Install Taiga in Production [OLD]

Table of Contents

1. Docker

⚠ *This documentation is OUTDATED*

Intended for older versions of Taiga (prior to ver. 6.6.0). It is maintained for archival purposes only.

ℹ Updated information can be found here. If you are using this version, we strongly recommend to migrate to the new version.

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`.

Note If you're updating from serving taiga6 in subdomain to subpath, check the specific instructions at the end of this tutorial.

1.1. Requirements

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 with Environment Variables

We've exposed the **Basic configuration** and the most commonly modified settings in Taiga to an `.env` file. We strongly recommend you to change it, or at least review its content, to avoid using the default values.

Both `docker-compose.yml` and `docker-compose-inits.yml` will read from this file to populate their environment variables, so, initially you don't need to change them. Edit these files just in case you require to enable **Additional customization**, or an **Advanced configuration**.

Refer to these sections for further information.

1.4. Basic Configuration

You will find basic **configuration variables** in the `.env` file. As stated before, we encourage you to edit these values, especially those affecting the security.

Database settings

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

```
POSTGRES_DB=taiga  # PostgreSQL database name
POSTGRES_USER=taiga  # user to connect to PostgreSQL
POSTGRES_PASSWORD=taiga  # database user's password
```

URLs settings

These vars set where your Taiga instance should be served, and the security protocols to use in the communication layer.

```
TAIGA_SCHEME=http  # serve Taiga using "http" or "https" (secured) connection
TAIGA_DOMAIN=localhost:9000  # Taiga's base URL
SUBPATH=""  # it'll be appended to the TAIGA_DOMAIN (use either "" or a "/subpath")
WEBSOCKETS_SCHEME=ws  # events connection protocol (use either "ws" or "wss")
```

The default configuration assumes Taiga is being served in a **subdomain**. For example:

```
TAIGA_SCHEME=https
TAIGA_DOMAIN=taiga.mycompany.com
SUBPATH=""
WEBSOCKETS_SCHEME=wss
```

If Taiga is being served in a **subpath**, instead of a subdomain, the configuration should be something like this:

```
TAIGA_SCHEME=https
TAIGA_DOMAIN=mycompany.com
SUBPATH="/taiga"
WEBSOCKETS_SCHEME=wss
```

Secret Key settings

This variable allows you to set the secret key in Taiga, used in the cryptographic signing.

```
SECRET_KEY="taiga-secret-key"  # Please, change it to an unpredictable value!
```

Email settings

By default, emails will be printed in the standard output (`EMAIL_BACKEND=console`). If you have your own SMTP service, change it to `EMAIL_BACKEND=smtp` and configure the rest of these variables with the values supplied by your SMTP provider:

```
EMAIL_BACKEND=console  # use an SMTP server or display the emails in the console
(either "smtp" or "console")
EMAIL_HOST=smtp.host.example.com  # SMTP server address
EMAIL_PORT=587   # default SMTP port
EMAIL_HOST_USER=user  # user to connect the SMTP server
EMAIL_HOST_PASSWORD=password  # SMTP user's password
EMAIL_DEFAULT_FROM=changeme@example.com  # email address for the automated emails

# EMAIL_USE_TLS/EMAIL_USE_SSL are mutually exclusive (only set one of those to True)
EMAIL_USE_TLS=True  # use TLS (secure) connection with the SMTP server
EMAIL_USE_SSL=False  # use implicit TLS (secure) connection with the SMTP server
```

Queue manager settings

These variables are used to leave messages in the rabbitmq services.

```
RABBITMQ_USER=taiga  # user to connect to RabbitMQ
RABBITMQ_PASS=taiga  # RabbitMQ user's password
RABBITMQ_VHOST=taiga  # RabbitMQ container name
RABBITMQ_ERLANG_COOKIE=secret-erlang-cookie  # unique value shared by any connected
instance of RabbitMQ
```

Attachments settings

You can configure how long the attachments will be accessible by changing the token expiration timer. After that amount of seconds the token will expire, but you can always get a new attachment url with an active token.

```
ATTACHMENTS_MAX_AGE=360  # token expiration date (in seconds)
```

Telemetry settings

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

```
ENABLE_TELEMETRY=True
```

You can opt out by setting this variable to False. By default, it's True.

1.5. Additional customization

All these customization options are by default disabled and require you to edit `docker-compose.yml` and `docker-compose-inits.yml`.

You should add the corresponding environment variables in the proper services with a valid value in order to enable them. Please, do not modify it unless you know what you're doing.

Session cookies in Django Admin

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 the 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.

Public registration

If you want to allow a public register, configure this variable to "True". By default it's "False". The value should be the same in `taiga-front` and `taiga-back`.

Service: `taiga-back`

```
PUBLIC_REGISTER_ENABLED: "True"
```

Service: `taiga-front`

```
PUBLIC_REGISTER_ENABLED: "true"
```

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 OAuth login

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 OAuth login

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 integration

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

Service: `taiga-back`

```
ENABLE_SLACK: "True"
```

Service: `taiga-front`

```
ENABLE_SLACK: "true"
```

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

The advanced configuration **will ignore** the environment variables in `docker-compose.yml` or `docker-compose-inits.yml`. Skip this section if you're using env vars.

It requires you to map the configuration files of `taiga-back` and `taiga-front` services to local files in order to unlock further configuration options.

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 `config.py` with your own configuration:

- Taiga secret key: **it's important** to change it. It must have the same value as the secret key in `taiga-events` and `taiga-protected`
- Taiga urls: configure where Taiga would be served using `TAIGA_URL`, `SITES` and `FORCE_SCRIPT_NAME` (see examples below)
- Connection to PostgreSQL; check `DATABASES` section in the file
- Connection to RabbitMQ for `taiga-events`; check "EVENTS" section in the file
- Connection to RabbitMQ for `taiga-async`; check "TAIGA ASYNC" section in the file
- Credentials for email; check "EMAIL" section in the file
- Enable/disable anonymous telemetry; check "TELEMETRY" section in the file

Example to configure Taiga in **subdomain**:

```
TAIGA_SITES_SCHEME = "https"
TAIGA_SITES_DOMAIN = "taiga.mycompany.com"
FORCE_SCRIPT_NAME = ""
```

Example to configure Taiga in **subpath**:

```
TAIGA_SITES_SCHEME = "https"
TAIGA_SITES_DOMAIN = "taiga.mycompany.com"
FORCE_SCRIPT_NAME = "/taiga"
```

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:

- Taiga urls: configure where Taiga would be served using `api`, `eventsUrl` and `baseHref` (see examples below)

Example to configure Taiga in **subdomain**:

```
# conf.json
{
    "api": "https://taiga.mycompany.com/api/v1/",
    "eventsUrl": "wss://taiga.mycompany.com/events",
    "baseHref": "/",
```

Example to configure Taiga in **subpath**:

```
# conf.json
{
    "api": "https://mycompany.com/taiga/api/v1/",
    "eventsUrl": "wss://mycompany.com/taiga/events",
    "baseHref": "/taiga/",
```

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

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
```

1.9. Configure the proxy

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

If Taiga is being served in a **subdomain**:

```
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;
  }

  # TLS: Configure your TLS following the best practices inside your company
  # Logs and other configurations
}
```

If Taiga is being served in a **subpath** instead of a subdomain, the configuration should be something like:

```
server {
  server_name mycompany.com;

  location /taiga/ {
    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 /taiga/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;
  }

  # TLS: Configure your TLS following the best practices inside your company
  # Logs and other configurations
}
```

1.10. Change between subpath and subdomain

If you're changing Taiga configuration from default subdomain (https://taiga.mycompany.com) to subpath (http://mycompany.com/subpath) or vice versa, on top of adjusting the configuration as said above, you should consider changing the TAIGA_SECRET_KEY so the refresh works properly for the end user.