QUOREM

Mike Hall

Sep 09, 2022

CONTENTS:

CHAPTER

ONE

OVERVIEW OF QUOREM

QUOREM is an open-source web server for storing microbial ecology data. The primary goal of the QUOREM project is to make organizing and analyzing microbial ecology data easier for small research groups.

QUOREM runs on a computer running Linux and Python, and can be run as a development server and accessed locally, or hosted on a network-accessible computer with Apache to provide access to provide more robust access to distributed groups. Source code is available on GitHub at: https://github.com/mwhall/QUOREM

1.1 QUOREM Features

- Drag-and-drop upload of spreadsheets and QIIME2 artifact files
- Stores data about ASVs, OTUs, and taxonomic groups:
  - Taxonomic classifications
  - Abundances
  - Related biological sequences
  - Trees
- Keeps your samples and their metadata organized
- Download stored artifacts and metadata in convenient formats to take your analysis further

1.2 Ways to Use QUOREM

Web: A simple web interface helps users search, download, and visualize their microbial ecology data. This is suitable for all users. The web interface can easily be run as a test server on a local machine, or with a bit more work as a proper web server on the internet with appropriate security configuration.

Code: Objects in the QUOREM database can be accessed directly with Python code through the Django ORM. JupyterHub gives trusted users the ability to access samples, features, metadata, trees, tables, etc. programmatically. This gives a powerful query and retrieval interface for coders.

1.3 Objects

On a QUOREM server, data are organized as objects that are accessed through an [Object-relational mapping (ORM)](https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping) provided by the [Django project](https://www.djangoproject.com/).

Sample A sample is typically biological material taken from an environment, or something derived from a sample. For example, soil taken from a forest may be a sample, while replicates made or sequencing runs based on that material may be derived samples.

Feature A feature is anything that has measures that are tracked across samples. More specific examples may be amplicon sequence variants (ASVs), operational taxonomic units (OTUs), functional genes, or metabolites.

Step A step represents a transformation on either a physical sample or a data file.

Process A process is a set of linked steps and parameters. For example, a set of QIIME2 commands would be a process.

Result A result is a set of values from a given computational or wet-lab step. For example, a QIIME2 artifact is considered a result and contains many values that are imported to QUOREM, such as abundances or taxonomic classifications.

Analysis An analysis is any time a process is run, i.e., it is a specific instantiation of a set of steps. For example, running a denoising pipeline would be one analysis, while running an OTU clustering pipeline on the same data would be a second analysis.

Investigation An investigation is the motivation for an analysis.

Value A value is a piece of data that can be attached to any of the other objects, including multiple types and multiple of the same type. For example, a taxonomic classification is a piece of data that is attached to a feature and a result.

Each of these objects are connected to one another through the ORM, allowing for data from one object to be retrieved through other objects, as long as those objects are connected.

CHAPTER

TWO

INSTALLATION

QUOREM is a series of servers (web, process, database), so installation is complex on existing systems. The web server is powered by Django, the process server Redis and Celery, and the database Postgresql. It can be run pretty effectively through the Docker container system, but a native install has some advantages. For trying out QUOREM or for machines that will only have local access (not on the web), the test server is sufficient. For servers with domain names pointing at them, an example of production deployment using Apache2 is provided.

To start, you must download a copy of the repository. You can do this with the `git` or `gh` utilities:

```
git clone git@github.com:mwhall/QUOREM.git
```

or

```
gh repo clone mwhall/QUOREM
```

2.1 Quickstart Development Mode with Docker

Install the Docker engine using Docker's most up-to-date installation instructions: https://docs.docker.com/engine/install/

These instructions will only work to set up a server accessible by `localhost` or `127.0.0.1` from the Docker host machine. To deploy to an outside network, see the Production Deployment section below.

Make a copy of the *docker/example_secrets.env* file outside of the repository and **change the default passwords and secret key** and other settings, if desired:

```
cp docker/example_secrets.env ~/quorem_secrets.env
#Edit any desired settings in ~/quorem_secrets.env
```

Because we are setting up a database, cache, and web server as separate images, we'll be using `docker compose` from within the QUOREM project's root directory (where docker-compose.yml is located):

```
sudo docker compose --env-file ~/quorem_secrets.env build
```

The build process installs a QIIME2 environment for QUOREM to utilize, so it can take 20 minutes to complete on the first build.

If using `sudo` with Docker, root needs access to the launch entrypoint for Django:

```
chmod o+x docker/django_entrypoint.sh
```

To start the Docker up in the future use the `up` subcommand:

```
sudo docker compose --env-file ~/quorem_secrets.env up
```

Note: The database files will be in /docker/persistence/postgresql/ but this folder **will be owned by root on your host system**. If you have Docker access but not root access on your host system, this could be a problem for you.

List the running containers with `sudo docker ps`, and find the name of the container for quorem_django (likely quorem-django-1). You can attach to a bash shell the running container with `sudo docker exec -it quorem-django-1 bash` and after activating the QUOREM/QIIME2 conda environment with `conda activate quorem`, you have full access to Django's management command line utilities through `python manage.py`.

2.2 Native System Install

2.2.1 Install non-Conda dependencies

A number of packages must be installed at the system level, which typically requires root access. On an Ubuntu server, you can install the required packages with `apt`:

```
sudo apt install tzdata gcc-multilib g++-multilib curl graphviz apache2 apache2-dev␣
↪postgresql celery redis-server
```

2.2.2 Install miniconda

Miniconda is an environment manager that enables the creation of virtual environments. This keeps the many software dependencies of QUOREM safely isolated from your system's other versions. The gist is below butfull information is available at: https://docs.conda.io/en/latest/miniconda.html

```
curl -LO "http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh"
chmod u+x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh
```

After you have configured the installation using the interactive prompts, you will create and activate a conda environment for QUOREM with the appropriate Python version:

```
conda update -y conda
conda create --name quorem python=3.8.12
conda activate quorem
```

Next, you must install QIIME2 in this conda environment:

```
conda env update --file scripts/qiime2/qiime2-2022.8-py38-linux-conda.yml
```

2.2.3 Set up Postgresql Database

Configure the Postgresql database. We do this by setting environment variables (see docker/example_secrets.env). You should keep your QUOREM deployment's environment file out of version control to avoid accidental pushing or overwriting.

```
POSTGRES_USER=postgres
POSTGRES_DB=quoremdb
POSTGRES_PASSWORD=abcdefg12345
```

```
sudo -u postgres bash -c "psql -c "CREATE USER ${POSTGRES_USER} WITH PASSWORD '$
↪{POSTGRES_PASSWORD}';""
sudo -u postgres createdb --owner=${POSTGRES_USER} ${POSTGRES_DB}
```

2.2.4 Django Configuration

We need a secret key for our Django installation, and one can be generated from https://djecrety.ir/:

```
DJANGO_SECRET_KEY="@v-n8hwx!@@jex(jqr-w^94^#_=%ub3ypd#*epx1&-rnv@@qj@"
```

Make sure to save these environment variables, as they must be set in the shell whenever QUOREM is run.

In the `quorem/settings.py` file, some default strings that are needed for the Docker install must be changed for a full system install:

Line 30: If your QUOREM server is using a qualified domain name or a static IP instead of `localhost` for remote access, either the domain or IP must be added to the `ALLOWED_HOSTS` list.

Line 102: `CELERY_HOSTNAME` must be set to `127.0.0.1`

Line 148: `HOST` must be set to `localhost` for Postgresql.

Lines 181-186: (optional) Set up e-mail credentials to allow QUOREM to send password and account e-mails to users.

Finally, some Django commands must be run to set up the web server:

```
python manage.py makemigrations
python manage.py migrate
python manage.py collectstatic
python manage.py initialize
```

Once these have completed successfully, you must make a superuser account to approve any new users:

```
python manage.py createsuperuser
```

You can now start the Django test server with:

```
python manage.py runserver
```

This server works very well for local, single-user applications. Launching `127.0.0.1` in your web browser should bring up your new QUOREM instance. After signing up, be sure to log in with your superuser account and check the `Has Access` checkbox at `127.0.0.1/admin/`.

2.3 Production Deployment

In this section, we describe the general steps to tighten up configuration to allow secure remote access over the web to a QUOREM instance. We'll use the Apache2 webserver with the `mod_wsgi` plugin, a recommended approach for Django apps. This is a finnicky, often error-prone procedure. There are many ways to secure a production server, and this is one example. Report any issues or struggles to: https://github.com/mwhall/QUOREM/issues

First, in your QUOREM conda environment, ensure you have the `mod_wsgi` package.

```
pip install mod_wsgi
```

It is _very_ important that you install this via `pip` in your conda environment. The `mod_wsgi` package used by Django and Apache must be the same version of Python as the other packages, which is typically not your system-level Python installation.

Find the location of your `mod_wsgi` compiled library with:

```
mod_wsgi-express module-config
```

This will return two lines, but only the `LoadModule` line is needed. It should look something like (but may not be exactly):

```
LoadModule wsgi_module "/home/quorem/miniconda3/envs/quorem/lib/python3.8/site-packages/
↪mod_wsgi/server/mod_wsgi-py38.cpython-38-x86_64-linux-gnu.so"
```

Copy this line and with your favourite editor (and `sudo`), edit the Apache2 configuration file at `/etc/apache2/sites-available/000-default.conf`. The `LoadModule` line should go first, outside of any `<VirtualHost>` tags.

Inside the `<VirtualHost>` tag, set `ServerName` to your server's domain name, and `DocumentRoot` to the location of your QUOREM repository (e.g., `/home/quorem/QUOREM/`).

Now it is time to run Certbot to get a Let's Encrypt certificate for SSL (secure web browsing) connections with your server. It will create a certificate and automatically modify your Apache2 configuration to forward your non-secure HTTP connections through SSL HTTPS encryption. *If you do not secure your server, all logins and data (including passwords!) sent and received will visible to those monitoring your traffic. ALWAYS SECURE YOUR TRAFFIC.* The instructions are available in full at: https://certbot.eff.org/instructions?ws=apache&os=ubuntufocal

This process, if successful, will have modified your `000-default.conf` and created a new `000-default-le-ssl.conf` file in the same `/etc/apache2/sites-available` directory. Once again with `sudo`, edit this new file. Inside the `<VirtualHost>` tags, add the following lines, but be sure to *replace the directory names as appropriate* by replacing `/home/quorem/QUOREM` with the path to your QUOREM repository directory and `/home/quorem/miniconda3/envs/quorem/` to the path of the conda environment created earlier:

```
WSGIProcessGroup quorem
WSGIDaemonProcess quorem python-path=/home/quorem/QUOREM/ python-home=/home/quorem/
↪miniconda3/envs/quorem/ user=quorem group=quorem
WSGIScriptAlias / /home/quorem/QUOREM/quorem/wsgi.py application-group=%{GLOBAL} process-
↪group=quorem

Alias /static /home/quorem/QUOREM/staticfiles
Alias /data /home/quorem/QUOREM/uploaddata
<Directory /home/quorem/QUOREM/staticfiles>
    Require all granted
</Directory>
<Directory /home/quorem/QUOREM/uploaddata>
    Require all granted
```

(continues on next page)

(continued from previous page)

```
</Directory>
<Directory /home/quorem/miniconda3/envs/quorem>
    Require all granted
</Directory>

<Directory /home/quorem/QUOREM/quorem>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>
```

Finally, restart your Apache2 server with this new configuration:

```
sudo systemctl restart apache2
```

If there are any errors (especially if navigating to your domain produces "Internal Server Error"), you can start debugging by looking at the Apache2 logs at `/var/log/apache2/error.log`.

CHAPTER

THREE

DEVELOPMENT GUIDE

This section describes important files and useful functions for developing with QUOREM.

3.1 Development Resources

QUOREM is built on powerful open-source tools and their documentation extends our own. Within QUOREM, Django's ORM can be used to perform complex data queries on data input from QIIME2, so these are particularly helpful sets of external documentation. QUOREM makes heavy use of Django's QuerySets.

QIIME2 https://docs.qiime2.org/2022.2/

Django https://docs.djangoproject.com/en/4.0/

django-filter https://django-filter.readthedocs.io/en/stable/

Plotly https://plotly.com/python-api-reference/

scikit-learn https://scikit-learn.org/

3.2 Project Structure Walkthrough

```
QUOREM/
|      manage.py
|
└───── quorem/
|   |    settings.py
|   |    urls.py
|   |
|   └── templates/
|
└───── db/
    |   artifacts.py
    |   spreadsheets.py
    |   tasks.py
    |   plot.py
    |   ml.py
    |
    └── models/
    |
    └── forms/
```

(continues on next page)

(continued from previous page)

```
    |
    └── views/
```

Django projects are split into “apps”, which are represented by subdirectories. QUOREM’s apps are: *quorem*, *db*, *landingpage*, *accounts*. The *landingpage* and *accounts* are separated to enable easier customization of these aspects without affecting other areas.

3.2.1 quorem/

This app contains the main server configuration, including all settings and routing.

settings.py Django file that contains all server-specific configurations (including passwords!). Don’t let secrets in this file get onto the internet.

urls.py Django file that maps server URLs onto Django Views (defined in the *db/* app principally).

3.2.2 db/

This app contains all of the database code.

artifacts.py Code for processing and parsing QIIME2 artifacts as server input.

spreadsheets.py Code for processing and parsing different spreadsheet formats as server input.

tasks.py Functions that are wrapped by Celery to be asynchronous tasks. These are typically functons that take too long in a web context, so the user launches the task and returns to get the result.

plot.py Plotting functions that support database visualization.

ml.py Machine learning functions for analyzing data in QUOREM.

models/ This directory contains the [Django Models](https://docs.djangoproject.com/en/4.0/topics/db/models/) that define QUOREM’s database schema. Each of the major *Object* concepts (Analysis, Process, Step, Result, Feature, Sample, Investigation) and their connections to one another are defined in these files. These Models all have useful methods that extend their utility and provide convenience when developing new features. Other concepts, such as the metadata Values and all of their valid data types are also defined in this directory.

forms/ This directory contains [Django Forms](https://docs.djangoproject.com/en/4.0/topics/forms/) that define the widgets and GET/POST arguments for web pages that take in user input or other parameters.

views/ This directory has the [Django Views](https://docs.djangoproject.com/en/4.0/topics/http/views/) that define each of QUOREM’s web pages. These Views can be simple template rendering, or can take on a form that allows it to process input queries.

3.3 Walkthrough: Adding a new QUOREM webpage

In this walkthrough, we'll go through the process of adding a new web page with user input that redirects to a plot or some other HTML output.

3.3.1 Planning

Start by developing the function that produces the output you want. For example, you might have a function that takes in an ASV table, a set of taxonomic classifications, and a query taxonomy string and produces some statistics on the distribution of taxa that match the query. Generally, widgets will take names or primary keys ("pk") of database objects to pass as arguments, so this is what your function should expect.

Here is a function that takes in the primary key of a Result that contains a table/matrix, a Result that contains taxonomic classifications, and a query string to filter the results:

```python
def sample_taxonomy_query(table_pk, taxonomy_pk, query_string):
    asv_table = Result.objects.get(pk=table_pk).get_value("asv_table")
    taxonomy = Result.objects.get(pk=taxonomy_pk)
    taxonomy = taxonomy.get_value(value_names=["taxonomic_classification"],
                                  additional_fields=["feature__name"])
    # Filter out the taxonomy table to only features matching the query
    taxonomy = taxonomy[taxonomy['value_data'].str.contains(query_string)]
    match_asvs = [x for x in taxonomy['features__name'] if x in asv_table.index ]
    filtered_table = asv_table.loc[match_asvs]
    n_not_found = sum(filtered_table.sum(axis=0) <= 0)
    n_found = asv_table.shape[1] - n_not_found
    output_string = "Found in %d samples (%.2f%% of total)" % (n_found,
                                                               100*float(n_found)/
                                                               (n_found+n_not_found))
    return output_string
```

With this function designed, we can start to plan the web page that we will build to present this sophisticated analysis. We need to consider the URL we want, the widgets we want to use to collect our parameters/options, and how we want it all displayed.

- search