Install Taiga in Production

Table of Contents

1. Docker

This is the easiest and **recommended** way to run Taiga in production. This document explains how to deploy a full Taiga service for a production environment with `docker`.

1.1. Requirements and caveats

Prior to start the installation, ensure you have installed:

- `docker`: version >= 17.09.0+

- `docker-compose`: version >= 1.27.0+

Additionally, it's necessary to have familiarity with Docker, docker-compose and Docker repositories.

1.2. Get repository

Clone this repository.

```
$ cd taiga-docker/
$ git checkout stable
```

1.3. Configuration and Customisation with Environment Variables

This configuration is likely to suit what you need. Edit environment variables in `docker-compose.yml` and `docker-compose-inits.yml`. Have in mind that some of the variables are in both files, and you need to edit both.

Configuration variables are in `docker-compose.yml` with default values that we strongly recommend that you change. Those variables are needed to run Taiga. Apart from this configuration, you can have some **customisation** in Taiga, that add features that are disabled by default. Find those variables in **Customisation** section and add the corresponding environment variables whenever you want to enable them.

1.4. Configuration

Database configuration

These vars will be used to create the database for Taiga and connect to it.

Important: these vars should have the same values in `taiga-back` and `taiga-db`.

Service: taiga-db

```
POSTGRES_DB: taiga
POSTGRES_USER: taiga
POSTGRES_PASSWORD: taiga
```

Service: taiga-back

```
POSTGRES_DB: taiga
POSTGRES_USER: taiga
POSTGRES_PASSWORD: taiga
```

Additionally, you can also configure `POSTGRES_PORT` in `taiga-back`. Defaults to '5432'.

Taiga Settings

Service: taiga-back

```
TAIGA_SECRET_KEY: taiga-back-secret-key
TAIGA_SITES_SCHEME: http
TAIGA_SITES_DOMAIN: localhost:9000
```

Service: taiga-events

```
TAIGA_SECRET_KEY: taiga-back-secret-key
```

Service: taiga-protected

```
TAIGA_SECRET_KEY: taiga-back-secret-key
```

Service: taiga-front

```
TAIGA_URL: "http://localhost:9000"
TAIGA_WEBSOCKETS_URL: "ws://localhost:9000"
```

`TAIGA_SECRET_KEY` is the secret key of Taiga. Should be the same as this var in `taiga-back`, `taiga-events` and `taiga-protected`. `TAIGA_URL` is where this Taiga instance should be served. It should be the same as `TAIGA_SITES_SCHEME`://`TAIGA_SITES_DOMAIN`. `TAIGA_WEBSOCKETS_URL` is used to connect to the events. This should have the same value as `TAIGA_SITES_DOMAIN`, ie: ws://taiga.mycompany.com.

Session Settings

Taiga doesn't use session cookies in its API as it stateless. However, the Django Admin (`/admin/`) uses session cookie for authentication. By default, Taiga is configured to work behind HTTPS. If you're using HTTP (despite de strong recommendations against it), you'll need to configure the following environment variables so you can access the Admin:

Service: taiga-back

```
SESSION_COOKIE_SECURE: "False"
CSRF_COOKIE_SECURE: "False"
```

More info about those variables can be found here.

Email Settings

By default, email is configured with the **console** backend, which means that the emails will be shown in the stdout. If you have an smtp service, uncomment the "Email settings" section in `docker-compose.yml` and configure those environment variables:

Service: taiga-back

```
EMAIL_BACKEND: "django.core.mail.backends.smtp.EmailBackend"
DEFAULT_FROM_EMAIL: "no-reply@example.com"
EMAIL_HOST: "smtp.host.example.com"
EMAIL_PORT: 587
EMAIL_HOST_USER: "user"
EMAIL_HOST_PASSWORD: "password"
EMAIL_USE_TLS: "True"
EMAIL_USE_SSL: "True"
```

Uncomment `EMAIL_BACKEND` variable, but do not modify unless you know what you're doing.

Telemetry Settings

Telemetry anonymous data is collected in order to learn about the use of Taiga and improve the platform based on real scenarios.

Service: taiga-back

```
ENABLE_TELEMETRY: "True"
```

You can opt out by setting this variable to "False". By default is "True".

Rabbit settings

These variables are used to leave messages in the rabbitmq services. These variables should be the same as in `taiga-back`, `taiga-async`, `taiga-events`, `taiga-async-rabbitmq` and `taiga-events-rabbitmq`.

Service: taiga-back

```
RABBITMQ_USER: taiga
RABBITMQ_PASS: taiga
```

Two other variables `EVENTS_PUSH_BACKEND_URL` and `CELERY_BROKER_URL` can also be used to set the events push backend URL and celery broker URL.

```
EVENTS_PUSH_BACKEND_URL: "amqp://taiga:taiga@taiga-events-rabbitmq:5672/taiga"
CELERY_BROKER_URL: "amqp://taiga:taiga@taiga-async-rabbitmq:5672/taiga"
```

Service: taiga-events

```
RABBITMQ_USER: taiga
RABBITMQ_PASS: taiga
```

Service: taiga-async-rabbitmq

```
RABBITMQ_ERLANG_COOKIE: secret-erlang-cookie
RABBITMQ_DEFAULT_USER: taiga
RABBITMQ_DEFAULT_PASS: taiga
RABBITMQ_DEFAULT_VHOST: taiga
```

Service: taiga-events-rabbitmq

```
RABBITMQ_ERLANG_COOKIE: secret-erlang-cookie
RABBITMQ_DEFAULT_USER: taiga
RABBITMQ_DEFAULT_PASS: taiga
RABBITMQ_DEFAULT_VHOST: taiga
```

Taiga protected settings

Service: taiga-protected

```
MAX_AGE: 360
```

The attachments will be accesible with a token during MAX_AGE (in seconds). After that, the token will expire.

1.5. Customisation

All these features are disabled by default. You should add the corresponding environment variables with a proper value to enable them.

Registration Settings

Service: taiga-back

```
PUBLIC_REGISTER_ENABLED: "True"
```

Service: taiga-front

```
PUBLIC_REGISTER_ENABLED: "true"
```

If you want to allow a public register, configure this variable to "True". By default is "False". Should be the same as this var in `taiga-front` and `taiga-back`.

Important: Taiga (in its default configuration) disables both Gitlab or Github oauth buttons whenever the public registration option hasn't been activated. To be able to use Github/ Gitlab login/registration, make sure you have public registration activated on your Taiga instance.

Github settings

Used for login with Github.

Follow the documentation in Github, when save application Github displays the ID and Secret.

Set variables in docker-compose.yml:

Note `ENABLE_GITHUB_AUTH` and `GITHUB_API_CLIENT_ID / GITHUB_CLIENT_ID` should have the same value in `taiga-back` and `taiga-front` services.

Service: taiga-back

```
ENABLE_GITHUB_AUTH: "True"
GITHUB_API_CLIENT_ID: "github-client-id"
GITHUB_API_CLIENT_SECRET: "github-client-secret"
PUBLIC_REGISTER_ENABLED: "True"
```

Service: taiga-front

```
ENABLE_GITHUB_AUTH: "true"
GITHUB_CLIENT_ID: "github-client-id"
PUBLIC_REGISTER_ENABLED: "true"
```

Gitlab settings

Used for login with GitLab.

Follow the documentation in Gitlab, when save application GitLab displays the ID and Secret.

Set variables in docker-compose.yml:

Note `ENABLE_GITLAB_AUTH`, `GITLAB_API_CLIENT_ID / GITLAB_CLIENT_ID` and `GITLAB_URL` should have the same value in `taiga-back` and `taiga-front` services.

Service: taiga-back

```
ENABLE_GITLAB_AUTH: "True"
GITLAB_API_CLIENT_ID: "gitlab-client-id"
GITLAB_API_CLIENT_SECRET: "gitlab-client-secret"
GITLAB_URL: "gitlab-url"
PUBLIC_REGISTER_ENABLED: "True"
```

Service: taiga-front

```
ENABLE_GITLAB_AUTH: "true"
GITLAB_CLIENT_ID: "gitlab-client-id"
GITLAB_URL: "gitlab-url"
PUBLIC_REGISTER_ENABLED: "true"
```

Slack Settings

Service: taiga-back

```
ENABLE_SLACK: "True"
```

Service: taiga-front

```
ENABLE_SLACK: "true"
```

Enable Slack integration in your Taiga instance. By default is "False". Should have the same value as this variable in `taiga-front` and `taiga-back`.

Github importer

Service: taiga-back

```
ENABLE_GITHUB_IMPORTER: "True"
GITHUB_IMPORTER_CLIENT_ID: "client-id-from-github"
GITHUB_IMPORTER_CLIENT_SECRET: "client-secret-from-github"
```

Service: taiga-front

```
ENABLE_GITHUB_IMPORTER: "true"
```

Jira importer

Service: taiga-back

```
ENABLE_JIRA_IMPORTER: "True"
JIRA_IMPORTER_CONSUMER_KEY: "consumer-key-from-jira"
JIRA_IMPORTER_CERT: "cert-from-jira"
JIRA_IMPORTER_PUB_CERT: "pub-cert-from-jira"
```

Service: taiga-front

```
ENABLE_JIRA_IMPORTER: "true"
```

Trello importer

Service: taiga-back

```
ENABLE_TRELLO_IMPORTER: "True"
TRELLO_IMPORTER_API_KEY: "api-key-from-trello"
TRELLO_IMPORTER_SECRET_KEY: "secret-key-from-trello"
```

Service: taiga-front

```
ENABLE_TRELLO_IMPORTER: "true"
```

1.6. Advanced configuration and customisation

In an advanced configuration, you ignore the environment variables in `docker-compose.yml` or `docker-compose-inits.yml`.

Map a `config.py` file

From taiga-back download the file `settings/config.py.prod.example` and rename it:

```
mv settings/config.py.prod.example settings/config.py
```

Edit it with your own configuration:

- connection to PostgreSQL
- connection to RabbitMQ for `taiga-events` and `taiga-async`
- credentials for email
- Enable/disable anonymous telemetry

Check as well the rest of the configuration if you need to enable some advanced features.

Map the file into `/taiga-back/settings/config.py`. Have in mind that you have to map it both in `docker-compose.yml` and `docker-compose-inits.yml`. You can check the `x-volumes` section in docker-compose.yml with an example.

Map a `conf.json` file

From taiga-front download the file `dist/conf.example.json` and rename it:

```
mv dist/conf.example.json dist/conf.json
```

Edit it with your own configuration and map the file into `/taiga-front/dist/config.py`.

1.7. Configure an admin user

```
$ docker-compose up -d

$ docker-compose -f docker-compose.yml -f docker-compose-inits.yml run --rm taiga-
manage createsuperuser
```

1.8. Up and running

Once everything has been installed, launch all the services and check the result:

```
$ docker-compose up -d
```

If you're testing it in your own machine, you can access your Taiga Platform in **http://localhost:9000**. If you're deploying in a server, you'll need to configure hosts and nginx as described later.

1.9. Configure the proxy

Your host configuration needs to make a proxy to `http://localhost:9000`. Example:

```
server {
  server_name taiga.mycompany.com;

  ...

  location / {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Scheme $scheme;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_redirect off;
    proxy_pass http://localhost:9000/;
  }

  # Events
  location /events {
      proxy_pass http://localhost:9000/events;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
      proxy_set_header Host $host;
      proxy_connect_timeout 7d;
      proxy_send_timeout 7d;
      proxy_read_timeout 7d;
  }
}
```

2. From source code

2.1. Introduction

This document explains how to deploy a full Taiga service for a production environment. A Taiga service consists of multiple Taiga modules which altogether make the Taiga platform.

The standard Taiga platform consists of several modules, and each one has its own dependencies

both at compile time and runtime:

- **taiga-back** (API)
- **taiga-async-tasks** (async tasks, like bulk email or exports generation)
- **taiga-front-dist** (frontend)
- **taiga-events** (websockets gateway)
- **taiga-protected** (protected attachments)

Each module can be run on a unique machine or all of them can be installed to a different machine as well. In this tutorial we will setup everything on a single machine. This type of setup should suffice for small/medium production environments with low traffic.

2.2. Pre-requisites

- A clean, recently updated **Ubuntu 20.04** image
- At least 1GB RAM
- At least 20GB of free storage
- TLS certificate to serve Taiga with HTTPS

Taiga installation must be done with a "regular" user, never with root!

During the tutorial, the following conditions are assumed:

- **IP:** `80.88.23.45`
- **Hostname:** `example.com` (which points to 80.88.23.45)
- **Username:** `taiga`
- **Working directory:** `/home/taiga/` (default for user `taiga`)

2.3. Dependencies

The typical Taiga setup described in this documentation depends on the following standalone major software installed separately from Taiga:

- Python 3 - taiga-back, taiga-async and taiga-protected (Python >= 3.6)
- Node.js - taiga-events
- NGINX - web server and reverse proxy
- PostgreSQL - database (PostgreSQL >= 9.4)
- RabbitMQ - message broker, for taiga-async and taiga-events

2.4. Install System Dependencies

Install the following dependencies:

```
sudo apt-get update
sudo apt-get install -y build-essential binutils-doc autoconf flex bison libjpeg-dev
sudo apt-get install -y libfreetype6-dev zlib1g-dev libzmq3-dev libgdbm-dev
libncurses5-dev
sudo apt-get install -y automake libtool curl git tmux gettext
sudo apt-get install -y nginx
sudo apt-get install -y rabbitmq-server
```

Install PostgreSQL and remember to start the database server:

```
sudo apt-get install -y postgresql-12 postgresql-contrib-12 postgresql-doc-12
postgresql-server-dev-12
sudo pg_ctlcluster 12 main start
```

Python 3 must be installed along with a few third-party libraries:

```
sudo apt-get install -y python3 python3-pip python3-dev python3-venv
sudo apt-get install -y libxml2-dev libxslt-dev
sudo apt-get install -y libssl-dev libffi-dev
```

Install Node.js

```
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt-get install -y nodejs
```

2.5. Create a user `taiga`

Create a user with root privileges named `taiga`:

```
sudo adduser taiga
sudo adduser taiga sudo
sudo su taiga
cd ~
```

NOTE	Do **not** change back to the root user (`uid=0`) at this point. Taiga deployment must be finished with the `taiga` user!

2.6. Configuring PostgreSQL and RabbitMQ

Configure PostgreSQL with the initial user and database:

```
sudo -u postgres createuser taiga --interactive --pwprompt
sudo -u postgres createdb taiga -O taiga --encoding='utf-8' --locale=en_US.utf8
--template=template0
```

Create a rabbitmquser named `taiga` and a virtualhost for RabbitMQ (taiga-events and async tasks)

```
sudo rabbitmqctl add_user rabbitmquser rabbitmqpassword
sudo rabbitmqctl add_vhost taiga
sudo rabbitmqctl set_permissions -p taiga rabbitmquser ".*" ".*" ".*"
```

NOTE	As the password will be used inside the Postgresql URL later, use only web safe characters: a-z, A-Z, 0-9, and - . _ ~

2.7. Backend Setup

This section describes the installation and configuration of the **taiga-back** and **taiga-async** modules which serves the REST API endpoints and the async tasks respectively.

Get the code:

```
cd ~
git clone https://github.com/kaleidos-ventures/taiga-back.git taiga-back
cd taiga-back
git checkout stable
```

Create a virtualenv:

```
python3 -m venv .venv --prompt taiga-back
source .venv/bin/activate
(taiga-back) pip install --upgrade pip wheel
```

Install all Python dependencies:

```
(taiga-back) pip install -r requirements.txt
```

Install taiga-contrib-protected:

```
(taiga-back) pip install git+https://github.com/kaleidos-ventures/taiga-contrib-
protected.git@stable#egg=taiga-contrib-protected
```

Settings file:

Create a `settings/config.py` file based on the example provided:

```
cp settings/config.py.prod.example settings/config.py
```

Edit the file and configure:

- connection to PostgreSQL
- connection to RabbitMQ for `taiga-events` and `taiga-async`

- credentials for email
- Enable/disable anonymous telemetry
- Enable/disable public registration

Check as well the rest of the configuration if you need to enable some adavanced features.

Execute all migrations to populate the database with basic necessary initial data:

```
source .venv/bin/activate
(taiga-back) DJANGO_SETTINGS_MODULE=settings.config python manage.py migrate --noinput
# create an administrator with strong password
(taiga-back) CELERY_ENABLED=False DJANGO_SETTINGS_MODULE=settings.config python
manage.py createsuperuser
(taiga-back) DJANGO_SETTINGS_MODULE=settings.config python manage.py loaddata
initial_project_templates
(taiga-back) DJANGO_SETTINGS_MODULE=settings.config python manage.py compilemessages
(taiga-back) DJANGO_SETTINGS_MODULE=settings.config python manage.py collectstatic
--noinput
```

OPTIONAL: If you would like to have some example data loaded into Taiga, execute the following command to populate the database with sample projects and random data (useful for demos):

```
(taiga-back) CELERY_ENABLED=False DJANGO_SETTINGS_MODULE=settings.config python
manage.py sample_data
```

Verification

To make sure that everything works, execute the following commands to run the backend in development mode for a quick test:

```
source .venv/bin/activate
(taiga-back) DJANGO_SETTINGS_MODULE=settings.config python manage.py runserver
```

Open your browser at http://localhost:8000/api/v1/. If your configuration is correct, you will see a JSON representation of REST API endpoints. Open your browser at http://localhost:8000/admin/ and log-in with your admin credentials. Stop the developtment server (Ctrl+C) before continuing.

2.8. Frontend Setup

This section describes the installation and configuration of the **taiga-front** module which serves the frontend application.

Get the code

```
cd ~
git clone https://github.com/kaleidos-ventures/taiga-front-dist.git taiga-front-dist
cd taiga-front-dist
git checkout stable
```

Copy the example config file:

```
cp ~/taiga-front-dist/dist/conf.example.json ~/taiga-front-dist/dist/conf.json
```

Edit the example configuration following the pattern below (replace with your own details):

```
{
    "api": "https://example.com/api/v1/",
    "eventsUrl": "wss://example.com/events",
    "debug": "true",
    "publicRegisterEnabled": true,
    "feedbackEnabled": true,
    "privacyPolicyUrl": null,
    "termsOfServiceUrl": null,
    "GDPRUrl": null,
    "maxUploadFileSize": null,
    "contribPlugins": []
}
```

Having **taiga-front-dist** downloaded and configured is insufficient. The next step is to expose the code (in **dist** directory) under a static file web server.

In this tutorial We use **NGINX** as a static file web server and reverse-proxy. The configuration of NGINX is explained later.

2.9. Events Setup

This section provides instructions on downloading **taiga-events**, installing its dependencies and configuring it for use in production:

The **taiga-events** module is the Taiga websocket server which allows **taiga-front** to show realtime changes in the backlog, taskboard, kanban and issues listing.

Get the code:

```
cd ~
git clone https://github.com/kaleidos-ventures/taiga-events.git taiga-events
cd taiga-events
git checkout stable
```

Install the required JavaScript dependencies:

```
npm install
```

Create `.env` file based on the provided example.

```
cp .env.example .env
```

Update it with your RabbitMQ URL and your unique secret key. Your final `.env` should look similar to the following example:

```
RABBITMQ_URL="amqp://rabbitmquser:rabbitmqpassword@rabbitmqhost:5672/taiga"
SECRET="taiga-back-secret-key"
WEB_SOCKET_SERVER_PORT=8888
APP_PORT=3023
```

The `secret` value in `.env` must be the same as the `SECRET_KEY` in `~/taiga-back/settings/config.py`.

2.10. Taiga protected Setup

This section describes the installation and configuration of the **taiga-protected** modules which protects the attachments from external downloads.

Get the code:

```
cd ~
git clone https://github.com/kaleidos-ventures/taiga-protected.git taiga-protected
cd taiga-protected
git checkout stable
```

Create a virtualenv:

```
python3 -m venv .venv --prompt taiga-protected
source .venv/bin/activate
(taiga-protected) pip install --upgrade pip wheel
```

Install all Python dependencies:

```
(taiga-protected) pip install -r requirements.txt
```

Copy the example config file:

```
cp ~/taiga-protected/env.sample ~/taiga-protected/.env
```

The `SECRET_KEY` value in `.env` must be the same as the `SECRET_KEY` in `~/taiga-back/settings/config.py`.

2.11. Start Taiga

Now it's time to create the different systemd services to serve different modules of Taiga.

Create a new systemd file at `/etc/systemd/system/taiga.service` *to run* **taiga-back***:*

```
[Unit]
Description=taiga_back
After=network.target

[Service]
User=taiga
WorkingDirectory=/home/taiga/taiga-back
ExecStart=/home/taiga/taiga-back/.venv/bin/gunicorn --workers 4 --timeout 60 --log
-level=info --access-logfile - --bind 0.0.0.0:8001 taiga.wsgi
Restart=always
RestartSec=3

Environment=PYTHONUNBUFFERED=true
Environment=DJANGO_SETTINGS_MODULE=settings.config

[Install]
WantedBy=default.target
```

Reload the systemd daemon and start the `taiga` service:

```
sudo systemctl daemon-reload
sudo systemctl start taiga
sudo systemctl enable taiga
```

To verify that the service is running, execute the following command:

```
sudo systemctl status taiga
```

Create a new systemd file at `/etc/systemd/system/taiga-async.service` *to run* **taiga-async***:*

```
[Unit]
Description=taiga_async
After=network.target

[Service]
User=taiga
WorkingDirectory=/home/taiga/taiga-back
ExecStart=/home/taiga/taiga-back/.venv/bin/celery -A taiga.celery worker -B
--concurrency 4 -l INFO
Restart=always
RestartSec=3
ExecStop=/bin/kill -s TERM $MAINPID

Environment=PYTHONUNBUFFERED=true
Environment=DJANGO_SETTINGS_MODULE=settings.config

[Install]
WantedBy=default.target
```

Reload the systemd daemon and start the `taiga-async` service:

```
sudo systemctl daemon-reload
sudo systemctl start taiga-async
sudo systemctl enable taiga-async
```

To verify that the service is running, execute the following command:

```
sudo systemctl status taiga-async
```

Create a new systemd file at `/etc/systemd/system/taiga-events.service` *to run* **taiga-events***:*

```
[Unit]
Description=taiga_events
After=network.target

[Service]
User=taiga
WorkingDirectory=/home/taiga/taiga-events
ExecStart=npm run start:production
Restart=always
RestartSec=3

[Install]
WantedBy=default.target
```

Reload the systemd daemon and start the `taiga-events` service:

```
sudo systemctl daemon-reload
sudo systemctl start taiga-events
sudo systemctl enable taiga-events
```

To verify that the service is running, execute the following command:

```
sudo systemctl status taiga-events
```

Create a new systemd file at `/etc/systemd/system/taiga-protected.service` *to run* **taiga-protected***:*

```
[Unit]
Description=taiga_protected
After=network.target

[Service]
User=taiga
WorkingDirectory=/home/taiga/taiga-protected
ExecStart=/home/taiga/taiga-protected/.venv/bin/gunicorn --workers 4 --timeout 60
--log-level=info --access-logfile - --bind 0.0.0.0:8003 server:app
Restart=always
RestartSec=3

Environment=PYTHONUNBUFFERED=true

[Install]
WantedBy=default.target
```

Reload the systemd daemon and start the `taiga-protected` service:

```
sudo systemctl daemon-reload
sudo systemctl start taiga-protected
sudo systemctl enable taiga-protected
```

To verify that the service is running, execute the following command:

```
sudo systemctl status taiga-protected
```

2.12. Expose Taiga with NGINX

The recommended way to serve Taiga is to use NGINX proxy server.

Remove the default NGINX config file to avoid collision with Taiga:

```
sudo rm /etc/nginx/sites-enabled/default
```

Create the logs folder (mandatory)

```
mkdir -p ~/logs
```

To configure a new NGINX virtualhost for Taiga, create and edit the `/etc/nginx/conf.d/taiga.conf` *file, as follows:*

```
server {
    listen 80 default_server;
    server_name _;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 default_server;
    server_name _;  #  See http://nginx.org/en/docs/http/server_names.html

    large_client_header_buffers 4 32k;
    client_max_body_size 50M;
    charset utf-8;

    access_log /home/taiga/logs/nginx.access.log;
    error_log /home/taiga/logs/nginx.error.log;

    # Frontend
    location / {
        root /home/taiga/taiga-front-dist/dist/;
        try_files $uri $uri/ /index.html;
    }

    # Backend
    location /api {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8001/api;
        proxy_redirect off;
    }

    # Admin access (/admin/)
    location /admin {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8001$request_uri;
        proxy_redirect off;
```

```
    }

    # Static files
    location /static {
        alias /home/taiga/taiga-back/static;
    }

    # Media
    location /_protected {
        internal;
        alias /home/taiga/taiga-back/media/;
        add_header Content-disposition "attachment";
    }

    # Unprotected section
    location /media/exports {
        alias /home/taiga/taiga-back/media/exports/;
        add_header Content-disposition "attachment";
    }

    location /media {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8003/;
        proxy_redirect off;
    }

    # Events
    location /events {
        proxy_pass http://127.0.0.1:8888/events;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_connect_timeout 7d;
        proxy_send_timeout 7d;
        proxy_read_timeout 7d;
    }

    # TLS
    # Configure your TLS following the best practices inside your company
}
```

Execute the following command to verify the NGINX configuration and to track any error in the service:

```
sudo nginx -t
```

Finally, restart the `nginx` service:

```
sudo systemctl restart nginx
```

Restart all Taiga services after updating the configuration:

```
sudo systemctl restart 'taiga*'
```

Now you should have the service up and running on: `https://example.com/`

2.13. Other methods to expose Taiga

Caddy server

It's possible to serve Taiga with Caddy as well, following the next guides:

- install caddy >= 2.4.1
- create a symlink from `media` to `_protected`

```
cd ~/taiga-back
ln -s media/ _protected
```

- use a Caddyfile based on this

2.14. Extend Taiga

With this installation, you have access to a fair amount of features of Taiga. However, you may want to extend it with other functionalities or plugins, such Slack integration or login with Github. To extend Taiga, check all the available options at https://resources.taiga.io/extend/how-to-extend-taiga/.

2.15. Troubleshooting

If you face any issue during or after installing Taiga, please check the content of the following files:

- `/etc/nginx/conf.d/taiga.conf`
- `/etc/systemd/system/taiga.service`
- `/etc/systemd/system/taiga-async.service`
- `/etc/systemd/system/taiga-events.service`
- `/etc/systemd/system/taiga-protected.service`
- `/home/taiga/taiga-back/settings/config.py`
- `/home/taiga/taiga-front-dist/dist/conf.json`
- `/home/taiga/taiga-events/.env`
- `/home/taiga/taiga-protected/.venv`
- The result of command `sudo systemctl status 'taiga*'`

Execute the following commands to check the status of services used by Taiga:

```
sudo systemctl status nginx
sudo systemctl status rabbitmq-server
sudo systemctl status postgresql
```

Check If you see any error in the service statuses and make sure all service status is `Active: active (running)`.