## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Installation</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Clone the Repository</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Docker Setup</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Manual Setup</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Application Programming Interface (API)</td>
<td>22</td>
</tr>
</tbody>
</table>
**Maine-eDNA Metadata** is a data management system for tracking environmental DNA samples from field data collection to bioinformatics analyses.

The Sequencing and Bioinformatics Metadata project is funded through the Maine-eDNA RII Track-1 EPSCoR grant supported by the National Science Foundation award #OIA-1849227. Maine-eDNA is a Maine coast-wide research effort to determine the efficacy of ecosystem monitoring through environmental DNA (eDNA). For more in-depth information related to Maine-eDNA, see the [Maine-eDNA home page](#). The metadata project is directly related to the goals of **Theme 3: Macrosystem eDNA Integration**, but also involved in **Theme 1: Sustainable Fisheries** and **Theme 2: Harmful and Shifting Species**. Our efforts contribute to the Data Management and Data Analysis Pipeline workgroups. The Sequencing and Bioinformatics Metadata project is a collaboration between the Coordinated Operating Research Entities (CORE) environmental DNA (eDNA) Laboratory and the Data Management Team to facilitate the development of a web application for the management and tracking of eDNA samples (MeDNA-Metadata; [GitHub](#)).

Check out the *Installation* section for further information, including how to setup the project.

---

**Note:** This project is under active development.
1.1 Installation

Important: Ubuntu (20.04 LTS is recommended)

See also:
If you’ve never set up a server before, DigitalOcean provides an extensive variety of tutorials that we highly recommend. Several of the steps below were adapted from their tutorials.

- Initial Server Setup With Ubuntu 20.04
- Django, PostgreSQL, NGINX, Gunicorn, and Ubuntu 20.04

It’s recommended to visit these tutorials to understand some of the rationale behind the steps below.

The following instructions provide guidance on installing MeDNA-Metadata on Ubuntu 20.04.

Important: Instances of youruser need to be replaced with a relevant username. For example, references to /home/youruser/ must be adjusted to the active Ubuntu username. The simplest solution would be to use the same username throughout the setup process, as long as it is not the root username. Never use root. For more information, see the aforementioned Initial Server Setup With Ubuntu 20.04.

1.2 Clone the Repository

To clone the most recent stable release as read-only:

```sh
git clone -b stable/1.0.1 https://github.com/Maine-eDNA/medna-metadata.git
```

To clone the development branch:

```sh
git clone -b main git@github.com:Maine-eDNA/medna-metadata.git
```
1.3 Docker Setup

1.3.1 Install Requirements

Ubuntu 20.04

Deploying with docker requires docker and docker-compose. Docker-compose must be able to run at least version 3.8. DigitalOcean provides tutorials on How to Install and Use Docker on Ubuntu 20.04 and How To Install and Use Docker Compose on Ubuntu 20.04.

After following DigitalOcean’s tutorials, create a symbolic link of the installation to the `docker-compose` command

```
sudo ln -s ~/.docker/cli-plugins/docker-compose /usr/bin/docker-compose
```

**Important:** This allows you to call `docker-compose` from the command line, and is highly recommended to avoid future headaches.

The `/docker` directory has `medna.env.txt` and `medna.env.db.txt`, which contain all environmental variables for docker deployment. These files are necessary for docker. Other files that affect docker are:

- `entrypoint.sh`
- `.dockerignore`
- contents of `/docker` directory
- settings in `medna_metadata/settings.py`

Navigate to the `/docker` directory

```
 cd medna-metadata/docker/
```

Make a copy of `medna.env.txt` and `medna.env.db.txt` in the same `/docker` directory with the `.txt` extension removed (e.g., `medna.env.db`, `medna.env`)

```
 cp medna.env.txt medna.env
 cp medna.env.db.txt medna.env.db
```

`medna.env`:

```
# django general settings; loaded in settings.py
# The secret key must be a large random value and it must be kept secret.
# https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-SECRET_KEY
# https://docs.djangoproject.com/en/4.0/ref/settings/#std-setting-TIME_ZONE

# Django secret key=your_secret_key
# DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 your_ip_address [:1]
# DJANGO_SETTINGS_MODULE=medna_metadata.settings
# DJANGO_DEBUG=False
# DJANGO_LOG_LEVEL=info
# TIME_ZONE=America/New_York
```

(continues on next page)
# Creates a superuser account (a user who has all permissions).
# https://docs.djangoproject.com/en/4.0/ref/django-admin/#createsuperuser

DJANGO_SUPERUSER_PASSWORD=your_superuser_password
DJANGO_SUPERUSER_EMAIL=your_superuser_email@domain.com
DJANGO_SUPERUSER_USERNAME=your_superuser_email@domain.com

# web_start.sh settings
# DJANGO_MANAGEPY_MIGRATE - build database tables; calls the migrate management cmd →
# https://docs.djangoproject.com/en/4.0/topics/migrations/
# DJANGO_DATABASE_FLUSH - empty database of all data; calls the flush management cmd →
# https://docs.djangoproject.com/en/4.0/ref/django-admin/#flush
# DJANGO_SUPERUSER_CREATE - create a superuser account; calls thecreatesuperuser management cmd - https://docs.djangoproject.com/en/4.0/ref/django-admin/
# DJANGO_DEFAULT_GROUPS_CREATE - create default permissions groups; calls custom create_default_groups management cmd (admin, gradstudent, intern)
# DJANGO_DEFAULT_USERS_CREATE - create default user; calls custom create_default_user management cmd
# DJANGO_DATABASE_LOADDATA - load database with base data from fixtures; calls loaddata management cmd - https://docs.djangoproject.com/en/4.0/ref/contrib/staticfiles/
# entrypoint.sh settings
# ENTRYPOINT_DATABASE & ENTRYPOINT_MESSAGING - name of entrypoint to wait for; do not change these values unless updating entrypoint.sh and the type of messaging server or database

DJANGO_MANAGEPY_MIGRATE=on
DJANGO_DATABASE_FLUSH=on
DJANGO_SUPERUSER_CREATE=on
DJANGO_DEFAULT_GROUPS_CREATE=on
DJANGO_DEFAULT_USERS_CREATE=on
DJANGO_DATABASE_LOADDATA=on
DJANGO_COLLECT_STATIC=on
ENTRYPOINT_DATABASE=postgres
ENTRYPOINT_MESSAGING=rabbitmq

# django database settings; loaded in settings.py
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DJANGO_DATABASE_NAME=medna_metadata
DJANGO_DATABASE_TESTNAME=test_medna_metadata
DJANGO_DATABASE_USERNAME=your_db_user
DJANGO_DATABASE_PASSWORD=your_db_password
DJANGO_DATABASE_HOST=medna_metadata_pgdb
DJANGO_DATABASE_PORT=5432

1.3. Docker Setup
# django database backup custom setting; loaded in settings.py
# setting DB_BACKUPS to True will automatically backup the full database
daily at 4:30AM local time - this can be changed in medna_metadata/settings.py
# under CELERYBEAT_SCHEDULE

DB_BACKUPS=False

# django smtp (email) settings; loaded in settings.py
# https://docs.djangoproject.com/en/4.0/topics/email/

DJANGO_EMAIL_HOST_USER=your_email@domain.com
DJANGO_EMAIL_HOST_PASSWORD=your_email_password

# django-storages settings; loaded in settings.py

AWS_ACCESS_KEY_ID=your_access_key_id
AWS_SECRET_ACCESS_KEY=your_access_key
AWS_REGION=your_aws_region
AWS_STORAGE_BUCKET_NAME=your_storage_bucket_name
AWS_STORAGE_BUCKET_SUBFOLDER_NAME=your_storage_bucket_subfolder_name

# RabbitMQ docker settings
# https://www.rabbitmq.com/configure.html
# https://hub.docker.com/_/rabbitmq

RABBITMQ_DEFAULT_USER=your_rabbitmq_user
RABBITMQ_DEFAULT_PASS=your_rabbitmq_password
RABBITMQ_DEFAULT_VHOST=your_rabbitmq_vhost
RABBITMQ_HOST=medna_metadata_rabbitmq
RABBITMQ_PORT=5672

# Celery settings

CELERY_RESULT_BACKEND=rpc
CELERY_BROKER_URL=pyamqp://your_rabbitmq_user:your_rabbitmq_password@medna_metadata_rabbitmq/vhost
CELERYD_NODES=worker
CELERYD_NUM_NODES=1
CELERY_BIN=/home/django/.virtualenvs/mednaenv/bin/celery
CELERY_APP=medna_metadata.celery.app
CELERYD_MULTI=multi
CELERYD_OPTS=''
CELERYD_PID_FILE=/var/run/celery/%n.pid
CELERYD_LOG_FILE=/var/log/celery/%nI.log
CELERYD_LOG_LEVEL=INFO
CELERYBEAT_PID_FILE=/var/run/celery/beat.pid
CELERYBEAT_LOG_FILE=/var/log/celery/beat.log

####################################################################
# Gunicorn settings
# https://github.com/benoitc/gunicorn/blob/master/examples/logging.conf
####################################################################
GUNICORN_CONFIG_FILE=/your/path/to/medna-metadata/docker/gunicorn-logging.conf

medna.env.db:

# PostGIS docker settings; loaded in settings.py
# https://registry.hub.docker.com/r/postgis/postgis/
POSTGRES_USER=your_db_user
POSTGRES_PASSWORD=your_db_password
POSTGRES_DB=medna_metadata

Open and update the environmental variables with desired settings in a text editor such as VIM

```
sudo vim medna.env
```

Write and exit the VIM text editor:

```
:q!
```

Repeat these steps with medna.env.db.

Once settings are verified, run `sudo docker-compose -f docker-compose.prod.yml up -d` from the /docker directory to build and deploy Maine-eDNA Metadata, PostgreSQL with PostGIS, RabbitMQ, Celery, Gunicorn, and NGINX.

If you want to build and load the docker containers directly from the Dockerfiles in /web and /nginx, run `sudo docker-compose -f docker-compose.yml up -d --build`.

After the containers are up, it will take a moment for the application to set itself up. Once it is done, it will be accessible at `http://localhost:8000`

## 1.4 Manual Setup

### 1.4.1 Install Requirements

**Ubuntu 20.04**

Install Ubuntu Requirements:

```
cd medna-metadata
sudo bash requirements/ubuntu-requirements.sh
```
It will run the following installation commands:

```bash
# Ubuntu 20.04
# sudo bash ubuntu-requirements.sh
apt-get update & & apt-get install -y --no-install-recommends apt-utils
# Django with Postgres, Nginx, and Gunicorn on Ubuntu 20.04
apt update & & apt -y install \ 
libpq-dev postgresql postgresql-contrib nginx curl vim netcat
# GDAL, GEOS, PROJ.4, python deps
apt-get -y install \ 
binutils libproj-dev gdal-bin \ 
python3-dev python3-pip python3-venv python3-wheel \ 
rabbitmq-server
# PostGIS
apt -y install \ 
postgis postgresql-12-postgis-3
# Certbot and Nginx plugin
apt -y install \ 
certbot python3-certbot-nginx
```

Create a Virtual Environment and Set Environmental Variables

Create a virtual environment with venv:

```bash
sudo -H pip install --upgrade pip
sudo -H pip install virtualenvwrapper
sudo -H pip install virtualenv
```

Dotenv is a python library for reading in environmental variables from a file:

```bash
pip install django-dotenv
```

Add environmental variables to the bottom of bashrc, which reloads environmental variables anytime the server reboots. The environmental variables required to run this application are listed in docker/bashrc.txt:

```
# django general settings; loaded in settings.py
# The secret key must be a large random value and it must be kept secret.
# https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-SECRET_KEY
# https://docs.djangoproject.com/en/4.0/ref/settings/#std-setting-TIME_ZONE
export DJANGO_SECRET_KEY='your_secret_key'
export DJANGO_ALLOWED_HOSTS='localhost www.yourdomain.com [::1]' export DJANGO_SETTINGS_MODULE=medna_metadata.settings
export DJANGO_DEBUG=False
export DJANGO_LOG_LEVEL='info'
export TIME_ZONE='America/New_York'
```

```
# django database settings; loaded in settings.py
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases
```
These environmental variables must be updated with values specific to your deployment.
Open ~/.bashrc with a text editor such as VIM:

```bash
sudo vim ~/.bashrc
```

Scroll to the bottom of bashrc and type `i` to insert into the file. Write and exit the VIM text editor:

```bash
:wq!
```

**Tip:** To write and quit within the VIM text editor, `[esc]`, `:wq!`, and `[enter]`

Load the environmental variables:

```bash
source ~/.bashrc
```

Make a copy of gunicorn.env.txt in the same /docker directory with the .txt extension removed (e.g., gunicorn.env)

```bash
cp docker/gunicorn.env.txt docker/gunicorn.env
```

```bash
gunicorn.env:
```

```bash
############################################################
# django general settings; loaded in settings.py
# The secret key must be a large random value and it must be kept secret.
# https://docs.djangoproject.com/en/4.0/ref/settings/#std:setting-SECRET_KEY
# https://docs.djangoproject.com/en/4.0/ref/settings/#std-setting-TIME_ZONE
############################################################
DJANGO_SECRET_KEY=your_secret_key
DJANGO_ALLOWED_HOSTS=localhost [:1]
DJANGO_DEBUG=True
DJANGO_LOG_LEVEL=info
TIME_ZONE=America/New_York

############################################################
# django database settings; loaded in settings.py
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases
############################################################
DJANGO_DATABASE_NAME=medna_metadata
DJANGO_DATABASE_TESTNAME=test_medna_metadata
DJANGO_DATABASE_USERNAME=your_db_user
DJANGO_DATABASE_PASSWORD=your_db_password
DJANGO_DATABASE_HOST=localhost
DJANGO_DATABASE_PORT=''

############################################################
# django database backup custom setting; loaded in settings.py
# setting DB_BACKUPS to True will automatically backup the full database
# daily at 4:30AM local time - this can be changed in medna_metadata/settings.py
# under CELERYBEAT_SCHEDULE
############################################################
DB_BACKUPS=False
```

(continues on next page)
# django smtp (email) settings; loaded in settings.py
# https://docs.djangoproject.com/en/4.0/topics/email/
DJANGO_EMAIL_HOST_USER=your_email@domain.com
DJANGO_EMAIL_HOST_PASSWORD=your_email_password

# django-storages settings; loaded in settings.py
AWS_ACCESS_KEY_ID=your_access_key_id
AWS_SECRET_ACCESS_KEY=your_access_key
AWS_REGION=your_aws_region
AWS_STORAGE_BUCKET_NAME=your_storage_bucket_name
AWS_STORAGE_BUCKET_SUBFOLDER_NAME=your_storage_bucket_subfolder_name

# Celery settings
CELERY_RESULT_BACKEND=rpc
CELERY_BROKER_URL=pyamqp://your_rabbitmq_user:your_rabbitmq_password@localhost:5672/your_rabbitmq_vhost
CELERYD_NODES=worker
CELERYD_NUM_NODES=1
CELERYD_BIN=/your/path/to/bin/celery
CELERYD_APP=medna_metadata.celery.app
CELERYD_MULTI=multi
CELERYD_OPTS=''
CELERYD_PID_FILE=/var/run/celery/%n.pid
CELERYD_LOG_FILE=/var/log/celery/%n.log
CELERYD_LOG_LEVEL=INFO
CELERYBEAT_PID_FILE=/var/run/celery/beat.pid
CELERYBEAT_LOG_FILE=/var/log/celery/beat.log

# Gunicorn settings
# https://docs.gunicorn.org/en/stable/settings.html#logging
# https://github.com/benoitc/gunicorn/blob/master/examples/logging.conf
GUNICORN_CONFIG_FILE=/your/path/to/medna-metadata/docker/gunicorn-logging.conf

Update the environmental variables with values specific to your deployment in a text editor such as VIM.

```
sudo vim docker/gunicorn.env
```

Type `i` to insert into the file. Write and exit the VIM text editor:
Warning: MeDNA-Metadata will not successfully deploy without setting these environmental variables.

Create and activate virtualenvwrapper:

```
mkvirtualenv --python /usr/bin/python3.8 mednaenv
```

Important:

The version of python varies and you will have to check or install it.

- Python with Virtualenvwrapper
- Change Ubuntu’s Default Python Version

Activate the virtual environment:

```
workon mednaenv
```

Install python requirements to the virtualenv:

```
pip install -U -r requirements/prod.txt
```

The following python libraries will install: .. literalinclude:: ../../requirements/base.txt

```
language
env
-r base.txt
gunicorn==20.1.0
```

### 1.4.2 Create PostgreSQL database with PostGIS Extension

Create the database for your project:

```
sudo -u postgres psql -c "CREATE DATABASE medna_metadata;"
```

Create a database user and add them to the project with a secure password:

```
sudo -u postgres psql -c "CREATE USER youruser WITH PASSWORD 'yourdbpassword';"
```

Recommended settings from the Django project:

```
sudo -u postgres psql -c "ALTER ROLE youruser SET client_encoding TO 'utf8';"
sudo -u postgres psql -c "ALTER ROLE youruser SET default_transaction_isolation TO 'readcommitted';"
sudo -u postgres psql -c "ALTER ROLE youruser SET timezone TO 'UTC';"
```

Set youruser as the administrator for the medna_metadata database:
Grant privileges to the youruser database user to create databases for Django tests:

```
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE medna_metadata TO youruser;"
```

Add the PostGIS extension to medna_metadata:

```
sudo -i -u postgres psql -d medna_metadata -c "CREATE EXTENSION postgis;"
```

**Tip:** It would be advantageous here to use the same username as your selected Ubuntu username.

### Migrate the Database Tables

Migrate the database schema to PostgreSQL from within the same directory as manage.py Within each app there is a migration directory which contains files which tell the database how to create the database tables. These have been pre-generated and added to this repository to simplify the process of deploying the Maine-eDNA Metadata application:

```
python manage.py migrate users
python manage.py migrate utility
python manage.py migrate field_site
python manage.py migrate sample_label
python manage.py migrate field_survey
python manage.py migrate wet_lab
python manage.py migrate freezer_inventory
python manage.py migrate bioinformatics
```

Now, if everything looked good (e.g., no error messages), complete the remaining migrations:

```
python manage.py migrate
```

### 1.4.3 Create Superuser

Creating a superuser adds an administrative user with full privileges to the MeDNA-Metadata project.

Create a superuser:

```
python manage.py createsuperuser
```

When prompted, enter in your preferred credentials (youremail, yourpassword).
1.4.4 Collect Static

Collecting static files will copy all static content into the directory specified in settings.py.

Collect static files:

```
python manage.py collectstatic
```

1.4.5 Test The Deployment

Temporarily create an exception for port 8000:

```
sudo ufw allow 8000
```

Test the project deployment:

```
python manage.py runserver 0.0.0.0:8000
```

In your web browser, visit the server’s IP address followed by :8000
http://youripaddress:8000

If you’re able to see a live project, then enter [CTRL-C] in to shut down the test deployment.

Test Gunicorn

Now test to see if the project can be deployed with Gunicorn:

```
gunicorn --bind 0.0.0.0:8000 medna_metadata.wsgi
```

See also:

If you were able to visit the same page while deployed with Gunicorn, continue onward. If not, some helpful troubleshoot steps can be found in the DigitalOcean tutorial on setting up Django with PostgreSQL, Gunicorn, and Nginx.

• Django with PostgreSQL, Gunicorn, and Nginx on Ubuntu 20.04

1.4.6 Create Gunicorn Socket and Service files (e.g., daemonizing!)

Leave the virtualenv:

```
deactivate
```

Create a systemd Socket file:

```
sudo vim /etc/systemd/system/gunicorn.socket
```

Write the following to the file:

```
[Unit]
Description=gunicorn socket

[Socket]
```

(continues on next page)
ListenStream=/run/gunicorn.sock

[Install]
WantedBy=sockets.target

Write and exit the VIM text editor:
:wq!

Create a systemd service file:
sudo vim /etc/systemd/system/gunicorn.service

Modify then write the following to the file:

[Unit]
Description=gunicorn daemon
Requires=gunicorn.socket
After=network.target

[Service]
User=youruser
Group=youruser
WorkingDirectory=/path/to/medna-metadata
EnvironmentFile=/path/to/medna-metadata/docker/gunicorn.env
ExecStart=/path/to/.virtualenvs/mednaenv/bin/gunicorn 
        --log-config ${GUNICORN_CONFIG_FILE} 
        --workers 3 
        --timeout 300 
        --bind unix:/run/gunicorn.sock 
        medna_metadata.wsgi:application

[Install]
WantedBy=multi-user.target

Warning: You will need to replace the User and Group to the correct Ubuntu username and group and modify the WorkingDirectory, EnvironmentFile, and ExecStart to the actual directory MeDNA-Metadata is in.

Write and exit the VIM text editor:
:wq!

Enable the socket and service:
sudo systemctl start gunicorn.socket
sudo systemctl enable gunicorn.socket

Check for the socket file’s status (it should be green):
sudo systemctl status gunicorn.socket

Check for the existence of the sock file:
file /run/gunicorn.sock

If there is no file, check the socket’s logs and troubleshoot:

```bash
sudo journalctl -u gunicorn.socket
```

Test the socket activation (should be grey & inactive):

```bash
sudo systemctl status gunicorn
```

Test the socket activation mechanism through curl:

```bash
curl --unix-socket /run/gunicorn.sock localhost
```

Now see if Gunicorn is “running”:

```bash
sudo systemctl status gunicorn
```

If you need to troubleshoot, check journalctl and daemon-reload and restart gunicorn when the service, socket, settings, or env files are edited:

```bash
sudo journalctl -u gunicorn

sudo systemctl daemon-reload

sudo systemctl restart gunicorn
```

### 1.4.7 Setup Celery and RabbitMQ

Celery task management and the RabbitMQ messaging server are used for task queues within the backend application. This allows for things such as queues of data transformations and workers that will spawn as resources are available.

**See also:**

- Celery First Steps
- Celery, RabbitMQ, and Ubuntu
- Celery and Django

**Note:** Celery and RabbitMQ should have already been installed with the requirements.txt and ubuntu-requirements.sh, but the commands are also provided here.

Install RabbitMQ messaging server:

```bash
sudo apt-get update

sudo apt-get install rabbitmq-server
```

**Important:** If you named your venv something other than mednaenv, use that here instead.

Activate the virtualenv:

```bash
workon mednaenv
```

Install Celery:
pip install -U "celery>=5.2.3"

**Setup RabbitMQ**

Add a user and a virtual host:

```
sudo rabbitmqctl add_user youruser yourpassword
sudo rabbitmqctl add_vhost yourvhost
sudo rabbitmqctl set_user_tags youruser yourtag
sudo rabbitmqctl set_permissions -p yourvhost youruser "." "." "." "."
```

**Warning:** You will need to replace youruser, yourpassword, yourvhost, yourtag, to the password, username, tag and hostname of your choosing. These are specific to RabbitMQ and not dependent on other applications. What is chosen is used to construct your `CELERY_BROKER_URL`.

Stop RabbitMQ:

```
sudo systemctl stop rabbitmq-server
```

Check to verify it is actually stopped:

```
sudo rabbitmqctl cluster_status
```

Start it up again:

```
sudo systemctl start rabbitmq-server
sudo systemctl restart rabbitmq-server
```

**Warning:** For Celery and RabbitMQ to function, the `CELERY_RESULT_BACKEND` and `CELERY_BROKER_URL` variables must be set in `~/.bashrc` and `docker/gunicorn.env`. These variables should resemble the following:

- `CELERY_RESULT_BACKEND='rpc'`
- `CELERY_BROKER_URL='pyamqp://youruser:yourpassword@localhost:port/yourvhost'`

**Create Celery Worker and Beat files (e.g., daemonizing!)**

Like Gunicorn, celery should be run as a Systemd service. –

First, open your environment file:

```
sudo vim docker/gunicorn.env
```

Update or add the following Celery config settings to your environment file:

```
CELERY_RESULT_BACKEND='your_result_backend'
CELERY_BROKER_URL='transport://your_rabbitmq_user:your_rabbitmq_password@hostname:port/...your_rabbitmq_vhost'
```

(continues on next page)
CELERYD_NODES='worker'
CELERY_BIN='/path/to/.virtualenvs/mednaenv/bin/celery'
CELERY_APP='medna_metadata.celery.app'
CELERYD_MULTI='multi'
CELERYD_OPTS=''
CELERYD_PID_FILE='/var/run/celery/%n.pid'
CELERYD_LOG_FILE='/var/log/celery/%n%I.log'
CELERYD_LOG_LEVEL='INFO'
CELERYBEAT_PID_FILE='/var/run/celery/beat.pid'
CELERYBEAT_LOG_FILE='/var/log/celery/beat.log'

Write and exit the VIM text editor:

:wq!

Now we need to create the Celery log and run directories and update their permissions:

```bash
sudo mkdir /var/run/celery/
sudo mkdir /var/log/celery/
sudo chgrp yourgroup /var/run/celery/
sudo chgrp yourgroup /var/log/celery/
sudo chmod g+rwx /var/run/celery/
sudo chmod g+rwx /var/log/celery/
```

Create a Celery service file:

```bash
sudo vim /etc/systemd/system/celery.service
```

Modify then write the following to the file:

```bash
[Unit]
Description=Celery Service
After=network.target rabbitmq-server.service
Requires=rabbitmq-server.service

[Service]
Type=forking
User=youruser
Group=yourgroup
EnvironmentFile=/path/to/medna-metadata/docker/gunicorn.env
WorkingDirectory=/path/to/medna-metadata
Environment=DJANGO_SETTINGS_MODULE=medna_metadata.settings
ExecStart=/bin/sh -c '$(CELERY_BIN) -A $(CELERY_APP) multi start $(CELERYD_NODES) 
--pidfile=$(CELERYD_PID_FILE) --logfile=$(CELERYD_LOG_FILE) 
--loglevel="$(CELERYD_LOG_LEVEL)"
$(CELERYD_OPTS)'
ExecStop=/bin/sh -c '$(CELERY_BIN) multi stopwait $(CELERYD_NODES) 
--pidfile=$(CELERYD_PID_FILE) --logfile=$(CELERYD_LOG_FILE) 
--loglevel="$(CELERYD_LOG_LEVEL)"
$(CELERYD_OPTS)''
ExecReload=/bin/sh -c '$(CELERY_BIN) -A $(CELERY_APP) multi restart $(CELERYD_NODES) 
--pidfile=$(CELERYD_PID_FILE) --logfile=$(CELERYD_LOG_FILE) 
--loglevel="$(CELERYD_LOG_LEVEL)"
$(CELERYD_OPTS)'
```

(continues on next page)
Restart=always

[Install]
WantedBy=multi-user.target

Write and exit the VIM text editor:

:wq!

**Warning:** You will need to replace the User and Group to the correct Ubuntu username and group and modify the WorkingDirectory and EnvironmentFile to the actual directory MeDNA-Metadata is in.

We also need a **celerybeat** **Systemd service** for scheduling tasks.

Create a celerybeat file:

```bash
sudo vim /etc/systemd/system/celerybeat.service
```

Modify then write the following to the file:

```
[Unit]
Description=Celery Beat Service
After=network.target rabbitmq-server.service
Requires=rabbitmq-server.service

[Service]
Type=simple
User=youruser
Group=yourgroup
EnvironmentFile=/path/to/medna-metadata/docker/gunicorn.env
WorkingDirectory=/path/to/medna-metadata
Environment=DJANGO_SETTINGS_MODULE=medna_metadata.settings
ExecStart=/bin/sh -c \
  '$(CELERY_BIN) -A $(CELERY_APP) beat \n  --pidfile=$(CELERYBEAT_PID_FILE) \n  --logfile=$(CELERYBEAT_LOG_FILE) --loglevel=$(CELERYD_LOG_LEVEL)'
Restart=always

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

Write and exit the VIM text editor:

:wq!

**Warning:** You **must** replace the User and Group to the correct Ubuntu username and group and modify the WorkingDirectory and EnvironmentFile to the actual directory MeDNA-Metadata is in.

We can now start the celery and celerybeat services:
sudo systemctl daemon-reload
sudo systemctl start celery.service
sudo systemctl start celerybeat.service
sudo systemctl status celery.service

If any modifications are made to any tasks.py or medna_metadata/celery.py, restart celerybeat and celeryworker:

```bash
sudo systemctl restart celery.service && sudo systemctl restart celerybeat.service &&
  sudo systemctl daemon-reload && sudo systemctl restart gunicorn
```

If you want these services to start automatically on boot, you can enable them as follows:

```bash
sudo systemctl enable celery.service
sudo systemctl enable celerybeat.service
```

### Troubleshooting Celery

If you are trying to troubleshoot celery or celerybeat, be sure to check system logs for error messages:

```bash
sudo cat /var/log/syslog
sudo tail /var/log/syslog -n 40
```

You can also check RabbitMQ logs:

```bash
sudo tail /var/log/rabbitmq/rabbit@medna-metadata.log -n 50
```

To view Celery tasks as they are sent by RabbitMQ:

```bash
celery -A medna_metadata worker --pool=solo -l info
```

[CTRL-C] to exit.

### 1.4.8 Collect Static Files

Any time there is a change made to the python code, run the following to reload changes:

```bash
git pull && python manage.py collectstatic --noinput --clear && sudo systemctl daemon-reload && sudo systemctl restart gunicorn.socket gunicorn.service
```

### 1.4.9 Configure Nginx

Create a nginx config file:

```bash
sudo vim /etc/nginx/sites-available/medna-metadata
```

Modify and write the following:
server {
    listen 80;
    server_name youripaddress yourdomain.com;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /path/to/medna-metadata;
    }

    location / {
        include proxy_params;
        proxy_pass http://unix:/run/gunicorn.sock;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_connect_timeout 300s;
        proxy_read_timeout 300s;
    }
}

**Warning:** You **must** edit the `server_name` and `location` to the actual IP address or domain name and to the actual directory MeDNA-Metadata is in.

Write and exit the VIM text editor:

```
:wq!
```

Enable the file by linking it to sites-enabled:

```
sudo ln -s /etc/nginx/sites-available/medna-metadata /etc/nginx/sites-enabled
```

Test the Nginx configuration for syntax errors:

```
sudo nginx -t
```

If there are no errors, restart Nginx:

```
sudo systemctl restart nginx
```

Delete port 8000 and allow Nginx in the firewall:

```
sudo ufw delete allow 8000
sudo ufw allow 'Nginx Full'
```
Troubleshooting Nginx and Gunicorn

See also:

For more information on troubleshooting Nginx and Gunicorn, please see the following:

- Django with PostgreSQL, NGINX, and Gunicorn on Ubuntu 20.04

1.4.10 SSL Certificates with Certbot

Run certbot:

```
sudo certbot --nginx -d yourdomain.com
```

Follow the prompt by enter in your email address, [A], [n], and [2].

Verify Certbot auto-renewal:

```
sudo certbot renew --dry-run
```

You should now be good to go and running with your desired server and domain.

1.5 Application Programming Interface (API)

1.5.1 API Documentation

MeDNA-Metadata’s API was written in the Django REST Framework and all API documentation are automatically generated with drf-yasg library. With an active deployment, simply navigate to https://yourdomain.com/swagger/ to view details on all Create, Read, Update, and Delete (CRUD) operations.
1.5.2 Database Design

Field Data Collection

Field data collection is project wide, and pertains to environmental and collection information related to sediment or water samples. These samples are barcoded and that barcode is used as a unique identifier. Barcodes are placed on filters for water samples, and on sub-cores for sediment samples. Each water collection is filtered using multiple filter types (Nitex, Glass Fiber Filter, Supor, or Cellulose Nitrate) and each filter represents one sample. If sediment is collected, it is initially collected as a core. After a core of sediment is collected, it is subdivided into sub-cores, and each sub-core represents one sample.

Filters, cores, and sub-cores are stored within -80°C lab freezers until extraction.
**Wet Lab Processing**

Wet lab processing is conducted by the Wet Laboratory. Wet lab processing pertains to all processing from extraction, Digital Droplet PCR (ddPCR), quantitative PCR (qPCR), and Next Generation Sequencing (NGS). After filtration or sub-coring, the samples are extracted. Extraction converts field samples into solutions of eDNA. Extractions share the same unique barcode identifier as samples.

The tables for LibraryPrep, PooledLibrary, RunPrep, RunResults, and FastqFile are all directly related to Next Generation Sequencing (NGS) and the generation of FastQ files. FastQ files contain the resulting sequences from NGS, and sequencing is typically performed by hardware equivalent to an Illumina MiSeq. These NGS tables also relate to information in the IndexPair and PrimerPair tables. The PCR and PCRRreplicate tables pertain to additional analyses that can amplify or quantify eDNA samples and are also related to the PrimerPair table.

The PrimerPair table contains information related to the target gene, or DNA region of interest. Primers allow for the amplification of a specific region of DNA, which is important for the positive identification of a particular taxon or taxonomic group. The IndexPair table contains information related to a unique sequence that is used to identify each sample. During NGS, the samples are pooled together. Adding a unique index adapter to the sample sequences enables the separation of pooled samples after sequencing is complete. Separation of pooled samples occurs during a process called demultiplexing, which converts raw BCL files into two FastQ files for each sample.

Extractions and pooled libraries are stored within the -80°C lab freezers until they are depleted.
Freezer Inventory Tracking

Freezer inventory tracking is conducted by the Wet Laboratory, and pertains to the tracking of all filters, subcores, extractions, or pooled libraries that may be stored in -80°C lab freezers.

Occasionally a sediment core is not sub-cored within the same day it was collected, so it will be stored in -80°C lab freezers until it is sub-cored. Water collections will not be stored in -80°C lab freezers, but the filters that are taken from water are typically stored in freezers until they are extracted.

Extractions are stored within -80°C lab freezers, where they are temporarily removed from the freezer when further processing is performed. Only small portions are taken from an extraction each time they are removed from the freezer. After small portions are taken, extractions are placed back in the -80°C lab freezers until there is no more extraction remaining. The Wet Laboratory will track the amount taken from extractions, who has the extraction, and whether it was returned to the same freezer location.

A pooled library may also function in a similar way to an extraction, in that it is a solution of eDNA that may be stored in the -80°C lab freezers. The Wet laboratory will also track the amount taken from a pooled library, who has the pooled library, and whether it was returned to the same freezer location.
Bioinformatics: Denoising or Clustering

The bioinformatic process of denoising (converting sequences within FastQ files to Amplicon Sequence Variants) or clustering (converting sequences within FastQ files to Operational Taxonomic Units), are represented by the DenoiseClusterMetadata, FeatureReads, and FeatureOutputs tables.

The DenoiseClusterMetadata table covers information related to the specifics of the bioinformatics analysis pipeline. This table enables users to track the process through which ASVs or OTUs were generated. The resulting sequences are listed in the FeatureOutput table. The number of reads, or count of each sequence in each sequencing run, is represented in the FeatureRead table.

Bioinformatics: Taxonomic Annotation

The bioinformatic process of taxonomic annotation is represented by the AnnotationMetadata, TaxonomicAnnotation, ReferenceDatabase, TaxonDomain, TaxonKingdom, TaxonClass, TaxonOrder, TaxonFamily, TaxonGenus, and TaxonSpecies tables.

The AnnotationMetadata table covers information related to the specifics of the bioinformatics analysis pipeline. This table enables us to track the process through which taxonomies are annotated to feature outputs from denoising or clustering. It is possible that taxonomic annotation may be performed multiple times on the same set of feature outputs. The TaxonomicAnnotation table retains the results of either BLAST or a trained classifier and also enables the annotation of manually verified taxonomy. The TaxonDomain, TaxonKingdom, TaxonClass, TaxonOrder, TaxonFamily, TaxonGenus, and TaxonSpecies tables represent curated regional species lists that can be referenced to manually associate verified taxonomy to a sequence through the TaxonomicAnnotation table.

1.5. Application Programming Interface (API)