Pytition

Release 2.0

Yann Sionneau

Jun 26, 2021

CONTENTS:

Pytition is an application for privacy-friendly online petitions you can host on your own server. *Pytition* uses the renown *Django* framework and is easy to install.

Demo: https://pytitiondemo.sionneau.net/

CHAPTER

ONE

INSTALLATION

1.1 Manual installation (recommended for production)

Install system dependencies:

1.1.1 On Debian derivatives

```
$ sudo apt update
$ sudo apt install git virtualenv python3-dev build-essential mariadb-server gettext␣
↪libzip-dev libssl-dev
```

On Ubuntu 18.04 LTS you need to install libmariadbclient-dev:

```
$ sudo apt install libmariadbclient-dev
```

On Ubuntu 20.04 LTS you need to install libmariadb-dev-compat:

```
$ sudo apt install libmariadb-dev-compat
```

1.1.2 On Centos/Fedora derivatives

```
$ sudo yum install MariaDB-server galera-4 MariaDB-client MariaDB-shared MariaDB-
↪backup MariaDB-common git python3 python3-virtualenv make gcc gettext
```

1.1.3 On Arch Linux

```
$ sudo pacman -S mariadb mariadb-libs python make gcc gettext
```

1.1.4 Get the source, configure and initialize Pytition

Get the latest release git tag:

```
$ version=$(curl -s https://api.github.com/repos/pytition/pytition/releases/latest |␣
↪grep "tag_name" | cut -d : -f2,3 | tr -d \" | tr -d ,)
```

Create a directory to host your Pytition instance and it's static files:

```
$ mkdir -p www/static www/mediaroot
```

Create a Python3 virtualenv to install Pytitiont's dependencies:

```
$ virtualenv -p python3 pytition_venv
```

Clone Pytition git repository and checkout latest release:

```
$ cd www
$ git clone https://github.com/pytition/pytition
$ cd pytition
$ git checkout $version
```

Enter your virtualenv and install Pytition's dependencies:

```
$ source ../../pytition_venv/bin/activate
(pytition_venv) $ pip3 install -r requirements.txt
```

Create a MySQL database and user for Pytition:

```
$ password="ENTER_A_SECURE_PASSWORD_YOU_WILL_REMEMBER_HERE"
$ sudo mysql -h localhost -u root -Bse "CREATE USER pytition@localhost IDENTIFIED BY '
↪${password}'; CREATE DATABASE pytition; GRANT USAGE ON *.* TO 'pytition'@localhost;␣
↪GRANT ALL privileges ON pytition.* TO pytition@localhost; FLUSH PRIVILEGES;"
```

Write your SQL credential file in *my.cnf* outside of *www*:

```
[client]
database = pytition
user = pytition
password = YOUR_PASSWORD_HERE
default-character-set = utf8
```

If your SQL server is MariaDB <= 10.2.1, you need to setup your SQL server to use table format compatible with larger-than-767-bytes columns. From 10.2.2 onward, row format is already DYNAMIC by default. So, if you have an old MariaDB, add the following lines after *[server]* in */etc/mysql/mariadb.conf.d/50-server.cnf* (This path is for Ubuntu 18.04):

```
innodb_large_prefix=true
innodb_file_format=barracuda
innodb_file_per_table=true
innodb_default_row_format=DYNAMIC
```

Create your Pytition instance config file by copying the example one:

```
$ cd www/pytition
$ cp pytition/pytition/settings/config_example.py pytition/pytition/settings/config.py
```

Now you can edit your config file in *pytition/pytition/settings/config.py* according to *Configuration*.

You **must** *at least* configure the settings described in the *Mandatory settings* section of the *Configuration* page.

Those are:

- SECRET_KEY
- STATIC_URL
- STATIC_ROOT
- MEDIA_URL
- MEDIA_ROOT
- DATABASES
- ALLOWED_HOSTS

Note: Do not forget to put a correct path to your *my.cnf* MySQL credential file in your config *DATABASES* setting.

Initialize Pytition project database. Pay attention to be in your virtualenv to enter the following commands:

```
$ cd www/pytition/pytition
$ export DJANGO_SETTINGS_MODULE="pytition.settings.config"
$ python3 manage.py migrate
$ python3 manage.py collectstatic
$ python3 manage.py compilemessages
$ python3 manage.py createsuperuser
```

Note: You will be asked to enter a *username*, *email* and *password* for the administrator's account.

Before trying to configure a web server you can try to see if your configuration is OK by running:

```
$ DEBUG=1 DJANGO_SETTINGS_MODULE=pytition.settings.config python3 ./manage.py␣
↪runserver
```

You can then point your browser to *http://yourdomain.tld:8000* and check that you can see Pytitiont's home page and log-in with your newly created admin account.

Warning: If you've set `USE_MAIL_QUEUE` to `True` and `MAIL_EXTERNAL_CRON_SET` to `False`, running Pytition via `manage.py runserver` might not work well since you need to be run via *uwsgi*. Especially emails might not be sent.

Note: If you switch `USE_MAIL_QUEUE` from `False` to `True` at some point, you might have to re-run `python3 manage.py migrate` to create the database structures needed for the mail queues.

1.1.5 Configure your web server

Nginx + uwsgi (recommended)

First install Nginx web server:

```
$ sudo apt install nginx
```

Here is an example of Nginx configuration that you can put in */etc/nginx/sites-available/pytition*:

```
server {
  server_name pytition.mydomain.tld;
  keepalive_timeout   70;

  location / {
    include         uwsgi_params;
    uwsgi_pass      unix:/var/run/uwsgi/app/pytition/socket;
  }
  location /static {
    alias /home/pytition/www/static;
  }

  location /mediaroot {
    alias /home/pytition/www/mediaroot;
  }

  listen 443 ssl; # managed by Certbot
  ssl_certificate /etc/letsencrypt/live/pytition.mydomain.tld/fullchain.pem; #␣
↪managed by Certbot
  ssl_certificate_key /etc/letsencrypt/live/pytition.mydomain.tld/privkey.pem; #␣
↪managed by Certbot
  include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
  server_name pytition.mydomain.tld;
  listen 80;
  return 301 https://pytition.mydomain.tld$request_uri;
}
```

The previous example automatically redirects HTTP/80 to HTTPS/443 and uses Let's Encrypt generated certificate.

Enable your new Nginx config:

```
$ sudo ln -s /etc/nginx/sites-available/pytition /etc/nginx/sites-enabled/pytition
$ sudo systemctl reload nginx
```

Install uwsgi dependency:

```
sudo apt install uwsgi uwsgi-plugin-python3 python3-uwsgidecorators
```

Put the UNIX user of your install in *www-data* group (for Debian like systems) if your user wasn't *www-data* already. For instance in our case we use the *pytition* unix username:

```
sudo usermod -a -G pytition www-data
```

Give both uwsgi and nginx access to your mediaroot directory:

```
sudo chown -R pytition:www-data /home/pytition/www/mediaroot
```

Now let's create our uwsgi configuration in */etc/uwsgi/apps-available/pytition.ini*:

```
[uwsgi]
chdir = /home/pytition/www/pytition/pytition
module = pytition.wsgi
home = /home/pytition/pytition_venv
master = true
processes = 10
vacuum = true
socket = /run/uwsgi/app/pytition/socket
uid = ENTER_HERE_PYTITION_UNIX_USER
gid = www-data
chmod-socket = 664
plugins = python3
env = DJANGO_SETTINGS_MODULE=pytition.settings.config
```

Create a symlink to enable or uwsgi configuration:

```
sudo ln -s /etc/uwsgi/apps-available/pytition.ini /etc/uwsgi/apps-enabled/pytition.ini
```

Start uwsgi and nginx servers:

```
$ sudo systemctl start uwsgi
$ sudo systemctl start nginx
```

Your Pytition home page should be available over there: http://mydomain.tld

Now it's time to *Configure* your Pytition instance the way you want!

1.2 Installation via Docker (recommended for development)

Warning: Please, do **NOT** use this in production. You would have tons of security and performance issues. You could lose your SECRET_KEY, you would run with Django's DEBUG setting enabled, you would be serving static files via Django basic webserver. You would be running with no HTTPS possibility at all. etc etc. Please : don't.

Clone latest development version of Pytition:

```
$ git clone https://github.com/pytition/pytition
```

Install docker and docker-compose:

```
$ sudo apt install docker.io docker-compose
```

Put your user in the docker group (needed for Ubuntu 18.04) and start docker daemon:

```
$ sudo usermod -a -G docker $USER
$ # log-in again as your user for group change to take effect
$ # or just type the following line
$ su -l $USER
$ sudo systemctl enable docker
$ sudo systemctl start docker
```

For the first run you need to create the database container and let it be ready:

```
$ docker-compose up --build db
```

Wait until it prints something like:

```
LOG:  database system is ready to accept connections
```

Then hit ^C (ctrl+C) to shutdown the database container.

From now on, you can just type this to run Pytition in a container:

```
$ docker-compose up --build
```

Last command before being able to click on the "http://0.0.0.0:8000/" link that the "web" container prints to out on the console. You need to run migrations, install static files, compile language files, create an admin account and lastly populate your database with some dummy data. You can do all of this with the *dev/initialize.sh* script:

```
$ docker-compose exec web ./dev/initialize.sh
```

Aaaand that's it! You can now just click on the links:

- http://0.0.0.0:8000/ for the Pytition interface
- http://0.0.0.0:8080/ for the mail server web interface

Next time, just run `$ docker-compose up --build`

CHAPTER

TWO

MULTI-DOMAIN INSTALLATION EXAMPLE

2.1 Objectif

Mutualize Pytition's code so that database and mediaroot directory stay separate for each organization. In practice, on a single hosting server, you will have for instance 2 organizations that will each have their own Pytition instance: pytition.orga1.org and pytition.orga2.org and each web site will share Pytition's source code but will have its own independant database and mediaroot directory. Because the source code will be shared, it will be easier to keep all the web sites up-to-date, using a dedicated administration account.

2.2 Creating user accounts and directories

```
$ sudo useradd -m -s /bin/bash pytition-admin
$ sudo useradd -m -s /bin/bash orga1-user
$ sudo useradd -m -s /bin/bash orga2-user
```

pytition-admin will be the user account dedicated to Pytition's code maintenance.

```
$ sudo mkdir -p /etc/pytition/{orga1,orga2,admin}
$ sudo touch /etc/pytition/{orga1,orga2,admin}/__init__.py
$ sudo touch /etc/pytition/__init__.py
```

/etc/pytition will contain database config and credentials as well as Pytition's config file for each site.

```
$ sudo mkdir -p /srv/pytition/www/mediaroot/{admin,orga1,orga2}
$ sudo mkdir -p /srv/pytition/www/static
```

2.3 Install system dependencies

```
$ sudo apt update
$ sudo apt install git virtualenv python3-dev build-essential default-libmysqlclient-
↪dev gettext libzip-dev libssl-dev apache2 uwsgi
```

2.4 Get the source, configure and initialize Pytition

Get the latest release git tag:

```
$ version=$(curl -s https://api.github.com/repos/pytition/pytition/releases/latest |␣
↪grep "tag_name" | cut -d : -f2,3 | tr -d \" | tr -d ,)
```

Create a Python3 virtualenv to install Pytitiont's dependencies:

```
$ cd /srv/pytition/
$ sudo virtualenv -p python3 pytition_venv
```

Clone Pytition git repository and checkout latest release:

```
$ cd www
$ sudo git clone https://github.com/pytition/pytition
$ cd pytition
$ sudo git checkout $version
```

Set correct ownership and group to directories:

```
$ sudo chown -R pytition-admin:www-data /srv/pytition
$ sudo chown orga1-user:www-data /srv/pytition/www/mediaroot/orga1
$ sudo chown orga2-user:www-data /srv/pytition/www/mediaroot/orga2
$ sudo chmod g+s /srv/pytition/www/static/
```

Enter your virtualenv and install Pytition's dependencies:

```
$ sudo su pytition-admin
$ source /srv/pytition/pytition_venv/bin/activate
(pytition_venv) $ pip3 install -r /srv/pytition/www/pytition/requirements.txt
```

Create db-pytition-orga, db-pytition-orga2, db-pytition-admin as well as associated SQL users db-user-orga1, db-user-orga2 and db-user-admin on your MariaDB SQL server.

You need to write a /etc/pytition/{orga1,orga2,admin}/my.cnf file for each organization.

```
[client]
host = your-data-base-server
database = db-pytition-orga1
user = db-user-orga1
password = YOUR_PASSWORD_HERE
default-character-set = utf8
```

For the administration account, you can use an sqlite3 database instead of creating a new database on MariaDB.

Create the /etc/pytition/{orga1,orga2,admin}/config.py file for each organization. You can start by copying the configuration example file /src/pytition/www/config_example.py

The my.cnf and config.py files must have the correct permissions.

E.g. for orga1:

```
$ sudo chown orga1:pytition-admin /etc/pytition/orga1/{my.cnf,config.py}
$ sudo chmod u=rw,g=r,o=--- /etc/pytition/orga1/{my.cnf,config.py}
```

Now you can edit your config file in *pytition/pytition/settings/config.py* according to *Configuration*.

You **must** *at least* configure the settings described in the *Mandatory settings* section of the *Configuration* page.

Those are:

- SECRET_KEY
- STATIC_URL
- STATIC_ROOT
- MEDIA_URL
- MEDIA_ROOT
- DATABASES
- ALLOWED_HOSTS

Warning: Pay attention to the following config values:

```
STATIC_ROOT = "/srv/pytition/www/static"
MEDIA_ROOT = "/srv/pytition/www/mediaroot/orga1 (pour le config.py de l'orga1)
```

The *DATABASE* config value should point to */etc/pytition/orga1/my.cnf*

Note: Do not forget to put a correct path to the *my.cnf* MySQL credential file in your each config *DATABASES* setting.

Initialize Pytition as well as its databases. You must be in the virtualenv while entering the following commands:

```
$ export PYTHONPATH="/etc/pytition"
$ cd /srv/pytition/www/pytition/pytition
$ sudo -u pytition-admin -- DJANGO_SETTINGS_MODULE="admin.config" python3 manage.py␣
↪migrate
$ sudo -u pytition-admin -- DJANGO_SETTINGS_MODULE="admin.config" python3 manage.py␣
↪collectstatic
$ sudo -u pytition-admin -- DJANGO_SETTINGS_MODULE="admin.config" python3 manage.py␣
↪compilemessages
$ sudo -u pytition-admin -- DJANGO_SETTINGS_MODULE="admin.config" python3 manage.py␣
↪createsuperuser
$ sudo -u orga1-user -- DJANGO_SETTINGS_MODULE="orga1.config" python3 manage.py␣
↪migrate
$ sudo -u orga2-user -- DJANGO_SETTINGS_MODULE="orga2.config" python3 manage.py␣
↪migrate
```

Note: You will be asked to enter a *username*, *email* and *password* for the administrator's

Before trying to configure a web server you can try to see if your configuration is OK by running: E.g. for orga1:

```
$ DEBUG=1 DJANGO_SETTINGS_MODULE=orga1.config python3 ./manage.py runserver
```

You can then point your browser to *http://yourdomain.tld:8000* and check that you can see Pytitiont's home page and log-in with your newly created admin account.

Warning: If you've set `USE_MAIL_QUEUE` to `True` and `MAIL_EXTERNAL_CRON_SET` to `False`, running Pytition via `manage.py runserver` might not work well since you need to be run via *uwsgi*. Especially emails might not be sent.

Note: If you switch `USE_MAIL_QUEUE` from `False` to `True` at some point, you might have to re-run `python3 manage.py migrate` to create the database structures needed for the mail queues.

2.5 Apache and uwsgi configuration

Install uwsgi dependency:

```
$ sudo apt install uwsgi uwsgi-plugin-python3 python3-uwsgidecorators
```

and enable proxy_uwsgi on apache:

```
$ sudo a2enmod proxy_uwsgi
```

Here is an example of Apache configuration that you can put in */etc/apache2/sites-available/orga1*:

```
<VirtualHost *:80>

ServerName pytition.orga1.org

Alias /static /srv/pytition/www/static
Proxypass /static !
Alias /mediaroot /srv/pytition/www/mediaroot/orga1/
Proxypass /mediaroot !

ProxyPass / unix:/var/run/uwsgi/app/pytition.orga1.org/socket|uwsgi://localhost/

<Directory /srv/pytition/www/static>
Require all granted
</Directory>

<Directory /srv/pytition/www/mediaroot>
Require all granted
</Directory>

CustomLog /var/log/apache2/access.log combined
CustomLog /var/log/apache2/pytition.orga1.org.log combined

</VirtualHost>
```

Here is an example of uwsgi configuration that you can put in /etc/uwsgi/app-available/. Don't forget to create a symbolic link in /etc/uwsgi/app-enabled pointing to the previously created file.

```
[uwsgi]
chdir = /srv/pytition/www/pytition/pytition
module = pytition.wsgi
home = /srv/pytition/pytition_venv
master = true
enable-threads = true
processes = 5
vacuum = true
socket = /var/run/uwsgi/app/pytition.orga1.org/socket
uid = orga1-user
gid = www-data
```

(continues on next page)

(continued from previous page)

```
chmod-socket = 664
pythonpath = /etc/pytition/
plugins = python3
env = DJANGO_SETTINGS_MODULE=orga1.config
stats = 127.0.0.1:9191
need-app = true
max-requests = 5000
max-worker-lifetime = 3600
reload-on-rss = 2048
worker-reload-mercy = 60
harakiri = 120
py-callos-afterfork = true
auto-procname = true
procname-prefix = orga1->
```

Start uwsgi and nginx servers:

```
$ sudo systemctl start uwsgi
$ sudo systemctl start apache2
```

Your Pytition home page should be available over there: http://pytition.orga1.org

Now it's time to *Configure* your Pytition instance the way you want!

2.6 Regular maintenance (update)

In order to update all your Pytition sites, here is a bach script (run by pytition-admin user) which can be used in a cron task:

```
#!/bin/bash
set -e
DJANGO_MANAGE="/srv/pytition/www/pytition/pytition/manage.py"
source /srv/pytition/pytition_venv/bin/activate
export PYTHONPATH="/etc/pytition/"
echo
echo "############################"
echo "Updating admin Pytition"
echo "############################"
echo
DJANGO_SETTINGS_MODULE="admin.config" python3 $DJANGO_MANAGE maintenance_mode on
DJANGO_SETTINGS_MODULE="admin.config" python3 $DJANGO_MANAGE update
DJANGO_SETTINGS_MODULE="admin.config" python3 $DJANGO_MANAGE maintenance_mode off
for site in $(ls /etc/pytition|grep -vE "^admin$|^__init__\.py$")
do
echo
echo "################################################"
echo "Updating $site Pytition"
echo "################################################"
echo
  DJANGO_SETTINGS_MODULE="$site.config" python3 $DJANGO_MANAGE maintenance_mode on
  DJANGO_SETTINGS_MODULE="$site.config" python3 $DJANGO_MANAGE migrate
  DJANGO_SETTINGS_MODULE="$site.config" python3 $DJANGO_MANAGE maintenance_mode off
done
deactivate
```

CHAPTER

THREE

CONFIGURATION

A configuration example is provided in *pytition/settings/config_example.py*. You should copy and edit it to configure Pytition.

3.1 Mandatory settings

You **must** set the following variables:

ALLOWED_HOSTS = ['127.0.0.1', 'localhost', '[::1]']

Enter the hostname(s) (aka VirtualHost(s)) Django should accept.
For instance mydomain.tld or petition.mydomain.tld

See also:

Details on how to set this up are available in Django documentation: ALLOWED_HOSTS

Example:

```
ALLOWED_HOSTS = ['www.mysuperpetition.org', 'mysuperpetition.org']
```

DATABASES = {}

Enter a database setting.
This will tell Django what database engine you want to use (supported ones are listed there: https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-DATABASE-ENGINE)
It will also give parameters like user/password credentials, server host/port etc.

See also:

Details on how to set this up are available in Django documentation: https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-DATABASES

In the following example, credentials are in my.cnf file:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'OPTIONS': {
            'read_default_file': '/home/pytition/my.cnf',
            'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
        },
```

(continues on next page)

(continued from previous page)

```
    }
}
```

MEDIA_ROOT = ''

Enter the file system path to the directory that will be used to serve user uploaded files.
This must be an initially empty directory.
You must also configure a web server (apache, nginx or other) to serve
the content of this directory according to your *MEDIA_URL* setting
which default is `'/mediaroot/'` in the example config.

For instance you can have this kind of setting:

```
MEDIA_ROOT = '/home/pytition/www/mediaroot'
MEDIA_URL = '/mediaroot/'
```

And then in your apache config:

```
Alias /mediaroot /home/pytition/www/mediaroot
```

Or in your nginx config:

```
location /mediaroot {
    alias /home/pytition/www/mediaroot;
}
```

See also:

https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-MEDIA_ROOT for more details from Django Documentation

MEDIA_URL = '/mediaroot/'

enter the prefix that will be used for the url to refer to uploaded files.
it must end with a forward slash '/'.
you must also configure a web server (apache, nginx or other) to serve
the content of the directory configured as MEDIA_ROOT according to this setting
it defaults to `'/mediaroot/'` in the example config.

for instance you can have this kind of setting:

```
MEDIA_ROOT = '/home/pytition/www/mediaroot'
MEDIA_URL = '/mediaroot/'
```

and then in your apache config:

```
alias /mediaroot /home/pytition/www/mediaroot
```

or in your nginx config:

```
location /mediaroot {
    alias /home/pytition/www/mediaroot;
}
```

See also:

https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-MEDIA_URL for more details from django documentation

SECRET_KEY = ''

Enter a **random**, **unique** and **private** secret key.
Pytition won't start without it.
Never share it, don't commit in git.
See https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-SECRET_KEY for more details from Django documentation
To generate it, you can use the following command from your virtualenv with Django installed:

```
$ python3 -c "from django.core.management.utils import
get_random_secret_key as g; print(g())"
```

Example:

```
SECRET_KEY = 'my secret key here'
```

STATIC_ROOT = None

Enter the file system path to the directory that will be used to serve your static files.
This must be an initially empty directory.
You must also configure a web server (apache, nginx or other) to serve
the content of this directory according to your *STATIC_URL* setting
which default is `'/static/'` in the example config.

For instance you can have this kind of setting:

```
STATIC_ROOT = '/home/pytition/www/static'
STATIC_URL = '/static/'
```

And then in your apache config:

```
Alias /static /home/pytition/www/static
```

Or in your nginx config:

```
location /static {
    alias /home/pytition/www/static;
}
```

See also:

https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-STATIC_ROOT for more details from Django Documentation

STATIC_URL = '/static/'

enter the prefix that will be used for the url to refer to static files.
it must end with a forward slash '/'.
you must also configure a web server (apache, nginx or other) to serve
the content of the directory configured as STATIC_ROOT according to this setting
it defaults to `'/static/'` in the example config.

for instance you can have this kind of setting:

```
STATIC_ROOT = '/home/pytition/www/static'
STATIC_URL = '/static/'
```

and then in your apache config:

```
alias /static /home/pytition/www/static
```

or in your nginx config:

```
location /static {
    alias /home/pytition/www/static;
}
```

See also:

https://docs.djangoproject.com/en/2.2/ref/settings/#std:setting-static_url for more details from django documentation

3.2 Not mandatory but important settings

You are **highly encouraged** to set the following variables in a production environment:

3.2.1 Pytition specific settings

`USE_MAIL_QUEUE = False`

Set it to `True` if you want email sending to retry upon failure.
Email transmition naturally have retries *if the first SMTP server accepts it*
If your SMTP server refuses to handle the email (anti-flood throttle?) then it is up to you to retry, and this is what the mail queue does for you.
This is especially needed if you don't own the first-hop SMTP server and cannot configure it to always accept your emails regardless of the sending frequency.
It is **HIGHLY** recommended to set this to `True`.
If you chose to use the mail queue, you must also either

- set a cron job (automatic task execution), or
- serve the Django app through uwsgi (recommended setup)

Warning: The first time you switch this setting from `False` to `True`, you must run the `DJANGO_SETTINGS_MODULE=pytition.settings.config python3 pytition/manage.py migrate` command again. Beware to run it while being in your virtualenv.

`ALLOW_REGISTER = True`

Whether you want to allow anyone to create an account and host petitions

on your Pytition instance.
Set it to `False` for a private instance.
Set it to `True` for a public instance.

DEFAULT_NOREPLY_MAIL = 'noreply@domain.tld'

Default address for 'Reply to' field in mail sent on account creation

3.2.2 Django settings

The following settings are important to set so that the email sent by Pytition are less likely to be considered as spam/junk. You should configure a real SMTP email account and not just rely on "fake" email address from local sendmail:

- DEFAULT_FROM_EMAIL
- SERVER_EMAIL
- EMAIL_HOST
- EMAIL_HOST_PASSWORD
- EMAIL_HOST_USER
- EMAIL_PORT
- EMAIL_USE_TLS
- EMAIL_USE_SSL
- others when necessary

3.3 Other optional settings

Those are things you can configure to customize your Pytition instance:

SITE_NAME = 'Pytition'

The name of your Pytition instance.

FOOTER_TEMPLATE = None

Leave it set to None for no footer.
This should contain the relative path to your footer template.
That would be the location for any "legal mention" / "GDPR" / "TOS" link.

Example:

```
FOOTER_TEMPLATE = 'layouts/footer.html.example'
```

DISABLE_USER_PETITION = False

If set to True, users won't be able to create petitions in their name, but only for an organization

RESTRICT_ORG_CREATION = False

If set to True, regular users won't be able to create new organizations.
Only superusers will be allowed to

CHAPTER

FOUR

UPDATE

4.1 Backup your files

Backup your media files, those are the pictures uploaded by your users in the petition contents and metadata.

The files to backup are in the mediaroot directory that you configured in your settings in the *MEDIA_ROOT* variable.

```
$ source path/to/pytition_venv/bin/activate
$ export DJANGO_SETTINGS_MODULE="pytition.settings.config" # path to your config
$ backup_dir=pytition_backup_$(date +%Y%m%d_%H%M%S)
$ mediaroot_dir=$(python3 pytition/manage.py shell -c 'from django.conf import␣
↪settings; print(settings.MEDIA_ROOT)')
$ rsync -av $mediaroot_dir $backup_dir
```

4.2 Backup your Database

For this, I would advise to use the tools provided with your database server.

- SQLite: just copy your .db file and you're done!
- PostgreSQL: use *pg_dump* to backup and *psql* to restore
- MariaDB / MySQL: use *mysqldump* to backup and *mysql* to restore

You can also try to backup using the django tool:

```
$ source path/to/pytition_venv/bin/activate
$ export DJANGO_SETTINGS_MODULE="pytition.settings.config" # path to your config
$ # let's dump data
$ python3 pytition/manage.py dumpdata --all --output data.json
$ # now let's restore it
$ python3 pytition/manage.py loaddata data.json
```

Warning: Always *test* your backup mechanism. If not tested, you can only suppose your backups are worthless. You need to try to restore them on a dummy and empty instance, in order to make sure the backup is OK. Untested backups do not work.

4.3 Update to a newer Pytition version

You can simply run the *update* command of the *manage.py* CLI:

```
$ source pytition_venv/bin/activate
$ python3 pytition/manage.py update
```

Or go through the following document and do it manually.

Download latest Pytition release tarball or update your git clone:

```
$ git stash && git pull
$ version=$(curl -s https://api.github.com/repos/pytition/pytition/releases/latest |␣
↪grep "tag_name" | cut -d : -f2,3 | tr -d \" | tr -d ,)
$ git checkout $version
```

Then upgrade Pytition's dependencies:

```
$ source pytition_venv/bin/activate
(pytition_venv) $ pip3 install --upgrade -r requirements.txt
```

Then update your database scheme, update static files, compile new translation files:

```
$ export DJANGO_SETTINGS_MODULE="pytition.settings.config" # path to your config
$ python3 pytition/manage.py migrate
$ python3 pytition/manage.py collectstatic
$ python3 pytition/manage.py compilemessages
```

Then restart your web server, be it apache or nginx, and also your application server (uWSGI). Congratulations! You should now be OK with a brand new Pytition release!

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX