# Technical documentation

# How to install?

## Bare metal

You can download the entire code base from github, make a deployment of it and put in on your server. (see <https://flask.palletsprojects.com/en/3.0.x/deploying/>). Of course, you could use the current implementation as is, running “python3 -m swagger\_server”, but this starts a development server.

## Docker

There is also a docker available with the content of the github at the docker hub (edwinvandenbelt/nl\_profiel\_cds\_m). To run it, execute “docker run --name test -p 8080:8080 -v [local path]/:/usr/src/app/ edwinvandenbelt/nl\_profiel\_cds\_m”.

## Kubernetes

The kubernetes file is included in the folder ‘kubernetes’ on github. Of course, you must look at the host paths, but when you’re using K8, you should be familiar with working the volumes. In these volumes the import files should be placed.

# How to give access to read endpoints?

## Generate token

In the export files, the municipality should be referred to (look at the examples in the ‘import’ folder). This name or ID (it doesn’t matter, use whatever you prefer) must be taken and copied in this website, in the field ‘name’. <https://jwt.io/#debugger-io>

Afbeelding met tekst, schermopname, software, Webpagina

Automatisch gegenereerde beschrijving

Before copying and distributing the endcoded JWT, you must make up a secret word. You have to place it in the signature.

Afbeelding met tekst, schermopname, software, Webpagina

Automatisch gegenereerde beschrijving

The encoded token (on the left side) can be distributed to the data-consumer. If you have multiple municipalities to provide data for, you have to make multiple JWT-tokens, just by replacing the ‘name’. This secret word must also be in the config file (see next chapter).

# How to configure?

## Config file

The config file contains a few key-value pairs:

|  |  |
| --- | --- |
| Jwt\_secret | This is needed to decrypt the incoming requests, to authenticate them and to decipher the municipality that’s making the request (or a platform in their behalf, see also ‘generating tokens’). |
| Allowed\_tokens | A list of allowed IDs/names of the registered municipalities. These should match the values you’ve provided in the ‘name’ field (see generating tokens). |
| Filename\_vehicles | The file path containing the CSV of the vehicles |
| Filename\_vehicles\_status | The file path containing the CSV of the vehicles’ states |
| Filenames\_areas | A list of filenames with geojsons, per municipality a single geoJSON |
| Directoryname\_trips | A directory path where the CSV files of the trips can be found, per hour. |
| Trip\_retention\_days | The number of days the trips should be served by the app. (required for the /admin/clear) |
| Provider\_id | Your ID, as registered (just create a new entry!) in <https://github.com/openmobilityfoundation/mobility-data-specification/blob/main/providers.csv> |

Especially when using docker or Kubernetes, don’t change the file and directory names.

## Export files from production

The export scripts can be created and scheduled on your own production system.

We advise to schedule the status-export every 5 minutes, the trips can be exported once a day. If there are changes in the fleet, you must export the vehicles file, to stay consistent (as long as vehicles are used in the other endpoints, they must be in this file as well).

The exported CSVs should match the example files (same data, same order of columns). In the example files there is a header, but this is not required.

## Import files

The import can be done at any time, using the /admin/import endpoint. It can also be separately done per file:

* /admin/import/vehicles
* /admin/import/statusses
* /admin/import/areas
* /admin/import/trips

These endpoints you shouldn’t expose externally to the internet.

## Clear data

When the app is starting, it reads all the imports automatically, and can pick up new imports using the endpoint(s) described above. When the app stays active, data normally won’t be cleared automatically. You have to call the endpoint /admin/clear, it will clear all trips older than [trip\_retention\_days, see the config-section].

This endpoint you shouldn’t expose externally to the internet.

## Scheduled processes

A typical installation should have a schedule like this:

* Per 5 minutes:
  + export statuses (batch process, a simple example in github folder scripts)
  + import statuses (/admin/import/statusses)
  + export vehicles (only when changed, batch process)
  + import vehicles (/admin/import/vehicles)
* Per hour (or day):
  + export trips (batch process)
  + import trips (/admin/import/trips)
  + clear data (/admin/clear)

Whenever one of the areas have changed, you have to replace it as well, and call the /admin/import/areas (no need to schedule).