Taiga: Setup Production Environment

Table of Contents

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 three main modules, and each one has its own dependencies both at compile time and runtime:

- **taiga-back** (backend/API)
- **taiga-front-dist** (frontend)
- **taiga-events** (websockets gateway) (optional)

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, and will be installing all three main Taiga modules.
This type of setup should suffice for small/medium production environments with low traffic.

2. Overview

This tutorial assumes that you are using a clean, recently updated **Ubuntu 16.04** image.

Due to the nature of the frontend, Taiga is accessed through a domain/public IP address, because the frontend application runs in your browser. The frontend must be able to communicate with the backend/API, therefore both the frontend and the backend must be accessible through a domain/public IP address too.

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`
- **System memory:** `>=1GB` (needed for compilation of lxml)
- **Working directory:** `/home/taiga/` (default for user `taiga`)

Changing the user from `taiga` to something else is not recommended at any point during deployment unless you're well aware of what you're doing.
Changing user may result in unexpected behavior or failed deployment!

2.1. Dependencies

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

- Python 3 - programming language and runtime environment of **taiga-back**
- NGINX - web server and reverse proxy
- PostgreSQL - database
- Gunicorn - Python WSGI HTTP server
- RabbitMQ - message broker
- Redis - in-memory key-value database
- Node.js - JavaScript runtime (required if **taiga-events** is present)
- Virtualenv and virtualenvwrapper - Python virtual environment managers

NOTE	This list does not contain libraries and the Python dependencies required by **taiga-back**. For Python dependencies, refer to taiga-back/stable/requirements.txt.

2.2. System Architecture Description

This is a short system architecture description to help you understand the way Taiga is built and

works. Before you go any further in the installation procedure, make sure you read this description to get a high-level overview of the architecture.

Taiga consists of 2 core modules:

- **taiga-back**
- **taiga-front**

taiga-back is built with Django and written in Python 3. This serves the API endpoints for the frontend.
taiga-front is written mostly in AngularJS and CoffeeScript. This consumes the API endpoints provided by the backend.

The Python backend is exposed by gunicorn (port `8001/tcp`), which is a Python WSGI HTTP server. The process manager is systemd, which runs gunicorn and taiga-back together. Technically the backend communicates with the RDBMS, and through the frontend it allows the user to use the features of Taiga. The communication between the front- and backend is done using REST APIs.

The **backend** is publicly exposed by NGINX which acts as a reverse-proxy in this case.
The **frontend** is located in the `dist` folder and is also exposed publicly by NGINX which acts as a static webserver in this case.

3. Prerequisites

This guide describes a configuration of Taiga which consists of three Taiga modules. Each module has its own dependencies on various packages.
The following subsections list the required packages and provide instructions on their installation and configuration for a successful Taiga deployment.

3.1. Installing Dependencies

Execute the following commands to install all dependencies for all modules. Optional dependencies are marked with an inline comment, i.e.: `# Optional: taiga-events`.

Essential packages:

```
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 redis-server  # Optional: taiga-events or
async tasks
```

The **taiga-back** *module depends on PostgreSQL (>= 9.4) as its database:*

```
sudo apt-get install -y postgresql-9.5 postgresql-contrib-9.5
sudo apt-get install -y postgresql-doc-9.5 postgresql-server-dev-9.5
```

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

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

NOTE	**virtualenvwrapper** helps keeping the system clean of third party libraries, installed with the language package manager by installing these packages in an isolated virtual environment.

Restart the shell or type `bash` and press `Enter` to reload the shell environment with the new virtualenvwrapper variables and functions.

This step is mandatory before continuing with the deployment!

Create a user with root privileges named `taiga`*:*

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

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

3.2. Configuring Dependencies

Configure PostgreSQL with the initial user and database:

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

Create a user named `taiga`*, and a virtualhost for RabbitMQ (Optional: taiga-events or async tasks)*

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

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

4. Backend Setup

This section describes the installation and configuration of the **taiga-back** module which serves the REST API endpoints.

Download the code:

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

Create a new virtualenv named **taiga***:*

```
mkvirtualenv -p /usr/bin/python3 taiga
```

Install all Python dependencies:

```
pip install -r requirements.txt
```

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

```
python manage.py migrate --noinput
python manage.py loaddata initial_user
python manage.py loaddata initial_project_templates
python manage.py compilemessages
python manage.py collectstatic --noinput
```

The above migrations create an administrator account. The login credentials are the following:

- **username**: admin
- **password**: 123123

NOTE	**Attention!** Change the administrator account password to a strong password before enabling public access to your Taiga instance. If you miss this action, unauthorized persons can log in to your Taiga instance equipped with administrator privileges.

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):

```
python manage.py sample_data
```

To finish the setup of **taiga-back**, create the initial configuration file for proper static/media file resolution, optionally with email sending support:

Copy-paste the following config into `~/taiga-back/settings/local.py` and update it with your own details:

```
from .common import *

MEDIA_URL = "http://example.com/media/"
STATIC_URL = "http://example.com/static/"
SITES["front"]["scheme"] = "http"
SITES["front"]["domain"] = "example.com"

SECRET_KEY = "theveryultratopsecretkey"

DEBUG = False
PUBLIC_REGISTER_ENABLED = True

DEFAULT_FROM_EMAIL = "no-reply@example.com"
SERVER_EMAIL = DEFAULT_FROM_EMAIL

#CELERY_ENABLED = True

EVENTS_PUSH_BACKEND = "taiga.events.backends.rabbitmq.EventsPushBackend"
EVENTS_PUSH_BACKEND_OPTIONS = {"url":
"amqp://taiga:PASSWORD_FOR_EVENTS@localhost:5672/taiga"}

# Uncomment and populate with proper connection parameters
# to enable email sending. `EMAIL_HOST_USER` should end by @<domain>.<tld>
#EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
#EMAIL_USE_TLS = False
#EMAIL_HOST = "localhost"
#EMAIL_HOST_USER = ""
#EMAIL_HOST_PASSWORD = ""
#EMAIL_PORT = 25

# Uncomment and populate with proper connection parameters
# to enable GitHub login/sign-in.
#GITHUB_API_CLIENT_ID = "yourgithubclientid"
#GITHUB_API_CLIENT_SECRET = "yourgithubclientsecret"
```

4.1. Verification

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

```
workon taiga
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.

NOTE	At this stage, the backend has been deployed successfully, but to run the Python backend in production, a WSGI server must be configured first. The WSGI server configuration is explained later in this documentation.

5. Frontend Setup

Download the code from GitHub:

Download the code

```
cd ~
git clone https://github.com/taigaio/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": "http://example.com/api/v1/",
    "eventsUrl": "ws://example.com/events",
    "debug": "true",
    "publicRegisterEnabled": true,
    "feedbackEnabled": true,
    "privacyPolicyUrl": null,
    "termsOfServiceUrl": null,
    "GDPRUrl": null,
    "maxUploadFileSize": null,
    "contribPlugins": []
}
```

NOTE	Be careful using copy-paste from browser to avoid `http://` duplication.
NOTE	Also, make sure to remove the `:8000` portion of the api string. Causes 404 page on public facing sites if left in.

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.

6. Events Setup

This step is optional and can be skipped

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.

The **taiga-events** module depends on rabbitmq-server as its message broker.

Download the code:

```
cd ~
git clone https://github.com/taigaio/taiga-events.git taiga-events
cd taiga-events
```

Install Node.js

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Install the required JavaScript dependencies:

```
npm install
```

Create `config.json` *file based on the provided example.*

```
cp config.example.json config.json
```

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

```
{
    "url": "amqp://taiga:PASSWORD_FOR_EVENTS@localhost:5672/taiga",
    "secret": "theveryultratopsecretkey",
    "webSocketServer": {
        "port": 8888
    }
}
```

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

Add **taiga-events** to systemd configuration.

Copy-paste the code below into `/etc/systemd/system/taiga_events.service`

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

[Service]
User=taiga
WorkingDirectory=/home/taiga/taiga-events
ExecStart=/bin/bash -c "node_modules/coffeescript/bin/coffee index.coffee"
Restart=always
RestartSec=3

[Install]
WantedBy=default.target
```

Reload the systemd configurations:

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

7. Start and Expose Taiga

Before moving further, make sure you installed **taiga-back** and **taiga-front-dist**, however, having installed them is insufficient to run Taiga.

taiga-back should run under an application server, which in turn, should be executed and monitored by a process manager. For this task we will use **gunicorn** and **systemd** respectively.

Both **taiga-front-dist** and **taiga-back** must be exposed to the outside using a proxy/static-file web server. For this purpose, Taiga uses **NGINX**.

7.1. systemd and gunicorn

systemd is the process supervisor used by Ubuntu, and Taiga uses it to run **gunicorn**.
systemd is not only for executing processes, but it also has utils for monitoring them, collecting logs, and restarting processes if something goes wrong, and for starting processes on system boot.

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
Environment=PYTHONUNBUFFERED=true
WorkingDirectory=/home/taiga/taiga-back
ExecStart=/home/taiga/.virtualenvs/taiga/bin/gunicorn --workers 4 --timeout 60 -b
127.0.0.1:8001 taiga.wsgi
Restart=always
RestartSec=3

[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
```

7.2. NGINX

NGINX is used as a static file web server to serve **taiga-front-dist** and send proxy requests to **taiga-back**.

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 _;  #  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 files
    location /media {
        alias /home/taiga/taiga-back/media;
    }

    # 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;
    }
}
```

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
```

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

8. Optional Modules and Settings

The following items are completely optional and are up for you to configure them. Taiga-events module is also an optional feature, but its installation is part of this tutorial.

8.1. Async tasks (Optional)

The default behavior in Taiga is to do all tasks in a synchronous way, but some of them can be completely asynchronous (such as webhook or import/export tasks). To do this, you have to configure and install the Celery service requirements.

There is an exception related to exported files:

- If your instance works in **asynchronous mode**, the exported project files will be automatically removed from the storage after 24 hours.
- If your instance works in **synchronous mode**, the exported project files will not be automatically removed (taiga-back doesn't know anything about the existence of such files).

Install `rabbitmq-server` and `redis-server`:

```
sudo apt-get install -y rabbitmq-server redis-server
```

To run Celery with Taiga, include the following code line in the `local.py` file:

```
CELERY_ENABLED = True
```

You can configure other broker or results backend. If you need more info about it, check the Celery

documentation web page: http://docs.celeryproject.org/en/latest/index.html

Once you have configured Celery on Taiga, you have to add Celery to systemd configuration. See Systemd and gunicorn section.

Taiga Celery configuration block for systemd on `/etc/systemd/system/taiga_celery.service`

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

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

[Install]
WantedBy=default.target
```

Reload the circus configuration, restart taiga, then start taiga-celery:

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

8.2. Taiga Hardening - HTTPS

Follow the instructions in this section to server Taiga under HTTPS.

Place the SSL certificates in `/etc/nginx/ssl`. It is recommended to replace the original configuration for `port 80` so that users are redirected to the HTTPS version automatically.

Second, we need to generate a strong DH parameter:

```
cd /etc/ssl
sudo openssl dhparam -out dhparam.pem 4096
```

Update the configuration in `/etc/nginx/conf.d/taiga.conf` *as follows:*

```
server {
    listen 80 default_server;
    server_name _;
```

```
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl default_server;
    server_name _;

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

    index index.html;

    # 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 files
    location /media {
        alias /home/taiga/taiga-back/media;
    }
```

```
    # 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;
    }

    # SSL
    add_header Strict-Transport-Security "max-age=63072000; includeSubdomains;
preload";
    add_header Public-Key-Pins 'pin-
sha256="klO23nT2ehFDXCfx3eHTDRESMz3asj1muO+4aIdjiuY="; pin-
sha256="633lt352PKRXbOwf4xSEa1M517scpD3l5f79xMD9r9Q="; max-age=2592000;
includeSubDomains';

    ssl on;
    ssl_certificate /etc/nginx/ssl/example.com/ssl-bundle.crt;
    ssl_certificate_key /etc/nginx/ssl/example.com/example_com.key;
    ssl_session_timeout 5m;
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-
AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-
RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-
SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-
AES256-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!3DES:!MD5:!PSK';
    ssl_session_cache shared:SSL:10m;
    ssl_dhparam /etc/ssl/dhparam.pem;
    ssl_stapling on;
    ssl_stapling_verify on;

}
```

Before activating the HTTPS site, the configuration for the frontend and the backend must be updated. Change the scheme from `http` to `https` throughout the configurations.

Update ~/taiga-back/settings/local.py:

```
from .common import *

MEDIA_URL = "https://example.com/media/"
STATIC_URL = "https://example.com/static/"
SITES["front"]["scheme"] = "https"
SITES["front"]["domain"] = "example.com"

SECRET_KEY = "theveryultratopsecretkey"

DEBUG = False
PUBLIC_REGISTER_ENABLED = True

DEFAULT_FROM_EMAIL = "no-reply@example.com"
SERVER_EMAIL = DEFAULT_FROM_EMAIL

# Uncomment and populate with proper connection parameters
# to enable email sending.
#EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
#EMAIL_USE_TLS = False
#EMAIL_HOST = "localhost"
#EMAIL_HOST_USER = ""
#EMAIL_HOST_PASSWORD = ""
#EMAIL_PORT = 25

# Uncomment and populate with proper connection parameters
# to enable GitHub login/sign-in.
#GITHUB_API_CLIENT_ID = "yourgithubclientid"
#GITHUB_API_CLIENT_SECRET = "yourgithubclientsecret"
```

Update ~/taiga-front-dist/dist/conf.json:

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

Restart all Taiga services after updating the configuration:

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

Reload the `nginx` configuration:

```
sudo systemctl reload nginx
```

9. Troubleshooting

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

- `/etc/nginx/conf.d/taiga.conf`
- `/etc/systemd/system/taiga.service`
- `/etc/systemd/system/taiga_celery.service`
- `/etc/systemd/system/taiga_events.service`
- `/home/taiga/taiga-back/settings/local.py`
- `/home/taiga/taiga-front-dist/dist/conf.json`
- `/home/taiga/taiga-events/config.json`
- 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)`.