DTaaS

Digital Twin as a Service (DTaaS)

DTaaS Development Team

Table of contents

3.6

1. What is DTaaS?

> ❌ **Failure**
>
> Please note that the development documentation is being rewritten heavily. This rework is being done to prepare the documentation for the next release. Please use DTaaS v0.4 for installation and use. Thank you.

The Digital Twin as a Service (DTaaS) software platform is useful to **Build, Use and Share** digital twins (DTs).

💪 **Build**: The DTs are built on the software platform using the reusable DT components available on the platform.

👷👨‍💻 **Use**: Use the DTs on the software platform.

🤝 **Share**: Share ready to use DTs with other users. It is also possible to share the services offered by one DT with other users.

There is an overview of the software available in the form of slides, video, and feature walkthrough.

1.1 License

This software is owned by The INTO-CPS Association and is available under the INTO-CPS License.

The DTaaS software platform uses third-party open-source software. These software components have their own licenses.

2. Admin

2.1 Overview

2.1.1 Goal

The goal is to set up the DTaaS infrastructure in order to enable your users to use the DTaaS. As an admin you will administrate the users and the servers of the system.

2.1.2 Optional Requirements

There are three optional requirements for installing the DTaaS. These requirements are not needed for **localhost** installation. They are only required for installation of the DTaaS on a web server.

OAuth Provider

The DTaaS software is uses OAuth for user authorization. It is possible to use either *foo.com* or your own OAuth service provider.

Domain name

The DTaaS software is a web application and is preferably hosted on a server with a domain name like *foo.com*. However, it is possible to install the software on your computer and use access it at *localhost*.

Reverse Proxy

The installation setup recommends that the *foo.com* server is behind a reverse proxy / load balancer that provides https termination. You can still use the DTaaS software even if you do not have this reverse proxy.

2.1.3 Install

The DTaaS can be installed in different ways. Each version serves a different purpose. Follow the installation that fits your usecase.

Installation Setup	Purpose
Trial installation on localhost	Install DTaaS on your computer for a single user; does not need a web server. *This setup also does not require reverse proxy and domain name.*
Production installation on single host	Install DTaaS on server for multiple users.
One vagrant machine	Install DTaaS on a virtual machine; can be used for single or multiple users.
Two vagrant machines	Install DTaaS on two virtual machines; can be used for single or multiple users.
	The core DTaaS application is installed on the first virtual machine and all the services (RabbitMQ, MQTT, InfluxDB, Grafana and MongoDB) are installed on second virtual machine.
Seperater Packages: client website and lib microservice	Can be used independently; do not need full installation of DTaaS.

2.2 Authorization

2.2.1 OAuth for React Client

> ❌ **Failure**
>
> Add screenshots from dtaas-digitaltwin.com demo.

To enable user authorization on DTaaS React client website, you will use the OAuth authorization protocol, specifically the PKCE authorization flow. Here are the steps to get started:

1. Choose Your GitLab Server:

- You need to set up OAuth authorization on a GitLab server. The commercial gitlab.com is not suitable for multi-user authorization (DTaaS requires this), so you'll need an on-premise GitLab instance.
- You can use GitLab Omnibus Docker for this purpose.
- Configure the OAuth application as an instance-wide authorization type.

2. Determine Your Website's Hostname:

- Before setting up OAuth on GitLab, decide on the hostname for your website. It's recommended to use a self-hosted GitLab instance, which you will use in other parts of the DTaaS application.

3. Define Callback and Logout URLs:

- For the PKCE authorization flow to function correctly, you need two URLs: a callback URL and a logout URL.
- The callback URL informs the OAuth provider of the page where signed-in users should be redirected. It's different from the landing homepage of the DTaaS application.
- The logout URL is where users will be directed after logging out.

4. OAuth Application Creation:

- During the creation of the OAuth application on GitLab, you need to specify the scope. Choose openid, profile, read_user, read_repository, and api scopes.

5. Application ID:

- After successfully creating the OAuth application, GitLab generates an application ID. This is a long string of HEX values that you will need for your configuration files.

6. Required Information from OAuth Application:

- You will need the following information from the OAuth application registered on GitLab:

GitLab Variable Name	Variable Name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.foo.com/
Application ID	REACT_APP_CLIENT_ID	
Callback URL	REACT_APP_REDIRECT_URI	https://foo.com/Library
Scopes	REACT_APP_GITLAB_SCOPES	openid, profile, read_user, read_repository, api

7. Create User Accounts:

Create user accounts in gitlab for all the usernames chosen during installation. The *trial* installation script comes with two default usernames - *user1* and *user2*. For all other installation scenarios, accounts with specific usernames need to be created on gitlab.

Development Environment

There needs to be a valid callback and logout URLs for development and testing purposes. You can use the same oauth application id for both development, testing and deployment scenarios. Only the callback and logout URLs change. It is possible to register multiple callback URLs in one oauth application. In order to use oauth for development and testing on developer computer (localhost), you need to add the following to oauth callback URL.

```
1   DTaaS application URL: http://localhost:4000
2   Callback URL: http://localhost:4000/Library
3   Logout URL: http://localhost:4000
```

The port 4000 is the default port for running the client website.

2.2.2 OAuth for Traefik Gateway

> ❌ **Failure**
>
> Add screenshots from dtaas-digitaltwin.com demo.

The traefik gateway is used to serve the DTaaS. All the services provided as part of the application are secured at the traefik gateway. The security is based on Traefik forward-auth.

An illustration of the docker containers used and the authorization setup is shown here.
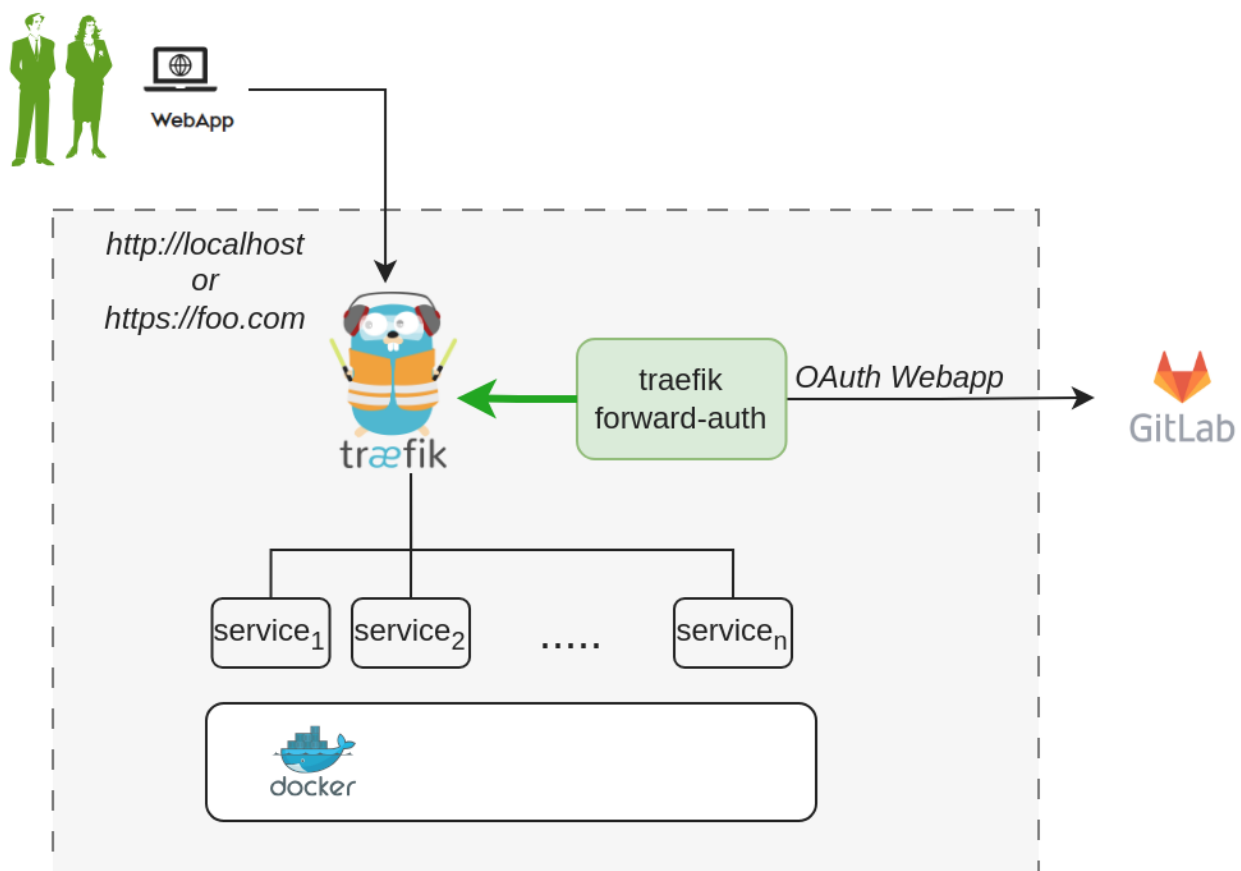
The **traefik forward-auth** can use any OAuth2 provider, but within the DTaaS gitlab is used as authorization provider. You will use the OAuth the web / server application authorization flow.

Here are the steps to get started:

1. Choose Your GitLab Server:

- You need to set up OAuth authorization on a GitLab server. The commercial gitlab.com is not suitable for multi-user authorization (DTaaS requires this), so you'll need an on-premise GitLab instance.
- You can use GitLab Omnibus Docker for this purpose.
- Configure the OAuth application as an instance-wide authorization type. Select option to generate client secret and also selection option for trusted application.

2. Determine Your Website's Hostname:

- Before setting up OAuth on GitLab, decide on the hostname for your website. It's recommended to use a self-hosted GitLab instance, which you will use in other parts of the DTaaS application.

3. Define Callback and Logout URLs:

- For the web / server authorization flow to function correctly, you need two URLs: a callback URL and a logout URL.
- The callback URL informs the OAuth provider of the page where signed-in users should be redirected. It is the landing homepage of the DTaaS application. (either http://foo.com/_oauth/ or http://localhost/_oauth/)
- The logout URL is the URL for signout of gitlab and clear authorization within traefik-forward auth. (either http://foo.com/_oauth/logout or http://localhost/_oauth/logout). The logout URL is to help users logout of traefik forward-auth. The logout URL should not be entered into Gitlab OAuth application setup.

4. OAuth Application Creation:

- During the creation of the OAuth application on GitLab, you need to specify the scope. Choose *read_user* scope.

5. Application Credentials:

- After successfully creating the OAuth application, GitLab generates an *application ID* and *client secret*. Both these values are long string of HEX values that you will need for your configuration files.

6. Required Information from OAuth Application:

- You will need the following information from the OAuth application registered on GitLab:

GitLab Variable Name	Variable Name in .env of docker compose file	Default Value
OAuth Provider	OAUTH_URL	https://gitlab.foo.com/
Application ID	CLIENT_ID	xx
Application Secret	CLIENT_SECRET	xx
Callback URL	(to be directly entered in Gitlab OAuth registration)	
Forward-auth secret	OAUTH_SECRET	*random-secret-string* (password for forward-auth, can be changed to your preferred string)
Scopes	read_user	

Development Environment

The development environment and server installation scenarios requires traefik forward-auth.

Configure Authorization Rules for Traefik Forward-Auth

The Traefik forward-auth microservices requires configuration rules to manage authorization for different URL paths. The *conf.** file can be used to configure the specific rules. There are broadly three kinds of URLs:

PUBLIC PATH WITHOUT AUTHORIZATION

To setup a public page, an example is shown below.

```
1  rule.noauth.action=allow
2  rule.noauth.rule=Path(`/public`)
```

Here, 'noauth' is the rule name, and should be changed to suit rule use. Rule names should be unique for each rule. The 'action' property is set to "allow" to make the resource public. The 'rule' property defines the path/route to reach the resource.

To setup a common page that requires Gitlab OAuth, but is available to all users of the Gitlab instance:

```
1    rule.all.action=auth
2    rule.all.rule=Path(`/common`)
```

The 'action' property is set to "auth", to enable Gitlab OAuth before the resource can be accessed.

SELECTIVE ACCESS

Selective Access refers to the scenario of allowing access to a URL path for a few users. To setup selective access to a page:

```
1    rule.onlyu1.action=auth
2    rule.onlyu1.rule=Path(`/user1`)
3    rule.onlyu1.whitelist = user1@localhost
```

The 'whitelist' property of a rule defines a comma separated list of email IDs that are allowed to access the resource. While signing in users can sign in with either their username or email ID as usual, but the email ID corresponding to the account should be included in the whitelist.

This restricts access of the resource, allowing only users mentioned in the whitelist.

Limitation

The rules in _conf._ *file are not dynamically loaded into the traefik-forward-auth microservice. Any change in the conf file requires retart of traefik-forward-auth for the changes to take effect. All the existing user sessions get invalidated when the traefik-forward-auth* restarts.*

Use a simple command on the terminal.

2.3 Docker

2.3.1 Install DTaaS on localhost

The installation instructions provided in this README are ideal for running the DTaaS on both localhost. This installation is ideal for single users intending to use DTaaS on their own computers.

Design

An illustration of the docker containers used and the authorization setup is shown here.
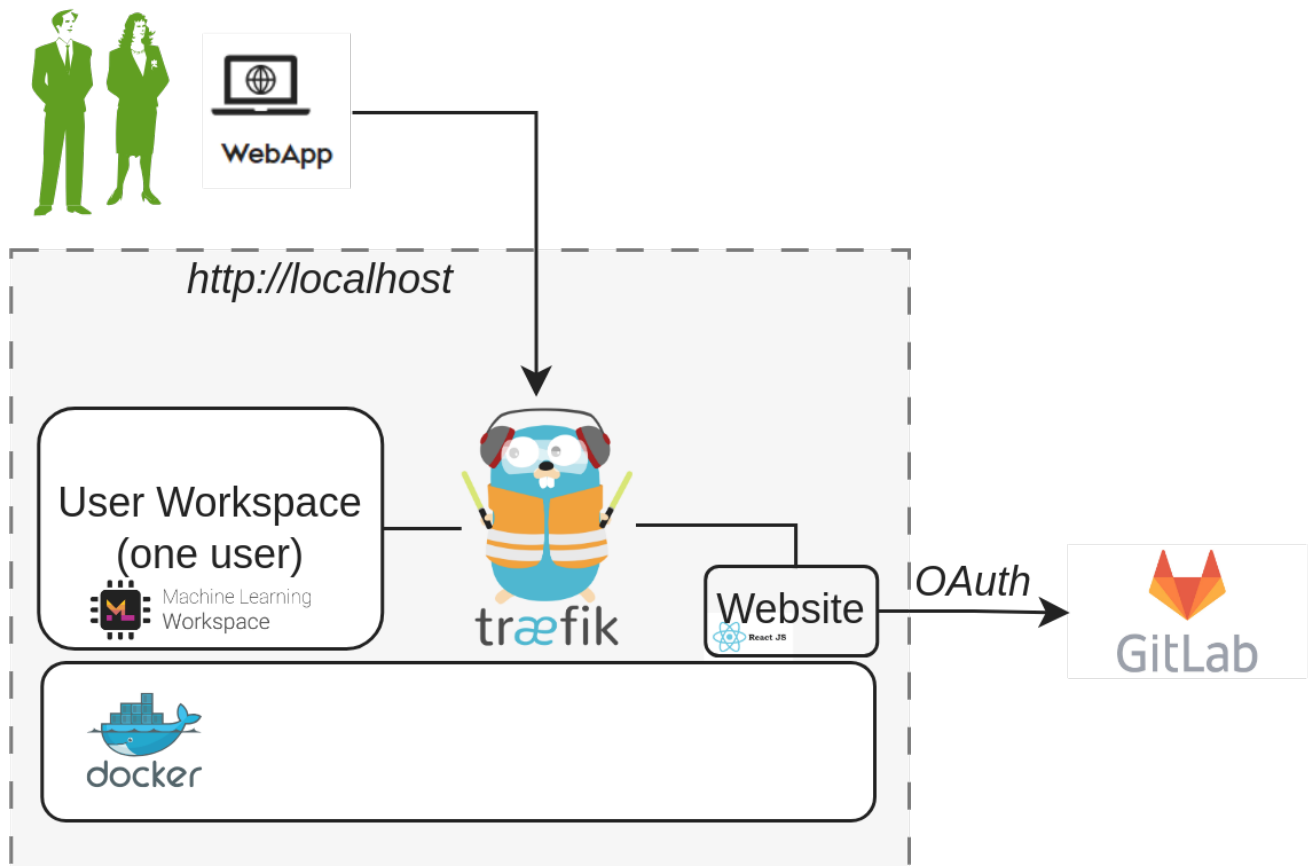
Requirements

The installation requirements to run this docker version of the DTaaS are:

- docker desktop / docker CLI with compose plugin
- User account on *gitlab.com*

> **Tip**
>
> The frontend website requires authorization. The default authorization configuration works for *gitlab.com*. If you desire to use locally hosted gitlab instance, please see the client docs.

Clone Codebase

```
1   git clone https://github.com/INTO-CPS-Association/DTaaS.git
2   cd DTaaS
```

> **Tip**
>
> 1. The filepaths shown here follow POSIX convention. The installation procedures also work with Windows paths.
>
> 2. The description below refers to filenames. All the file paths mentioned below are relatively to the top-level **DTaaS** directory.

Configuration

DOCKER COMPOSE

The docker compose configuration is in `deploy/docker/.env.local`; it is a sample file. It contains environment variables that are used by the docker compose files. It can be updated to suit your local installation scenario. It contains the following environment variables.

Edit all the fields according to your specific case.

URL Path	Example Value	Explanation
DTAAS_DIR	'/home/Desktop/DTaaS'	Full path to the DTaaS directory. This is an absolute path with no trailing slash.
username1	'user1'	Your gitlab username
CLIENT_CONFIG	'/home/Desktop/DTaaS/deploy/config/client/env.local.js'	Full path to env.js file for client

> **Tip**
>
> The path examples given here are for Linux OS. These paths can be Windows OS compatible paths as well.

CREATE USER WORKSPACE

The existing filesystem for installation is setup for `user1`. A new filesystem directory needs to be created for the selected user.

Please execute the following commands from the top-level directory of the DTaaS project.

```
1   cp -R files/user1 files/username
```

where *username* is the selected username registered on https://gitlab.com.

Run

The commands to start and stop the appliation are:

```
1   docker compose -f compose.local.yml --env-file .env.local up -d
2   docker compose -f compose.local.yml --env-file .env.local down
```

To restart only a specific container, for example `client``

```
1   docker compose -f compose.local.yml --env-file .env up -d --force-recreate client
```

Use

The application will be accessible at: http://localhost from web browser. Sign in using your https://gitlab.com account.

All the functionality of DTaaS should be available to you through the single page client now.

Limitations

The library microservice is not included in the localhost installation scenario.

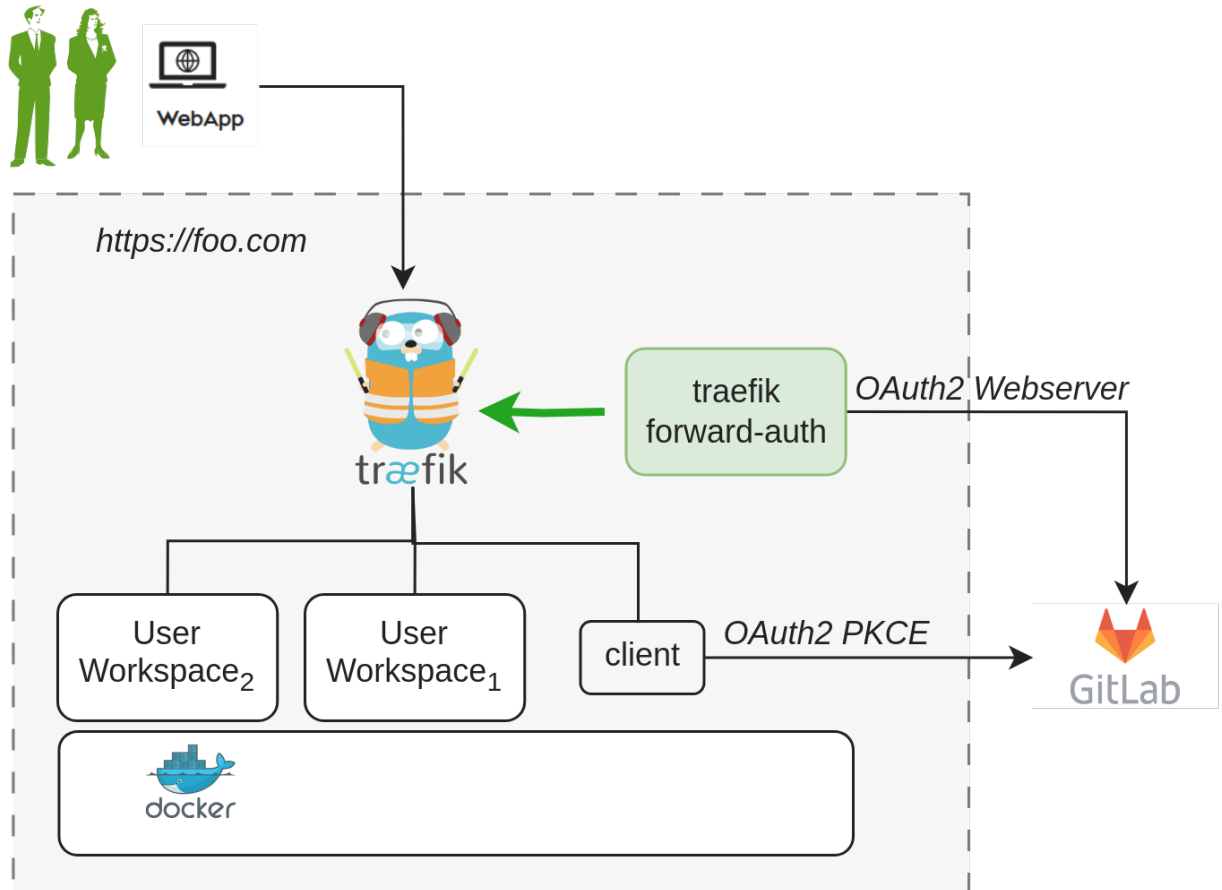**References**

Image sources: Traefik logo, ml-workspace, reactjs, gitlab

2.3.2 Install DTaaS on a Production Server

The installation instructions provided in this README are ideal for hosting the DTaaS as web application for multiple users.

Design

An illustration of the docker containers used and the authorization setup is shown here.

In the new application configuration, there are two OAuth2 applications.

Requirements

The installation requirements to run this docker version of the DTaaS are:

- DNS name for the server
- docker with compose plugin
- User accounts on a gitlab instance
- OAuth2 application registrations

CREATE USER ACCOUNTS

Create user accounts in a gitlab instance for all the users.

The default docker compose file contains two - *user1* and *user2*. These names need to be changed to suitable usernames.

It is possible to use https://gitlab.com as well for OAuth2 authorization purposes.

The multi-user installation setup requires dedicated authorization setup for both frontend website and backend services. Both these authorization requirements are satisfied using OAuth2 protocol.

- The frontend website is a React single page application (SPA).
- The details of Oauth2 app for the frontend website are in client docs.
- The Oauth2 authorization for backend services is managed by Traefik forward-auth. The details of this authorization setup are in server docs.

Based on your selection of gitlab instance, it is necessary to register these two OAuth2 applications and link them to your intended DTaaS installation.

Please see gitlab oauth provider documentation for further help with creating these two OAuth applications.

Clone Codebase

```
1   git clone https://github.com/INTO-CPS-Association/DTaaS.git
2   cd DTaaS
```

> **Tip**
>
> 1. The filepaths shown here follow Linux OS. The installation procedures also work with Windows OS.
>
> 2. The description below refers to filenames. All the file paths mentioned below are relatively to the top-level **DTaaS** directory.

Configuration

Three following configuration files need to be updated.

The docker compose configuration is in `deploy/docker/.env.server`. it is a sample file. It contains environment variables that are used by the docker compose files. It can be updated to suit your local installation scenario. It contains the following environment variables.

Edit all the fields according to your specific case.

URL Path	Example Value	Explanation
DTAAS_DIR	'/Users/username/DTaaS'	Full path to the DTaaS directory. This is an absolute path with no trailing slash.
SERVER_DNS	*foo.com*	The server DNS, if you are deploying with a dedicated server. Remember not use http(s) at the beginning of the DNS string
OAUTH_URL	*gitlab.foo.com*	The URL of your Gitlab instance. It can be *gitlab.com* if you are planning to use it for authorization.
CLIENT_ID	'xx'	The ID of your server OAuth application
CLIENT_SECRET	'xx'	The Secret of your server OAuth application
OAUTH_SECRET	'random-secret-string'	Any private random string. This is a password you choose for local installation.
username1	'user1'	The gitlab instance username of a user of DTaaS
username2	'user2'	The gitlab instance username of a user of DTaaS
CLIENT_CONFIG	'/Users/username/DTaaS/ deploy/config/client/env.js'	Full path to env.js file for client

> 🔥 **Tip**
>
> Important points to note:
>
> 1. The path examples given here are for Linux OS. These paths can be Windows OS compatible paths as well.
> 2. The Server DNS can also be an IP address. However, for proper working it is neccessary to use the same convention (IP/DNS) in the `CLIENT_CONFIG` file as well.

WEBSITE CLIENT

The frontend React website requires configuration which is specified via a filename provided in `CLIENT_CONFIG` variable of `deploy/docker/.env.server` file.

The `CLIENT_CONFIG` file is in relative directory of `deploy/config/client/env.js`.

Further explanation on the client configuration is available in client config.

> 🔥 **Tip**
>
> There is a default OAuth application registered on https://gitlab.com for client. The corresponding OAuth application details are:
>
> ```
> 1  REACT_APP_CLIENT_ID: '1be55736756190b3ace4c2c4fb19bde386d1dcc748d20b47ea8cfb5935b8446c',
> 2  REACT_APP_AUTH_AUTHORITY: 'https://gitlab.com/',
> ```
>
> **This can be used for test purposes**. Please use your own OAuth application for secure production deployments.

CREATE USER WORKSPACE

The existing filesystem for installation is setup for `files/user1`. A new filesystem directory needs to be created for the selected user.

Please execute the following commands from the top-level directory of the DTaaS project.

```
1  cp -R files/user1 files/username
```

where *username* is one of the selected usernames. This command needs to be repeated for all the selected users.

LIBRARY MICROSERVICE

The library microservice (Lib MS) acts like as a standalone file server for users. It does not interact directly with user workspaces and is not needed for the regular use cases of the DTaaS.

In case you wish to use the Lib MS, please update the `deploy/config/lib.docker` configuration file.

The config specified in INI format. The template configuration file is:

```
1  PORT='4001'
2  MODE='local'
3  LOCAL_PATH ='/Users/<username>/DTaaS/files'
4  LOG_LEVEL='debug'
5  APOLLO_PATH='/lib'
6  GRAPHQL_PLAYGROUND='true'
```

The `LOCAL_PATH` variable is the absolute filepath to the location of the local directory which will be served to users by the Library microservice. Replace the default values the appropriate values for your setup.

CONFIGURE AUTHORIZATION RULES FOR TRAEFIK FORWARD-AUTH

The Traefik forward-auth microservices requires configuration rules to manage authorization for different URL paths. The `deploy/docker/conf.server` file can be used to configure the authorization for user workspaces.

```
1   rule.onlyu1.action=auth
2   rule.onlyu1.rule=Path(`/user1`)
3   rule.onlyu1.whitelist = user1@localhost
4
5   rule.onlyu1.action=auth
6   rule.onlyu1.rule=Path(`/user2`)
7   rule.onlyu1.whitelist = user2@localhost
```

Please change the usernames and email addresses to the matching user accounts on the OAuth provider (either https://gitlab.foo.com or https://gitlab.com).

CAVEAT

The usernames in the `deploy/docker/.env.server` file need to match those in the `deploy/docker/conf.server` file.

Traefik routes are controlled by the `deploy/docker/.env.server` file. Authorization on these routes is controlled by the `deploy/docker/conf.server` file. If a route is not specified in `deploy/docker/conf.server` file but an authorisation is requested by traefik for this unknown route, the default behavior of traefik forward-auth kicks in. This default behavior is to enable endpoint being available to any signed in user.

If there are extra routes in `deploy/docker/conf.server` file but these are not in `deploy/docker/.env.server` file, such routes are not served by traefik; it will give **404 server response**.

Run

The commands to start and stop the appliation are:

```
1   docker compose -f compose.server.yml --env-file .env.server up -d
2   docker compose -f compose.server.yml --env-file .env.server down
```

To restart only a specific container, for example `client``

```
1   docker compose -f compose.server.yml --env-file .env.server up -d --force-recreate client
```

Use

The application will be accessible at: http://foo.com from web browser. Sign in using your account linked to either *gitlab.com* or your local gitlab instance.

All the functionality of DTaaS should be available to your users through the single page client now.

You may have to click Sign in to Gitlab on the Client page and authorize access to the shown application.

Adding a new user

To add a new user to your DTaaS instance, follow these steps:

- Use the DTaaS CLI to bring up the ML workspaces for new users. This brings up the containers, without the authorization enforced by Traefik forward-auth.
- Add three lines to the `conf.server` file

```
1   rule.onlyu3.action=auth
2   rule.onlyu3.rule=PathPrefix(`/user3`)
3   rule.onlyu3.whitelist = user3@emailservice.com
```

Run the appropriate command for a server/local installation:

```
1   docker compose -f compose.server.yml --env-file .env.server up -d --force-recreate traefik-forward-auth
```

The new users are now added to the DTaaS instance, with authorization enabled.

References

Image sources: Traefik logo, ml-workspace, reactjs, gitlab

2.4 DTaaS Command Line Interface

This is a command line tool for the INTO-CPS-Association Digital Twins as a Service.

2.4.1 Prerequisite

The DTaaS application with base users and essential containers should be up and running before using the CLI.

2.4.2 Installation

Simply install using:

We recommend installing this in a virutal environment.

Steps to install:

- Change the working folder:

```
1    cd <DTaaS-directory>/cli
```

- We recommend installing this in a virtual environment. Create and activate a virtual environment.
- To install, simply:

```
1    pip install dtaas
```

2.4.3 Usage

Setup

The base DTaaS system should be up and running before adding/deleting users with the CLI.

Additionally, Setup the *dtaas.toml* file in the *cli* directory:

- Set *common.server-dns* to domain name of your server. If you want to bring up the server locally, please set this to "*localhost*".
- Set the *path* to the full system path of the DTaaS directory.

```
1    [common]
2    # absolute path to the DTaaS application directory
3    server-dns = "localhost"
4    path = "/home/Desktop/DTaaS"
```

Add users

To add new users using the CLI, fill in the *users.add* list in *dtaas.toml* with the Gitlab instance usernames of the users to be added

```
1    [users]
2    # matching user info must present in this config file
3    add = ["username1","username2", "username3"]
```

Then simply:

```
1    dtaas admin user add
```

This brings up the containers, without the AuthMS authentication.

- Currently the *email* fields for each user in *dtaas.toml* are not in use, and are not necessary to fill in. These emails must be configured manually for each user in the docker/conf.local or docker/conf.server files and the *traefik-forward-auth* container must be restarted. This is done as follows:
- Go to the *docker* directory

```
1   cd <DTaaS>/deploy/docker
```

- Add two lines to the `conf.local` file

```
1   rule.onlyu4.action=allow
2   rule.onlyu4.rule=PathPrefix(`/user4`)
```

- Add three lines to the `conf.server` file

```
1   rule.onlyu3.action=auth
2   rule.onlyu3.rule=PathPrefix(`/user3`)
3   rule.onlyu3.whitelist = user3@emailservice.com
```

Run the appropritate command for a server/local installation:

```
1   docker compose -f compose.server.yml --env-file .env up -d --force-recreate traefik-forward-auth
```

```
1   docker compose -f compose.local.yml --env-file .env up -d --force-recreate traefik-forward-auth
```

The new users are now added to the DTaaS instance, with authorization enabled.

Delete users

TO delete existing users, fill in the *users.delete* list in *dtaas*.toml_ with the Gitlab instance usernames of the users to be deleted.

Make sure you are in the *cli* directory.

Then simply:

```
1   dtaas admin user delete
```

Additional Points to Remember

- The *user add* CLI will add and start a container for a new user. It can also start a container for an existing user if that container was somehow stopped. It shows a *Running* status for existing user containers that are already up and running, it doesn't restart them.
- *user add* and *user delete* CLIs return an error if the *add* and *delete* lists in *dtaas.toml* are empty, respectively.

2.5 Vagrant

2.5.1 DTaaS Vagrant Box

This README provides instructions on creating a custom Operating System virtual disk for running the DTaaS software. The virtual disk is managed by **vagrant**. The purpose is two fold:

- Provide cross-platform installation of the DTaaS application. Any operating system supporting use of vagrant software utility can support installation of the DTaaS software.
- Create a ready to use development environment for code contributors.

There are two scripts in this directory:

Script name	Purpose	Default
`user.sh`	user installation	✅
`developer.sh`	developer installation	❌

If you are installing the DTaaS for developers, the default installation caters to your needs. You can skip the next step and continue with the creation of vagrant box.

If you are a developer and would like additional software installed, you need to modify `Vagrantfile`. The existing `Vagrantfile` has two lines:

```
1    config.vm.provision "shell", path: "user.sh"
2    #config.vm.provision "shell", path: "developer.sh"
```

Uncomment the second line to have more software components installed. If you are not a developer, no changes are required to the `Vagrantfile`.

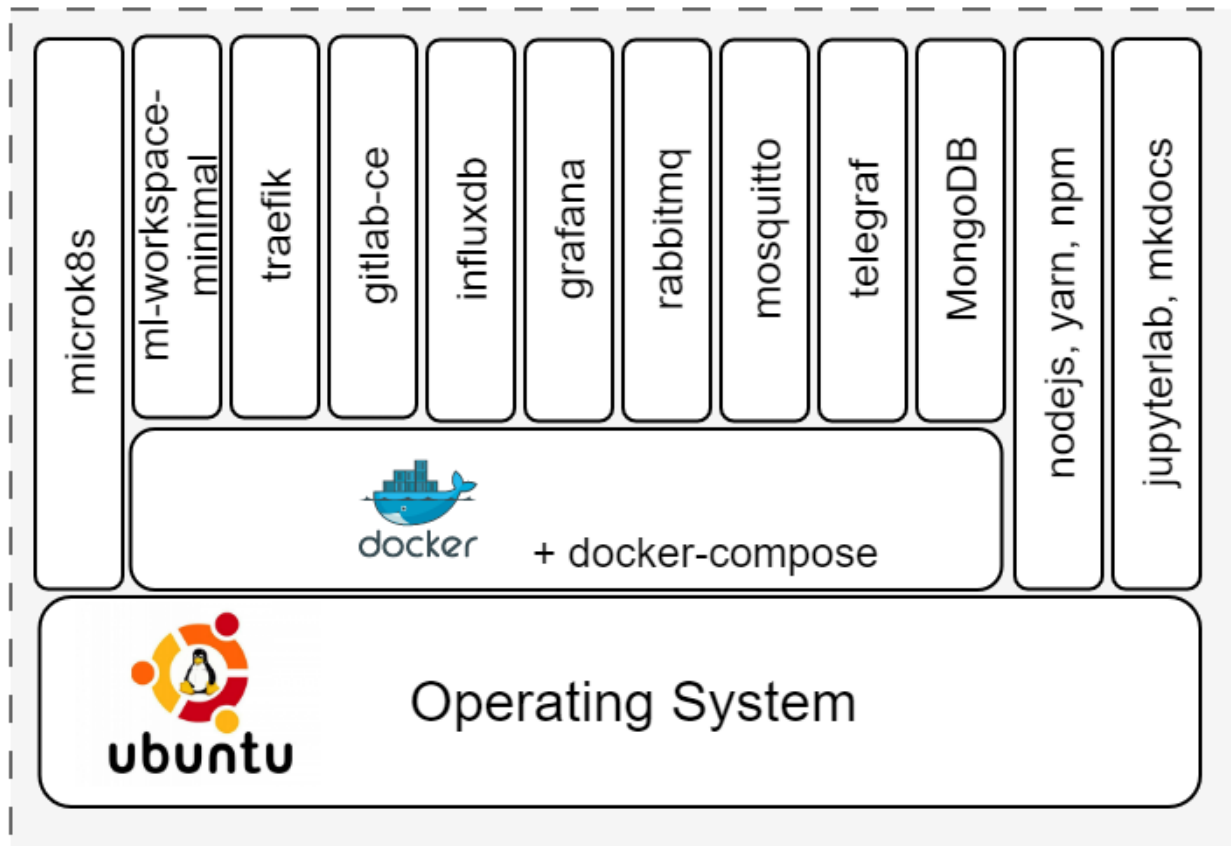This vagrant box installed for users will have the following items:

1. docker v24.0
2. nodejs v20.10
3. yarn v1.22
4. npm v10.2
5. containers - ml-workspace-minimal v0.13, traefik v2.10, gitlab-ce v16.4, influxdb v2.7, grafana v10.1, rabbitmq v3-management, eclipse-mosquitto (mqtt) v2, mongodb v7.0

   This vagrant box installed for developers will have the following items additional items:

   - docker-compose v2.20
   - microk8s v1.27
   - jupyterlab
   - mkdocs
   - container - telegraf v1.28

At the end of installation, the software stack created in vagrant box can be visualised as shown in the following figure.

vagrant box

microk8s | ml-workspace-minimal | traefik | gitlab-ce | influxdb | grafana | rabbitmq | mosquitto | telegraf | MongoDB | nodejs, yarn, npm | jupyterlab, mkdocs

docker + docker-compose

Operating System

ubuntu

The upcoming instructions will help with the creation of base vagrant box.

```
1   #create a key pair
2   ssh-keygen -b 4096 -t rsa -f vagrant -q -N ""
3
4   vagrant up
5
6   # let the provisioning be complete
7   # replace the vagrant ssh key-pair with personal one
8   vagrant ssh
9
10  # install the oh-my-zsh
11  sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
12  # install plugins: history, autosuggestions,
13  git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
14
15  # inside ~/.zshrc, modify the following line
16  plugins=(git zsh-autosuggestions history cp tmux)
17
18  # to replace the default vagrant ssh key-pair with
19  # the generated private key into authorized keys
20  cp /vagrant/vagrant.pub /home/vagrant/.ssh/authorized_keys
21
22  # exit vagrant guest machine and then
23  # copy own private key to vagrant private key location
24  cp vagrant .vagrant/machines/default/virtualbox/private_key
25
26  # check
27  vagrant ssh #should work
28
29  # exit vagrant guest machine and then
30  vagrant halt
31
32  vagrant package --base dtaas \
33  --info "info.json" --output dtaas.vagrant
34
35  # Add box to the vagrant cache in ~/.vagrant.d/boxes directory
36  vagrant box add --name dtaas ./dtaas.vagrant
37
38  # You can use this box in other vagrant boxes using
39  #config.vm.box = "dtaas"
```

References

Image sources: Ubuntu logo

2.5.2 DTaaS on Single Vagrant Machine

These are installation instructions for running DTaaS software inside one vagrant Virtual Machine. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space to the vagrant box.

Create Base Vagrant Box

Create **dtaas** Vagrant box. You would have created an SSH key pair - *vagrant* and *vagrant.pub*. The *vagrant* is the private SSH key and is needed for the next steps. Copy *vagrant* SSH private key into the current directory (`deploy/vagrant/single-machine`). This shall be useful for logging into the vagrant machines created for two-machine deployment.

Target Installation Setup

The goal is to use the **dtaas** Vagrant box to install the DTaaS software on one single vagrant machine. A graphical illustration of a successful installation can be seen here.

dtaas vagrant base box

post-install

http(s)://foo.com

http(s)://gitlab.foo.com

OAuth