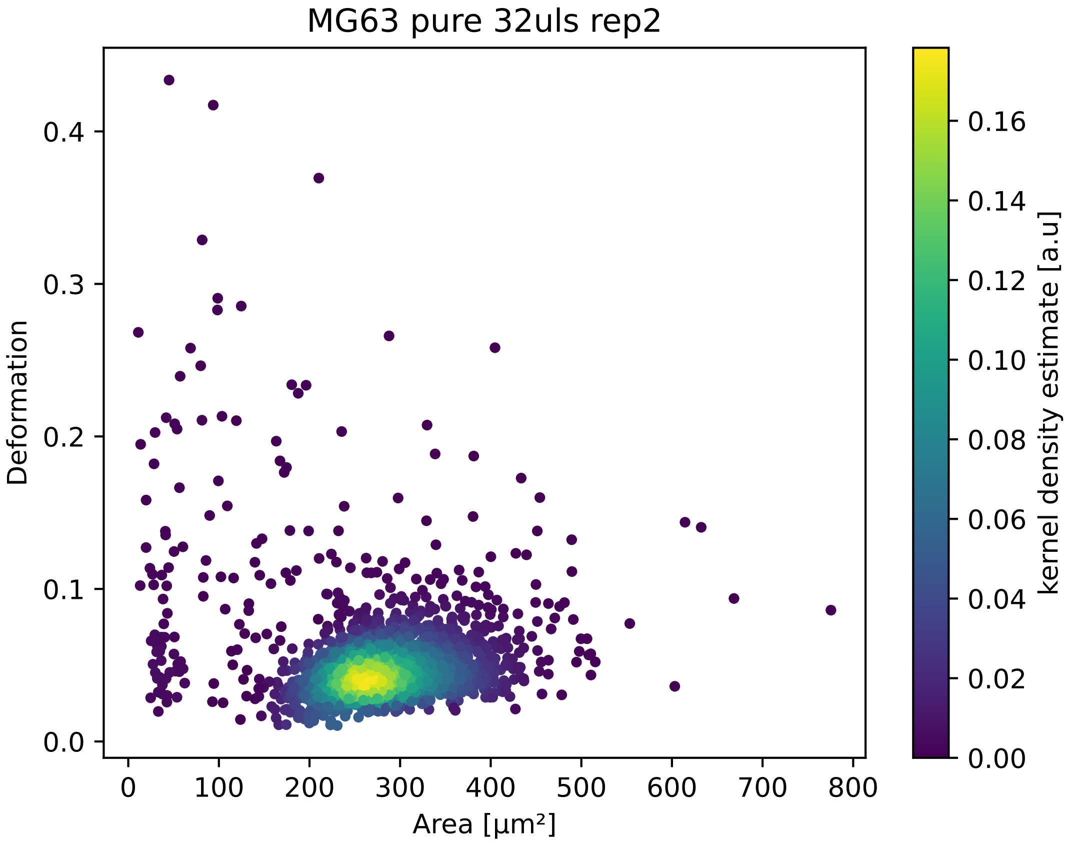
Let's say you are interested in all RT-DC data files in a DCOR circle, because you would like to run an automated analysis with dclab. The following script creates a list of IDs `resource_ids` with all RT-DC files in the Figshare mirror circle and plots one of the resources. For more information on how to access DCOR data with dclab, please refer to the dclab docs.

```
import dclab
import dcoraid
import matplotlib.pylab as plt

# name of the circle in question
circle_name = "figshare-import"

# initialize API (for private datasets, also provide `api_key`)
api = dcoraid.CKANAPI("dcor.mpl.mpg.de")
```

(continues on next page)

(continued from previous page)

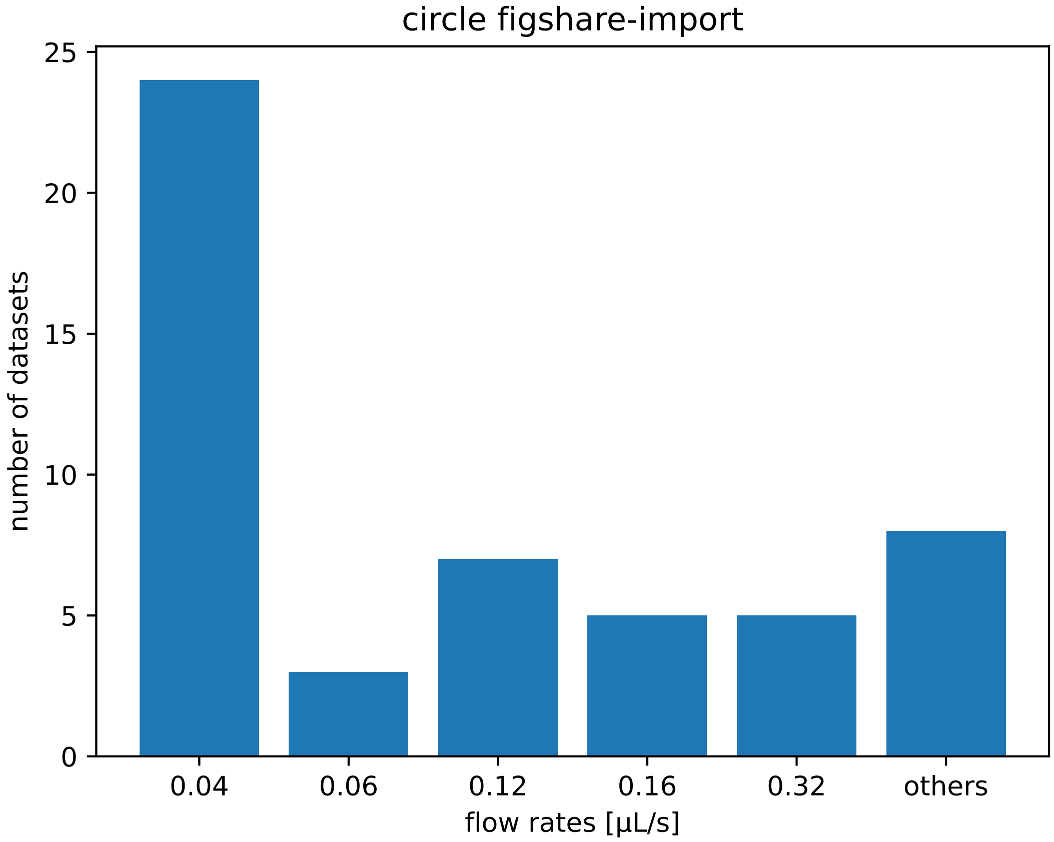
```
air = dcoraid.APIInterrogator(api)
# get a list of all datasets for `circle_name`
datasets = air.search_dataset(circles=[circle_name], limit=0)
# iterate over all datasets and populate our resources list
resource_ids = []
for ds_dict in datasets:
    # iterate over all resources of a dataset
    for res_dict in ds_dict["resources"]:
        # identify RT-DC data
        if res_dict["mimetype"] == "RT-DC":
            resource_ids.append(res_dict["id"])

# do something with one of the resources in dclab
with dclab.new_dataset(resource_ids[47]) as ds:
    kde = ds.get_kde_scatter(xax="area_um", yax="deform")
    ax = plt.subplot(111, title=ds.config['experiment']['sample'])
    sc = ax.scatter(ds["area_um"], ds["deform"], c=kde, marker=".")
    ax.set_xlabel(dclab.dfn.get_feature_label("area_um"))
    ax.set_ylabel(dclab.dfn.get_feature_label("deform"))
    plt.colorbar(sc, label="kernel density estimate [a.u]")
    plt.show()
```



Example: Order all resources of a DCOR circle according to flow rate

You may need to order your resources according to a certain metadata key. You can find all available metadata keys in the resource view in the DCOR web interface (scroll all the way down and click “show more”). In this example, we order all resources according to flow rate (the *“dc:setup:flow rate”* resource key).

```
import dclab
import dcoraid
import matplotlib.pylab as plt
import numpy as np

# name of the circle in question
circle_name = "figshare-import"

# dictionary with flow rates of interest
flow_rate_ids = {
    0.04: [],
    0.06: [],
    0.12: [],
    0.16: [],
    0.32: [],
    }

# list of flow rates that don't fit into the above dictionary
unsrt_ids = []

# initialize API (for private datasets, also provide `api_key`)
api = dcoraid.CKANAPI("dcor.mpl.mpg.de")
air = dcoraid.APIInterrogator(api)
# get a list of all datasets for `circle_name`
datasets = air.search_dataset(circles=[circle_name], limit=0)
# iterate over all datasets
for ds_dict in datasets:
    # iterate over all resources of a dataset
    for res_dict in ds_dict["resources"]:
        # identify RT-DC data
        if res_dict["mimetype"] == "RT-DC":
            flow_rate = res_dict.get("dc:setup:flow rate", np.nan)
            for fr in flow_rate_ids:
                if np.allclose(flow_rate, fr):
                    flow_rate_ids[fr].append(res_dict["id"])
                    break
            else:
                unsrt_ids.append((flow_rate, res_dict["id"]))

# plot some statistics
ax = plt.subplot(title=f"circle {circle_name}")
plt.bar([f"{fr}" for fr in flow_rate_ids] + ["others"],
        [len(flow_rate_ids[fr]) for fr in flow_rate_ids] + [len(unsrt_ids)])
ax.set_xlabel("flow rates [µL/s]")
ax.set_ylabel("number of datasets")
plt.show()
```


circle figshare-import
number of datasets
0
5
10
15
20
25
0.04
0.06
0.12
0.16
0.32
others
flow rates [µL/s]

2.1.4 Downloading data with wget

If you would like to download datasets, you can access it using the following URL

```
wget https://${SERVER}/dataset/${DATASET_ID}/resource/${RESOURCE_ID}/download/${RESOURCE_
↪NAME}
```

For private datasets, you would have to pass your API token

```
wget --header="Authorization: ${YOUR_API_KEY}" https://${SERVER}/dataset/${DATASET_ID}/
↪resource/${RESOURCE_ID}/download/${RESOURCE_NAME}
```

Example:

```
wget https://dcor.mpl.mpg.de/dataset/89bf2177-ffeb-9893-83cc-b619fc2f6663/resource/
↪fb719fb2-bd9f-817a-7d70-f4002af916f0/download/calibration_beads.rtdc
```

2.2 Uploading data to DCOR

2.2.1 Prerequisites

- DCKit: graphical toolkit for the management of RT-DC data (https://github.com/DC-analysis/DCKit/releases)
- DCOR-Aid: GUI for managing data on DCOR (https://github.com/DCOR-dev/DCOR-Aid/releases)

2.2.2 Data preparation with DCKit

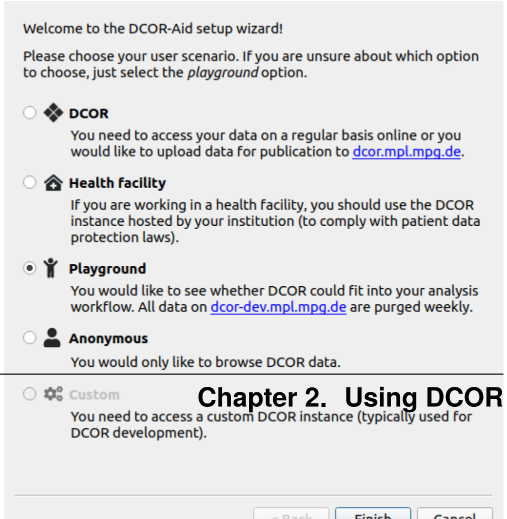
In many cases, you should not upload your experimental data right away to DCOR. There may be several reasons for that, such as missing metadata, uncompressed raw data, or log files that contain sensitive or unnecessary information (such as the user name of the person that recorded or processed the raw data). Please also note that DCOR only works with DC data in the HDF5 file format (.rtdc file extension).

DCKit to the rescue! In most cases, it is sufficient to to run your data through DCKit. Load the files in question, run the integrity check, complete or correct any missing or bad metadata keys and either convert the data to the .rtdc file format (for tdms data) or compress the data. You can verify that everything went as intended by running the integrity check for the newly generated files. If you are certain that you are not losing valuable information, you may also use the *repack and strip logs* option.

2.2.3 Data upload with DCOR-Aid

To upload your data to a DCOR instance, you first need to create an account. When you start DCOR-Aid for the first time, you will be given several options.

- If you select "Playground", DCOR-Aid will create a testing account at https://dcor-dev.mpl.mpg.de for you. All data on that **dev**elopment is pruned weekly. You can use the DCOR-dev instance for testing.
- If you select "DCOR", you will have to manually register at dcor.mpl.mpg.de and generate an API key for DCOR-Aid in the DCOR web interface.



Welcome to the DCOR-Aid setup wizard!

Please choose your user scenario. If you are unsure about which option to choose, just select the *playground* option.

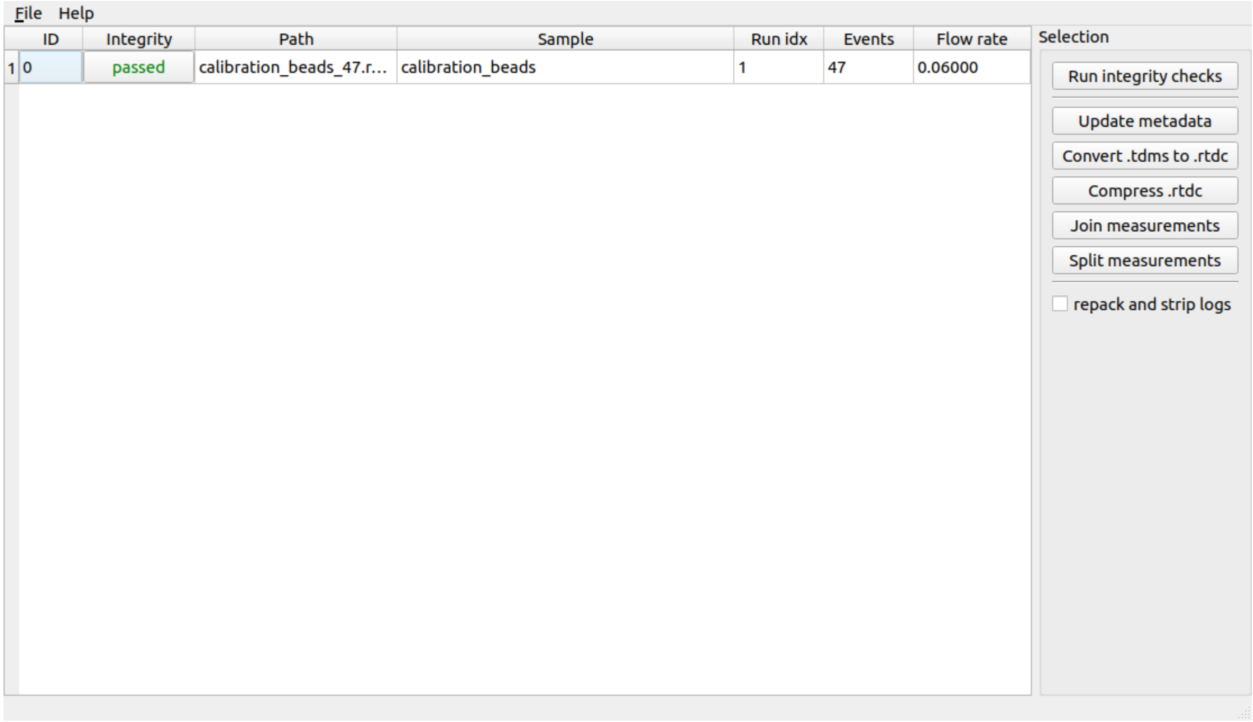
○ **DCOR**
You need to access your data on a regular basis online or you would like to upload data for publication to dcor.mpl.mpg.de.

○ **Health facility**
If you are working in a health facility, you should use the DCOR instance hosted by your institution (to comply with patient data protection laws).

● **Playground**
You would like to see whether DCOR could fit into your analysis workflow. All data on dcor-dev.mpl.mpg.de are purged weekly.

○ **Anonymous**
You would only like to browse DCOR data.

○ Custom
You need to access a custom DCOR instance (typically used for DCOR development).



	ID	Integrity	Path	Sample	Run idx	Events	Flow rate
1	0	passed	calibration_beads_47.r...	calibration_beads	1	47	0.06000

Fig. 2.3: DCKit user interface with one .rtdc file loaded that passed all integrity checks. DCKit can perform various tasks that are represented by the tool buttons on the right. Before uploading to DCOR, it is recommended to at least update the metadata such that the integrity checks pass.

You can always run the setup wizard again via the *File* menu to e.g. switch from “playground” to the production DCOR server.

Once DCOR-Aid is connected to a DCOR instance, go to the *Upload* tab. The *New manual upload* tool button directs you to the metadata entry and resource selection process. It is also possible to upload pre-defined upload tasks (see next section).

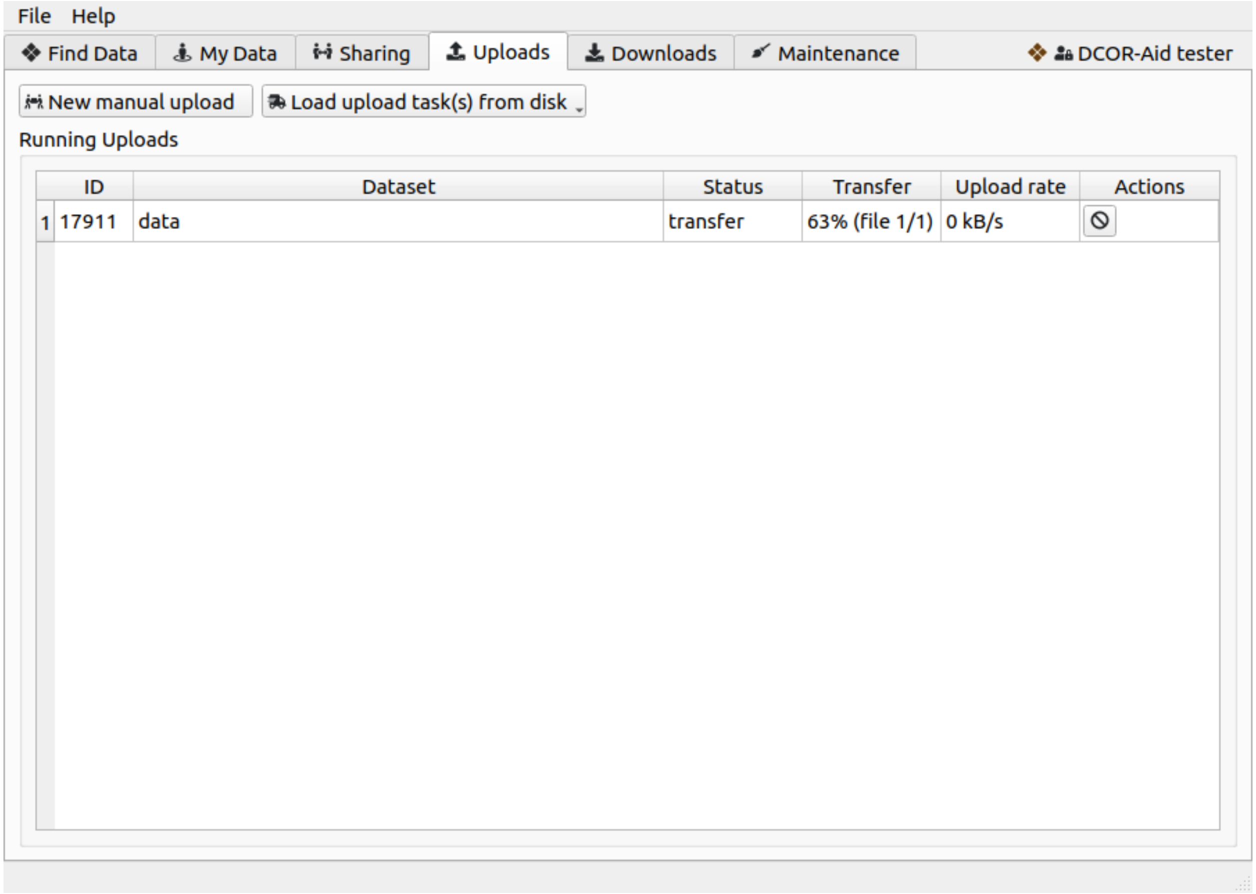
2.2.4 Generating DCOR-Aid upload tasks

If you need a way to upload many datasets in an automated manner you can make use of .dcoraid-task files. These files are essentially upload recipes that can be loaded into DCOR-Aid in the *Upload* tab via the *Load upload task(s) from disk* tool button.

The following script `upload_task_generation.py` recursively searches a directory tree for .rtdc files and generates .dcoraid-task files.

```
"""DCOR-Aid task creator

This script automatically generates *.dcoraid-task files recursively.
For each directory with *.rtdc files, an upload_job.dcoraid-task file
is generated. This task file can then be loaded into DCOR-Aid for the
actual upload. This script only serves as a template. Please go ahead
and edit it to your needs if necessary.
```

(continues on next page)



	ID	Dataset	Status	Transfer	Upload rate	Actions
1	17911	data	transfer	63% (file 1/1)	0 kB/s	⊘

Fig. 2.5: The upload tab gives you the option to manually upload datasets or to load auto-generated DCOR-Aid upload task (.dcoraid-task) files via the buttons at the top. Queued and running uploads are then displayed in the table below.

(continued from previous page)

```

Changelog
---------
2021-10-26
 - initial version
"""
import copy
import pathlib

import dclab
import dcoraid


#: Local directory to search recursively for .rtdc files
DATA_DIRECTORY = r"T:\Example_Data\Main_Directory"

#: List of file name suffixes of files to be included in the upload
#: (see :func:`dcoraid.CKANAPI.get_supported_resource_suffixes`)
DATA_FILE_SUFFIXES = [
    "*.ini",
    "*.csv",
    "*.tsv",
    "*.txt",
    "*.pdf",
    "*.jpg",
    "*.png",
    "*.so2",
    "*.poly",
    "*.sof",
    ]


#: Default values for the dataset upload
DATASET_TEMPLATE_DICT = {
    # Should the datasets by private or publicly visible (optional)?
    "private": False,
    # Under which license would you like to publish your data (mandatory)?
    "license_id": "CC0-1.0",
    # To which DCOR circle should the dataset be uploaded (optional)?
    "owner_org": "my-dcor-circle",
    # Who is responsible for this dataset (mandatory)?
    "authors": "Heinz Beinz Automated Upload",
}

#: Supplementary resource metadata
#: (see :func:`dcoraid.CKANAPI.get_supplementary_resource_schema`)
RSS_DICT = {
    "cells": {
        "organism": "human",
        "cell type": "blood",
        "fixed": False,
        "live": True,
```

(continues on next page)

(continued from previous page)

```
        "frozen": False,
        },
    "experiment": {
        "buffer osmolality": 284.0,
        "buffer ph": 7.4,
    }
}


def recursive_task_file_generation(path=DATA_DIRECTORY):
    """Recursively generate .dcoraid-task files in a directory tree

    Skips directories that already contain a .dcoraid-task file
    (This is important in case DCOR-Aid already imported that task
    file and gave that task a DCOR dataset ID).
    """
    # Iterate over all directories
    for pp in pathlib.Path(path).rglob("*"):
        if pp.is_dir():
            generate_task_file(pp)


def generate_task_file(path):
    """Generate the upload_job.dcoraid-task file in directory `path`

    A task file is only generated if the directory contains .rtdc
    files.
    """
    path = pathlib.Path(path)
    assert path.is_dir()

    path_task = path / "upload_job.dcoraid-task"
    if path_task.exists():
        print(f"Skipping creation of {path_task} (already exists)")
        return
    else:
        print(f"Processing {path}", end="", flush=True)

    # get all .rtdc files
    resource_paths = sorted(path.glob("*.rtdc"))
    # make sure they are ok
    for pp in copy.copy(resource_paths):
        try:
            with dclab.IntegrityChecker(pp) as ic:
                cues = ic.sanity_check()
                if len(cues):
                    raise ValueError(f"Sanity Check failed for {pp}!")
        except BaseException:
            print(f"\n...Excluding corrupt resource {pp.name}",
                  end="", flush=True)
            resource_paths.remove(pp)
```

(continues on next page)

(continued from previous page)

```
    # proceed with task generation
    if resource_paths:
        # DCOR dataset dictionary
        dataset_dict = copy.deepcopy(DATASET_TEMPLATE_DICT)
        # Set the directory name as the dataset title
        dataset_dict["title"] = path.name

        # append additional resources
        for suffix in DATA_FILE_SUFFIXES:
            resource_paths += path.glob(suffix)

        # create resource dictionaries for all resources
        resource_dicts = []
        for pp in resource_paths:
            rsd = {"path": pp,
                   "name": pp.name}
            if pp.suffix == ".rtdc":
                # only .rtdc data can have supplementary resource metadata
                rsd["supplements"] = get_supplementary_resource_metadata(pp)
            resource_dicts.append(rsd)
        dcoraid.create_task(path=path_task,
                            dataset_dict=dataset_dict,
                            resource_dicts=resource_dicts)
        print(" - Done!")
    else:
        print("\n...No usable RT-DC files!")


def get_supplementary_resource_metadata(path):
    """Return dictionary with supplementary resource metadata

    You will probably want to modify this function to your liking.
    """
    path = pathlib.Path(path)
    assert path.suffix == ".rtdc"
    supplements = copy.deepcopy(RSS_DICT)
    # Here you may add additional information, e.g. if you want
    # to add a pathology depending on the folder name of the
    # containing folder:
    #
    # if path.parent.name.count("BH"):
    #     supplements["cells"]["pathology"] = "long covid"
    #
    return supplements


if __name__ == "__main__":
    recursive_task_file_generation()
```

2.3 Sharing (private) data with others

This section is about dataset sharing. Whether you are working with private or public datasets, DCOR offers you several ways of sharing datasets.

2.3.1 Permission system

Public and private datasets

When uploading a dataset, you have the choice of *public* and *private* datasets. Public datasets can be accessed by anyone. Private datasets can only be accessed by members of the DCOR circle they were uploaded to. You can also create DCOR collections, which may contain datasets that belong to different circles. Every member of that collection has access to the private datasets therein. You can always make private datasets public, but not the other way around.

Circles: Sharing with colleagues

DCOR circles are the “home” of a dataset. When you upload a dataset, you have to specify a circle. A circle comprises one or more users who can be attributed to the same company, organization, research group, or laboratory. Every user can create circles and add other users to them. Users can have different roles within a circle, depending on their level of authorization to create, edit, and publish. All members of a circle have access to the private datasets of that circle.

Collections: Sharing for collaborators

You can use DCOR collections to organize datasets across circles. This could be to catalog datasets for a particular project or team, or on a particular theme, or as a very simple way to help people find and search your own published datasets. You can also use Collections to share private datasets with other users. All members of a collection have access to the private datasets in that collection.

Datasets: Simple sharing

It is also possible to share individual private datasets with other users.

Publications

Please note that circles and collections are in principle not citable. You may add new datasets to circles and collections and you may remove datasets from collections. You should always cite the datasets. If you would like to cite a dataset on DCOR, please read *Citing data on DCOR*.

2.3.2 Sharing data via DCOR-Aid

After you have *uploaded your datasets* via the *Uploads* tab, you can add individual datasets to a collection in the *My Data* tab.

Note: The *Sharing* tab is not implemented yet. Please manage members of collections and circles via the DCOR web interface.

2.4 Citing data on DCOR

Warning: Do not cite your data using DCOR collection IDs or DCOR circle IDs. Users may add new datasets to circles/collections or remove datasets from collections. You should always cite the datasets.

If you are using DCOR data for publication purposes, please make sure to properly cite the relevant dataset IDs. For instance, if you are using this dataset, you should use the ID `89bf2177-ffeb-9893-83cc-b619fc2f6663` specified on that page. You should also include the `dcor.mpl.mpg.de` domain.

If you wanted to include this dataset in a bibliography, you could cite it like this:

Philipp Rosendahl, Christoph Herold, Paul Müller, Jochen Guck (2020). *Real-time deformability cytometry reference data*, Deformability Cytometry Open Repository (DCOR) https://dcor.mpl.mpg.de/89bf2177-ffeb-9893-83cc-b619fc2f6663

Note: There are plans to add a DOI-service to DCOR, such that you can add DOI numbers to public datasets. Please let us know if you are interested in that feature.

CHAPTER

THREE

FREQUENTLY ASKED QUESTIONS

3.1 I need help with something that's not covered here

If you found a bug, have a feature request, or have any other question regarding DCOR, please open an issue in the DCOR repository that in your opinion best matches your query:

https://github.com/DCOR-dev

If in doubt, please create an issue in the DCOR-help repositry.

3.2 Why do I have to specify a creative commons license when uploading data?

This is a choice made by design. For all data that you upload, private or public, you must specify a creative commons license. This ensures that the data can be used by others in the future. DCOR is a free service hosted by an organization serving the public good. If you cannot meet those terms, then you still have the option to host your own DCOR instance (see *Self-Hosting*).

3.3 Can I upload a test dataset somewhere?

For all testing (or development) purposes, you can use the development instance at https://dcor-dev.mpl.mpg.de. All datasets on that server are purged on a regular basis, so feel free to play with it as you see fit.

3.4 What happens in the background when I upload a dataset?

For every DC file that you upload, DCOR performs the following tasks in the background:

- Generate a condensed version of the original data. This computationally expensive task is necessary to provide fast access to ancillary features, such as volume or principal inertia ratio, to Shape-Out 2 or dclab via the DCOR API. It also allows you to only upload the data you actually recorded (without any disadvantages).
- Generate a preview image and extract the configuration for visualization of the data in the web interface.
- The original file you uploaded is not changed. You can verify that the uploaded file is identical to the original file on your hard disk by comparing their sha256 sums. The sha256 sum is listed on each resource page under Additional Information.

Please note that, due to this data processing, it may take a few minutes until the preview is visible and the ancillary features are available via the DCOR API.

3.5 Why can't I add resources to existing datasets?

Not being able to modify a finalized dataset is part of the design of DCOR. The idea behind this design choice is that any user who uses a dataset (e.g. for a publication) will always work with the same resources. If you would be able to add resources (or even replace them), then this would impair reproducibility (or at least make things intransparent).

When you upload several resources in a dataset via DCOR-Aid, the DCOR-Aid first creates a *draft* dataset. When a dataset is a *draft*, resources may be uploaded and metadata may be edited. After the upload is complete, DCOR-Aid sets the state of the dataset (irreversibly) to *active*. In the active state, only the following actions are allowed:

1. setting the visibility of a private dataset to public
2. changing the license of a dataset to a less restrictive one

3.6 Why can't I delete datasets or resources?

Here, the same arguments about *prohibiting the addition of resources to active datasets* apply. Scientific data that once have been made available to the public should not be taken down again.

There are exceptions, for instance:

- The data are from a blood measurement of a patient at a hospital. The patient never gave consent for his/her data being made publicly available or the patient revoked a corresponding license.
- You have uploaded a dataset that contains sensitive patient data that could possibly be used to deanonymize the patient.

In such cases, it is possible to delete entire datasets. However, this procedure requires you to make a clear statement and proviee proof for your claims.

CHAPTER

FOUR

SELF-HOSTING

4.1 Installation

This section describes how to setup your own DCOR production instance.

4.1.1 Ubuntu and CKAN

Please use an Ubuntu 20.04 installation for any development or production usage. This makes it easier to give support and track down issues.

Before proceeding with the installation of CKAN, install the following packages:

```
apt update
# CKAN requirements
apt install -y libpq5 redis-server nginx supervisor
# needed for building packages that DCOR depends on (dclab)
apt install -y gcc python3-dev
# additional tools that you might find useful, but are not actually required
apt install -y aptitude net-tools mlocate screen needrestart python-is-python3
```

Install CKAN:

```
wget https://packaging.ckan.org/python-ckan_2.9-py3-focal_amd64.deb
dpkg -i python-ckan_2.9-py3-focal_amd64.deb
```

Note: Do *NOT* setup file uploads when following the instructions at https://docs.ckan.org. DCOR has its own dedicated directories for data uploads. The command `dcor inspect` will try to setup/fix that for you.

Follow the remainder of the installation guide at https://docs.ckan.org/en/2.9/maintaining/installing/install-from-package.html#install-and-configure-postgresql. Make sure to note down the PostgreSQL password which you will need in the initialization step.

Make sure to initiate the CKAN database with

```
source /usr/lib/ckan/default/bin/activate
export CKAN_INI=/etc/ckan/default/ckan.ini
ckan db init
```

DCOR by default stores all data on `/data`. This makes it easier to control backups and separate the CKAN/DCOR software from the actual data. If you have not mounted a block device or a network share on `/data`, please create this directory with

```
mkdir /data
```

4.1.2 Scratch Space

It is important that you have some scratch space of at least 100 GB available on you system, so that the *ckanext-dc_serve* extension can create temporary condensed datasets before uploading them to S3. By default, the cache is located at */data/tmp/ckanext-dc_serve* and is editable via the configuration option *ckanext.dc_serve.tmp_dir*.

4.1.3 Object Storage

You should use a cloud storage provider that you trust instead of setting this up yourself. If you know what you are doing (e.g. for testing) and would like to setup S3-compatible object storage yourself, you can use MinIO. On a Ubuntu/Debian machine, install the latest MinIO server like so:

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio_RELEASEDATE.0.0_amd64.deb
dpkg -i minio_RELEASEDATE.0.0_amd64.deb
```

This also installed the `minio` systemd service which we want to use. First, make sure that the user defined in the service:

```
systemctl show minio | grep User=
```

actually exists. You can add a system user via:

```
useradd -r minio-user
```

Then, create a file `/etc/defaul/minio` with the following content:

```
# Volume to be used for MinIO server (make sure minio-user has access).
MINIO_VOLUMES="/srv/minio"
# Use if you want to run MinIO on a custom port (console is the web interface).
MINIO_OPTS="--address :9000 --console-address :9001"
# Root user for the server.
MINIO_ROOT_USER=minio-root-user-account-name
# Root secret for the server.
MINIO_ROOT_PASSWORD=secret-password-for-minio-root-user
# set this for MinIO to reload entries with 'mc admin service restart'
MINIO_CONFIG_ENV_FILE=/etc/default/minio
```

Now you can enable and start the minio service:

```
systemctl enable minio
systemctl start minio
```

Create a "dcor" user (`http://minio.server.name:9001/identity/users/add-user`) with *readwrite* permissions and create an access key (via "Service Accounts") which you can then copy-paste to the `ckan.ini` configuration:

```
dcor_object_store.access_key_id = access-key-id
dcor_object_store.secret_access_key = secret-access-key
```

4.1.4 DCOR Extensions

Installation

Whenever you need to run the `ckan/dcor` commands or have to update Python packages, you have to first activate the CKAN virtual environment.

```
source /usr/lib/ckan/default/bin/activate
```

With the active environment, first install some basic requirements.

```
pip install --upgrade pip
pip install wheel
```

Then, install DCOR, which will install all extensions including their requirements.

```
pip install dcor_control
```

Background workers

DCOR comes with three job queues *dcor-short*, *dcor-normal*, and *dcor-long* for data processing after a resource is added to a dataset. The CKAN instance populates those queues and CKAN workers (e.g. via *ckan jobs worker dcor-short*) fetching and running the jobs in the background. The workers are run, like ckan itself, via *supervisor* and are defined via individual configuration files in */etc/supervisor/conf.d*. When you run *dcor inspect* (see next section), these files will be created with your approval.

Initialization

The `dcor_control` package installed the entry point `dcor` which allows you to manage your DCOR installation. Just type `dcor --help` to find out what you can do with it.

For the initial setup, you have to run the `inspect` command. You can run this command on a routinely basis to make sure that your DCOR installation is setup correctly.

```
source /usr/lib/ckan/default/bin/activate
dcor inspect
```

Testing

If you are setting up a development instance, then you might want to be able to run the DCOR tests. This step is not required if you are setting up an instance for production.

For testing purposes, you can use the DCOR vagrant box. It contains a full install of DCOR (including SOLR and object storage) and is updated regularly.

4.1.5 SSL

You have two options. If you server is reachable through the internet, you should use Let's encrypt (or a certificate from your organization) to set up SSL. If you are hosting your server on the intranet (clinics scenario), then you should create your own certificate and distribute it to your users

Creating an SSL certificate (Intranet only)

Start by creating your certificate (valid for 10 years):

```
openssl req -newkey rsa:4096 -x509 -sha256 -days 3650 -nodes -out fqdn.cert -keyout fqdn.key
```

where *fqdn* is your fully qualified domain name (FQDN) which maps to the server's IP address. Make sure to enter it in the dialog (otherwise use the IP address). This makes connection tests easier (e.g. if you only have SSH access to the machine and need to use SSH tunneling to connect to the CKAN instance by mapping its FQDN in the */etc/hosts* file to *127.0.0.1* on the testing client).

You may want to create an *encrypted access token* for your users.

Now proceed with the SSL configuration below, replacing "dcor.mpl.mpg.de" with your FQDN.

Configuring nginx (SSL and uWSGI proxy)

Encrypting data transfer should be a priority for you. If your server is available online, you can use e.g. Let's Encrypt to obtain an SSL certificate. If you are hosting CKAN/DCOR internally in your organization, you will have to create a self-signed certificate and distribute the public key to the client machines manually.

First copy the certificate to `/etc/ssl/private`:

```
cp dcor.mpl.mpg.de.cert /etc/ssl/certs/
cp dcor.mpl.mpg.de.key /etc/ssl/private/
```

Note: If dclab, Shape-Out, or DCOR-Aid cannot connect to your CKAN instance, it might be because the certificate in `/etc/ssl/certs/` does not contain the full certificate chain. In this case, just download the entire certificate chain using Firefox (right-lick on the shield symbol an look at the certificate - there should be a download option for the chained certificate somewhere) and replace the content of the .cert file with that.

Then, edit `/etc/nginx/sites-enabled/ckan` and replace its content with the following (change `dcor.mpl.mpg.de` to whatever domain you use):

Now, we need to modify the CKAN uWSGI file at `/etc/ckan/default/ckan-uwsgi.ini`:

```
[uwsgi]

; Since we are behind a webserver (proxy), we use the socket variant.
; We use HTTP1.1 (keep-alives)
http11-socket          =  127.0.0.1:8080
uid                    =  www-data
gid                    =  www-data
wsgi-file              =  /etc/ckan/default/wsgi.py
virtualenv             =  /usr/lib/ckan/default
module                 =  wsgi:application
```

(continues on next page)

(continued from previous page)

```
master                 =  true
pidfile                =  /tmp/%n.pid
; 10 hours for very long-lasting uploads
harakiri               =  36000
harakiri-verbose       =  true
; Restart workers after this many requests
max-requests           =  1000
; How long to wait before forcefully killing workers
worker-reload-mercy    =  30
; Delete sockets during shutdown
vacuum                 =  true
callable               =  application
; Disable threads only if not using threads and performance is critical
enable-threads         =  false
; Do not use multiple interpreters (since we only have one service)
single-interpreter     =  true
; Shutdown when receiving SIGTERM
die-on-term            =  true
; Fail to start if application cannot load
need-app               =  true
; Make sure all options in this file exist.
strict                 =  true

; Unfortunately, buffering the upload with nginx and then sending the upload
; to uWSGI does not work for some reason (uWSGI gets stuck when crunching the
; data). The intuitive choice would be to set this here to "1", but a look
; at the sources reveals that this should be set to the buffer size (2MB).
post-buffering         =  2097152
post-buffering-bufsize =  2097152

; Reduce or increase this number to limit POST requests. By default,
; the size of POST requests is unlimited.
limit-post             =  100000000000

; Set the number of workers to something > 1, otherwise
; only one client can connect via nginx to uWSGI at a time.
; See https://github.com/ckan/ckan/issues/5933
; In addition, use two threads per worker.
workers                =  4
; Use lazy apps to avoid the `__Global` error.
; See https://github.com/ckan/ckan/issues/5933#issuecomment-809114593
lazy-apps              =  true
; If we don't want to cache the files that users want to download
; (i.e. set `proxy_max_temp_file_size 0;` in nginx), then we have to
; set socket-timeout to a very large number (e.g. 7200).
; We may also want to increase this number if the storage location for
; resources has a low write speed (e.g. NFS). From the uWSGI sources,
; it looks like the default value is 4s.
socket-timeout         =  500
; (Note that we are serving CKAN via http11-socket behind nginx).
; Otherwise, downloads will fail with `uwsgi_response_sendfile_do() TIMEOUT !!!`,
; because the client cannot download the file from nginx as fast as
```

(continues on next page)

(continued from previous page)

```
; uWSGI can send the file to nginx. But in this case, we can really only
; have as many connections as we have workers.
; On the other hand, if we, set `proxy_max_temp_file_size 100000m;`
; in nginx, then all downloads will be cached by nginx. And nginx will
; handle all users. The purpose of setting `workers` to `4` in uWSGI
; is now only so that CKAN does not block for as long as it takes the
; system to copy the download from uwsgi to nginx's `proxy_temp_path`.
; In other words, CKAN will only be unresponsive if 4 downloads are
; started at the same time for as long as it takes the smallest download
; to be copied over the http socket from uWSGI to nginx.

; Custom logging
; disable logging in general (files easily get above 50MB)
disable-logging        =  true
; enable logging for a few specific cases
log-4xx                =  true
log-5xx                =  true
log-ioerror            =  true
; set the log format to match that of CKAN
log-date               =  %%Y-%%m-%%d %%H:%%M:%%S
logformat-strftime     =  true
logformat              =  %(ftime) uWSGI %(addr) (%(proto) %(status)) %(method) %(uri) =>
↪ %(size) bytes in %(msecs) msecs to %(uagent)
threaded-logger        =  true

; https://stumbles.id.au/how-to-fix-uwsgi-oserror-write-error.html
disable-write-exception = true
ignore-write-errors     = true
ignore-sigpipe          = true
```

4.1.6 Unattended upgrades

Unattended upgrades offer a simple way of keeping the server up-to-date and patched against security vulnerabilities.

```
apt-get install unattended-upgrades apt-listchanges
```

Edit the file */etc/apt/apt.conf.d/50unattended-upgrades* to your liking. The default settings should already work, but you might want to setup email notifications and automated reboots.

Note: If you have access to an internal email server and wish to get email notifications from your system, install

```
apt install bsd-mailx ssmtp
```

and edit `/etc/ssmtp/ssmtp.conf`:

Note that this is something different than CKAN email notifications.

In order for unattended upgrades to work properly: whenever updates are installed, make sure that *needrestart* automatically restarts the services by editing the file */etc/needrestart/needrestart.conf* and setting:

```
$nrconf{restart} = 'a';
```

4.1.7 Supervisor

Sometimes the ckan-uwsgi start job might take a little longer and the default (1s) is not long enough so supervisor becomes impatient. Edit the file `/etc/supervisor/conf.d/ckan-uwsgi.conf` and add `startsecs=60`.

4.1.8 Systemd

It is important that all services required for CKAN to run should be started before starting `supervisor`. This can be achieved by running `systemctl edit supervisor` and pasting the following config:

```
[Unit]
Requires=solr.service
After=solr.service
Requires=redis.service
After=redis.service
Requires=postgresql.service
After=postgresql.service

[Service]
Restart=always
RestartSec=20
```

If *solr* is slow when starting up, add this to its unit file `systemctl edit solr`:

```
[Service]
ExecStartPost=/bin/sleep 250
Restart=on-failure
RestartSec=10s
```

Afterwards run:

```
systemctl daemon-reload
```

4.2 Operations and maintenance

4.2.1 Creating an encrypted access token

Encrypted access tokens are used to safely transfer the SSL certificate and the user’s API Key from the server to the user. This is especially important in scenarios where self-signed SSL certificates are used (medical branding) and where users are not allowed to register on their own to prevent man-in-the-middle attacks.

An encrypted access token is an encrypted zip file with the suffix “.dcor-access” that contains the server’s SSL certificate “server.cert” and the user’s API key “api_key.txt”. DCOR-Aid can use such an access token to automatically setup the server connection.

Note: To create good passwords, you can use this command:

```
dd if=/dev/urandom bs=1M count=10 status=none | md5sum | awk '{ print $1 }'
```

Steps to create an access token:

1. create a CKAN user:

```
# set-up the CKAN environment
source /usr/lib/ckan/default/bin/activate
export CKAN_INI=/etc/ckan/default/ckan.ini
# create a user (use a good password)
ckan user add your_username
# obtain the API key (if this does not work, you have to login
# as that user and create an api key)
ckan user show your_username | grep apikey
# write the API key to a text file
echo 7c0c7203-4e25-4b14-a118-553c496a7a52 > api_key.txt
# copy the public SSL certificate to the current directory
cp /etc/ssl/certs/fqdn.cert ./server.cert
# creat the encrypted access token (use a good encryption passoword)
zip -e your_username.dcor-access api_key.txt server.cert
# cleanup
rm api_key.txt server.cert
```

You should send the file *your_username.dcor-access* to your user. Please send the encryption password of the access token via a different channel. Especially in the context of hospitals (i.e. data protection), this is critical.

4.2.2 Creating an encrypted database backup

The CKAN database may contain sensitive information, such as email addresses, which means that any backup should be encrypted. The following script should be self-explanatory:

```
#!/bin/bash
#
# Create an encrypted database backup on /data/encrypted_db_dumps.
# You have to import a private key with `gpg --import dcor_public.key`
# and trust it with `gpg --edit-key 8FD98B2183B2C228` (command 'trust').
# Then also make sure that the key id in the example below is correct.
#
# Put this script in /root/scripts, make it executable and add the
# following cron job:
#
# # create encrypted database backups every day
# 2 0 * * * root /root/scripts/encrypted_database_backup.sh > /dev/null
#
source /usr/lib/ckan/default/bin/activate
export CKAN_INI=/etc/ckan/default/ckan.ini
dcor encrypted-database-backup --key-id 8FD98B2183B2C228
```

4.3 Upgrading DCOR

4.3.1 DCOR only

Updating DCOR is done via the command:

```
dcor update
```

This will update all extensions to the latest release (if installed from PyPI) or to the latest commit (if installed from git repositories).

After each update, you should make sure that your installation is still set up correctly. The following command will check your configuration files (amongst other things):

```
dcor inspect
```

4.3.2 Upgrading CKAN/DCOR

If you would like to upgrade CKAN via a .deb package (recommended), you may have to install DCOR again (because the environment will be reset). First, follow the steps to upgrade CKAN from the CKAN docs:

1. Make sure your system is up-to-date and not running any outdated binaries.
2. Activate the virtual environment:

   ```
   source /usr/lib/ckan/default/bin/activate
   export CKAN_INI=/etc/ckan/default/ckan.ini
   ```

4. Shut down all services:

   ```
   systemctl stop supervisor
   systemctl stop nginx
   ```

5. Create a database backup:

   ```
   mkdir -p CKAN_updates
   DATE=$(printf '%(%Y-%m-%d)T\n' -1)
   CKANVER=$(python -c "import ckan; print(ckan.__version__)")
   sudo -u postgres pg_dump --format=custom -d ckan_default > CKAN_updates/ckan_$
   ↪{CKANVER}_${DATE}.dump
   ```

6. Check the CKAN Changelog to see if there are any incompatibilities. Making a patch release upgrade should not be problematic, but if you are planning to upgrade to a minor release, please be careful. You should probably also check the CKAN upgrade docs.

   Install the latest version of CKAN for your system:

   ```
   CKANMINOR=2.10
   CKANPATCH=2.10.4
   UBUNTURELEASE=$(lsb_release -cs)
   DLNAMESERVR=python-ckan_${CKANMINOR}-${UBUNTURELEASE}_amd64.deb
   DLNAMELOCAL=python-ckan_${CKANPATCH}-${UBUNTURELEASE}_amd64.deb
   mkdir -p CKAN_updates
   wget https://packaging.ckan.org/${DLNAMESERVR} -O CKAN_updates/${DLNAMELOCAL}
   ```

(continues on next page)

(continued from previous page)

```
apt-get -y remove python-ckan
rm -rf /usr/lib/ckan/
dpkg -i CKAN_updates/${DLNAMELOCAL}
```

7. Reactivate the environment:

```
deactivate
source /usr/lib/ckan/default/bin/activate
export CKAN_INI=/etc/ckan/default/ckan.ini
```

8. For the upgrade from CKAN 2.0 to 2.10, Solr must be installed (previous versions of Solr were installed via `apt`):

```
apt purge tomcat9 solr-tomcat
```

Then, install solr manually as described in (installing `openjdk-8-jdk-headless` is sufficient, you don't have to install `openjdk-8-jdk`). https://docs.ckan.org/en/2.10/maintaining/installing/solr.html?highlight=solr#installing-solr-manually (CKAN 2.10 only supports solr 8.x). Note that solr by default listens to tcp6 (IPv6). Thus, any setting in the ckan.ini file that uses *127.0.0.1* will not work - use *localhost* instead. To test solr, make sure that the following URL returns JSON data: `http://localhost:8983/solr/ckan/select/?q=*:*&rows=1&wt=json`

9. Install DCOR (either via *pip* or as described in the *development section*):

```
# might be necessary if pip is still broken
wget https://gitlab.gwdg.de/pmuelle3/ckan-release-mirror/-/raw/main/get-pip.py?
↪inline=false -O CKAN_updates/get-pip.py
# Make sure there is no conflict between setuptools and distutils
# (https://github.com/pypa/setuptools/issues/2993#issuecomment-1003765389)
export SETUPTOOLS_USE_DISTUTILS=stdlib
python CKAN_updates/get-pip.py
pip install --upgrade pip wheel
pip install --upgrade --force-reinstall dcor_control
```

10. Rerun rebranding scripts:

```
sed -i 's/ckan.locale_default=en_US/ckan.locale_default=en_GB/g' /etc/ckan/default/
↪ckan.ini
ckan dcor-theme-main-css-branding
ckan dcor-theme-i18n-branding
```

11. Make sure the configuration is intact (you may skip scanning for orphaned files):

```
dcor inspect
```

12. If the CKAN upgrade requires a database upgrade (see CKAN changelog):

```
ckan db upgrade
# This will take some time. For installations with many datasets, consider
# running it in a screen session:
ckan search-index rebuild
```

13. Finally start nginx and supervisor:

```
systemctl start nginx
systemctl start supervisor
```

4.4 Troubleshooting

- When setting up CKAN error email notifications, emails are sent for every file accessed on the server. Set the logging level to “WARNING” in all sections in `/etc/ckan/default/ckan.ini`.
- If you get the following errors in `/var/log/ckan/ckan-uwsgi.stderr.log`:

  ```
  Error processing line 1 of /usr/lib/ckan/default/lib/python3.8/site-packages/
  ↪ckanext-dcor-theme-nspkg.pth:

    Traceback (most recent call last):
      File "/usr/lib/python3.8/site.py", line 175, in addpackage
        exec(line)
      File "<string>", line 1, in <module>
      File "<frozen importlib._bootstrap>", line 553, in module_from_spec
    AttributeError: 'NoneType' object has no attribute 'loader'

  Remainder of file ignored
  ```

  Not sure what is causing this, but it was solved for me by editing the relevant .pth file. Add a new line after the first semicolon.

  From

  ```
  import sys, types, os;has_mfs = sys.version_info > (3, 8);p = os.path.join(sys._
  ↪getframe(1).$
  ```

  to

  ```
  import sys, types, os;
  has_mfs = sys.version_info > (3, 8);p = os.path.join(sys._getframe(1).$
  ```

  ```
  sed -i -- 's/os;has_mfs/os;\nhas_mfs/g' /usr/lib/ckan/default/lib/python3.8/site-
  ↪packages/ckan*.pth
  ```

- If you get import errors like this and you are running a development server:

  ```
  Traceback (most recent call last):
    File "/etc/ckan/default/wsgi.py", line 12, in <module>
      application = make_app(config)
    File "/usr/lib/ckan/default/src/ckan/ckan/config/middleware/__init__.py", line 56,
  ↪ in make_app
      load_environment(conf)
    File "/usr/lib/ckan/default/src/ckan/ckan/config/environment.py", line 123, in␣
  ↪load_environment
      p.load_all()
    File "/usr/lib/ckan/default/src/ckan/ckan/plugins/core.py", line 140, in load_all
      load(*plugins)
    File "/usr/lib/ckan/default/src/ckan/ckan/plugins/core.py", line 154, in load
  ```

(continues on next page)

(continued from previous page)

```
    service = _get_service(plugin)
  File "/usr/lib/ckan/default/src/ckan/ckan/plugins/core.py", line 257, in _get_
↪service
    raise PluginNotFoundException(plugin_name)
ckan.plugins.core.PluginNotFoundException: dcor_schemas
```

Please make sure that the ckan process/user has read (execute for directories) permission. The following might help, or you run UWSGI as root:

```
chmod a+x /dcor-repos/*
find /dcor-repos -type d -name ckanext |  xargs -0 chmod -R a+rx
chmod -R a+rx /dcor-repos/dcor_control
chmod -R a+rx /dcor-repos/dcor_shared
```

- If you are having issues with HDF5 file locking and are storing your data on a network file storage:

```
Traceback (most recent call last):
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/rq/worker.py", line 812,␣
↪in perform_job
    rv = job.perform()
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/rq/job.py", line 588, in␣
↪perform
    self._result = self._execute()
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/rq/job.py", line 594, in _
↪execute
    return self.func(*self.args, **self.kwargs)
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/ckanext/dcor_schemas/jobs.
↪py", line 27, in set_dc_config_job
    with dclab.new_dataset(path) as ds:
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/dclab/rtdc_dataset/load.py
↪", line 63, in new_dataset
    return load_file(data, identifier=identifier, **kwargs)
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/dclab/rtdc_dataset/load.py
↪", line 22, in load_file
    return fmt(path, identifier=identifier, **kwargs)
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/dclab/rtdc_dataset/fmt_
↪hdf5.py", line 194, in __init__
    self._h5 = h5py.File(h5path, mode="r")
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/h5py/_hl/files.py", line␣
↪424, in __init__
    fid = make_fid(name, mode, userblock_size,
  File "/usr/lib/ckan/default/lib/python3.8/site-packages/h5py/_hl/files.py", line␣
↪190, in make_fid
    fid = h5f.open(name, flags, fapl=fapl)
  File "h5py/_objects.pyx", line 54, in h5py._objects.with_phil.wrapper
  File "h5py/_objects.pyx", line 55, in h5py._objects.with_phil.wrapper
  File "h5py/h5f.pyx", line 96, in h5py.h5f.open
OSError: Unable to open file (unable to lock file, errno = 37, error message = 'No␣
↪locks available')
```

You have to disable file locking via the environment variable *HDF5_USE_FILE_LOCKING=’FALSE’*. The most convenient fix is to add the line:

```
export HDF5_USE_FILE_LOCKING='FALSE'
```

to */usr/lib/ckan/default/bin/activate*.

Also, you will have to set the environment variable for all configuration files (uwsgi and worker jobs in */etc/supervisor/conf.d/*.conf*):

```
# put this before the "command=" option.
environment=HDF5_USE_FILE_LOCKING=FALSE
```

Just to be sure, you could also add this to */etc/environment*:

```
HDF5_USE_FILE_LOCKING="FALSE"
```

- If uploads to DCOR fail and you are getting these errors in the nginx logs:

```
[crit] 983#983: *623 pwrite() "/var/lib/nginx/body/0000000001" failed (28: No space␣
↪left on device)
```

This means that your root partition does not have enough free space to cache uploaded files. A workaround is to move the data directly to the block storage on */data*. Add this in the nginx configuration file (*server* section):

```
client_body_temp_path /data/tmp/nginx 1 2;
```

and make sure that *www-data* has rw access to this directory.

- If your root partition is suddenly full, this might be due to the systemd journal in */var/logs*. You can free up space by running:

```
journalctl --vacuum-files=2
```

To add a general limit on how large the journal may become, edit the file */etc/systemd/journald.conf* and set:

```
SystemMaxUse=200M
```

It might also help to remove-purge the *snapd* package (Don't do this if you are using snaps, e.g. for certbot!):

```
apt purge snapd
rm -rf /snap
rm -rf /var/snap
rm -rf /var/lib/snapd
```

- Problems wih *OSError: [Errno 28] No space left on device* upon uploads of large files. The reason might be that uwsgi stores temporary files in /tmp. You could check this with:

```
(default) root@server:/# lsof / | grep "/tmp"
uwsgi      1301         www-data    7u    REG    0,28 2038633555 1304952 /tmp/
↪#1304952 (deleted)
uwsgi      1301         www-data   12u    REG    0,28 1558086333 1304953 /tmp/
↪#1304953 (deleted)
```

You could also check whether your CKAN installation is responsible for this (*df -h* shows less space than there should be) by restarting all services:

```
supervisorctl restart all
```

According to a PDF file that I found somewhere, uwsgi always stores its temporary files under */tmp*, a behavior that can be controlled via the environment variable *TMPDIR*. Thus, the solution is to edit the uwsgi supervisor file */etc/supervisor/conf.d/ckan-uwsgi.conf* and set this *TPMDIR* to something under */data*:

```
environment=SOMEOTHERVAR=FALSE,TMPDIR=/data/tmp/uwsgi
```

- If downloads of large resources are aborted by the server after a short time, this might be because nginx caches the download on the root partition which does not have enough free space. You have to specify a cache location with sufficient free space in */etc/nginx/sites-enabled/ckan* by uncommenting the line:

```
proxy_temp_path /data/tmp/nginx/proxy 1 2;
```

- If uploads fail with a timeout error message and in the logs you get:

```
OSError: timeout during read(57344) on wsgi.input
2021-09-07 09:20:43 uWSGI 127.0.0.1 (HTTP/1.0 500) POST /api/3/action/resource_
↪create => 0 bytes in 8644 msecs to DCOR-Aid/0.6.4
```

that probably means that the socket-timeout value for uWSGI is too low. A reason for that could be e.g. that the resources are written to a location with low write speed (e.g. NFS). A solution is to add the socket-timeout to */etc/ckan/default/ckan-uwsgi.ini*:

```
socket-timeout     = 7200
```

- If uploads fail with the following error message in the ckan-uwsgi logs:

```
2021-09-08 18:46:16 - [uwsgi-body-read] Error reading 6563 bytes. Content-Length:␣
↪15428164609 consumed: 2150065757 left: 13278098852 message: Client closed␣
↪connection

[...]

OSError: error during read(8192) on wsgi.input
```

The you probably have to disable proxy-buffering in nginx.

- If you are getting RuntimeErrors in the CKAN logs on startup:

```
RuntimeError: CKAN config option not found: /usr/lib/ckan/default/src/ckan/ckan.ini
```

This is not a big problem, but to resolve it, you can add the *CKAN_INI* to the supervisor environment variable in */etc/supervisor/conf.d/ckan-uwsgi.conf*:

```
environment=SOMEVAR=FALSE,CKAN_INI=/etc/ckan/default/ckan.ini
```

- If on CKAN>=2.10.1 you are getting errors about not being able to connect to SOLR on startup, such as:

```
requests.exceptions.ConnectionError: HTTPConnectionPool(host='localhost',␣
↪port=8983): Max retries exceeded
  with url: /solr/ckan/select/?q=%2A%3A%2A&rows=1&wt=json (Caused by␣
↪NewConnectionError(
  '<urllib3.connection.HTTPConnection object at 0x7f18ec3f4310>: Failed to␣
↪establish a new connection:
  [Errno 111] Connection refused'))
2023-09-19 18:24:48,312 WARNI [ckan.lib.search] Problems were found while␣
```

(continues on next page)

(continued from previous page)

```
↪connecting to the SOLR server
pysolr.SolrError: Failed to connect to server at http://localhost:8983/solr/ckan/
↪select/?q=%2A%3A%2A&rows=1&wt=json:
  HTTPConnectionPool(host='localhost', port=8983): Max retries exceeded with url:
  /solr/ckan/select/?q=%2A%3A%2A&rows=1&wt=json (Caused by NewConnectionError('
↪<urllib3.connection.HTTPConnection
  object at 0x7f18ec3f4310>: Failed to establish a new connection: [Errno 111]␣
↪Connection refused'))
```

This means that supervisor is starting before SOLR (or at the same time). The solution is to edit the supervisor systemd unit via:

```
systemctl edit supervisor
```

and add the SOLR depenendency like so:

```
[Unit]
Requires=solr.service
After=solr.service
```

CHAPTER

FIVE

DCOR DEVELOPMENT

This section describes how to setup a DCOR development system (CKAN + DCOR extensions).

5.1 Ubuntu and CKAN

We recommend to setup a virtual machine for development. It also works with docker, but currently not out of the box (see https://github.com/ckan/ckan/issues/5572). Otherwise, the installation instructions are identical to those in the *self-hosting section*.

5.2 DCOR Extensions

5.2.1 Installation

This part differs from the installation for production. We want to have the DCOR extensions installed in editable mode.

Let's first set up our development environment. The `dcor_control` package comes with a convenient `dcor develop` command. This command will create the `/dcor-repos` directory, clone all DCOR-related repositories into it and install each of them in editable mode.

Note: If you have already installed all packages in editable mode from GitHub repositories, a simple *dcor update* will only update those (regardless of where they are located in the file system).

```
source /usr/lib/ckan/default/bin/activate
pip install dcor_control
dcor develop
```

5.2.2 Initialization

Please follow the *initialization steps for self-hosting*.

5.3 Load some test data into the database

You can test the basic functionalities of your DCOR installation by importing these publicly available datasets from figshare:

```
ckan import-figshare
```

5.4 robots.txt

If you don't want bots to index you site, add the following line to the server section in `/etc/nginx/sites-enabled/ckan` (right before `location / { [...]`):

```
location = /robots.txt { return 200 "User-agent: *\nDisallow: /\n"; }
```

5.5 Important commands

5.5.1 System

Restart CKAN

```
supervisorctl reload
```

Find out what went wrong in case of internal server errors:

```
supervisorctl status
tail -n500 /var/log/ckan/ckan-uwsgi.stderr.log
```

5.5.2 CLI

If you are using the CKAN or DCOR CLI, activate environment and set `CKAN_INI`.

```
source /usr/lib/ckan/default/bin/activate
export CKAN_INI=/etc/ckan/default/ckan.ini
```

User `ckan --help` and `dcor --help` to get a list of commands. E.g. to list all jobs, use

```
ckan jobs list
```

To reset the CKAN database and search index:

```
dcor reset
```

CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search