DTaaS

Digital Twin as a Service (DTaaS)

DTaaS Development Team

Table of contents

1. What is DTaaS?

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 Requirements

OAuth Provider (Mandatory)

You need to have an OAuth Provider running, which the DTaaS can use for authorization. This is described further in the authorization section.

Domain name (Optional)

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 (Optional)

The installation setup assumes 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. If you do not have a reverse proxy, please replace https://foo.com with http://foo.com in client env.js file and in OAuth registration. Other installation configuration remains the same.

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.*
Trial installation on single host	Install DTaaS on server for a single user.
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

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.

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

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

Multiple DTaaS applications

The DTaaS is a regular web application. It is possible to host multiple DTaaS applications on the same server. The only requirement is to have a distinct URLs. You can have three DTaaS applications running at the following URLs.

```
https://foo.com/au
https://foo.com/acme
https://foo.com/bar
```

All of these instances can use the same gitlab instance for authorization.

DTaaS application URL	Gitlab Instance URL	Callback URL	Logout URL	Application ID
https://foo.com/au	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab
https://foo.com/acme	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab
https://foo.com/bar	https://foo.gitlab.com	https://foo.com/au/Library	https://foo.com/au	autogenerated by gitlab

If you are hosting multiple DTaaS instances on the same server, do not install DTaaS with a null basename on the same server. Even though it works well, the setup is confusing to setup and may lead to maintenance issues.

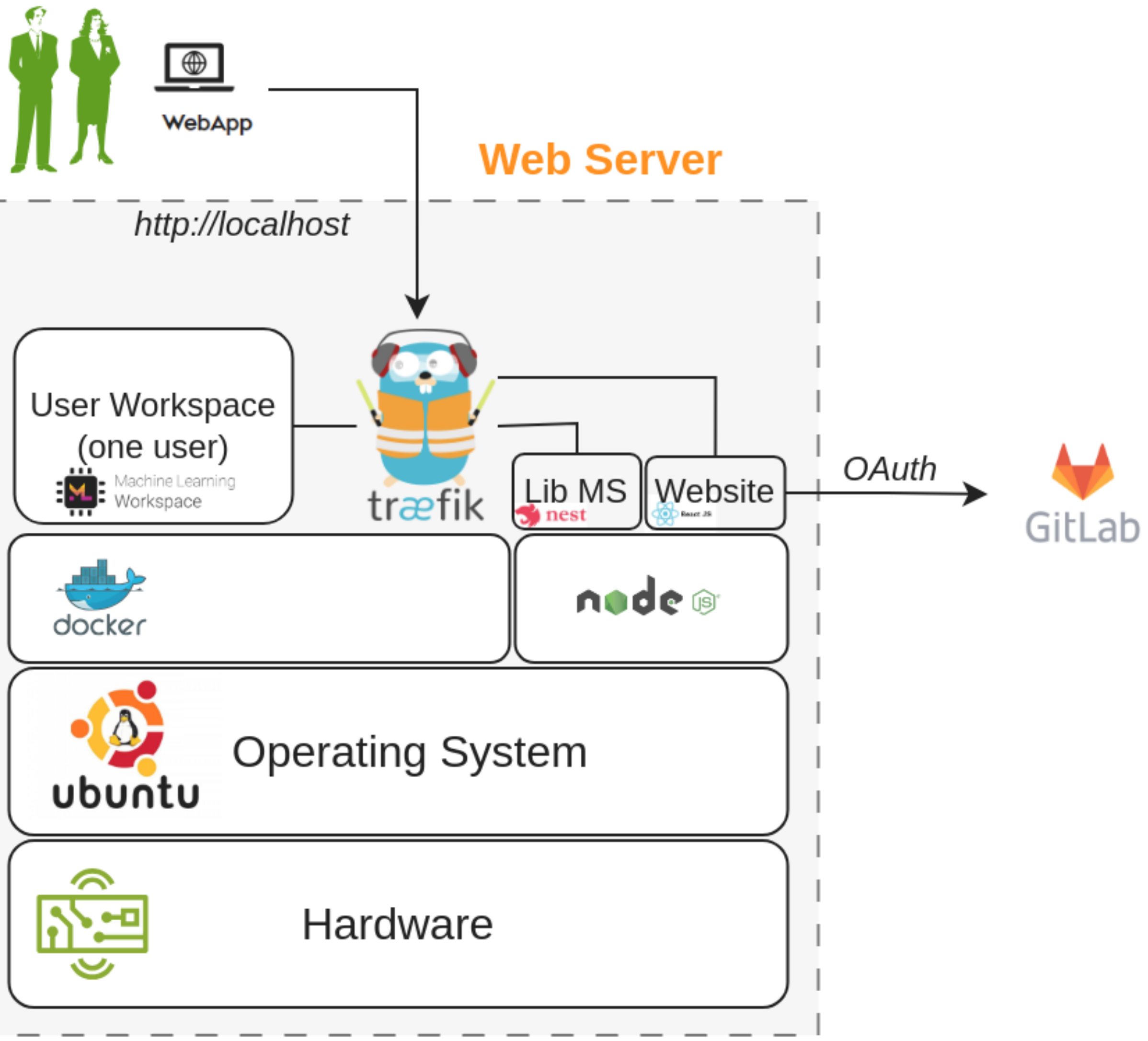
If you choose to host your DTaaS application with a basename (say bar), then the URLs in `env.js` change to:

```
DTaaS application URL: https://foo.com/bar
Gitlab instance URL: https://foo.gitlab.com
Callback URL: https://foo.com/bar/Library
Logout URL: https://foo.com/bar
```

2.3 Host Install

2.3.1 Localhot Installation

To try out the software, you can install it on Ubuntu 22.04 Desktop Operating System. The setup requires a machine which can spare 4GB RAM, 2 vCPUs and 15GB Hard Disk space to a the DTaaS application. A successful installation will create a setup similar to the one shown in the figure.



A one-step installation script is provided on this page. This script sets up the DTaaS software for a single user. You can use it to check a test installation of the DTaaS software.

Pre-requisites

1. GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

> **Information**
>
> It is sufficient to have user-owned oauth application. You can create this application in your gitlab account.

You need the following information from the Gitlab OAuth application registered on Gitlab:

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

You can also see Gitlab help page for getting the Gitlab OAuth application details.

Install

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Run the following commands.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.4/deploy/single-script-install.sh
bash single-script-install.sh --env local --username <username>
```

The `--env local` argument is added to the script specifies `localhost` as the installation scenario. The `--username username` uses your Gitlab username to configure the DTaaS application.

Post install

After the single-install-script is successfully run. Please change Gitlab OAuth details in

```
~/DTaaS/client/build/env.js
```

Post-install Check

Now when you visit http://localhost, you should be able to login through Gitlab OAuth Provider and access the DTaas web UI.

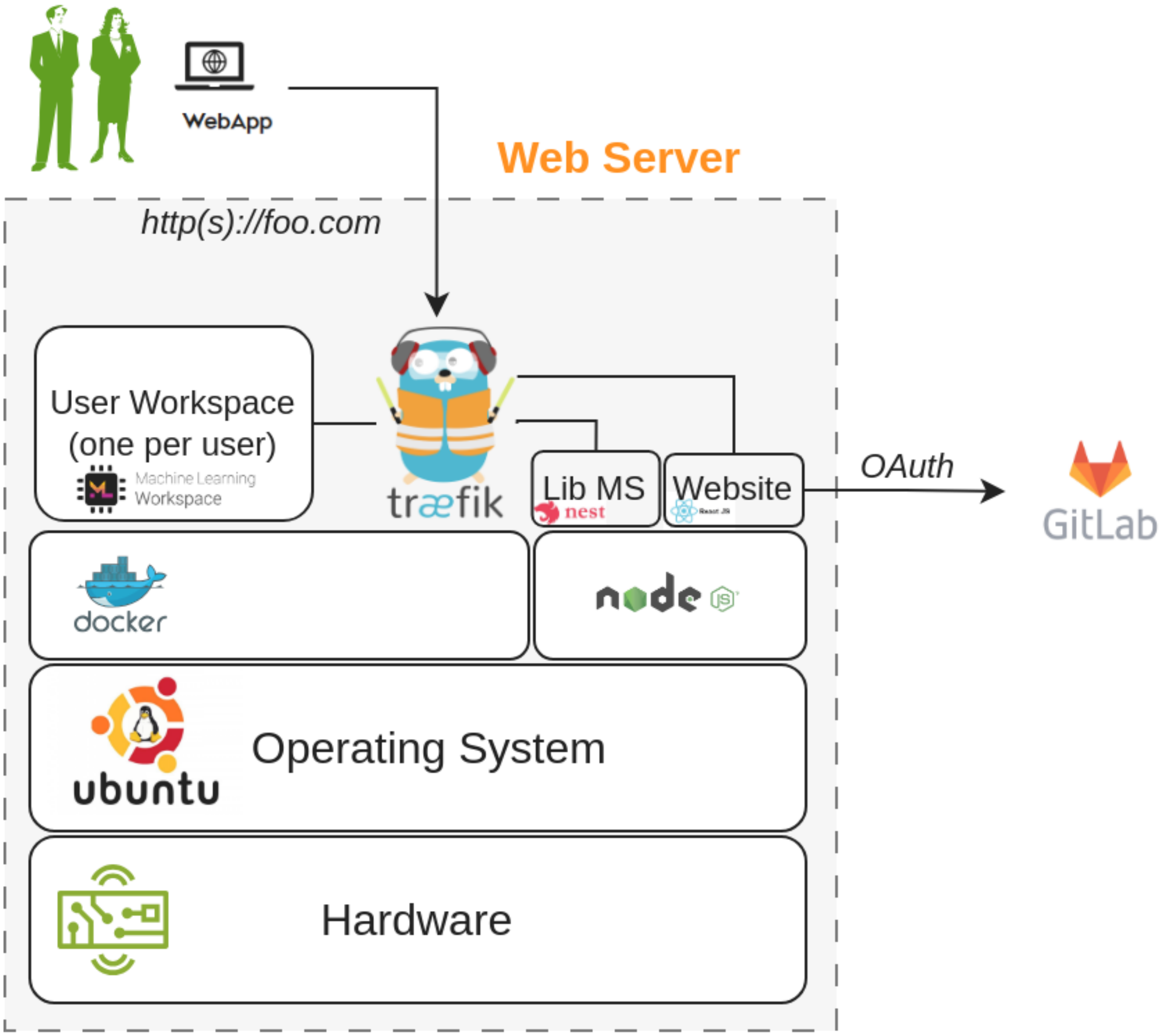
If you can following along to see all the screenshots from user website. Everything is correctly setup.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.3.2 Trial Installation

To try out the software, you can install it on Ubuntu Server 22.04 Operating System. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space to the DTaaS application (or the vagrant box hosting the DTaaS application). A successful installation will create a setup similar to the one shown in the figure.



A one-step installation script is provided on this page. This script sets up the DTaaS software with default gateway credentials for a single user. You can use it to check a test installation of DTaaS software.

> **Information**
>
> It is sufficient to have user-owned oauth application.

Pre-requisites

1. DOMAIN NAME

You need a domain name to run the application. The install script assumes **foo.com** to be your domain name. You will change this after running the script.

2. GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

You need the following information from the Gitlab OAuth application registered on Gitlab:

Gitlab Variable Name	Variable name in Client env.js	Default Value
OAuth Provider	REACT_APP_AUTH_AUTHORITY	https://gitlab.com or 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

You can also see Gitlab help page for getting the Gitlab OAuth application details.

Install

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Run the following commands.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.4/deploy/single-script-install.sh
bash single-script-install.sh --env trial --username <username>
```

The `--env trial` argument is added to the script specifies `trial` as the installation scenario. The `--username username` uses your Gitlab username to configure the DTaaS application.

> **Warning**
>
> This test installation has default gateway credentials and is thus highly insecure.

Post install

After the install-script. Please change **foo.com** and Gitlab OAuth details to your local settings in the following files.

```
~/DTaaS/client/build/env.js
~/DTaaS/servers/config/gateway/dynamic/fileConfig.yml
```

Post-install Check

Now when you visit https://foo.com, you should be able to login through Gitlab OAuth Provider and access the DTaas web UI.

If you can following along to see all the screenshots from user website. Everything is correctly setup.

References

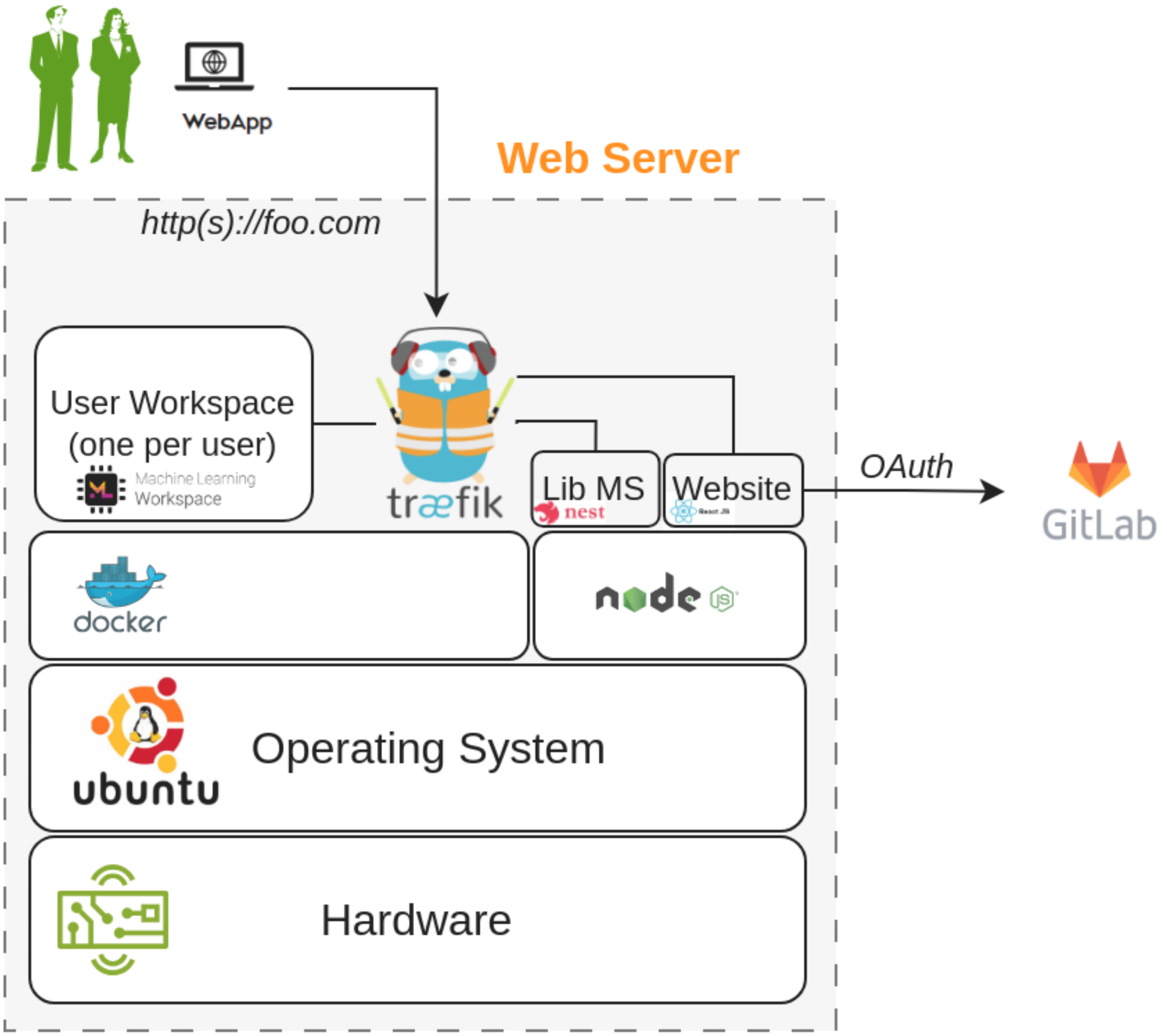
Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.3.3 DTaaS on Linux Operating System

These are installation instructions for running DTaaS application on a Ubuntu Server 22.04 Operating System. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space.

A dummy **foo.com** URL has been used for illustration. Please change this to your unique website URL. It is assumed that you are going to serve the application in only HTTPS mode.

A successful installation will create a setup similar to the one shown in the figure.



Please follow these steps to make this work in your local environment. Download the **DTaaS.zip** from the releases page. Unzip the same into a directory named **DTaaS**. The rest of the instructions assume that your working directory is **DTaaS**.

> **Note**
>
> If you only want to test the application and are not setting up a production instance, you can follow the instructions of trial installation.

Configuration

You need to configure the Traefik gateway, library microservice and react client website.

The first step is to decide on the number of users and their usenames. The traefik gateway configuration has a template for two users. You can modify the usernames in the template to the usernames chosen by you.

TRAEFIK GATEWAY SERVER

You can run the Traefik gateway server in both HTTP and HTTPS mode to experience the DTaaS application. The installation guide assumes that you can run the application in HTTPS mode.

The Traefik gateway configuration is at *deploy/config/gateway/fileConfig.yml*. Change `foo.com` to your local hostname and user1/user2 to the usernames chosen by you.

> **Tip**
>
> Do not use `http://` or `https://` in *deploy/config/gateway/fileConfig.yml*.

Authorization

This step requires `htpasswd` commandline utility. If it is not available on your system, please install the same by using

```
sudo apt-get update
sudo apt-get install -y apache2-utils
```

You can now proceed with update of the gateway authorization setup. The dummy username is `foo` and the password is `bar`. Please change this before starting the gateway.

```
rm deploy/config/gateway/auth
touch deploy/config/gateway/auth
htpasswd deploy/config/gateway/auth <first_username>
password: <your password>
```

The user credentials added in *deploy/config/gateway/auth* should match the usernames in *deploy/config/gateway/fileConfig.yml*.

Lib microservice

The library microservice requires configuration. A template of this configuration file is given in *deploy/config/lib* file. Please modify this file as per your needs.

The first step in this configuration is to prepare a filesystem for users. An example file system is in `files/` directory. You can rename the top-level user1/user2 to the usernames chosen by you.

The simplest possibility is to use `local` mode with the following example. The filepath is the absolute filepath to `files/` directory. You can copy this configuration into *deploy/config/lib* file to get started.

```
PORT='4001'
MODE='local'
LOCAL_PATH ='filepath'
LOG_LEVEL='debug'
APOLLO_PATH='/lib'
GRAPHQL_PLAYGROUND='true'
```

React Client Website

GITLAB OAUTH APPLICATION

The DTaaS react website requires Gitlab OAuth provider. If you need more help with this step, please see the Authorization page.

You 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

You can also see Gitlab help page for getting the Gitlab OAuth application details. Remember to Create gitlab accounts for usernames chosen by you.

UPDATE CLIENT CONFIG

Change the React website configuration in *deploy/config/client/env.js*.

```js
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: 'prod',
    REACT_APP_URL: 'https://foo.com/',
    REACT_APP_URL_BASENAME: 'dtaas',
    REACT_APP_URL_DTLINK: '/lab',
    REACT_APP_URL_LIBLINK: '',
    REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',
    REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',
    REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/lab',
    REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',

    REACT_APP_CLIENT_ID: '934b98f03f1b6f743832b2840bf7cccaed93c3bfe579093dd0942a433691ccc0',
    REACT_APP_AUTH_AUTHORITY: 'https://gitlab.foo.com/',
    REACT_APP_REDIRECT_URI: 'https://foo.com/Library',
    REACT_APP_LOGOUT_REDIRECT_URI: 'https://foo.com/',
    REACT_APP_GITLAB_SCOPES: 'openid profile read_user read_repository api',
  };
};
```

Update the installation script

Open `deploy/install.sh` and update user1/user2 to usernames chosen by you.

Perform the Installation

Go to the DTaaS directory and execute

```bash
source deploy/install.sh
```

You can run this script multiple times until the installation is successful.

> **Note**
>
> While installing you might encounter multiple dialogs asking, which services should be restarted. Just click **OK** to all of those.

Post-install Check

Now you should be able to access the DTaaS application at: https://foo.com

If you can following all the screenshots from user website. Everything is correctly setup.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.4 Vagrant

2.4.1 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:

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

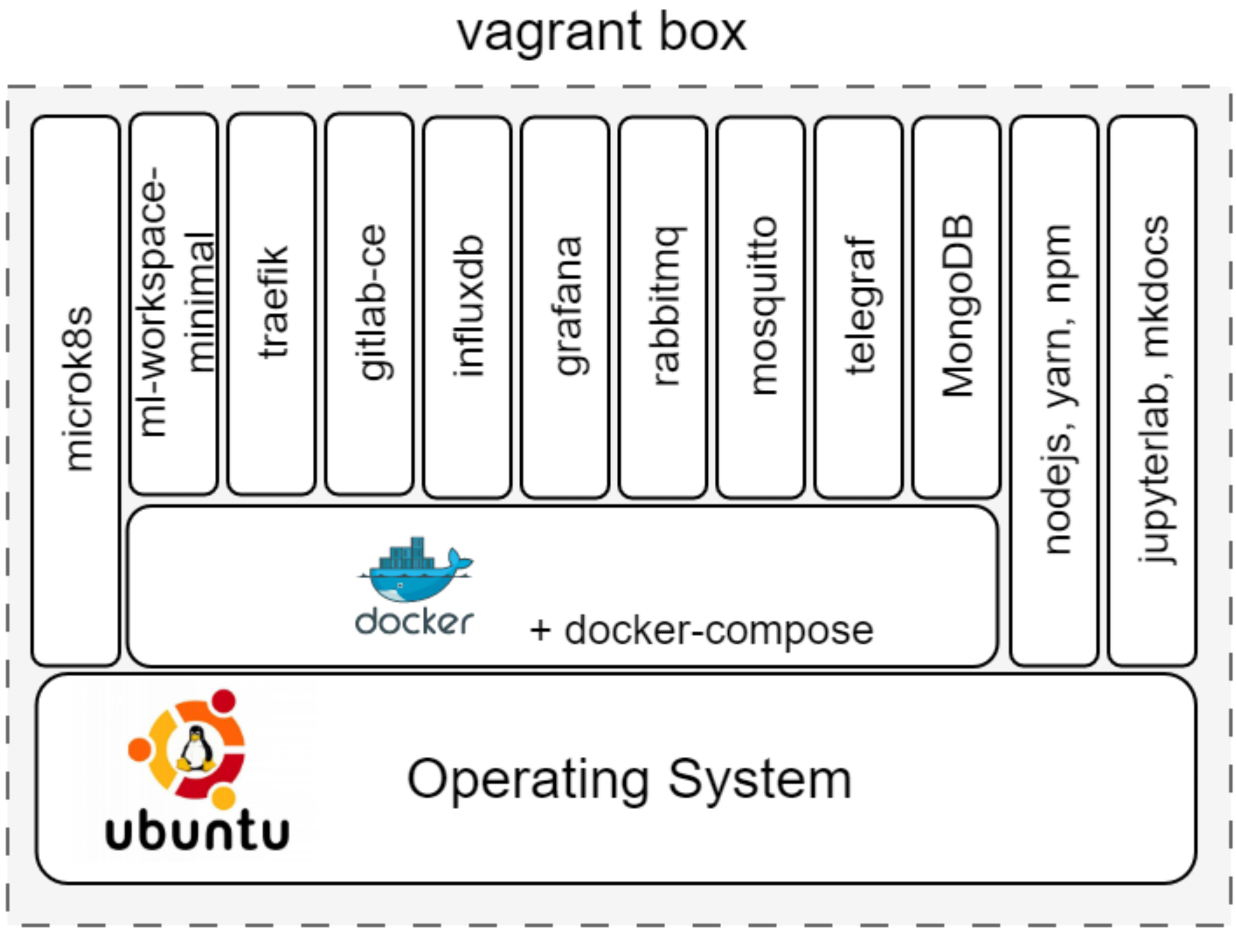
This vagrant box installed for users will have the following items:

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
ubuntu
Operating System

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

```
#create a key pair
ssh-keygen -b 4096 -t rsa -f vagrant -q -N ""

vagrant up

# let the provisioning be complete
# replace the vagrant ssh key-pair with personal one
vagrant ssh

# install the oh-my-zsh
sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
# install plugins: history, autosuggestions,
git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions

# inside ~/.zshrc, modify the following line
plugins=(git zsh-autosuggestions history cp tmux)

# to replace the default vagrant ssh key-pair with
# the generated private key into authorized keys
cp /vagrant/vagrant.pub /home/vagrant/.ssh/authorized_keys

# exit vagrant guest machine and then
# copy own private key to vagrant private key location
cp vagrant .vagrant/machines/default/virtualbox/private_key

# check
vagrant ssh #should work

# exit vagrant guest machine and then
vagrant halt

vagrant package --base dtaas \
--info "info.json" --output dtaas.vagrant

# Add box to the vagrant cache in ~/.vagrant.d/boxes directory
vagrant box add --name dtaas ./dtaas.vagrant

# You can use this box in other vagrant boxes using
#config.vm.box = "dtaas"
```

References

Image sources: Ubuntu logo

2.4.2 DTaaS on Single Vagrant Machine

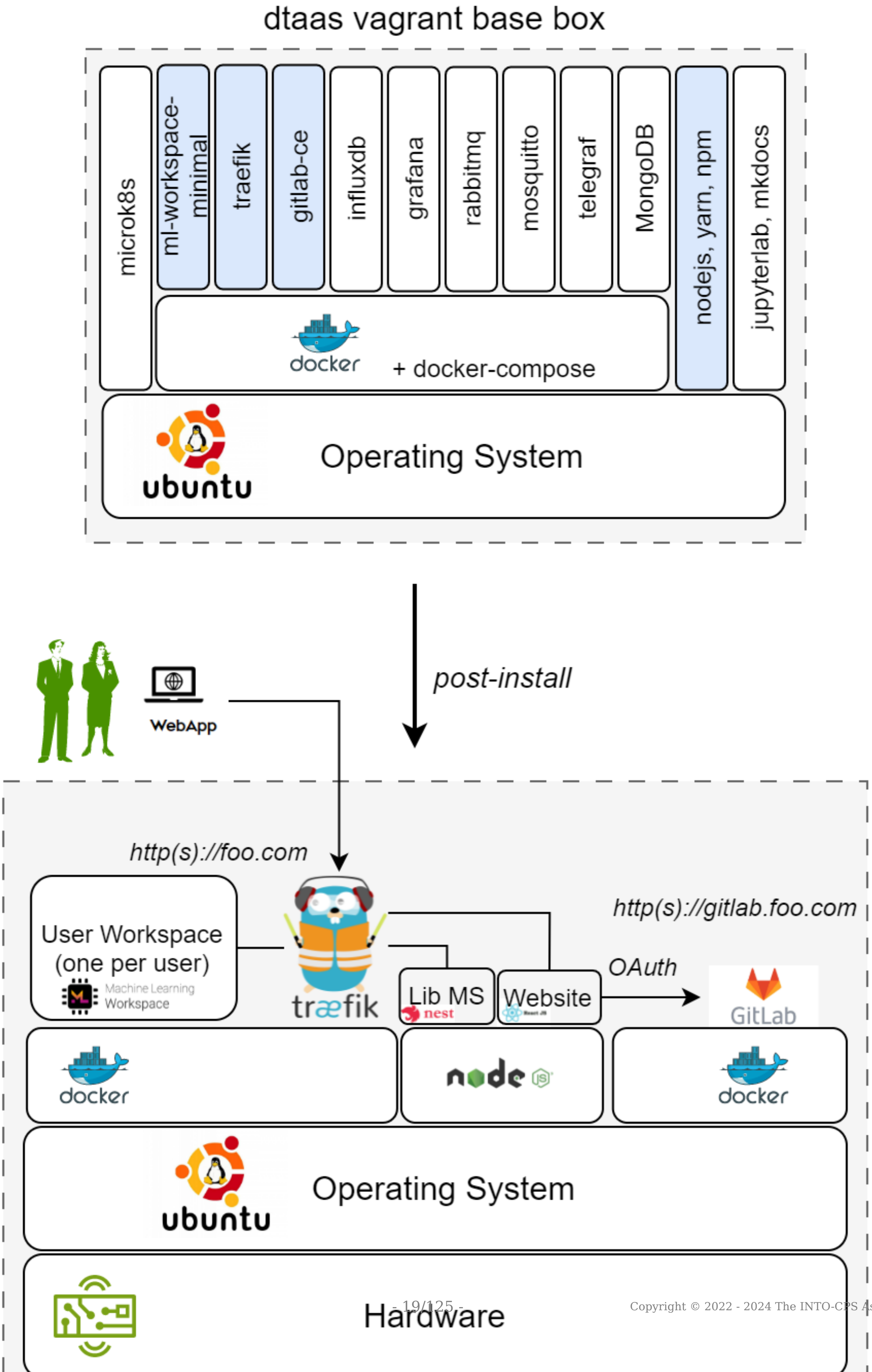
These are installation instructions for running DTaaS software inside one vagrant Virtual Machine. The setup requires a machine which can spare 16GB RAM, 8 vCPUs and 50GB Hard Disk space to the vagrant box.

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
microk8s
ml-workspace-minimal
traefik
gitlab-ce
influxdb
grafana
rabbitmq
mosquitto
telegraf
MongoDB
nodejs, yarn, npm
jupyterlab, mkdocs
docker
+ docker-compose
ubuntu
Operating System
post-install
WebApp
http(s)://foo.com
User Workspace
(one per user)
Machine Learning
Workspace
træfik
http(s)://gitlab.foo.com
OAuth
Lib MS
nest
Website
GitLab
docker
node
docker
ubuntu
Operating System
Hardware

There are many unused software packages/docker containers within the dtaas base box. The used packages/docker containers are highlighed in blue color.

> **Tip**
>
> The illustration shows hosting of gitlab on the same vagrant machine with *http(s)://gitlab.foo.com* The gitlab setup is outside the scope this installation guide. Please refer to gitlab docker install for gitlab installation.

Configure Server Settings

A dummy **foo.com** URL has been used for illustration. Please change this to your unique website URL.

Please follow the next steps to make this installation work in your local environment.

Update the **Vagrantfile**. Fields to update are:

1. Hostname (`node.vm.hostname = "foo.com"`)
2. MAC address (`:mac => "xxxxxxxx"`). This change is required if you have a DHCP server assigning domain names based on MAC address. Otherwise, you can leave this field unchanged.
3. Other adjustments are optional.

Installation Steps

Execute the following commands from terminal

```
vagrant up
vagrant ssh
```

Set a cronjob inside the vagrant virtual machine to remote the conflicting default route.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.4/deploy/vagrant/route.sh
sudo bash route.sh
```

If you only want to test the application and are not setting up a production instance, you can follow the instructions of single script install.

If you are not in a hurry and would rather have a production instance, follow the instructions of regular server installation setup to complete the installation.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.4.3 DTaaS on Two Vagrant Machines

These are installation instructions for running DTaaS application in two vagrant virtual machines (VMs). In this setup, all the user workspaces shall be run on server1 while all the platform services will be run on server2.

The setup requires two server VMs with the following hardware configuration:

server1: 16GB RAM, 8 x64 vCPUs and 50GB Hard Disk space

server2: 6GB RAM, 3 x64 vCPUs and 50GB Hard Disk space

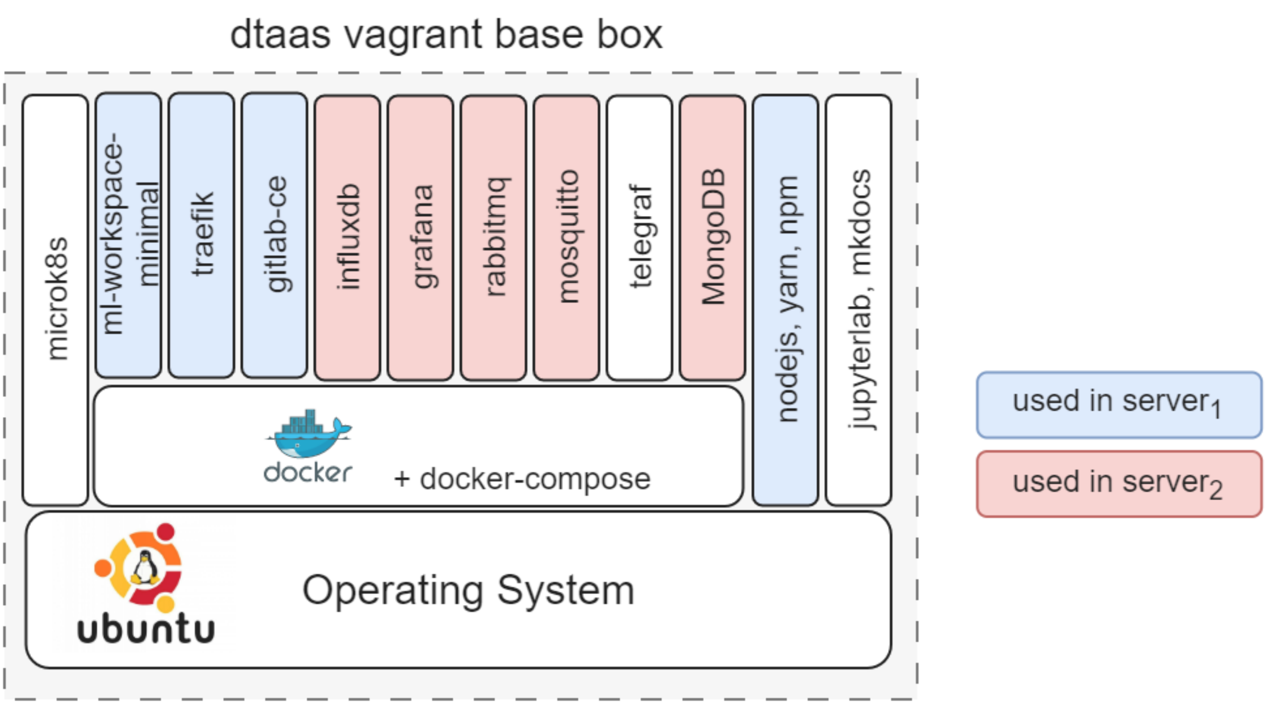
Under the default configuration, two user workspaces are provisioned on server1. The default installation setup also installs InfluxDB, Grafana, RabbitMQ and MQTT services on server2. If you would like to install more services, you can create shell scripts to install the same on server2.

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/two-machine`). This shall be useful for logging into the vagrant machines created for two-machine deployment.

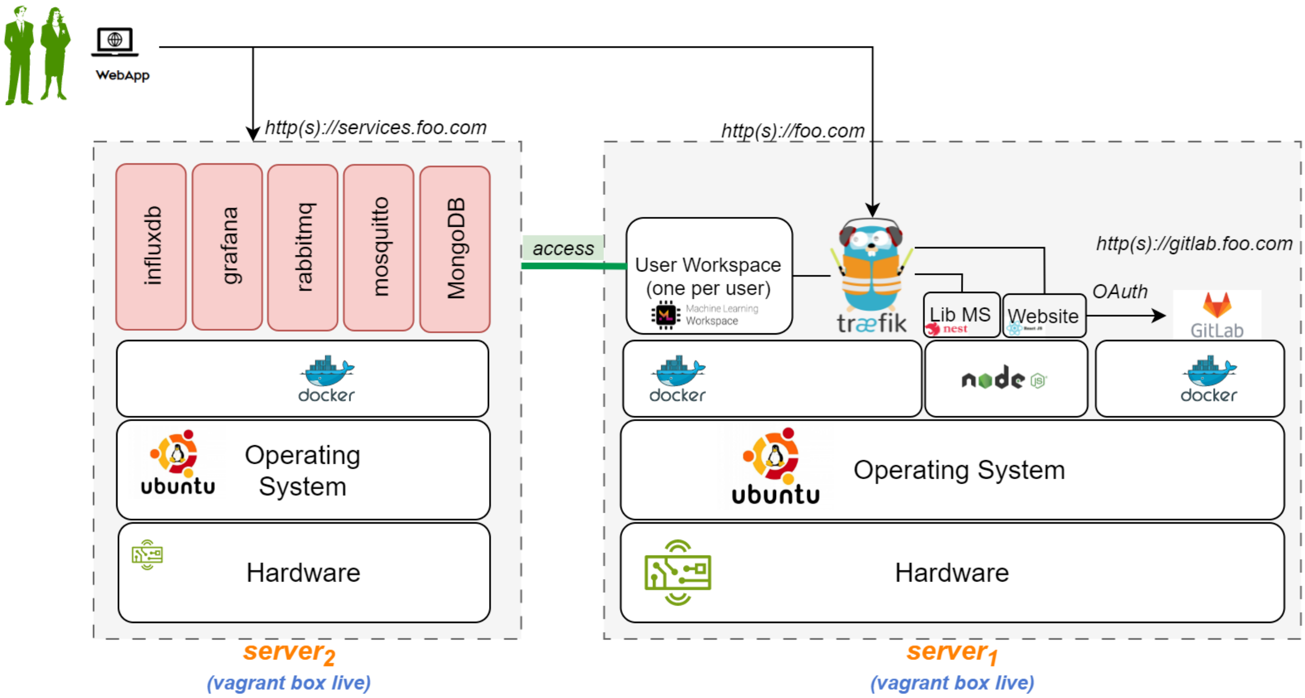
Target Installation Setup

The goal is to use this **dtaas** vagrant box to install the DTaaS software on server1 and the default platform services on server2. Both the servers are vagrant machines.



There are many unused software packages/docker containers within the dtaas base box. The used packages/docker containers are highlighed in blue and red color.

A graphical illustration of a successful installation can be seen here.



In this case, both the vagrant boxes are spawed on one server using two vagrant configuration files, namely *boxes.json* and *Vagrantfile*.

> **Tip**
>
> The illustration shows hosting of gitlab on the same vagrant machine with *http(s)://gitlab.foo.com* The gitlab setup is outside the scope this installation guide. Please refer to gitlab docker install for gitlab installation.

Configure Server Settings

NOTE: A dummy **foo.com** and **services.foo.com** URLs has been used for illustration. Please change these to your unique website URLs.

The first step is to define the network identity of the two VMs. For that, you need *server name*, *hostname* and *MAC address*. The hostname is the network URL at which the server can be accessed on the web. Please follow these steps to make this work in your local environment.

Update the **boxes.json**. There are entries one for each server. The fields to update are:

1. `name` - name of server1 (`"name" = "dtaas-two"`)
2. `hostname` - hostname of server1 (`"name" = "foo.com"`)
3. MAC address (`:mac => "xxxxxxxx"`). This change is required if you have a DHCP server assigning domain names based on MAC address. Otherwise, you can leave this field unchanged.
4. `name` - name of server2 (`"name" = "services"`)
5. `hostname` - hostname of server2 (`"name" = "services.foo.com"`)
6. MAC address (`:mac => "xxxxxxxx"`). This change is required if you have a DHCP server assigning domain names based on MAC address. Otherwise, you can leave this field unchanged.
7. Other adjustments are optional.

Installation Steps

The installation instructions are given separately for each vagrant machine.

LAUNCH DTAAS PLATFORM DEFAULT SERVICES

Follow the installation guide for services to install the DTaaS platform services.

After the services are up and running, you can see the following services active within server2 (*services.foo.com*).

service	external url
InfluxDB database	services.foo.com
Grafana visualization service	services.foo.com:3000
MQTT Broker	services.foo.com:1883
RabbitMQ Broker	services.foo.com:5672
RabbitMQ Broker management website	services.foo.com:15672
MongoDB database	services.foo.com:27017

INSTALL DTAAS APPLICATION

Execute the following commands from terminal

```
vagrant up --provision dtaas
vagrant ssh dtaas
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.4/deploy/vagrant/route.sh
sudo bash route.sh
```

If you only want to test the application and are not setting up a production instance, you can follow the instructions of single script install.

If you are not in a hurry and would rather have a production instance, follow the instructions of regular server installation setup to complete the installation.

References

Image sources: Ubuntu logo, Traefik logo, ml-workspace, nodejs, reactjs, nestjs

2.5 Separate Packages

2.5.1 Host the DTaaS Client Website

Tip

The Localhost installation setup is a faster way to use the DTaaS software. You might want to consider the localhost option before using only the **DTaaS-client.zip** package.

To host DTaaS client website on your server, follow these steps:

- Download the **DTaaS-client.zip** from the releases page.
- Inside the `DTaaS-client` directory, there is `site` directory. The `site` directory contains all the optimized static files that are ready for deployment.
- Setup the oauth application on gitlab instance. See the instructions in authorization page for completing this task.
- Locate the file `site/env.js` and replace the example values to match your infrastructure. The constructed links will be "`REACT_APP_URL` / `REACT_APP_URL_BASENAME` / `{username}` / `{Endpoint}`". See the definitions below:

```
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: "prod | dev",
    REACT_APP_URL: "URL for the gateway",
    REACT_APP_URL_BASENAME: "Base URL for the client website"(optional),
    REACT_APP_URL_DTLINK: "Endpoint for the Digital Twin",
    REACT_APP_URL_LIBLINK: "Endpoint for the Library Assets",
    REACT_APP_WORKBENCHLINK_VNCDESKTOP: "Endpoint for the VNC Desktop link",
    REACT_APP_WORKBENCHLINK_VSCODE: "Endpoint for the VS Code link",
    REACT_APP_WORKBENCHLINK_JUPYTERLAB: "Endpoint for the Jupyter Lab link",
    REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK:
      "Endpoint for the Jupyter Notebook link",
    REACT_APP_CLIENT_ID: 'AppID genereated by the gitlab OAuth provider',
    REACT_APP_AUTH_AUTHORITY: 'URL of the private gitlab instance',
    REACT_APP_REDIRECT_URI: 'URL of the homepage for the logged in users of the website',
    REACT_APP_LOGOUT_REDIRECT_URI: 'URL of the homepage for the anonymous users of the website',
    REACT_APP_GITLAB_SCOPES: 'OAuth scopes. These should match with the scopes set in gitlab OAuth provider',
  };
};

// Example values with no base URL. Trailing and ending slashes are optional.
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: 'prod',
    REACT_APP_URL: 'https://foo.com/',
    REACT_APP_URL_BASENAME: '',
    REACT_APP_URL_DTLINK: '/lab',
    REACT_APP_URL_LIBLINK: '',
    REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',
    REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',
    REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/lab',
    REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',
    REACT_APP_CLIENT_ID: '934b98f03f1b6f743832b2840bf7cccaed93c3bfe579093dd0942a433691ccc0',
    REACT_APP_AUTH_AUTHORITY: 'https://gitlab.foo.com/',
    REACT_APP_REDIRECT_URI: 'https://foo.com/Library',
    REACT_APP_LOGOUT_REDIRECT_URI: 'https://foo.com/',
    REACT_APP_GITLAB_SCOPES: 'openid profile read_user read_repository api',
  };
};


// Example values with "bar" as basename URL.
//Trailing and ending slashes are optional.
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: "dev",
    REACT_APP_URL: 'https://foo.com/',
    REACT_APP_URL_BASENAME: 'bar',
    REACT_APP_URL_DTLINK: '/lab',
    REACT_APP_URL_LIBLINK: '',
    REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',
    REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',
    REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/lab',
    REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',
    REACT_APP_CLIENT_ID: '934b98f03f1b6f743832b2840bf7cccaed93c3bfe579093dd0942a433691ccc0',
    REACT_APP_AUTH_AUTHORITY: 'https://gitlab.foo.com/',
    REACT_APP_REDIRECT_URI: 'https://foo.com/bar/Library',
    REACT_APP_LOGOUT_REDIRECT_URI: 'https://foo.com/bar',
    REACT_APP_GITLAB_SCOPES: 'openid profile read_user read_repository api',
  };
};
```

- Copy the entire contents of the build folder to the root directory of your server where you want to deploy the app. You can use FTP, SFTP, or any other file transfer protocol to transfer the files.
- Make sure your server is configured to serve static files. This can vary depending on the server technology you are using, but typically you will need to configure your server to serve files from a specific directory.
- Once the files are on your server, you should be able to access your app by visiting your server's IP address or domain name in a web browser.

ℹ The website depends on **Traefik gateway** and **ML Workspace** components to be available. Otherwise, you only get a skeleton non-functional website.

Complementary Components

The website requires background services for providing actual functionality. The minimum background service required is atleast one ML Workspace serving the following routes.

```
https://foo.com/<username>/lab
https://foo.com/<username>/terminals/main
https://foo.com/<username>/tools/vnc/?password=vncpassword
https://foo.com/<username>/tools/vscode/
```

The `username` is the user workspace created using ML Workspace docker container. Please follow the instructions in README. You can create as many user workspaces as you want. If you have two users - alice and bob - on your system, then the following the commands in will instantiate the required user workspaces.

```
mkdir -p files/alice files/bob files/common

printf "\n\n start the user workspaces"
docker run -d \
 -p 8090:8080 \
  --name "ml-workspace-alice" \
  -v "$(pwd)/files/alice:/workspace" \
  -v "$(pwd)/files/common:/workspace/common" \
  --env AUTHENTICATE_VIA_JUPYTER="" \
  --env WORKSPACE_BASE_URL="alice" \
  --shm-size 512m \
  --restart always \
  mltooling/ml-workspace-minimal:0.13.2

docker run -d \
 -p 8091:8080 \
  --name "ml-workspace-bob" \
  -v "$(pwd)/files/bob:/workspace" \
  -v "$(pwd)/files/common:/workspace/common" \
  --env AUTHENTICATE_VIA_JUPYTER="" \
  --env WORKSPACE_BASE_URL="bob" \
  --shm-size 512m \
  --restart always \
  mltooling/ml-workspace-minimal:0.13.2
```

Given that multiple services are running at different routes, a reverse proxy is needed to map the background services to external routes. You can use Apache, NGINX, Traefik or any other software to work as reverse proxy.

The website screenshots and usage information is available in user page.

2.5.2 Host Library Microservice

The **lib microservice** is a simplified file manager providing graphQL API. It has three features:

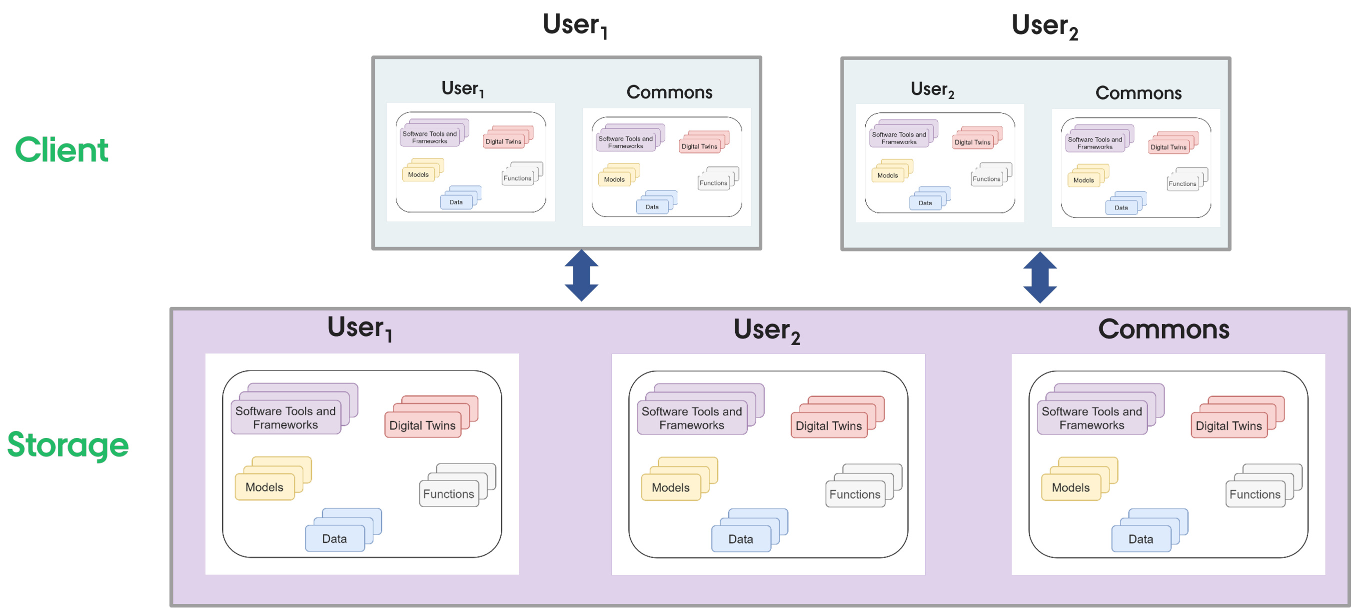
- provide a listing of directory contents.
- transfer a file to user.
- Source files can either come from local file system or from a gitlab instance.

The library microservice is designed to manage and serve files, functions, and models to users, allowing them to access and interact with various resources.

This document provides instructions for running a stand alone library microservice.

Setup the File System

The users expect the following file system structure for their reusable assets.



There is a skeleton file structure in DTaaS codebase. You can copy and create file system for your users.

Install

The package is available in Github packages registry and on npmjs.

Set the registry and install the package with the one of the two following commands

```
sudo npm install -g @into-cps-association/libms  # requires no login
sudo npm config set @into-cps-association:registry https://npm.pkg.github.com
```

The *github package registry* asks for username and password. The username is your Github username and the password is your Github personal access token. In order for the npm to download the package, your personal access token needs to have *read:packages* scope.

⚙️ Configure

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

```
PORT='4001'
MODE='local'
LOCAL_PATH='/Users/<Username>/DTaaS/files'
LOG_LEVEL='debug'
APOLLO_PATH='/lib' or ''
GRAPHQL_PLAYGROUND='false' or '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.

🚀 Use

Display help.

```
libms -h
```

The config is saved `.env` file by convention. The **libms** looks for `.env` file in the working directory from which it is run. If you want to run **libms** without explicitly specifying the configuration file, run

```
libms
```

To run **libms** with a custom config file,

```
libms -c FILE-PATH
libms --config FILE-PATH
```

If the environment file is named something other than `.env`, for example as `.env.development`, you can run

```
libms -c ".env.development"
```

You can press `Ctl+C` to halt the application. If you wish to run the microservice in the background, use

```
nohup libms [-c FILE-PATH] & disown
```

The lib microservice is now running and ready to serve files, functions, and models.

Service Endpoint

The URL endpoint for this microservice is located at: `localhost:PORT/lib`

The service API documentation is available on user page.

2.6 Third-party Services

The DTaaS software platform uses third-party software services to provide enhanced value to users.

InfluxDB, Grafana, RabbitMQ and Mosquitto are default services integrated into the DTaaS software platform.

2.6.1 Pre-requisites

All these services run on raw TCP/UDP ports. Thus a direct network access to these services is required for both the DTs running inside the DTaaS software and the PT located outside the DTaaS software.

There are two possible choices here:

- Configure Traefik gateway to permit TCP/UDP traffic
- Bypass Traefik altogether

Unless you are an informed user of Traefik, we recommend bypassing traefik and provide raw TCP/UDP access to these services from the Internet.

The InfluxDB service requires a dedicated hostname. The management interface of RabbitMQ service requires a dedicated hostname as well.

Grafana service can run well behind Traefik gateway. The default Traefik configuration makes permits access to Grafana at URL: http(s): *foo.com/vis*.

2.6.2 Configure and Install

If you have not cloned the DTaaS git repository, cloning would be the first step. In case you already have the codebase, you can skip the cloning step. To clone, do:

```
git clone https://github.com/into-cps-association/DTaaS.git
git checkout release-v0.4
cd DTaaS/deploy/services
```

The next step in installation is to specify the config of the services. There are two configuration files. The **services.yml** contains most of configuration settings. The **mqtt-default.conf** file contains the MQTT listening port. Update these two config files before proceeding with the installation of the services.

Now continue with the installation of services.

```
yarn install
node services.js
```

2.6.3 Use

After the installation is complete, you can see the following services active at the following ports / URLs.

service	external url
Influx	services.foo.com
Grafana	services.foo.com:3000
RabbitMQ Broker	services.foo.com:5672
RabbitMQ Broker Management Website	services.foo.com:15672
MQTT Broker	services.foo.com:1883
MongoDB database	services.foo.com:27017

The firewall and network access settings of corporate / cloud network need to be configured to allow external access to the services. Otherwise the users of DTaaS will not be able to utilize these services from their user workspaces.

2.7 Guides

2.7.1 Add a new user

This page will guide you on, how to add more users to the DTaas. Please do the following:

> **Important**
>
> Make sure to replace **<username>** and **<port>** Select a port that is not already being used by the system.

1. Add user:

Add the new user on the Gitlab instance.

2. Setup a new workspace:

The above code creates a new workspace for the new user based on *user2*.

```
cd DTaaS/files
cp -R user2 <username>
cd ..
docker run -d \
 -p <port>:8080 \
  --name "ml-workspace-<username>" \
  -v "${TOP_DIR}/files/<username>:/workspace" \
  -v "${TOP_DIR}/files/<username>:/workspace/common" \
  --env AUTHENTICATE_VIA_JUPYTER="" \
  --env WORKSPACE_BASE_URL="<username>" \
  --shm-size 512m \
  --restart always \
  mltooling/ml-workspace-minimal:0.13.2
```

3. Add username and password:

The following code adds basic authorization for the new user.

```
cd DTaaS/servers/config/gateway
htpasswd auth <username>
```

4. Add 'route' for new user:

We need to add a new route to the servers ingress.

Open the following file with your preffered editor (e.g. VIM/nano).

```
vi DTaaS/servers/config/gateway/dynamic/fileConfig.yml
```

Now add the new route and service for the user.

> **Important**

foo.com should be replaced with your own domain.

```
http:
  routers:
    ....
    <username>:
      entryPoints:
        - http
      rule: 'Host(`foo.com`) && PathPrefix(`/<username>`)'
      middlewares:
        - basic-auth
      service: <username>

  services:
    ...
    <username>:
      loadBalancer:
        servers:
          - url: 'http://localhost:<port>'
```

5. Access the new user:

Log into the DTaaS application as new user.

2.7.2 Add other services

Pre-requisite

You should read the documentation about the already available services

This guide will show you how to add more services. In the following example we will be adding **MongoDB** as a service, but these steps could be modified to install other services as well.

Adding other services requires more RAM and CPU power. Please make sure the host machine meets the hardware requirements for running all the services.

1. Add the configuration:

Select configuration parameters for the MongoDB service.

Configuration Variable Name	Description
username	the username of the root user in the MongoDB
password	the password of the root user in the MongoDB
port	the mapped port on the host machine (default is 27017)
datapath	path on host machine to mount the data from the MongoDB container

Open the file `/deploy/services/services.yml` and add the configuration for MongoDB:

```
services:
    rabbitmq:
        username: "dtaas"
        password: "dtaas"
        vhost: "/"
        ports:
            main: 5672
            management: 15672
    ...
    mongodb:
        username: <username>
        password: <password>
        port: <port>
        datapath: <datapath>
    ...
```

2. Add the script:

The next step is to add the script that sets up the MongoDB container with the configuraiton.

Create new file named `/deploy/services/mongodb.js` and add the following code:

```
#!/usr/bin/node
/* Install the optional platform services for DTaaS */
import { $ } from "execa";
import chalk from "chalk";
import fs from "fs";
import yaml from "js-yaml";

const $$ = $({ stdio: "inherit" });
const log = console.log;
let config;

try {
  log(chalk.blue("Load services configuration"));
  config = await yaml.load(fs.readFileSync("services.yml", "utf8"));
  log(
    chalk.green(
      "configuration loading is successful and config is a valid yaml file"
    )
  );
} catch (e) {
  log(chalk.red("configuration is invalid. Please rectify services.yml file"));
  process.exit(1);
}

log(chalk.blue("Start MongoDB server"));
const mongodbConfig = config.services.mongodb;

try {
  log(
    chalk.green(
      "Attempt to delete any existing MongoDB server docker container"
    )
  );
  await $$`docker stop mongodb`;
  await $$`docker rm mongodb`;
} catch (e) {}

log(chalk.green("Start new Mongodb server docker container"));
await $$`docker run -d -p ${mongodbConfig.port}:27017 \
  --name mongodb \
  -v ${mongodbConfig.datapath}:/data/db \
  -e MONGO_INITDB_ROOT_USERNAME=${mongodbConfig.username} \
  -e MONGO_INITDB_ROOT_PASSWORD=${mongodbConfig.password} \
  --restart always \
  mongo:7.0.3`;
log(chalk.green("MongoDB server docker container started successfully"));
```

3. Run the script:

Go to the directory `/deploy/services/` and run services script with the following commands:

```
yarn install
node mongodb.js
```

The MongoDB should now be available on **services.foo.com:<port>**.

2.7.3 Install Gitlab

This guide helps with installation of a dedicated Gitlab service. This Gitlab installation can be used as OAuth2 authorization provider to the DTaaS software.

There are two possible ways you can install Gitlab:

- At dedicated domain name (ex: *gitlab.foo.com*)
- At a URL path on existing WWW server (ex: foo.com/gitlab

2.7.4 Link services to local ports

Requirements

- User needs to have an account on server2.
- SSH server must be running on server2

To link a port from the service machine (server2) to the local port on the user workspace. You can use ssh local port forwarding technique.

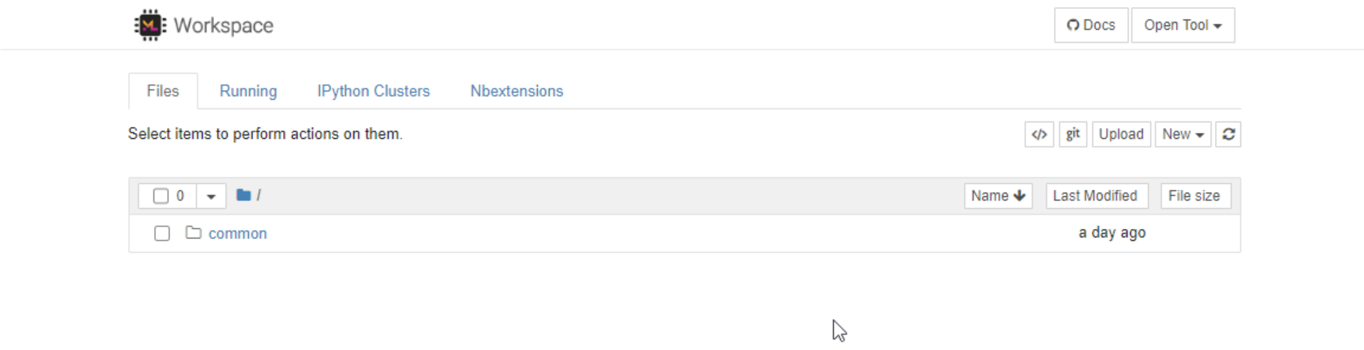
1. Step:

Go to the user workspace, on which you want to map from localhost to the services machine

- e.g. `foo.com/user1`

2. Step:

Open a terminal in your user workspace.



3. Step:

Run the following command to map a port:

```
ssh -fNT -L <local_port>:<destination>:<destination_port> <user>@<services.server.com>
```

Here's an example mapping the RabbitMQ broker service available at 5672 of *services.foo.com* to localhost port 5672.

```
ssh -fNT -L 5672:localhost:5672 vagrant@services.foo.com
```

Now the programs in user workspace can treat the RabbitMQ broker service as a local service running within user workspace.

2.7.5 Make common asset area read only

Why

In some cases you might want to restrict the access rights of some users to the common assets. In order to make the common area read only, you have to change the install script section performing the creation of user workspaces.

How

To make the common assets read-only for user2, the following changes need to be made to the install script, which is located one of the following places.

- trial installation: `single-script-install.sh`
- production installation: `DTaas/deploy/install.sh`

The line `-v "${TOP_DIR}/files/common:/workspace/common:ro"` was added to make the common workspace read-only for user2.

Here's the updated code:

```
docker run -d \
  -p 8091:8080 \
  --name "ml-workspace-user2" \
  -v "${TOP_DIR}/files/user2:/workspace" \
  -v "${TOP_DIR}/files/common:/workspace/common:ro" \
  --env AUTHENTICATE_VIA_JUPYTER="" \
  --env WORKSPACE_BASE_URL="user2" \
  --shm-size 512m \
  --restart always \
  mltooling/ml-workspace-minimal:0.13.2 || true
```

This ensures that the common area is read-only for user2, while the user's own (private) assets are still writable.

2.7.6 Hosting site without https

In the default trial or production installation setup, the https connection is provided by the reverse proxy. The DTaaS application by default runs in http mode. So removing the reverse proxy removes the https mode.

2.7.7 Update basepath/route for the application

The updates required to make the application work with basepath (say bar):

1. Change the Gitlab OAuth URLs to include basepath:

```
REACT_APP_AUTH_AUTHORITY: 'https://gitlab.foo.com/',
REACT_APP_REDIRECT_URI: 'https://foo.com/bar/Library',
REACT_APP_LOGOUT_REDIRECT_URI: 'https://foo.com/bar',
```

2. Update traefik gateway config (deploy/config/gateway/fileConfig.yml):

```
http:
  routers:
    dtaas:
      entryPoints:
        - http
      rule: "Host(`foo.com`)" #remember, there is no basepath for this rule
      middlewares:
        - basic-auth
      service: dtaas

    user1:
      entryPoints:
        - http
      rule: "Host(`foo.com`) && PathPrefix(`/bar/user1`)"
      middlewares:
        - basic-auth
      service: user1

  # Middleware: Basic authorization
  middlewares:
    basic-auth:
      basicAuth:
        usersFile: "/etc/traefik/auth"
        removeHeader: true

  services:
    dtaas:
      loadBalancer:
        servers:
          - url: "http://localhost:4000"

    user1:
      loadBalancer:
        servers:
          - url: "http://localhost:8090"
```

3. Update deploy/config/client/env.js:

See the client documentation for an example.

4. Update install scripts:

Update *deploy/install.sh* by adding basepath. For example, add *WORKSPACE_BASE_URL="bar/"* for all user workspaces.

For user1, the docker command changes to:

```
docker run -d \
 -p 8090:8080 \
  --name "ml-workspace-user1" \
  -v "${TOP_DIR}/files/user1:/workspace" \
  -v "${TOP_DIR}/files/common:/workspace/common" \
  --env AUTHENTICATE_VIA_JUPYTER="" \
  --env WORKSPACE_BASE_URL="bar/user1" \
  --shm-size 512m \
  --restart always \
  mltooling/ml-workspace-minimal:0.13.2 || true
```

5. Proceed with install using deploy/install.sh:

3. User

3.1 Motivation

How can DT software platforms enable users collaborate to:

- Build digital twins (DTs)
- Use DTs themselves
- Share DTs with other users
- Provide the existing DTs as Service to other users

In addition, how can the DT software platforms:

- Support DT lifecycle
- Scale up rather than scale down (flexible convention over configuration)

3.1.1 Existing Approaches

There are quite a few solutions proposed in the recent past to solve this problem. Some of them are:

- Focus on data from Physical Twins (PTs) to perform analysis, diagnosis, planning etc…
- Share DT assets across the upstream, downstream etc.…
- Evaluate different models of PT
- DevOps for Cyber Physical Systems (CPS)
- Scale DT / execution of DT / ensemble of related DTs
- Support for PT product lifecycle

3.1.2 Our Approach

- Support for transition from existing workflows to DT frameworks
- Create DTs from reusable assets
- Enable users to share DT assets
- Offer DTs as a Service
- Integrate the DTs with external software systems
- Separate configurations of independent DT components

3.2 Overview

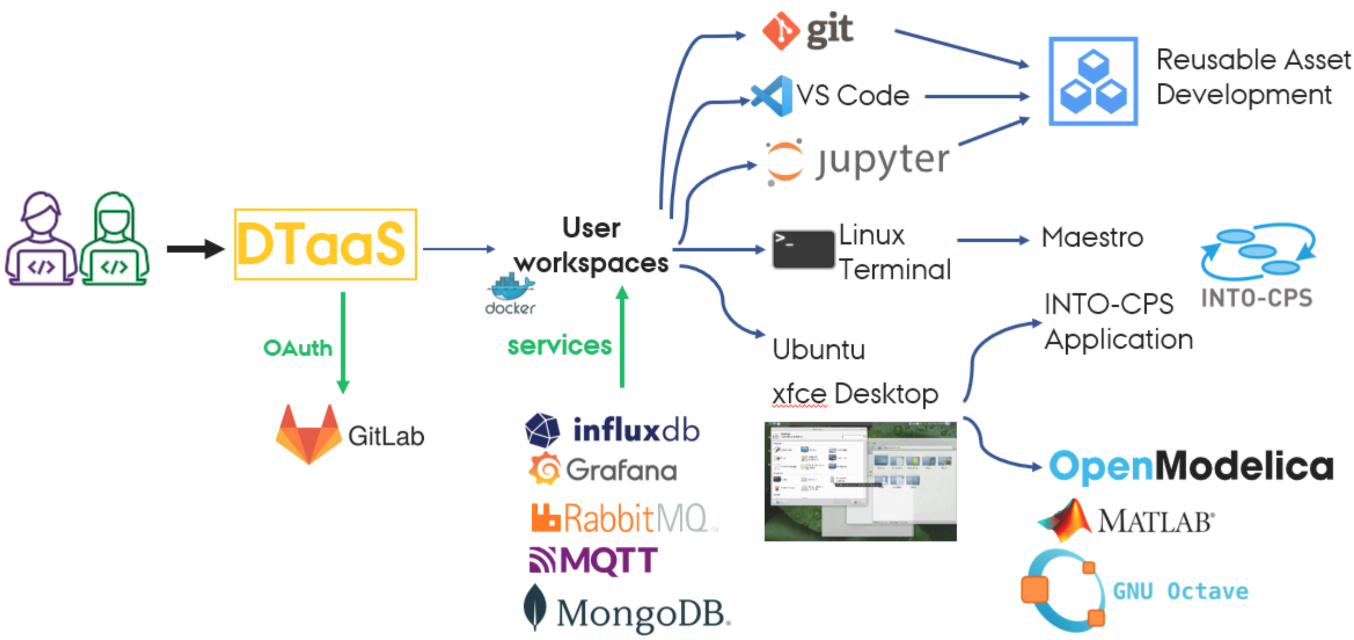
3.2.1 Advantages

The DTaaS software platform provides certain advantages to users:

- Support for different kinds of Digital Twins
- CFD, Simulink, co-simulation, FEM, ROM, ML etc.
- Integrates with other Digital Twin frameworks
- Facilitate availability of Digital Twin as a Service
- Collaboration and reuse
- Private workspaces for verification of reusable assets, trial run DTs
- Cost effectiveness

3.2.2 Software Features

Each installation of DTaaS platform comes with the features highlighted in the following picture.



All the users have dedicated workspaces. These workspaces are dockerized versions of Linux Desktops. The user desktops are isolated so the installations and customizations done in one user workspace do not effect the other user workspaces.

Each user workspace comes with some development tools pre-installed. These tools are directly accessible from web browser. The following tools are available at present:

Tool	Advantage
Jupyter Lab	Provides flexible creation and use of digital twins and their components from web browser. All the native Jupyterlab usecases are supported here.
Jupyter Notebook	Useful for web-based management of their files (library assets)
VS Code in the browser	A popular IDE for software development. Users can develop their digital twin-related assets here.
ungit	An interactive git client. Users can work with git repositories from web browser

In addition, users have access to xfce-based remote desktop via VNC client. The VNC client is available right in the web browser. The xfce supported desktop software can also be run in their workspace.

The DTaaS software platform has some pre-installed services available. The currently available services are:

Service	Advantage
InfluxDB	Time-series database primarly for storing time-series data from physical twins. The digital twins can use an already existing data. Users can also create visualization dashboards for their digital twins.
RabbitMQ	Communication broker for communication between physical and digital twins
Grafana	Visualization dashboards for their digital twins.
MQTT	Lightweight data transfer broker for IoT devices / physical twins feeding data into digital twins.
MongoDB	NoSQL document database for storing metadata of data from physical twins

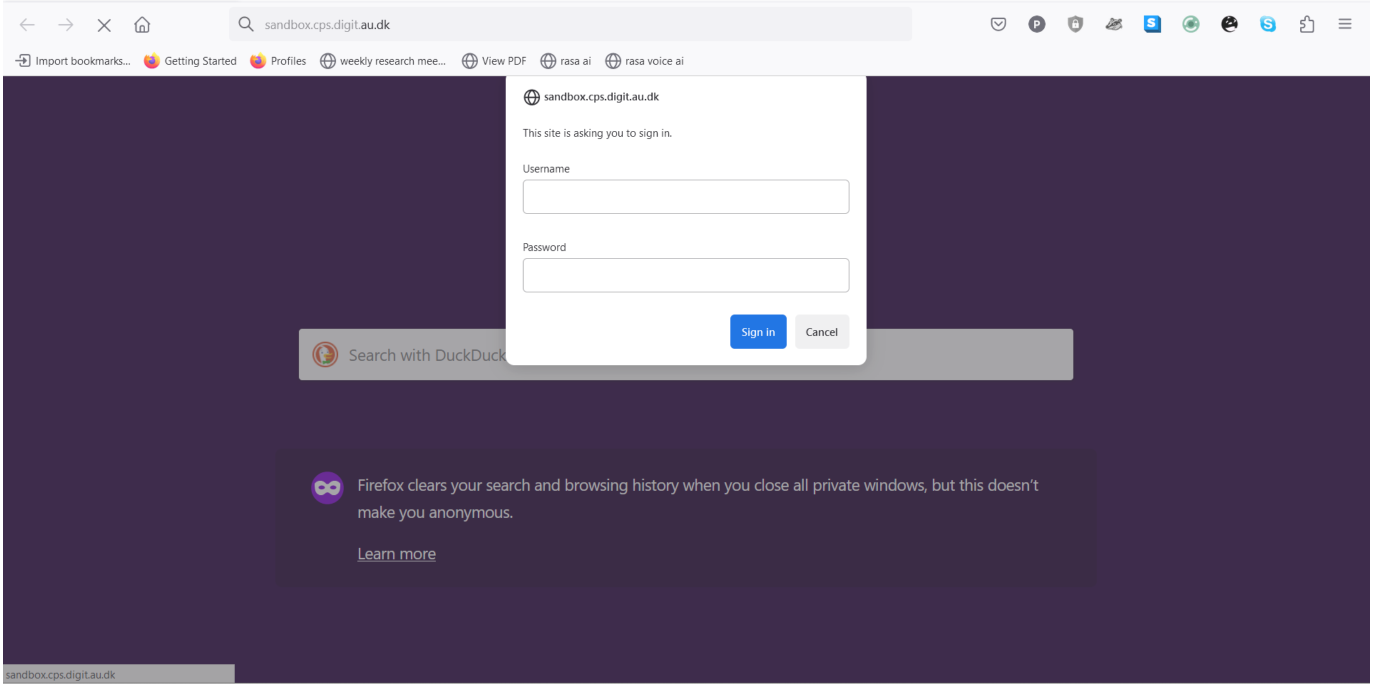
In addition, the workspaces are connected to the Internet so all the Digital Twins running in the workspace can interact with both the internal and external services.

The users can publish and reuse the digital twin assets available on the platform. In addition, users can run their digital twins and make these live digital twins available as services to their clients. The clients need not be users of the DTaaS software installation.

3.3 DTaaS Website Screenshots

This page contains a screenshot driven preview of the website serving the DTaaS software platform.

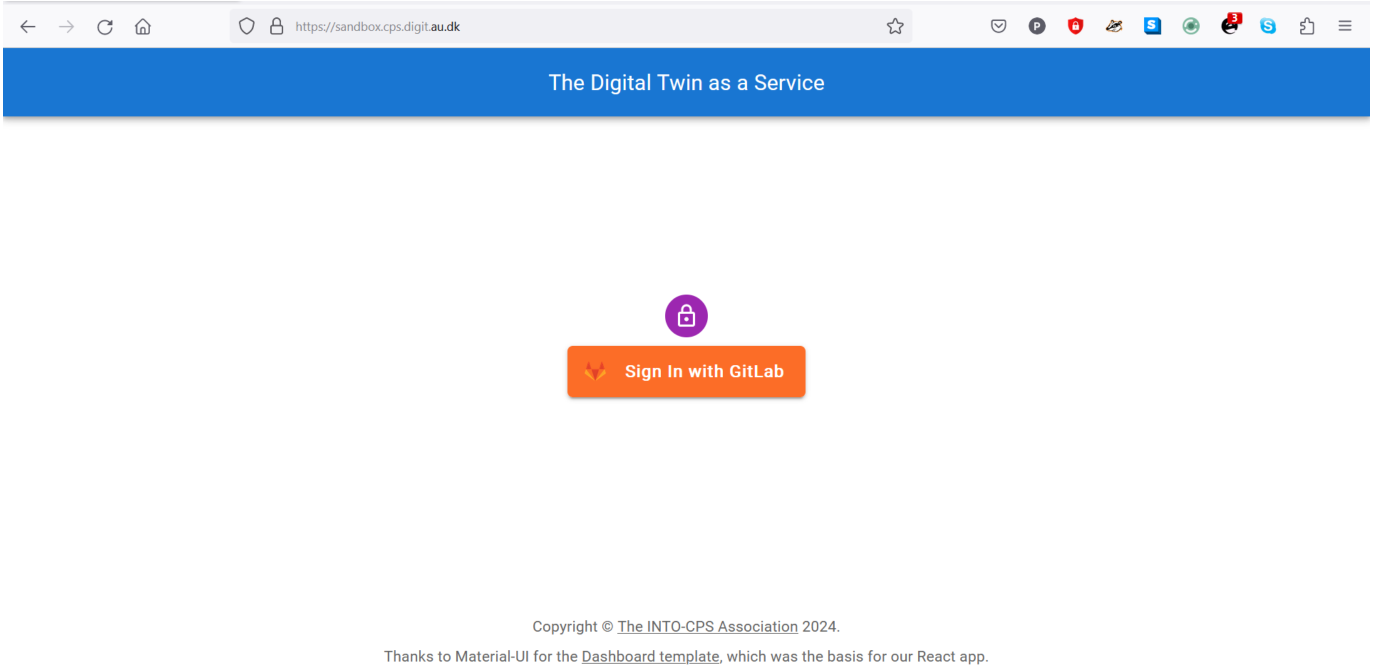
3.3.1 Login to enter the DTaaS software platform



The screen presents with HTTP authorization form. You can enter the user credentials. If the DTaaS is being served over HTTPS secure communication protocol, the username and password are secure.

3.3.2 Start the Authorization

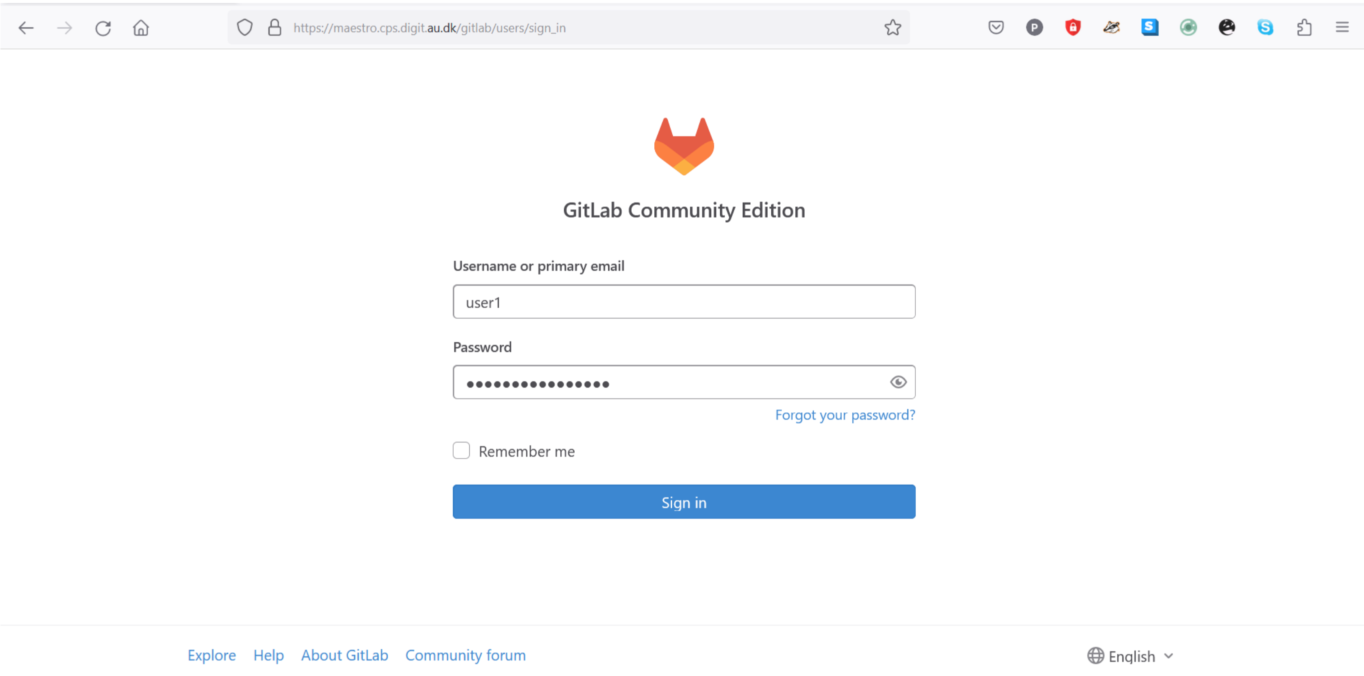
You are now logged into the DTaaS server. The DTaaS uses third-party authorization protocol known as OAuth. This protocol provides secure access to a DTaaS installation if users have a working active accounts at the selected OAuth service provider. The DTaaS uses Gitlab as OAuth provider.



You can see the Gitlab signin button. A click on this button takes you to Gitlab instance providing authorization for DTaaS.

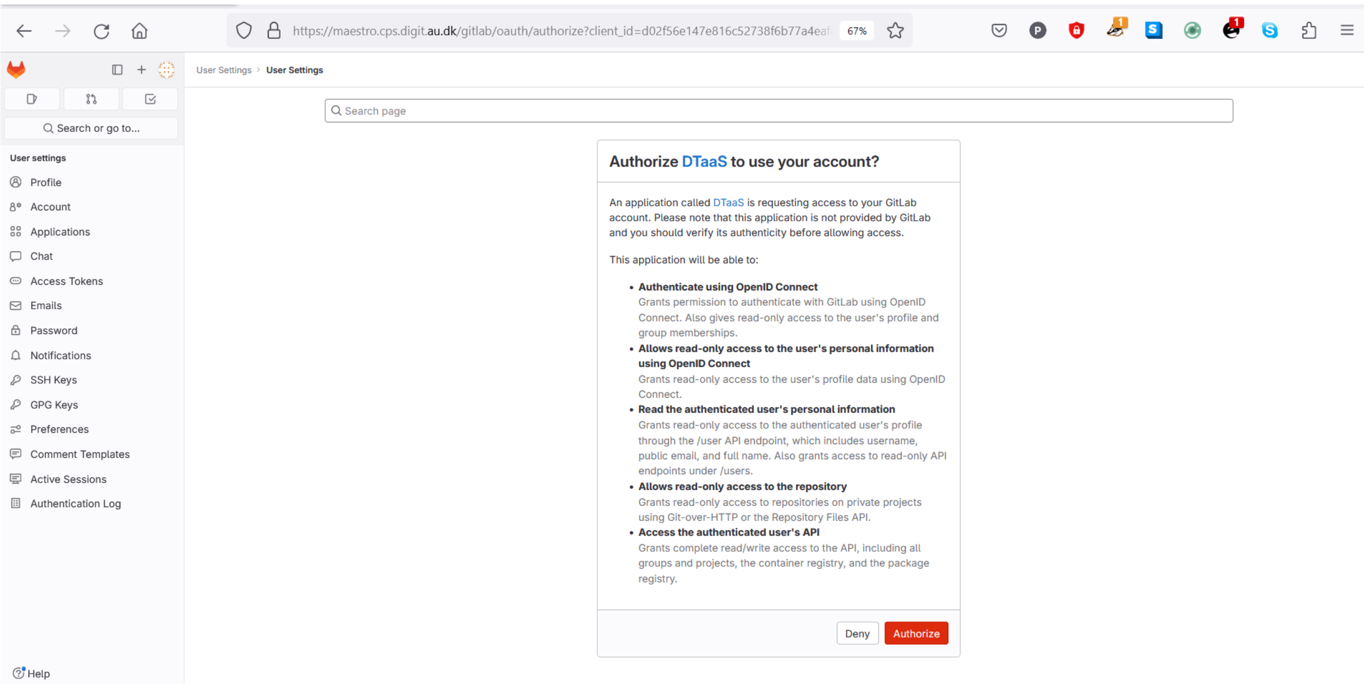
3.3.3 Authorize at Gitlab

The username and password authorization takes place on the gitlab website. Enter your username and password in the login form.



3.3.4 Permit DTaaS to Use Gitlab

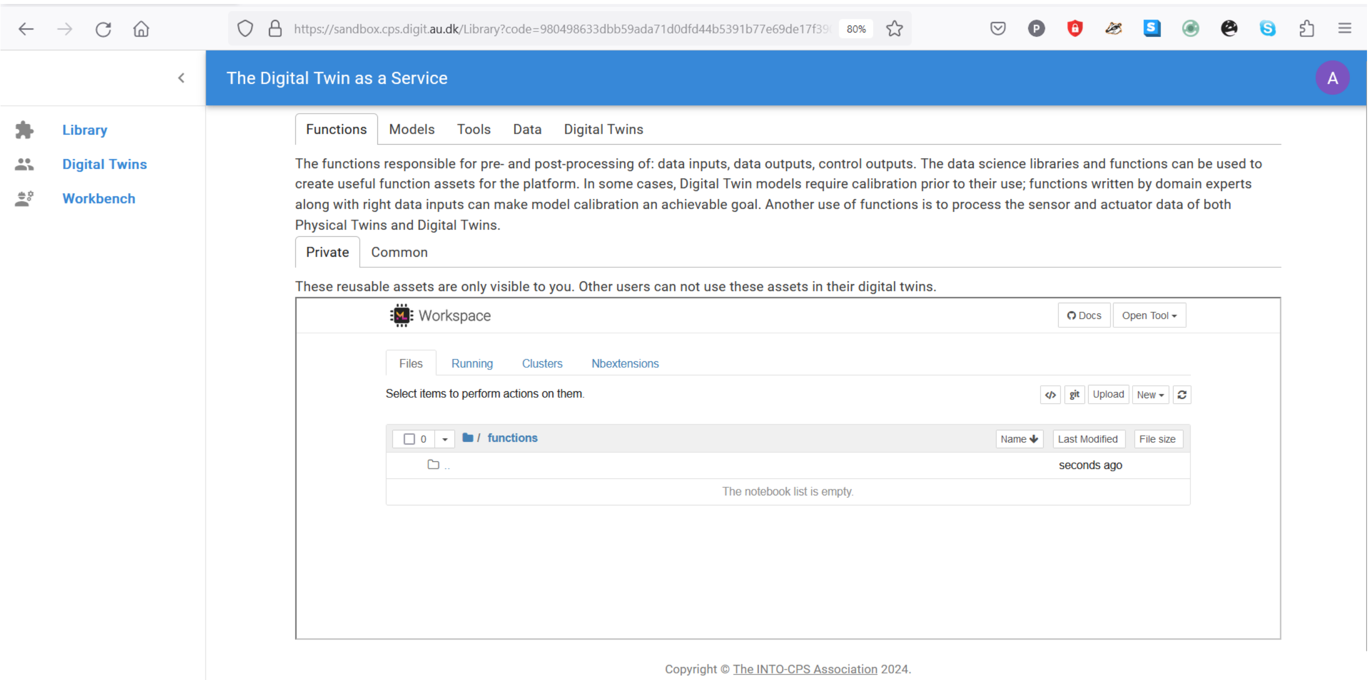
The DTaaS application needs your permission to use your Gitlab account for authorization. Click on **Authorize** button.



After successful authorization, you will be redirected to the **Library** page of the DTaaS website.

3.3.5 Overview of menu items

The menu is hidden by default. Only the icons of menu items are visible. You can click on the ☰ icon in the top-left corner of the page to see the menu.



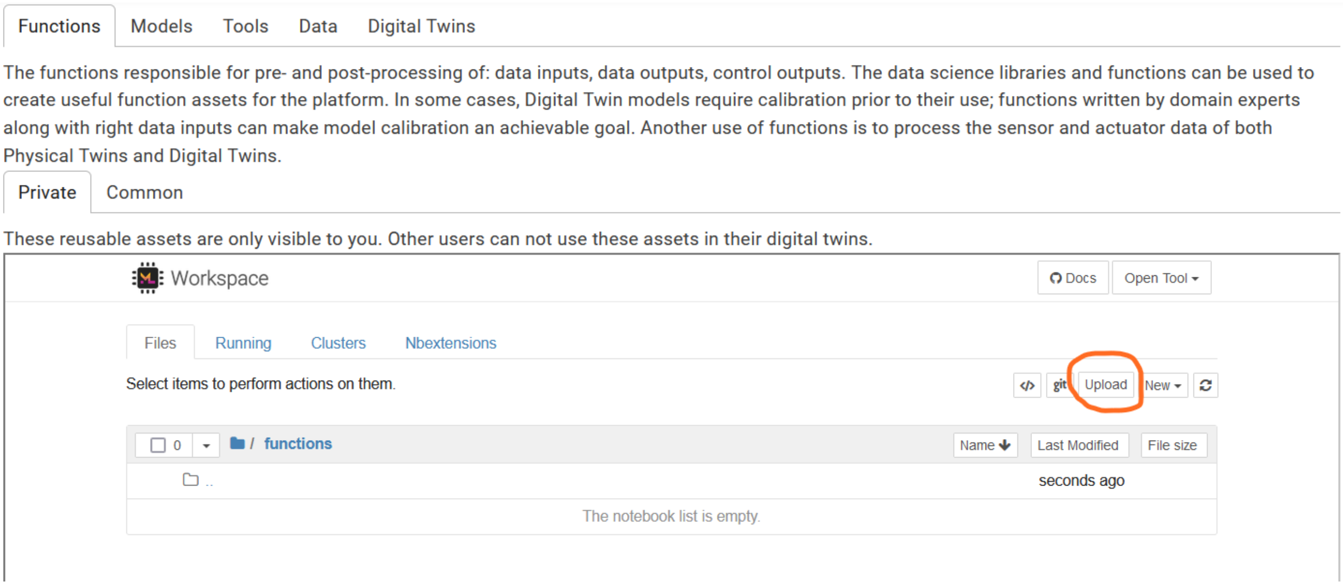
There are three menu items:

Library: for management of reusable library assets. You can upload, download, create and modify new files on this page.

Digital Twins: for management of digital twins. You are presented with the Jupyter Lab page from which you can run the digital twins.

Workbench: Not all digital twins can be managed within Jupyter Lab. You have more tools at your disposal on this page.

3.3.6 Library tabs and their help text



You can see the file manager and five tabs above the library manager. Each tab provides help text to guide users in the use of different directories in their workspace.

Functions

The functions responsible for pre- and post-processing of: data inputs, data outputs, control outputs. The data science libraries and functions can be used to create useful function assets for the platform. In some cases, Digital Twin models require calibration prior to their use; functions written by domain experts along with right data inputs can make model calibration an achievable goal. Another use of functions is to process the sensor and actuator data of both Physical Twins and Digital Twins.

Data

The data sources and sinks available to a digital twins. Typical examples of data sources are sensor measurements from Physical Twins, and test data provided by manufacturers for calibration of models. Typical examples of data sinks are visualization software, external users and data storage services. There exist special outputs such as events, and commands which are akin to control outputs from a Digital Twin. These control outputs usually go to Physical Twins, but they can also go to another Digital Twin.

Models

The model assets are used to describe different aspects of Physical Twins and their environment, at different levels of abstraction. Therefore, it is possible to have multiple models for the same Physical Twin. For example, a flexible robot used in a car production plant may have structural model(s) which will be useful in tracking the wear and tear of parts. The same robot can have a behavioural model(s) describing the safety guarantees provided by the robot manufacturer. The same robot can also have a functional model(s) describing the part manufacturing capabilities of the robot.

> **Tools**
>
> The software tool assets are software used to create, evaluate and analyze models. These tools are executed on top of a computing platforms, i.e., an operating system, or virtual machines like Java virtual machine, or inside docker containers. The tools tend to be platform specific, making them less reusable than models. A tool can be packaged to run on a local or distributed virtual machine environments thus allowing selection of most suitable execution environment for a Digital Twin. Most models require tools to evaluate them in the context of data inputs. There exist cases where executable packages are run as binaries in a computing environment. Each of these packages are a pre-packaged combination of models and tools put together to create a ready to use Digital Twins.

> **Digital**
>
> These are ready to use digital twins created by one or more users. These digital twins can be reconfigured later for specific use cases.

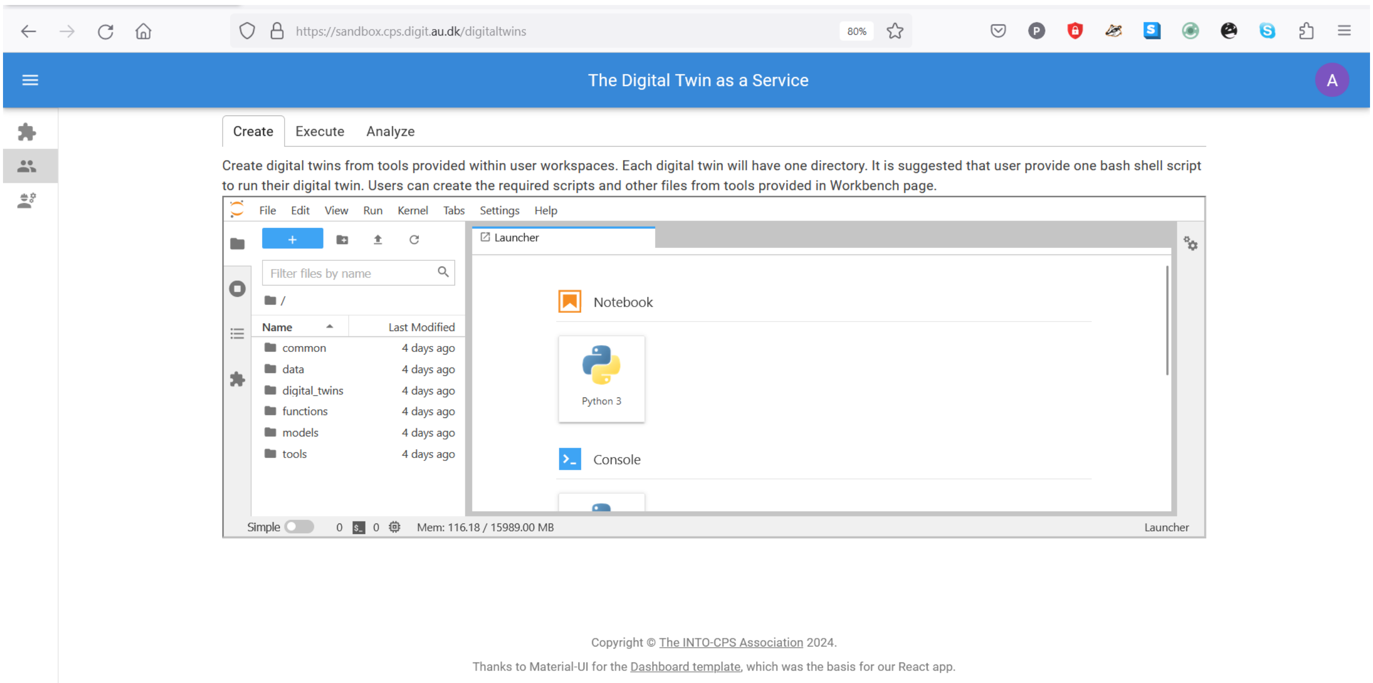
In addition to the five directories, there is also **common** directory in which five sub-directories exist. These sub-directories are: data, functions, models, tools and digital twins.

> **Common**
>
> The common directory again has four sub-directories: - data - functions - functions - models - tools - digital twins The assets common to all users are placed in **common**.

The items used by more than one user are placed in **common**. The items in the **common** directory are available to all users. Further explanation of directory structure and placement of reusable assets within the the directory structure is in the assets page

The file manager is based on Jupyter notebook and all the tasks you can perform in the Jupyter Notebook can be undertaken here.

3.3.7 Digital Twins page



The digital twins page has three tabs and the central pane opens Jupyter lab. There are three tabs with helpful instructions on the suggested tasks you can undertake in the **Create - Execute - Analyze** life cycle phases of digital twin. You can see more explanation on the life cycle phases of digital twin.

Create

Create digital twins from tools provided within user workspaces. Each digital twin will have one directory. It is suggested that user provide one bash shell script to run their digital twin. Users can create the required scripts and other files from tools provided in Workbench page.

Execute

Digital twins are executed from within user workspaces. The given bash script gets executed from digital twin directory. Terminal-based digital twins can be executed from VSCode and graphical digital twins can be executed from VNC GUI. The results of execution can be placed in the data directory.

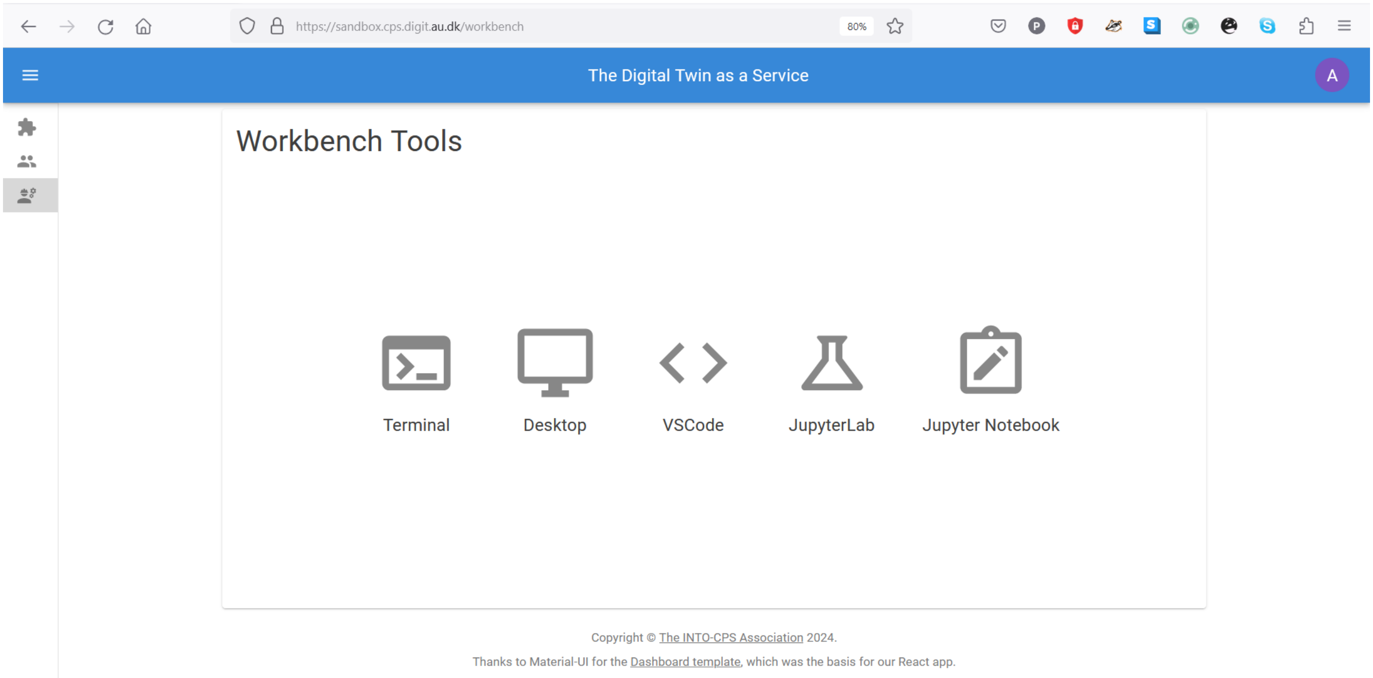
Analyze

The analysis of digital twins requires running of digital twin script from user workspace. The execution results placed within data directory are processed by analysis scripts and results are placed back in the data directory. These scripts can either be executed from VSCode and graphical results or can be executed from VNC GUI. The analysis of digital twins requires running of digital twin script from user workspace. The execution results placed within data directory are processed by analysis scripts and results are placed back in the data directory. These scripts can either be executed from VSCode and graphical results or can be executed from VNC GUI.

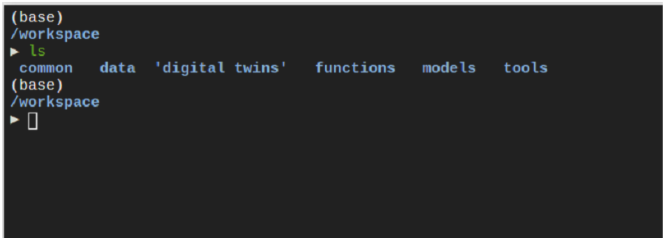
The reusable assets (files) seen in the file manager are available in the Jupyter Lab. In addition, there is a git plugin installed in the Jupyter Lab using which you can link your files with the external git repositories.

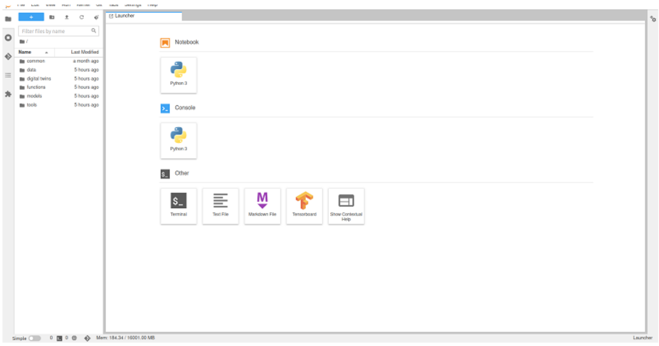
3.3.8 Workbench

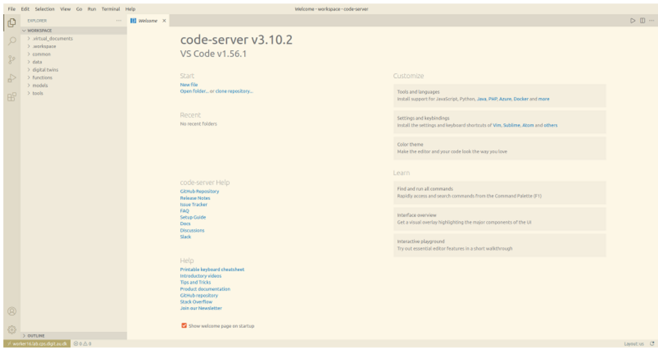
The **workbench** page provides links to four integrated tools.

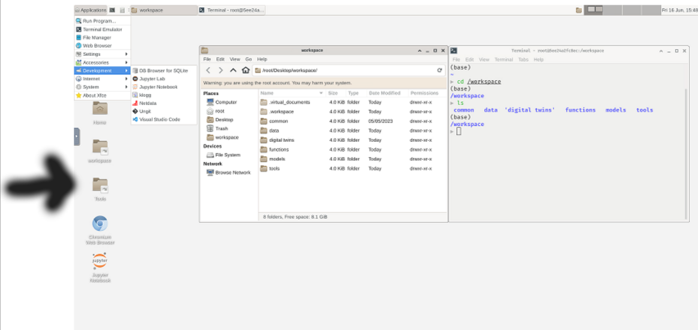


The hyperlinks open in new browser tab. The screenshots of pages opened in new browser are:



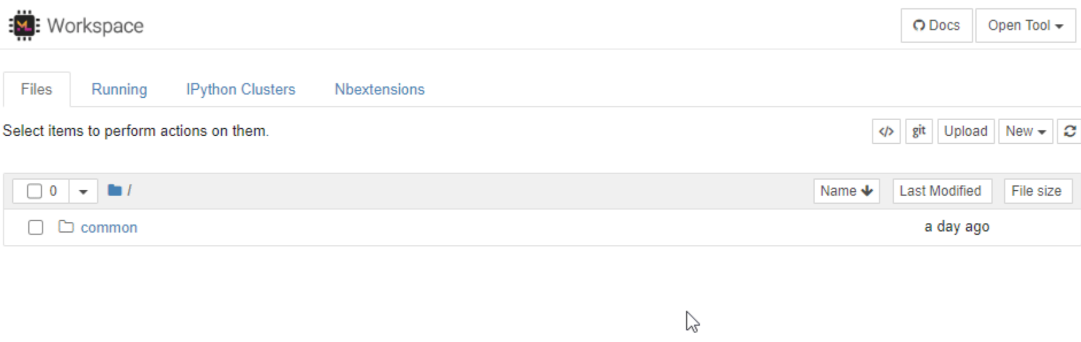




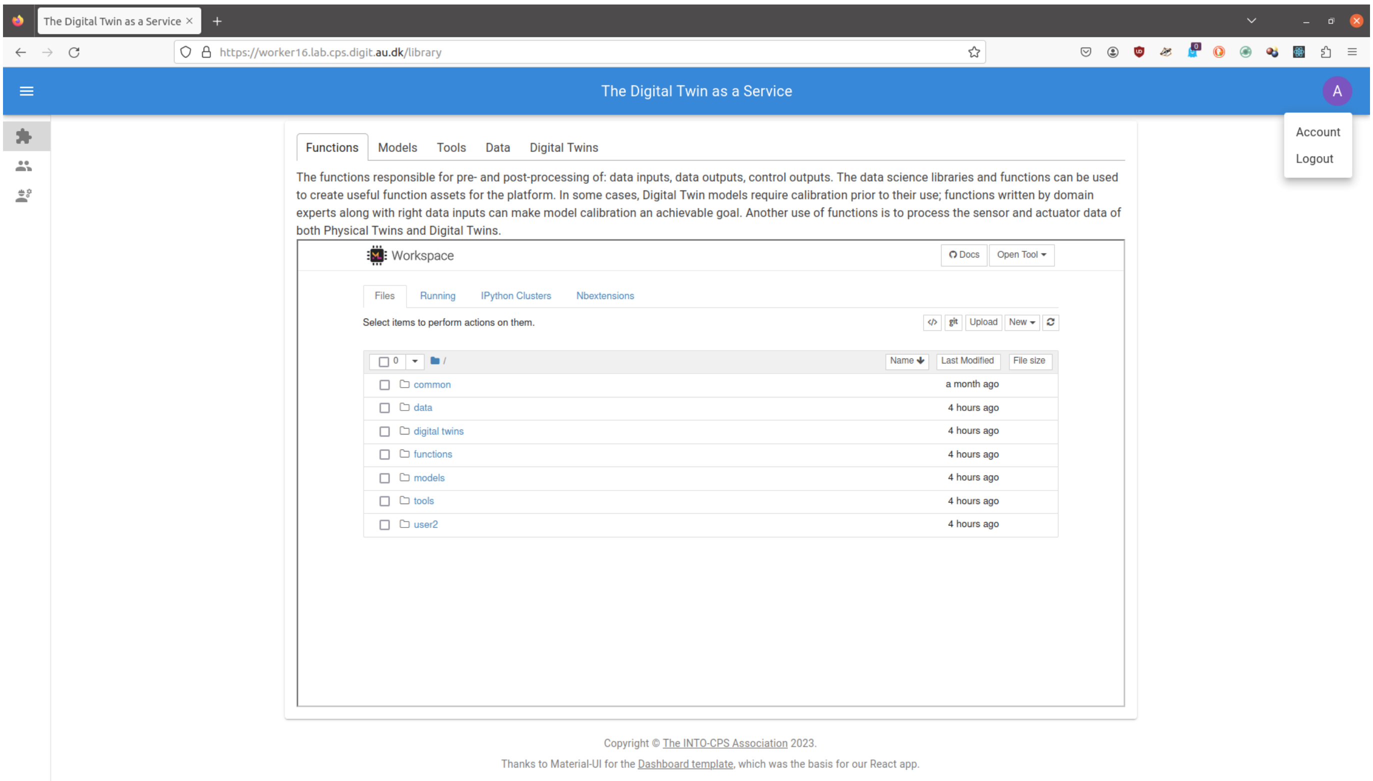


Bug

The Terminal hyperlink does not always work reliably. If you want terminal. Please use the tools dropdown in the Jupyter Notebook. The Terminal hyperlink does not always work reliably. If you want terminal, please use the tools dropdown in the Jupyter Notebook.



3.3.9 Finally logout



You have to close the browser in order to completely exit the DTaaS software platform.

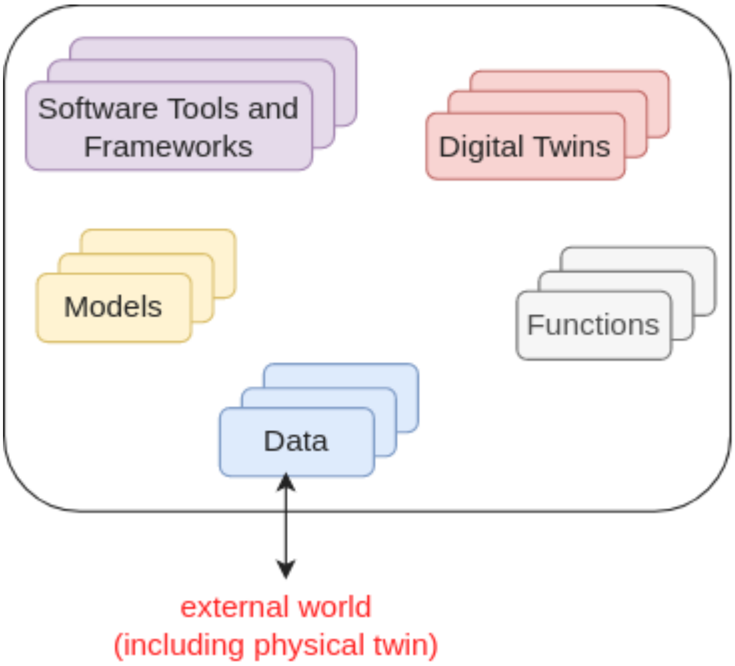
3.4 Library

3.4.1 Reusable Assets

The reusability of digital twin assets makes it easy for users to work with the digital twins. The reusability of assets is a fundamental feature of the platform.

Kinds of Reusable Assets

The DTaaS software categorizes all the reusable library assets into six categories:



DATA

The data sources and sinks available to a digital twins. Typical examples of data sources are sensor measurements from Physical Twins, and test data provided by manufacturers for calibration of models. Typical examples of data sinks are visualization software, external users and data storage services. There exist special outputs such as events, and commands which are akin to control outputs from a Digital Twin. These control outputs usually go to Physical Twins, but they can also go to another Digital Twin.

MODELS

The model assets are used to describe different aspects of Physical Twins and their environment, at different levels of abstraction. Therefore, it is possible to have multiple models for the same Physical Twin. For example, a flexible robot used in a car production plant may have structural model(s) which will be useful in tracking the wear and tear of parts. The same robot can have a behavioural model(s) describing the safety guarantees provided by the robot manufacturer. The same robot can also have a functional model(s) describing the part manufacturing capabilities of the robot.

TOOLS

The software tool assets are software used to create, evaluate and analyze models. These tools are executed on top of a computing platforms, i.e., an operating system, or virtual machines like Java virtual machine, or inside docker containers. The tools tend to be platform specific, making them less reusable than models. A tool can be packaged to run on a local or distributed virtual machine environments thus allowing selection of most suitable execution environment for a Digital Twin. Most models require tools to evaluate them in the context of data inputs. There exist cases where executable packages are run as binaries in a computing environment. Each of these packages are a pre-packaged combination of models and tools put together to create a ready to use Digital Twins.

FUNCTIONS

The functions responsible for pre- and post-processing of: data inputs, data outputs, control outputs. The data science libraries and functions can be used to create useful function assets for the platform. In some cases, Digital Twin models require calibration prior to their use; functions written by domain experts along with right data inputs can make model calibration an achievable goal. Another use of functions is to process the sensor and actuator data of both Physical Twins and Digital Twins.

DIGITAL TWINS

These are ready to use digital twins created by one or more users. These digital twins can be reconfigured later for specific use cases.

File System Structure

Each user has their assets put into five different directories named above. In addition, there will also be common library assets that all users have access to. A simplified example of the structure is as follows:

```
workspace/
  data/
    data1/ (ex: sensor)
      filename (ex: sensor.csv)
      README.md
    data2/ (ex: turbine)
      README.md (remote source; no local file)
    ...
  digital_twins/
    digital_twin-1/ (ex: incubator)
      code and config
      README.md (usage instructions)
    digital_twin-2/ (ex: mass spring damper)
      code and config
      README.md (usage instructions)
    digital_twin-3/ (ex: model swap)
      code and config
      README.md (usage instructions)
    ...
  functions/
    function1/ (ex: graphs)
      filename (ex: graphs.py)
      README.md
    function2/ (ex: statistics)
      filename (ex: statistics.py)
      README.md
    ...
  models/
    model1/ (ex: spring)
      filename (ex: spring.fmu)
      README.md
    model2/ (ex: building)
      filename (ex: building.skp)
      README.md
    model3/ (ex: rabbitmq)
      filename (ex: rabbitmq.fmu)
      README.md
    ...
  tools/
    tool1/ (ex: maestro)
      filename (ex: maestro.jar)
      README.md
    ...
  common/
    data/
    functions/
    models/
    tools/
```

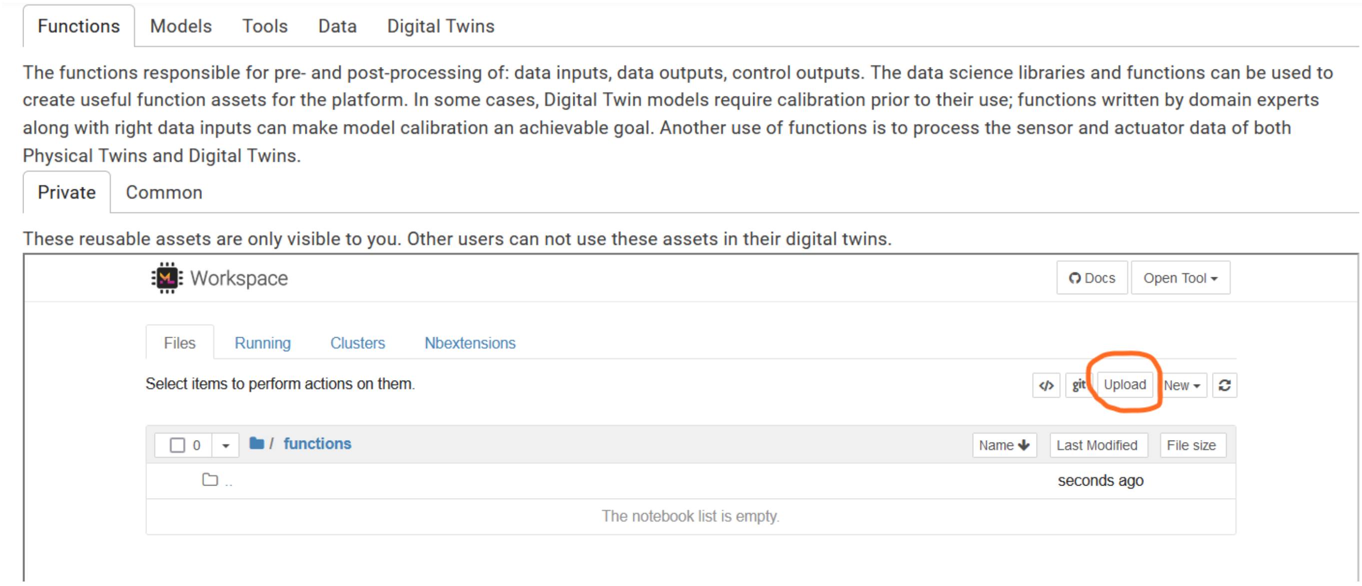
Tip

The DTaaS is agnostic to the format of your assets. The only requirement is that they are files which can be uploaded on the Library page. Any directories can be compressed as one file and uploaded. You can decompress the file into a directory from a Terminal or xfce Desktop available on the Workbench page.

A recommended file system structure for storing assets is also available in DTaaS examples.

Upload Assets

Users can upload assets into their workspace using Library page of the website.



You can go into a directory and click on the **upload** button to upload a file or a directory into your workspace. This asset is then available in all the workbench tools you can use. You can also create new assets on the page by clicking on **new** drop down menu. This is a simple web interface which allows you to create text-based files. You need to upload other files using **upload** button.

The user workbench has the following services:

- Jupyter Notebook and Lab
- VS Code
- XFCE Desktop Environment available via VNC
- Terminal

Users can also bring their DT assets into user workspaces from outside using any of the above mentioned services. The developers using *git* repositories can clone from and push to remote git servers. Users can also use widely used file transfer protocols such as FTP, and SCP to bring the required DT assets into their workspaces.

3.4.2 Library Microservice

The library microservice provides an API interface to reusable assets library. This is only for expert users who need to integrate the DTaaS with their own IT systems. Regular users can safely skip this page.

The lib microservice is responsible for handling and serving the contents of library assets of the DTaaS platform. It provides API endpoints for clients to query, and fetch these assets.

This document provides instructions for using the library microservice.

Please see assets for a suggested storage conventions of your library assets.

Once the assets are stored in the library, you can access the server's endpoint by typing in the following URL: `http://foo.com/lib`.

The URL opens a graphql playground. You can check the query schema and try sample queries here. You can also send graphql queries as HTTP POST requests and get responses.

API Queries

The library microservice services two API calls:

- Provide a list of contents for a directory
- Fetch a file from the available files

The API calls are accepted over GraphQL and HTTP API end points. The format of the accepted queries are:

PROVIDE LIST OF CONTENTS FOR A DIRECTORY

To retrieve a list of files in a directory, use the following GraphQL query.

Replace `path` with the desired directory path.

send requests to: https://foo.com/lib

GraphQL Query **GraphQL Response** **HTTP Request** **HTTP Response**

```graphql
query {
  listDirectory(path: "user1") {
    repository {
      tree {
        blobs {
          edges {
            node {
              name
              type
            }
          }
        }
        trees {
          edges {
            node {
              name
              type
            }
          }
        }
      }
    }
  }
}
```


```json
{
  "data": {
    "listDirectory": {
      "repository": {
        "tree": {
          "blobs": {
            "edges": []
          },
          "trees": {
            "edges": [
              {
                "node": {
                  "name": "common",
                  "type": "tree"
                }
              },
              {
                "node": {
                  "name": "data",
                  "type": "tree"
                }
              },
              {
                "node": {
                  "name": "digital twins",
                  "type": "tree"
                }
              },
              {
                "node": {
                  "name": "functions",
                  "type": "tree"
                }
              },
              {
                "node": {
                  "name": "models",
                  "type": "tree"
                }
              },
              {
                "node": {
                  "name": "tools",
                  "type": "tree"
                }
              }
            ]
          }
        }
      }
    }
  }
}
```


```http
POST /lib HTTP/1.1
Host: foo.com
Content-Type: application/json
Content-Length: 388

{
  "query":"query {\n  listDirectory(path: \"user1\") {\n    repository {\n      tree {\n        blobs {\n          edges {\n            node {\n              name\n              type\n            }\n          }\n        }\n        trees {\n          edges {\n            node {\n              name\n              type\n            }\n          }\n        }\n      }\n    }\n  }\n}"
}
```

```http
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: close
Content-Length: 306
Content-Type: application/json; charset=utf-8
Date: Tue, 26 Sep 2023 20:26:49 GMT
X-Powered-By: Express
{"data":{"listDirectory":{"repository":{"tree":{"blobs":{"edges":[]},"trees":{"edges":[{"node":{"name":"data","type":"tree"}},{"node":{"name":"digital twins","type":"tree"}},{"node":{"name":"functions","type":"tree"}},{"node":{"name":"models","type":"tree"}},{"node":{"name":"tools","type":"tree"}}]}}}}}}
```

FETCH A FILE FROM THE AVAILABLE FILES

This query receives directory path and send the file contents to user in response.

To check this query, create a file `files/user2/data/welcome.txt` with content of `hello world`.

GraphQL Request **GraphQL Response** **HTTP Request** **HTTP Response**

```
query {
  readFile(path: "user2/data/sample.txt") {
    repository {
      blobs {
        nodes {
          name
          rawBlob
          rawTextBlob
        }
      }
    }
  }
}
```

```
{
  "data": {
    "readFile": {
      "repository": {
        "blobs": {
          "nodes": [
            {
              "name": "sample.txt",
              "rawBlob": "hello world",
              "rawTextBlob": "hello world"
            }
          ]
        }
      }
    }
  }
}
```

```
POST /lib HTTP/1.1
Host: foo.com
Content-Type: application/json
Content-Length: 217
{
  "query":"query {\n  readFile(path: \"user2/data/welcome.txt\") {\n    repository {\n      blobs {\n        nodes {\n          name\n          rawBlob\n          rawTextBlob\n        }\n      }\n    }\n  }\n}"
}
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: close
Content-Length: 134
Content-Type: application/json; charset=utf-8
Date: Wed, 27 Sep 2023 09:17:18 GMT
X-Powered-By: Express
{"data":{"readFile":{"repository":{"blobs":{"nodes":[{"name":"welcome.txt","rawBlob":"hello world","rawTextBlob":"hello world"}]}}}}}
```

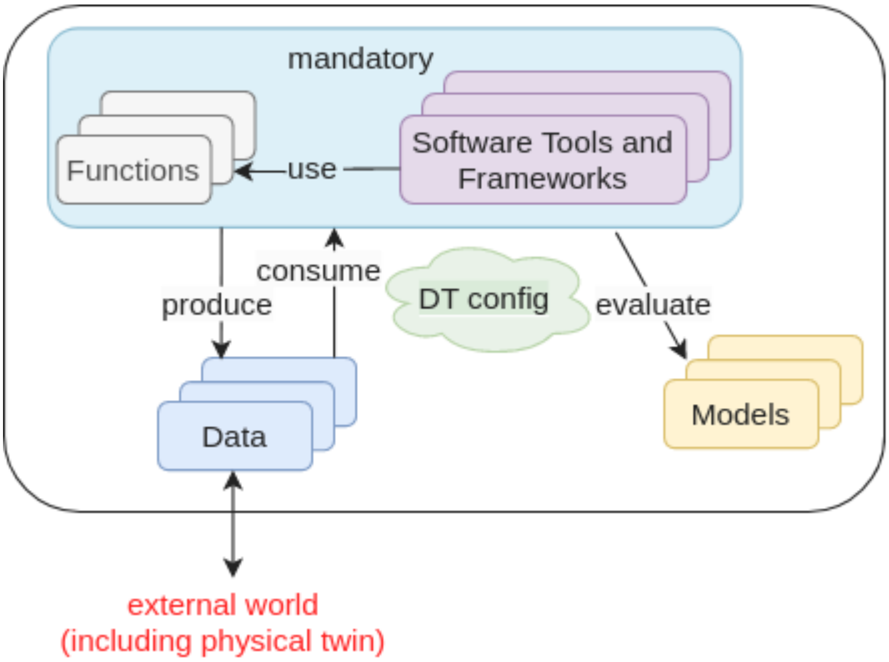
The *path* refers to the file path to look at: For example, *user1* looks at files of **user1**; *user1/functions* looks at contents of *functions/* directory.

3.5 Digital Twins

3.5.1 Create a Digital Twin

The first step in digital twin creation is to use the available assets in your workspace. If you have assets / files in your computer that need to be available in the DTaaS workspace, then please follow the instructions provided in library assets.

There are dependencies among the library assets. These dependencies are shown below.



A digital twin can only be created by linking the assets in a meaningful way. This relationship can be expressed using a mathematical equation:

where D denotes data, M denotes models, F denotes functions, T denotes tools, denotes DT configuration and is a symbolic notation for a digital twin itself. The expression denotes composition of DT from D,M,T and F assets. The indicates zero or one more instances of an asset and indicates one or more instances of an asset.

The DT configuration specifies the relevant assets to use, the potential parameters to be set for these assets. If a DT needs to use RabbitMQ, InfluxDB like services supported by the platform, the DT configuration needs to have access credentials for these services.

This kind of generic DT definition is based on the DT examples seen in the wild. You are at liberty to deviate from this definition of DT. The only requirement is the ability to run the DT from either commandline or desktop.

> **Tip**
>
> If you are stepping into the world of Digital Twins, you might not have distinct digital twin assets. You are likely to have one directory of everything in which you run your digital twin. In such a case we recommend that you upload this monolithic digital twin into **digital_twin/your_digital_twin_name** directory.

Example

The Examples repository contains a co-simulation setup for mass spring damper. This example illustrates the potential of using co-simulation for digital twins.

The file system contents for this example are:

```
workspace/
  data/
    mass-spring-damper
        input/
        output/

  digital_twins/
    mass-spring-damper/
      cosim.json
      time.json
      lifecycle/
        analyze
        clean
        evolve
        execute
        save
        terminate
      README.md

  functions/
  models/
    MassSpringDamper1.fmu
    MassSpringDamper2.fmu

  tools/
  common/
    data/
    functions/
    models/
    tools/
        maestro-2.3.0-jar-with-dependencies.jar
```

The `workspace/data/mass-spring-damper/` contains `input` and `output` data for the mass-spring-damper digital twin.

The two FMU models needed for this digital twin are in `models/` directory.

The co-simulation digital twin needs Maestro co-simulation orchestrator. Since this is a reusable asset for all the co-simulation based DTs, the tool has been placed in `common/tools/` directory.

The actual digital twin configuration is specified in `digital_twins/mass-spring-damper` directory. The co-simulation configuration is specified in two json files, namely `cosim.json` and `time.json`. A small explanation of digital twin for its users can be placed in `digital_twins/mass-spring-damper/README.md`.

The launch program for this digital twin is in `digital_twins/mass-spring-damper/lifecycle/execute`. This launch program runs the co-simulation digital twin. The co-simulation runs till completion and then ends. The programs in `digital_twins/mass-spring-damper/lifecycle` are responsible for lifecycle management of this digital twin. The lifecycle page provides more explanation on these programs.

Execution of a Digital Twin

A frequent question arises on the run time characteristics of a digital twin. The natural intuition is to say that a digital twin must operate as long as its physical twin is in operation. **If a digital twin runs for a finite time and then ends, can it be called a digital twin? The answer is a resounding YES**. The Industry 4.0 usecases seen among SMEs have digital twins that run for a finite time. These digital twins are often run at the discretion of the user.

You can run this digital twin by,

1. Go to Workbench tools page of the DTaaS website and open VNC Desktop. This opens a new tab in your browser
2. A page with VNC Desktop and a connect button comes up. Click on Connect. You are now connected to the Linux Desktop of your workspace.
3. Open a Terminal (black rectangular icon in the top left region of your tab) and type the following commands.
4. Download the example files by following the instructions given on examples overview.
5. Go to the digital twin directory and run

```
cd /workspace/examples/digital_twins/mass-spring-damper
lifecycle/execute
```

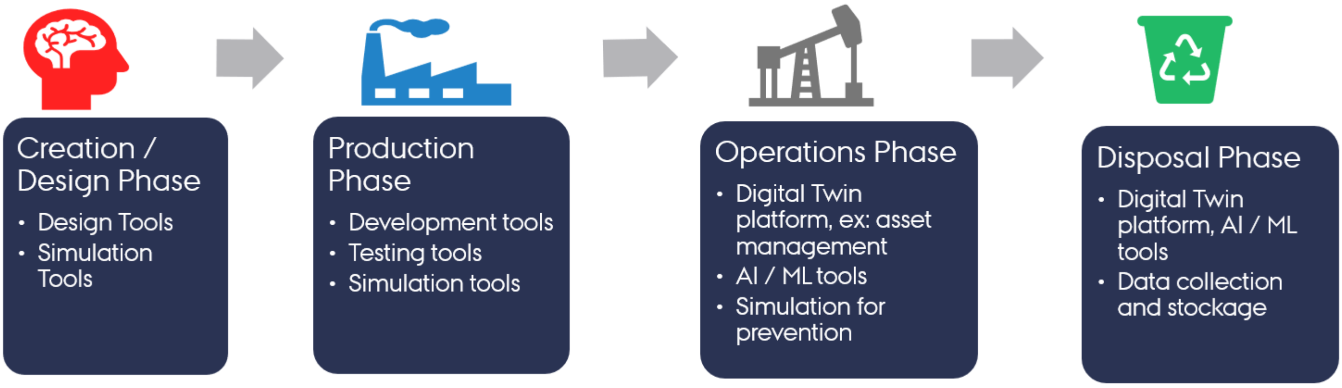
The last command executes the mass-spring-damper digital twin and stores the co-simulation output in `data/mass-spring-damper/output`.

3.5.2 ♻️ Digital Twin Lifecycle

The physical products in the real world have product lifecycle. A simplified four-stage product life is illustrated here.

A digital twin tracking the physical products (twins) need to track and evolve in conjunction with the corresponding physical twin.

The possible activities undertaken in each lifecycle phases are illustrated in the figure.



(Ref: Minerva, R, Lee, GM and Crespi, N (2020) Digital Twin in the IoT context: a survey on technical features, scenarios and architectural models. Proceedings of the IEEE, 108 (10). pp. 1785-1824. ISSN 0018-9219.)

Lifecycle Phases

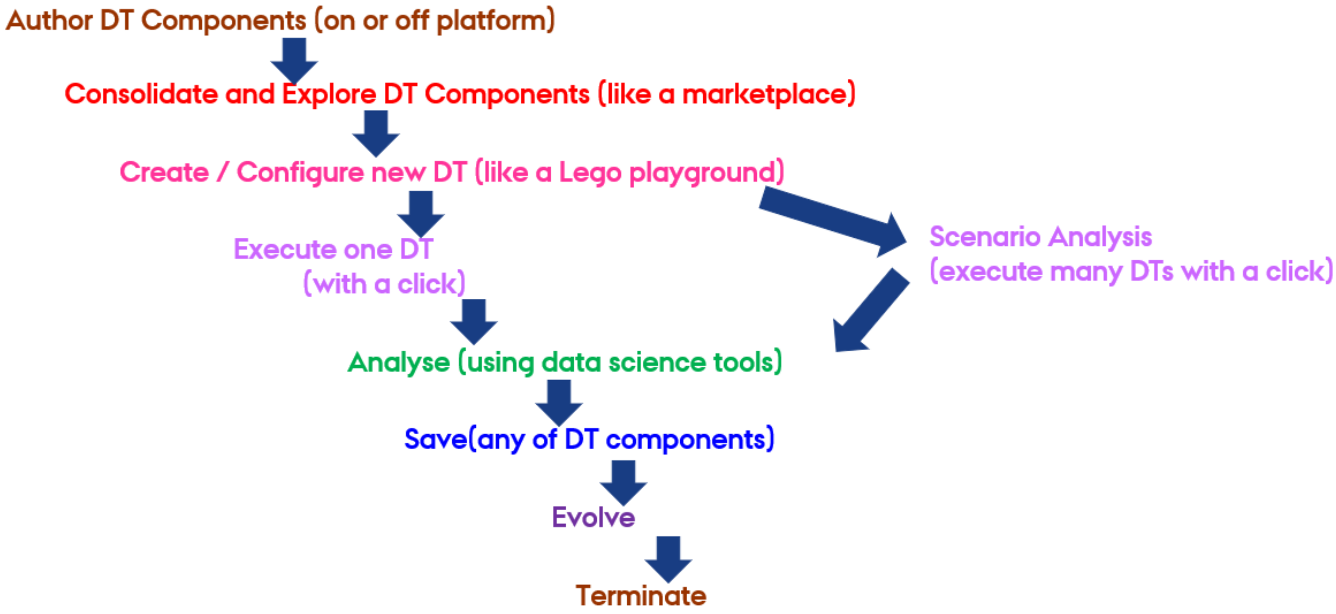
The four phase lifecycle has been extended to a lifecycle with eight phases. The new phase names and the typical activities undertaken in each phase are outlined in this section.

A DT lifecycle consists of **explore, create, execute, save, analyse, evolve** and **terminate** phases.

Phase	Main Activities
explore	selection of suitable assets based on the user needs and checking their compatibility for the purposes of creating a DT.
create	specification of DT configuration. If DT already exists, there is no creation phase at the time of reuse.
execute	automated / manual execution of a DT based on its configuration. The DT configuration must checked before starting the execution phase.
analyse	checking the outputs of a DT and making a decision. The outputs can be text files, or visual dashboards.
evolve	reconfigure DT primarily based on analysis.
save	involves saving the state of DT to enable future recovery.
terminate	stop the execution of DT.

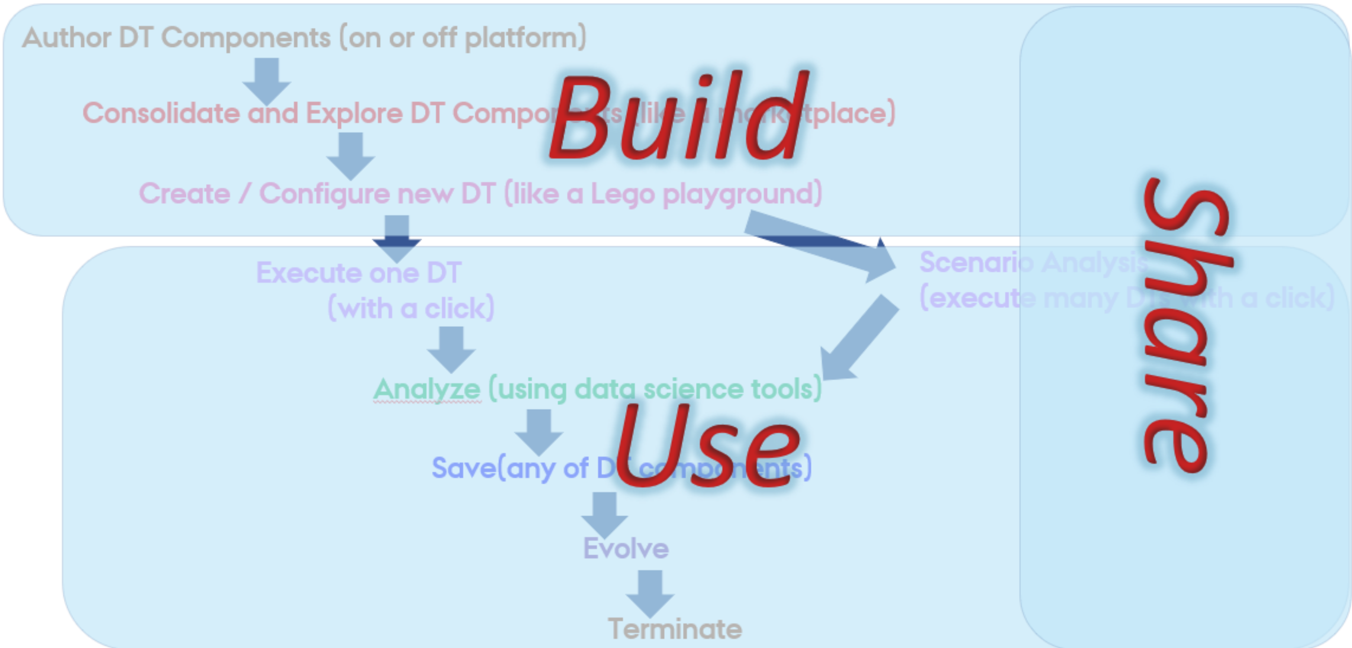
A digital twin faithfully tracking the physical twin lifecycle will have to support all the phases. It is also possible for digital twin engineers to add more phases to digital they are developing. Thus it is important for the DTaaS software platform needs to accommodate needs of different DTs.

A potential linear representation of the tasks undertaken in a digital twin lifecycle are shown here.



Again this is only a one possible pathway. Users are at liberty to alter the sequence of steps.

It is possible to map the lifecycle phases identified so far with the **Build-Use-Share** approach of the DTaaS software platform.



Even though not mandatory, having a matching coding structure makes it easy to for users to create and manage their DTs within the DTaaS. It is recommended to have the following structure:

```
workspace/
  digital_twins/
    digital-twin-1/
      lifecycle/
        analyze
        clean
        evolve
        execute
        save
        terminate
```

A dedicated program exists for each phase of DT lifecycle. Each program can be as simple as a script that launches other programs or sends messages to a live digital twin.

The recommended way to implement lifecycle phases within DTaaS is to create scripts. These scripts can be as simple as shell scripts.

Example Lifecycle Scripts

Here are the example programs / scripts to manage three phases in the lifecycle of **mass-spring-damper DT**.

```bash
#!/bin/bash
mkdir -p /workspace/data/mass-spring-damper/output
#cd ..
java -jar /workspace/common/tools/maestro-2.3.0-jar-with-dependencies.jar \
    import -output /workspace/data/mass-spring-damper/output \
    --dump-intermediate sg1 cosim.json time.json -i -vi FMI2 \
    output-dir>debug.log 2>&1
```

The execute phases uses the DT configuration, FMU models and Maestro tool to execute the digital twin. The script also stores the output of cosimulation in `/workspace/data/mass-spring-damper/output`.

It is possible for a DT not to support a specific lifecycle phase. This intention can be specified with an empty script and a helpful message if deemed necessary.

```bash
#!/bin/bash
printf "operation is not supported on this digital twin"
```

The lifecycle programs can call other programs in the code base. In the case of `lifecycle/terminate` program, it is calling another script to do the necessary job.

```bash
#!/bin/bash
lifecycle/clean
```

3.6 Examples

3.6.1 DTaaS Examples

There are some example digital twins created for the DTaaS software. Use these examples and follow the steps given in the **Examples** section to experience features of the DTaaS software platform and understand best practices for managing digital twins within the platform.

You can also see a short video on the use of DTaaS for creating a digital shadow. This video is recorded using DTaaS v0.2.0.

Copy Examples

The first step is to copy all the example code into your user workspace within the DTaaS. Use the given shell script to copy all the examples into `/workspace/examples` directory.

```
wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS-examples/main/getExamples.sh
bash getExamples.sh
```

Example List

The digital twins provided in examples vary in their complexity. It is best to use the examples in the following order.

1. Mass Spring Damper
2. Water Tank Fault Injection
3. Water Tank Model Swap
4. Desktop Robotti and RabbitMQ
5. Water Treatment Plant and OPC-UA
6. Three Water Tanks with DT Manager Framework
7. Flex Cell Digital Twin with Two Industrial Robots

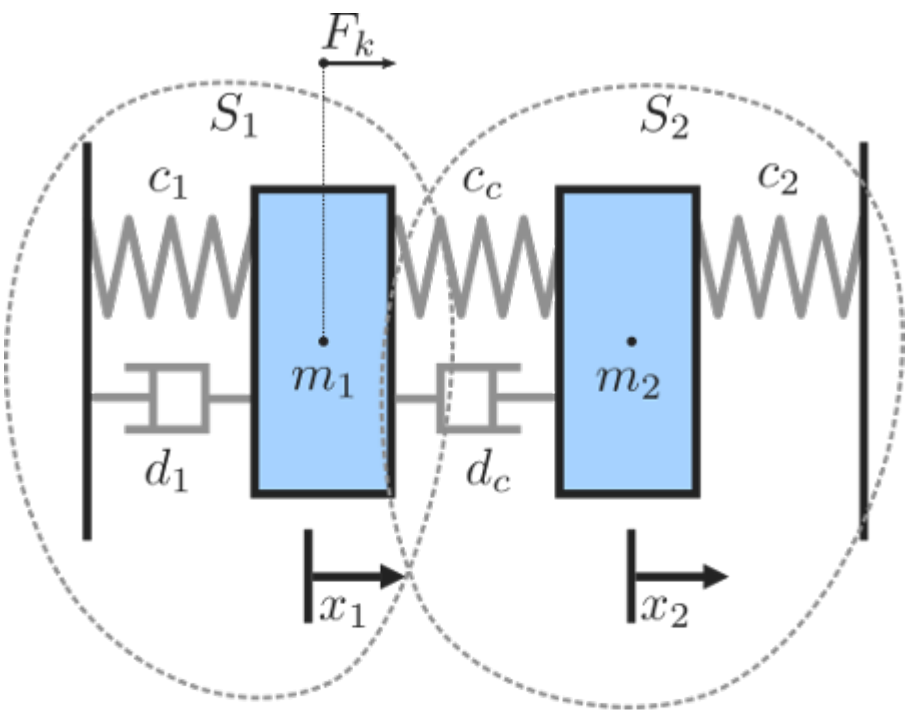
DTaaS examples

3.6.2 Mass Spring Damper

Overview

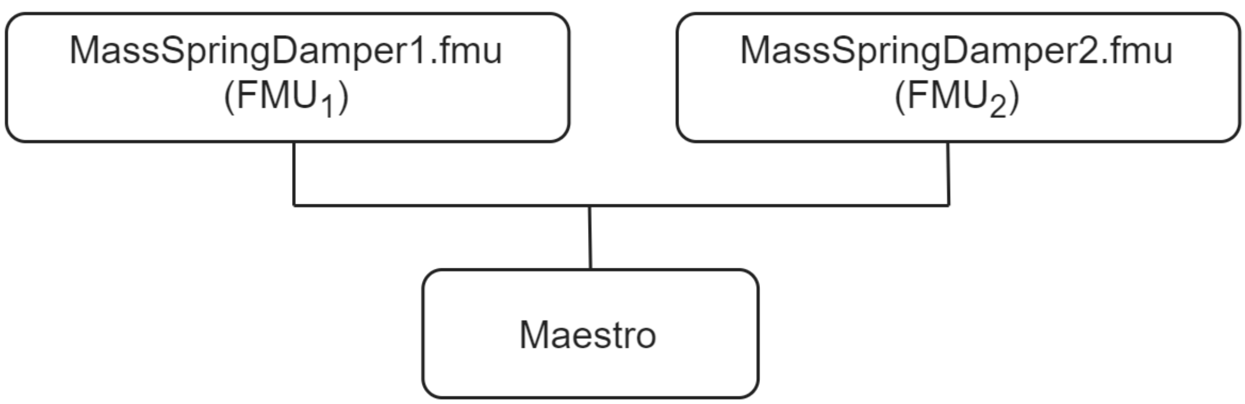
The mass spring damper digital twin (DT) comprises two mass spring dampers and demonstrates how a co-simulation based DT can be used within DTaaS.

Example Diagram



Example Structure

There are two simulators included in the study, each representing a mass spring damper system. The first simulator calculates the mass displacement and speed of for a given force acting on mass . The second simulator calculates force given a displacement and speed of mass . By coupling these simulators, the evolution of the position of the two masses is computed.



Digital Twin Configuration

This example uses two models and one tool. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	MassSpringDamper1.fmu	Private	Yes
	MassSpringDamper2.fmu	Private	Yes
Tool	maestro-2.3.0-jar-with-dependencies.jar	Common	Yes

The `co-sim.json` and `time.json` are two DT configuration files used for executing the digital twin. You can change these two files to customize the DT to your needs.

Lifecycle Phases

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool
Execute	Produces and stores output in data/mass-spring-damper/output directory
Clean	Clears run logs and outputs

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital_twins/mass-spring-damper
```

If required, change the execute permission of lifecycle scripts you need to execute, for example:

```
chmod +x lifecycle/create
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 17 in the workspace.

```
lifecycle/create
```

EXECUTE

Run the the Digital Twin. Since this is a co-simulation based digital twin, the Maestro co-simulation tool executes co-simulation using the two FMU models.

```
lifecycle/execute
```

Examine the results

The results can be found in the */workspace/examples/data/mass-spring-damper/output directory*.

You can also view run logs in the */workspace/examples/digital_twins/mass-spring-damper*.

TERMINATE PHASE

Terminate to clean up the debug files and co-simulation output files.

```
lifecycle/terminate
```

References

More information about co-simulation techniques and mass spring damper case study are available in:

```
Gomes, Cláudio, et al. "Co-simulation: State of the art."
arXiv preprint arXiv:1702.00686 (2017).
```

The source code for the models used in this DT are available in mass spring damper github repository.

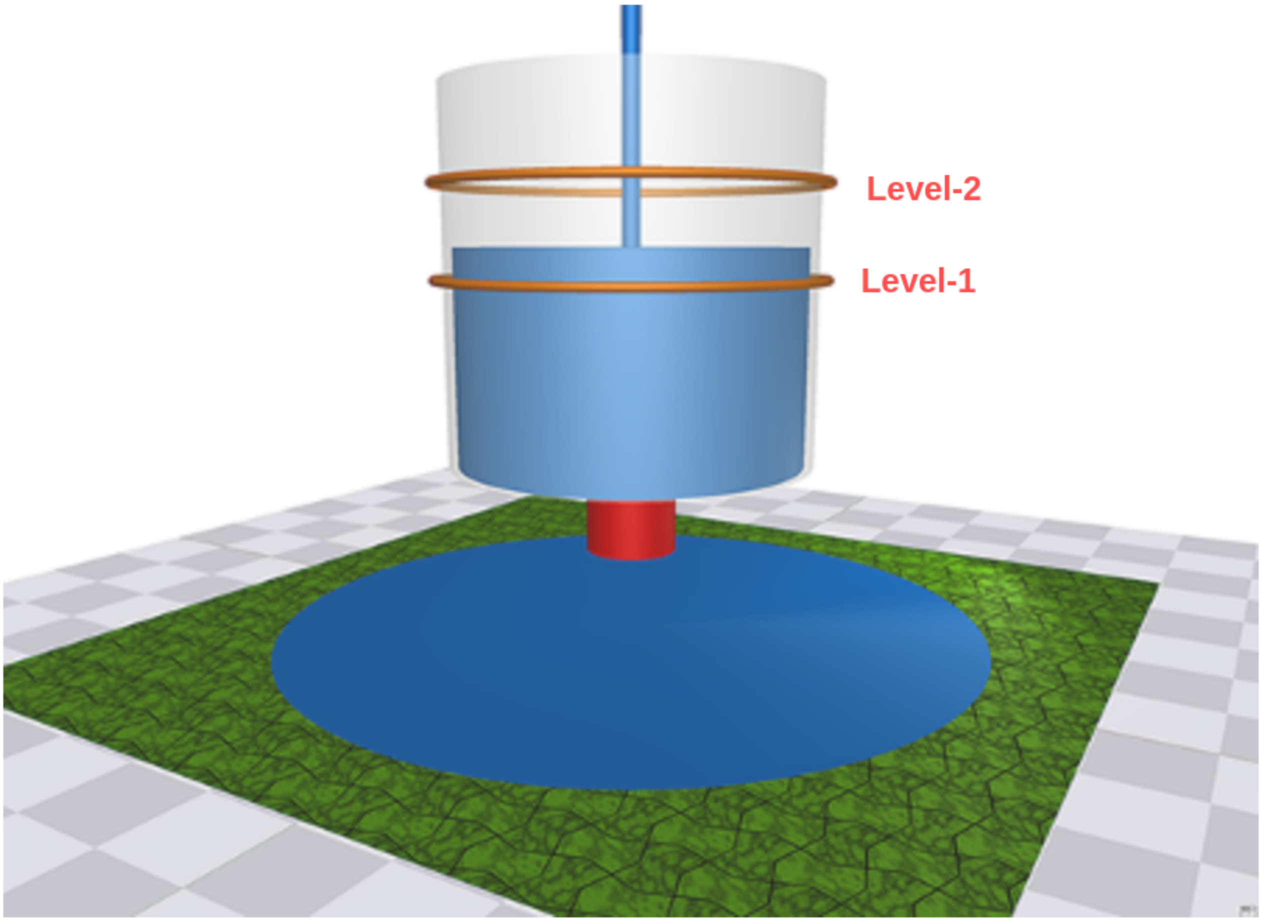
3.6.3 Water Tank Fault Injection

Overview

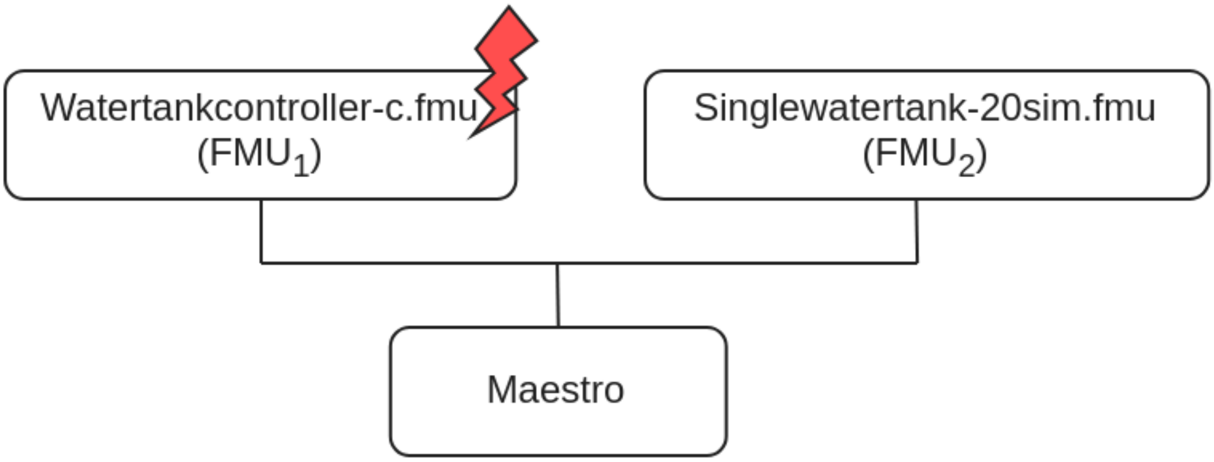
This example shows a fault injection (FI) enabled digital twin (DT). A live DT is subjected to simulated faults received from the environment. The simulated faults is specified as part of DT configuration and can be changed for new instances of DTs.

In this co-simulation based DT, a watertank case-study is used; co-simulation consists of a tank and controller. The goal of which is to keep the level of water in the tank between `Level-1` and `Level-2`. The faults are injected into output of the water tank controller (**Watertankcontroller-c.fmu**) from 12 to 20 time units, such that the tank output is closed for a period of time, leading to the water level increasing in the tank beyond the desired level (`Level-2`).

Example Diagram



Example Structure



Digital Twin Configuration

This example uses two models and one tool. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	watertankcontroller-c.fmu	Private	Yes
	singlewatertank-20sim.fmu	Private	Yes
Tool	maestro-2.3.0-jar-with-dependencies.jar	Common	Yes

The `multimodelFI.json` and `simulation-config.json` are two DT configuration files used for executing the digital twin. You can change these two files to customize the DT to your needs.

The faults are defined in **wt_fault.xml**.

Lifecycle Phases

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool
Execute	Produces and stores output in data/water_tank_FI/output directory
Clean	Clears run logs and outputs

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital_twins/water_tank_FI
```

If required, change the execute permission of lifecycle scripts you need to execute, for example:

```
chmod +x lifecycle/create
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 17 and pip dependencies. The pandas and matplotlib are the pip dependencies installated.

```
lifecycle/create
```

EXECUTE

Run the co-simulation. Generates the co-simulation output.csv file at `/workspace/examples/data/water_tank_FI/output`.

```
lifecycle/execute
```

ANALYZE PHASE

Process the output of co-simulation to produce a plot at: `/workspace/examples/data/water_tank_FI/output/plots/`.

```
lifecycle/analyze
```

Examine the results

The results can be found in the */workspace/examples/data/water_tank_FI/output directory*.

You can also view run logs in the */workspace/examples/digital_twins/water_tank_FI*.

TERMINATE PHASE

Clean up the temporary files and delete output plot

```
lifecycle/terminate
```

References

More details on this case-study can be found in the paper:

```
M. Frasheri, C. Thule, H. D. Macedo, K. Lausdahl, P. G. Larsen and
L. Esterle, "Fault Injecting Co-simulations for Safety,"
2021 5th International Conference on System Reliability and Safety (ICSRS),
Palermo, Italy, 2021.
```

The fault-injection plugin is an extension to the Maestro co-orchestration engine that enables injecting inputs and outputs of FMUs in an FMI-based co-simulation with tampered values. More details on the plugin can be found in fault injection git repository. The source code for this example is also in the same github repository in a example directory.

3.6.4 Water Tank Model Swap

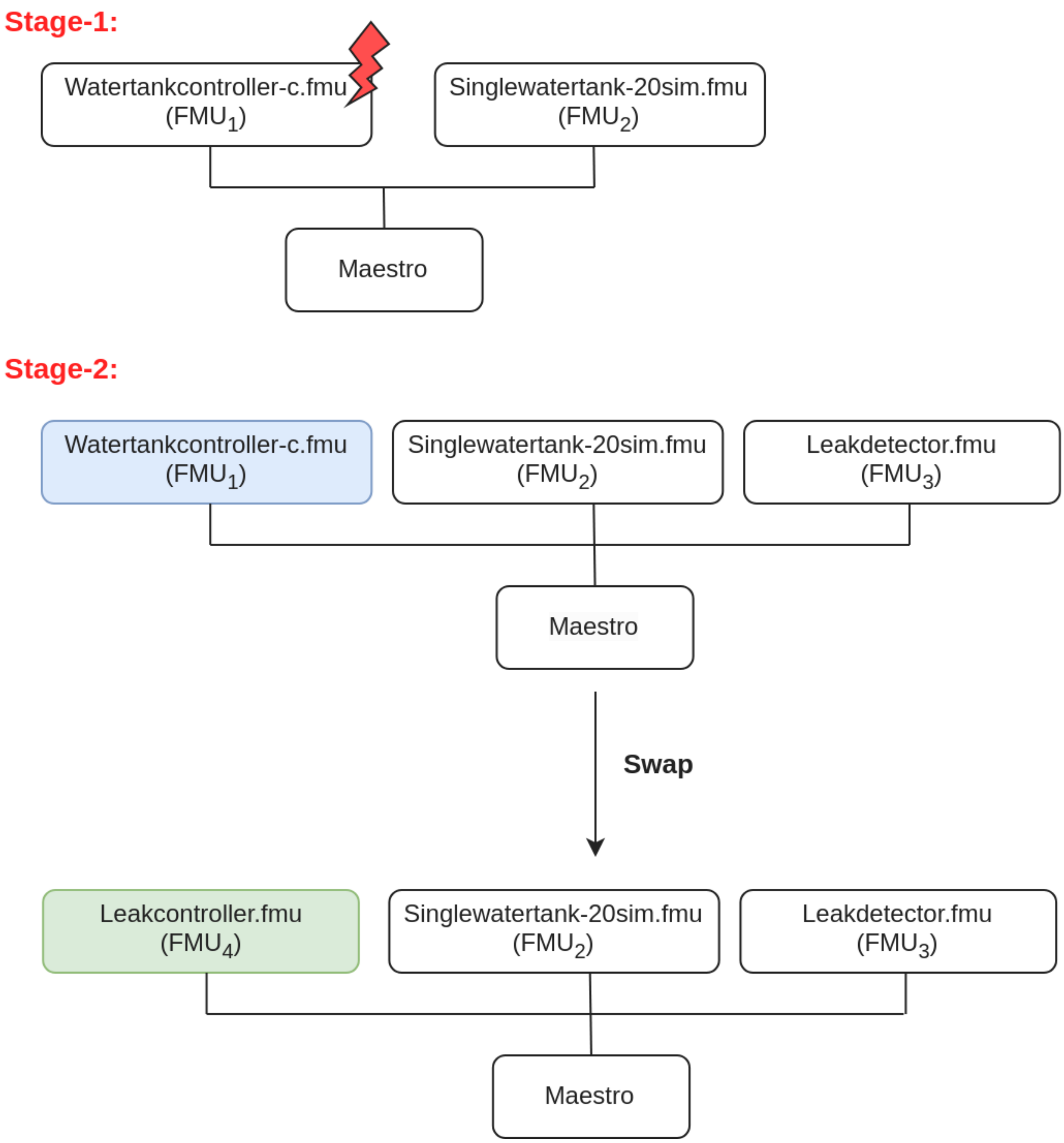
Overview

This example shows multi-stage execution and dynamic reconfiguration of a digital twin (DT). Two features of DTs are demonstrated here:

- Fault injection into live DT
- Dynamic auto-reconfiguration of live DT

The co-simulation methodology is used to construct this DT.

Example Structure



Configuration of assets

This example uses four models and one tool. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	Watertankcontroller-c.fmu	Private	Yes
	Singlewatertank-20sim.fmu	Private	Yes
	Leak_detector.fmu	Private	No
	Leak_controller.fmu	Private	No
Tool	maestro-2.3.0-jar-with-dependencies.jar	Common	Yes

This DT has many configuration files. The DT is executed in two stages. There exist separate DT configuration files for each stage. The following table shows the configuration files and their purpose.

Configuration file name	Execution Stage	Purpose
mm1. json	stage-1	DT configuration
wt_fault.xml, FaultInject.mabl	stage-1	faults injected into DT during stage-1
mm2.json	stage-2	DT configuration
simulation-config.json	Both stages	Configuration for specifying DT execution time and output logs

Lifecycle Phases

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool
Execute	Produces and stores output in data/water_tank_swap/output directory
Analyze	Process the co-simulation output and produce plots
Clean	Clears run logs, outputs and plots

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital_twins/water_tank_swap
```

If required, change the permission of files you need to execute, for example:

```
chmod +x lifecycle/create
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 17 and pip dependencies. The matplotlib pip package is also installated.

```
lifecycle/create
```

EXECUTE

This DT has two-stage execution. In the first-stage, a co-simulation is executed. The Watertankcontroller-c.fmu and Singlewatertank-20sim.fmu models are used to execute the DT. During this stage, faults are injected into one of the models (Watertankcontroller-c.fmu) and the system performance is checked.

In the second-stage, another co-simulation is run in which three FMUs are used. The FMUs used are: watertankcontroller, singlewatertank-20sim, and leak_detector. There is an in-built monitor in the Maestro tool. This monitor is enabled during the stage and a swap condition is set at the beginning of the second-stage. When the swap condition is satisfied, the Maestro swaps out Watertankcontroller-c.fmu model and swaps in Leakcontroller.fmu model. This swapping of FMU models demonstrates the dynamic reconfiguration of a DT.

The end of execution phase generates the co-simulation output.csv file at `/workspace/examples/data/water_tank_swap/output`.

```
lifecycle/execute
```

ANALYZE PHASE

Process the output of co-simulation to produce a plot at: `/workspace/examples/data/water_tank_FI/output/plots/`.

```
lifecycle/analyze
```

Examine the results

The results can be found in the *workspace/examples/data/water_tank_swap/output directory*.

You can also view run logs in the *workspace/examples/digital_twins/water_tank_swap*.

TERMINATE PHASE

Clean up the temporary files and delete output plot

```
lifecycle/terminate
```

References

The complete source of this example is available on model swap github repository.

The runtime model (FMU) swap mechanism demonstrated by the experiment is detailed in the paper:

```
Ejersbo, Henrik, et al. "fmiSwap: Run-time Swapping of Models for
Co-simulation and Digital Twins." arXiv preprint arXiv:2304.07328 (2023).
```

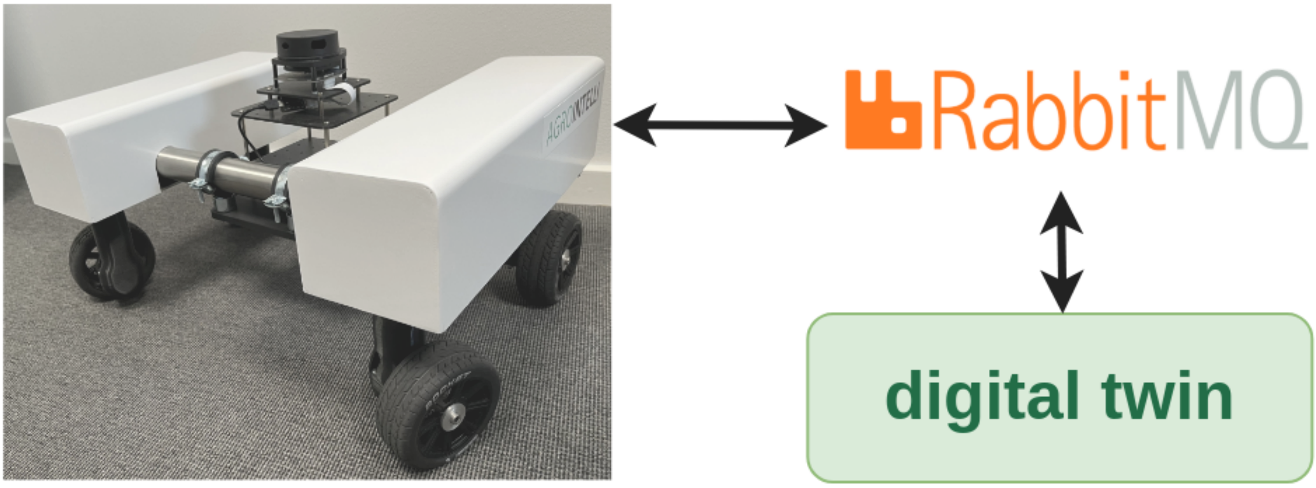
The runtime reconfiguration of co-simulation by modifying the Functional Mockup Units (FMUs) used is further detailed in the paper:

```
Ejersbo, Henrik, et al. "Dynamic Runtime Integration of
New Models in Digital Twins." 2023 IEEE/ACM 18th Symposium on
Software Engineering for Adaptive and Self-Managing Systems
(SEAMS). IEEE, 2023.
```

3.6.5 Desktop Robotti with RabbitMQ

Overview

This example demonstrates bidirectional communication between a mock physical twin and a digital twin of a mobile robot (Desktop Robotti). The communication is enabled by RabbitMQ Broker.

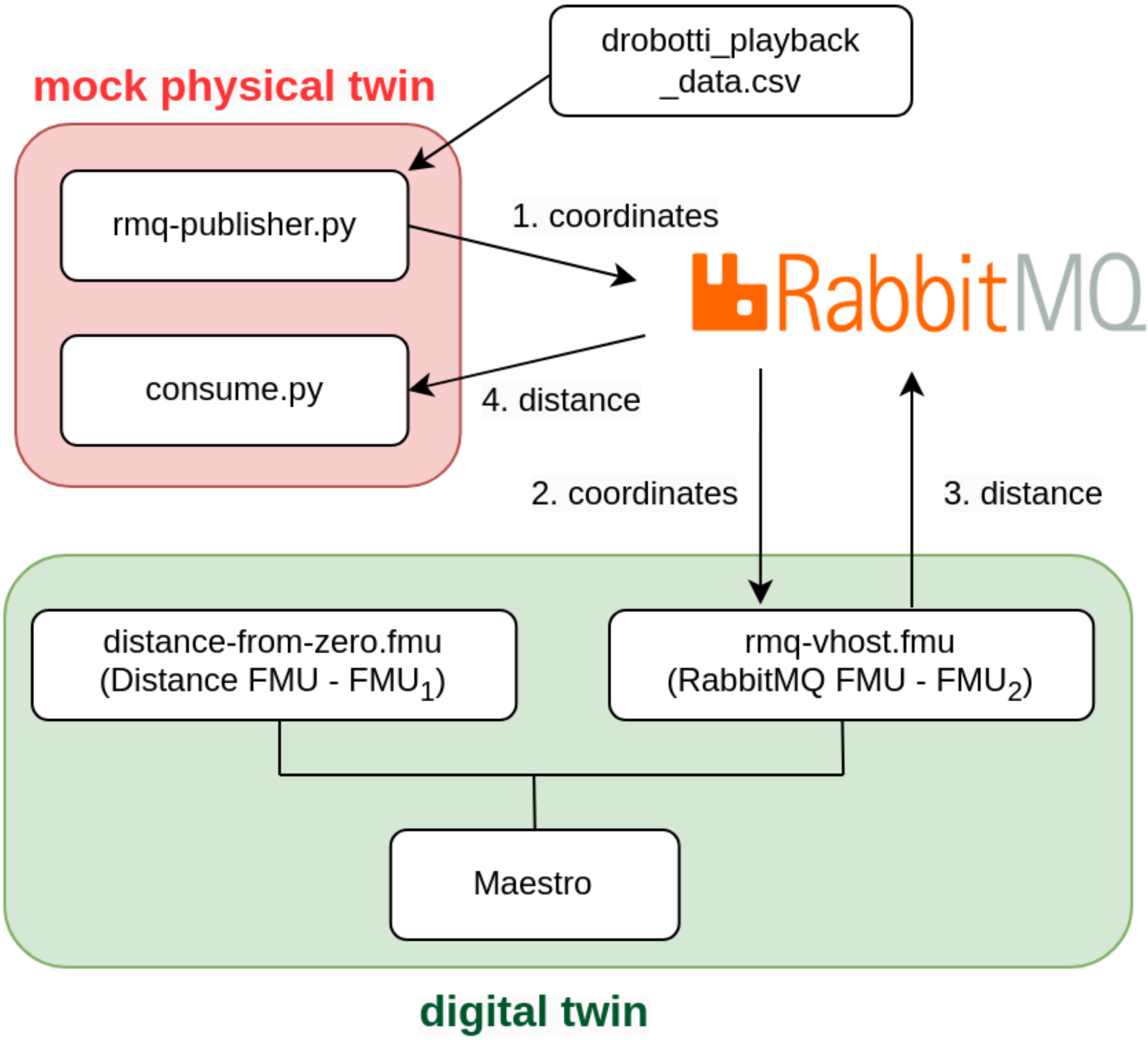


Example Structure

The mock physical twin of mobile robot is created using two python scripts

1. data/drobotti_rmqfmu/rmq-publisher.py
2. data/drobotti_rmqfmu/consume.py

The mock physical twin sends its physical location in *(x,y)* coordinates and expects a cartesian distance calculated from digital twin.



The *rmq-publisher.py* reads the recorded *(x,y)* physical coordinates of mobile robot. The recorded values are stored in a data file. These *(x,y)* values are published to RabbitMQ Broker. The published *(x,y)* values are consumed by the digital twin.

The *consume.py* subscribes to RabbitMQ Broker and waits for the calculated distance value from the digital twin.

The digital twin consists of a FMI-based co-simulation, where Maestro is used as co-orchestration engine. In this case, the co-simulation is created by using two FMUs - RMQ FMU (rabbitmq-vhost.fmu) and distance FMU (distance-from-zero.fmu). The RMQ FMU receives the *(x,y)* coordinates from *rmq-publisher.py* and sends calculated distance value to *consume.py*. The RMQ FMU uses RabbitMQ broker for communication with the mock mobile robot, i.e., *rmq-publisher.py* and *consume.py*. The distance FMU is responsible for calculating the distance between *(0,0)* and *(x,y)*. The RMQ FMU and distance FMU exchange values during co-simulation.

Digital Twin Configuration

This example uses two models, one tool, one data, and two scripts to create mock physical twin. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	distance-from-zero.fmu	Private	No
	rmq-vhost.fmu	Private	Yes
Tool	maestro-2.3.0-jar-with-dependencies.jar	Common	Yes
Data	drobotti_playback_data.csv	private	No
Mock PT	rmq-publisher.py	Private	No
	consume.py	Private	No

This DT has many configuration files. The `coe.json` and `multimodel.json` are two DT configuration files used for executing the digital twin. You can change these two files to customize the DT to your needs.

The RabbitMQ access credentials need to be provided in `multimodel.json`. The `rabbitMQ-credentials.json` provides RabbitMQ access credentials for mock PT python scripts. Please add your credentials in both these files.

Lifecycle Phases

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool and pip packages for python scripts
Execute	Runs both DT and mock PT
Clean	Clears run logs and outputs

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital_twins/drobotti_rmqfmu
```

If required, change the execute permission of lifecycle scripts you need to execute, for example:

```
chmod +x lifecycle/create
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 17 in the workspace. Also install the required python pip packages for *rmq-publisher.py* and *consume.py* scripts.

```
lifecycle/create
```

EXECUTE

Run the python scripts to start mock physical twin. Also run the the Digital Twin. Since this is a co-simulation based digital twin, the Maestro co-simulation tool executes co-simulation using the two FMU models.

```
lifecycle/execute
```

Examine the results

The results can be found in the */workspace/examples/digital_twins/drobotti_rmqfmu directory*.

Executing the DT will generate and launch a co-simulation (RMQFMU and distance FMU), and two python scripts. One to publish data that is read from a file. And one to consume what is sent by the distance FMU.

In this examples the DT will run for 10 seconds, with a stepsize of 100ms. Thereafter it is possible to examine the logs produce in `/workspace/examples/digital_twins/drobotti_rmqfmu/target`. The outputs for each FMU, xpos and ypos for the RMQFMU, and the distance for the distance FMU are recorded in the `outputs.csv` file. Other logs can be examined for each FMU and the publisher scripts. Note that, the RMQFMU only sends data, if the current input is different to the previous one.

TERMINATE PHASE

Terminate to clean up the debug files and co-simulation output files.

```
lifecycle/terminate
```

References

The RabbitMQ FMU github repository contains complete documentation and source code of the rmq-vhost.fmu.

More information about the case study is available in:

```
Frasheri, Mirgita, et al. "Addressing time discrepancy between digital
and physical twins." Robotics and Autonomous Systems 161 (2023): 104347.
```

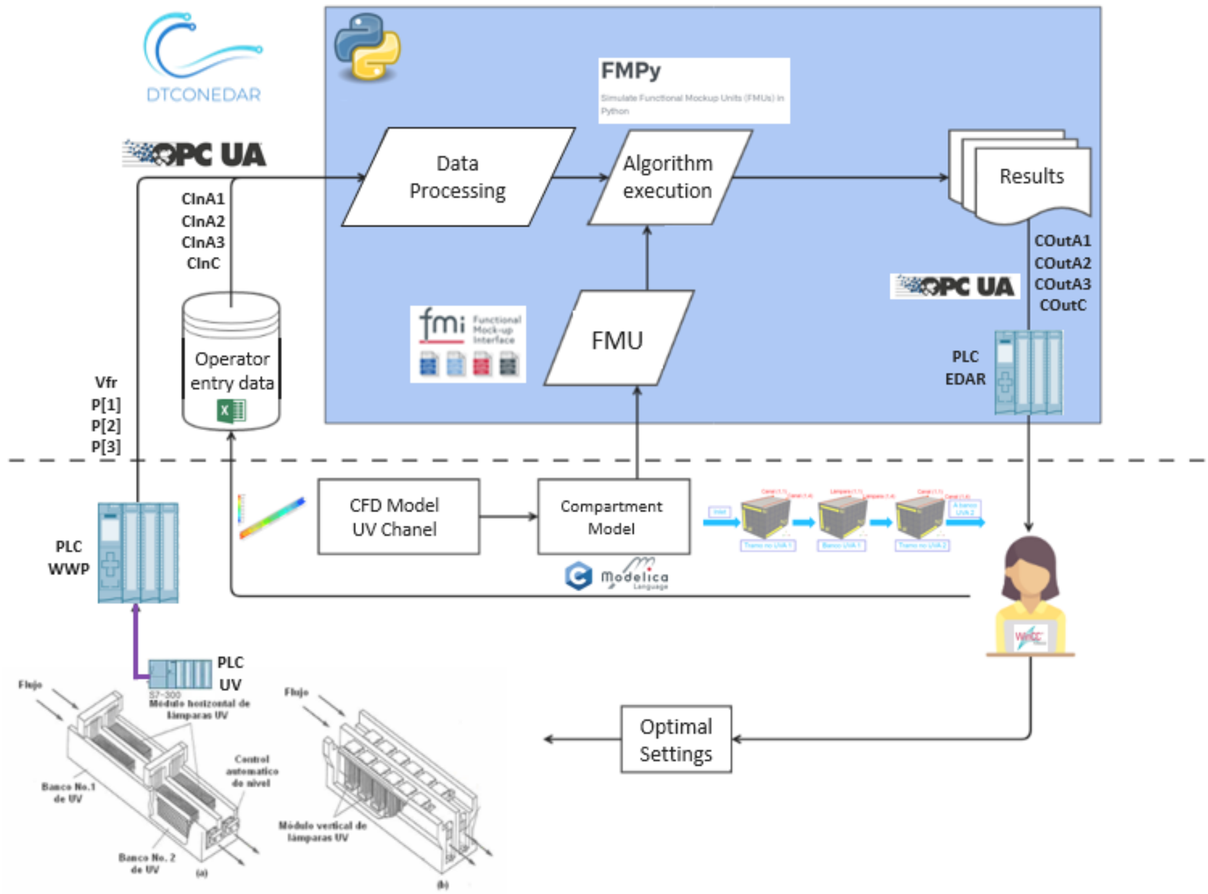
3.6.6 Waste Water Plant with OPC-UA

Introduction

Waste water treatment (WWT) plants must comply with substance and species concentration limits established by regulation in order to ensure the good quality of the water. This is usually done taking periodic samples that are analyzed in the laboratory. This means that plant operators do not have continuous information for making decisions and, therefore, operation setpoints are set to higher values than needed to guarantee water quality. Some of the processes involved in WWT plants consume a lot of power, thus adjusting setpoints could significantly reduce energy consumption.

Physical Twin Overview

This example demonstrates the communication between a physical ultraviolet (UV) disinfection process (the tertiary treatment of a WWT plant) and its digital twin, which is based on Computational Fluid Dynamics (CFD) and compartment models. The aim of this digital twin is to develop "virtual sensors" that provide continuous information that facilitates the decision making process for the plant operator.



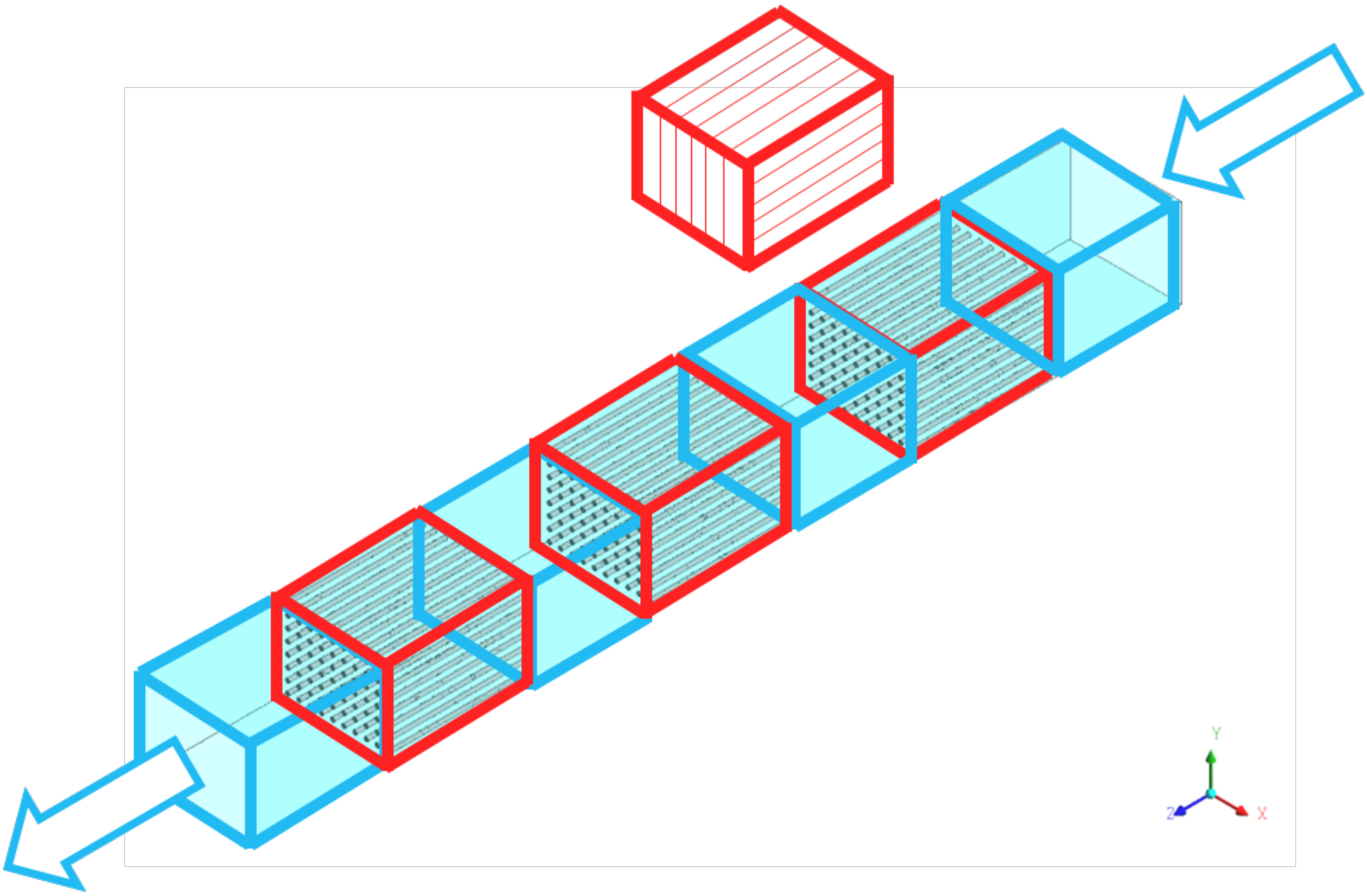
The physical twin of the waste water plant is composed of an ultraviolet channel controlled by a PLC that controls the power of the UV lamps needed to kill all the pathogens of the flow. The channel has 3 groups of UV lamps, therefore the real channel (and is mathematical model) is subdivided into 7 zones: 4 correspond to zones without UV lamps (2 for the entrance and exit of the channel + 2 zones between UV lamps) and the 3 reamaining for the UV lamps.

The dose to be applied (related with the power) changes according to the residence time (computed from the measure of the volume flow) and the UV intensity (measured by the intensity sensor).

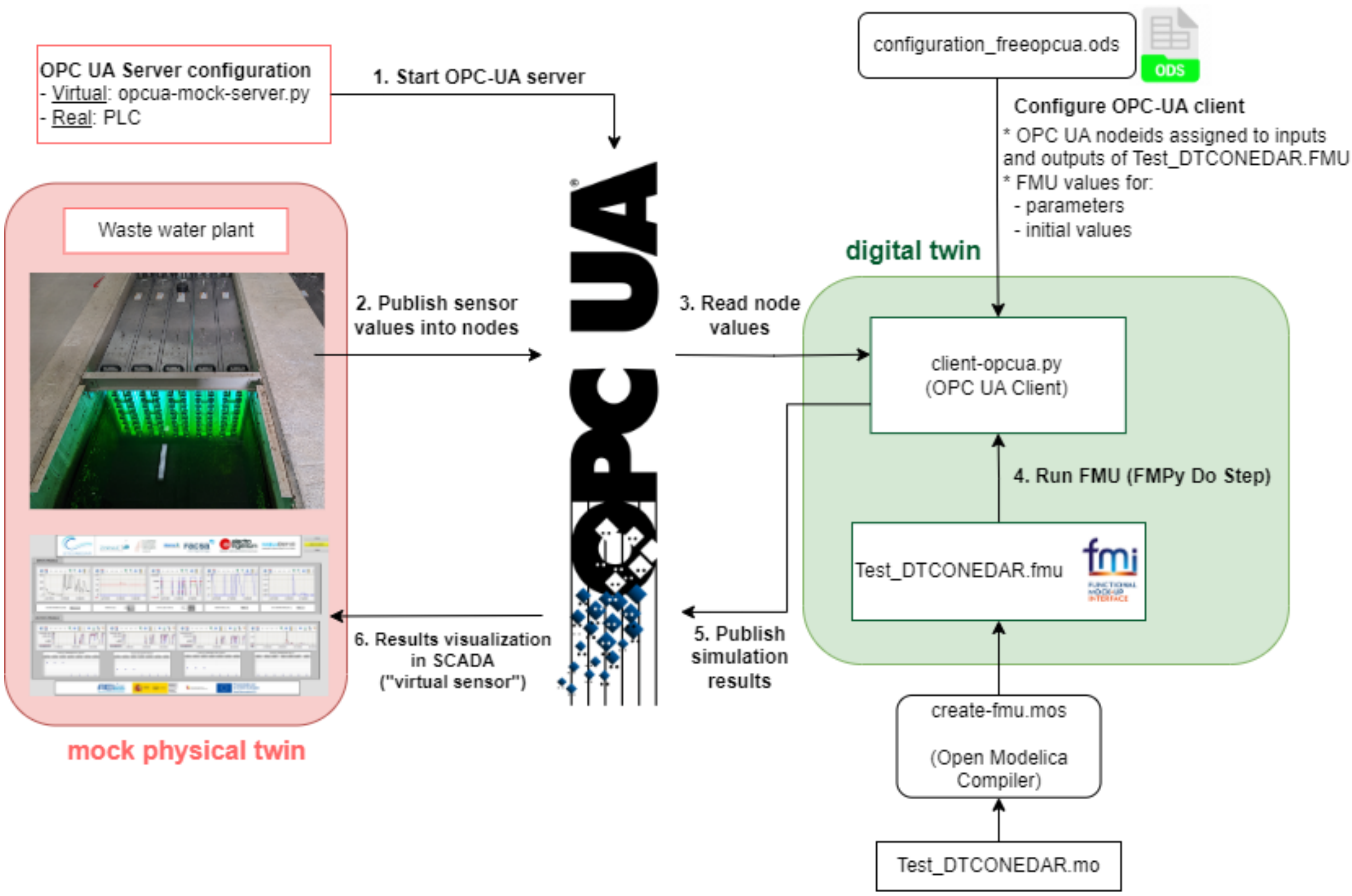
The information of the volumetric flow and power (in the three parts of the channel) is transmitted to the PLC of the plant. Furthermore, the PLC is working as OPC UA Server to send and receive data to and from an OPC UA Client. Additionally, some sizing parameters and initial values are read from a spreadsheet filled in by the plant operator. In this case, the spreadsheet is an

Open Office file (.ods) due to the software installed in the SCADA PC. Some of the variables like initial concentration of disinfectant and pathogens are included, among others. Some values defined by the plant operator correspond to input signals that are not currently being measured, but are expected to be measured in the future.

Digital Twin Overview



The digital twin is a reduced model (developed in C) that solves physical conservation laws (mass, energy and momentum), but simplifies details (geometry, mainly) to ensure real-time calculations and accurate results. The results are compared to the ones obtained by the CFD. C solver developed is used by the OpenModelica model. OpenModelica converts it into the FMI standard, to be integrated in the OPC UA Client (*client-opcua.py*).



Digital Twin Configuration

Asset Type	Name of Asset	Visibility	Reuse in Other Examples
Model	Test_DTCONEDAR.mo	private	No
Data	configuration_freopcua.ods	private	No
	model_description.csv (generated by client-asyncua.py)	private	No
	Mock OPC UA Server: opcua-mock-server.py	private	No
Tool	**OPC UA Client**: client-opcua.py	private	No
	FMU builder: create-fmu.mos	private	No

In this example, a dummy model representation of the plant is used, instead of the real model. The simplified model (with not the real equations) is developed in **Open Modelica (Test_DTCONEDAR.mo)**. The FMU is generated from the Open Modelica interface to obtain the needed binaries to run the FMU. It is possible to run an FMU previously generated, however, to ensure that we are using the right binaries it is recommended to install Open Modelica Compiler and run `script.mos` to build the FMU from the Modelica file `Test_DTCONEDAR.mo`.

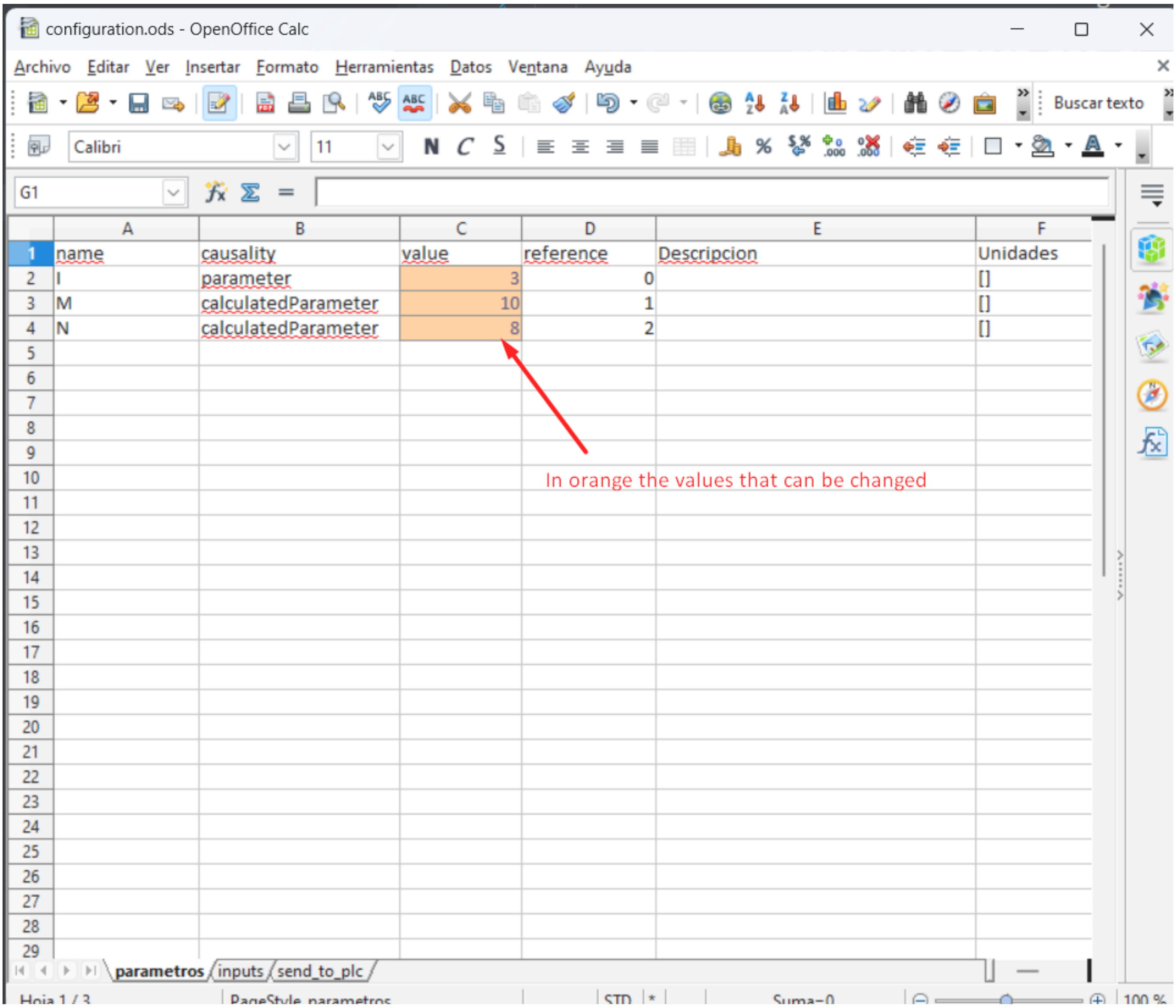
The FMU model description file (`modelDescription.xml` file inside `Test_DTCONEDAR.fmu`) has the information of the value references configuration_freeopcua.ods has the information of the OPC-UA node IDs And both have in common the variable name

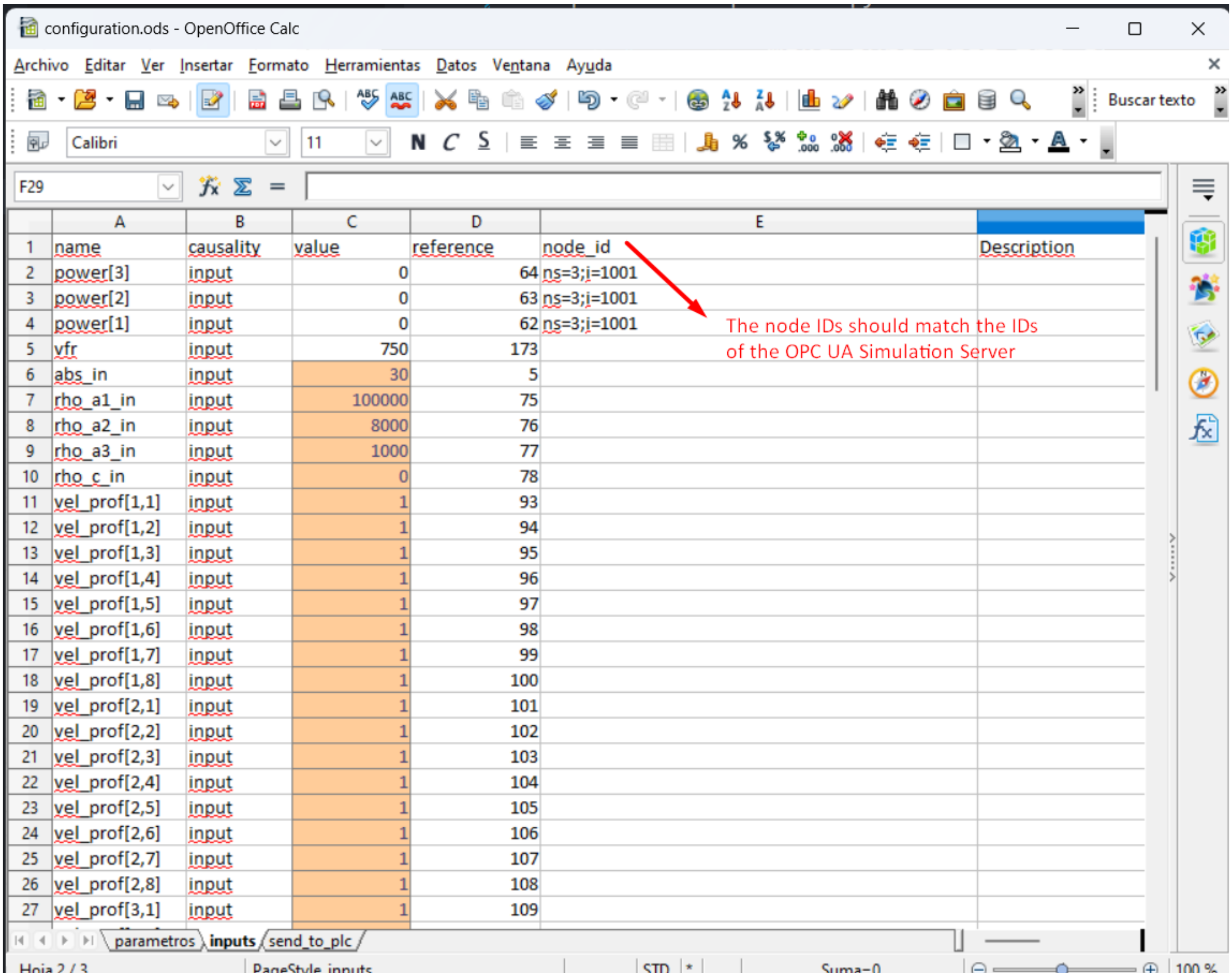
The client python script (`client-opcua.py`) does the following actions:

- Reads the variable names and the variable value references from the model description file of the Test_DTCONEDAR.FMU.
- Reads *configuration_freeopcua.ods* to obtain opcua node IDs and assigns those node IDs to the variables read from the FMU
- Read *configuration_freeopcua.ods* to fix initial values, parameters and some inputs (those inputs that are not being measured, a reasonable value is assumed).
- Read values from PLC using a client OPC.
- Execute the algorithm with the FMPy library using the .fmu created from the compartment model (based on CFD)
- Obtain results.
- Send by OPC UA protocol the result values to the PLC, to visualize them in the SCADA and with the aim to improve the decision-making process of the plant operator.

INPUT DATA VARIABLES

The *configuration_freeopcua.ods* date file is used for customizing the initial input data values used by the server.





name	causality	value	reference	node_id	Description
power[3]	input	0	64	ns=3;i=1001	
power[2]	input	0	63	ns=3;i=1001	
power[1]	input	0	62	ns=3;i=1001	
vfr	input	750	173		
abs_in	input	30	5		
rho_a1_in	input	100000	75		
rho_a2_in	input	8000	76		
rho_a3_in	input	1000	77		
rho_c_in	input	0	78		
vel_prof[1,1]	input	1	93		
vel_prof[1,2]	input	1	94		
vel_prof[1,3]	input	1	95		
vel_prof[1,4]	input	1	96		
vel_prof[1,5]	input	1	97		
vel_prof[1,6]	input	1	98		
vel_prof[1,7]	input	1	99		
vel_prof[1,8]	input	1	100		
vel_prof[2,1]	input	1	101		
vel_prof[2,2]	input	1	102		
vel_prof[2,3]	input	1	103		
vel_prof[2,4]	input	1	104		
vel_prof[2,5]	input	1	105		
vel_prof[2,6]	input	1	106		
vel_prof[2,7]	input	1	107		
vel_prof[2,8]	input	1	108		
vel_prof[3,1]	input	1	109		

DT CONFIG

The *config.json* specifies the configuration parameters for the OPC UA client.

config.json

```json
{
    "url"             : "opc.tcp://0.0.0.0:4840",
    "config_ods"      : "configuration_freeopcua.ods",
    "fmu_filename"    : "Test_DTCONEDAR_linux.fmu",
    "stop_time"       : 10.0,
    "step_size"       : 0.5,
    "record"          : false,
    "record_interval"   : 5.0,
    "record_variables"  : [],
    "enable_send"       : true
}
```

Optional parameters can be modified:

- stop_time
- step_size
- record = True, if we want to save the results of the simulation
- record_interval. Sometimes the simulation step_size is small and a the size of the results file can be too big. For instance, if the simulation step_size is 0.01 seconds, we can increase the record_interval so as to reduce the result file size.
- record_variables: we can specify the list of variables that we want to record.
- enable_send = True, if we want to send results to the OPC UA Server.

Lifecycle Phases

The lifecycles that are covered include:

Lifecycle Phase	Completed Tasks
Install	Installs Open Modelica, Python 3.10 and the required pip dependencies
Create	Create FMU from Open Modelica file
Execute	Run OPC UA mock server and normal OPC-UA client
Clean	Delete the temporary files

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital twins/opc-ua-waterplant
```

If required, change the execute permission of lifecycle scripts.

```
chmod +x lifecycle/*
```

Now, run the following scripts:

INSTALL

Installs Open Modelica, Python 3.10 and the required pip dependencies

```
lifecycle/install
```

CREATE

Create `Test_DTCONEDAR.fmu` co-simulation model from `Test_DTCONEDAR.mo open modelica file.

```
lifecycle/create
```

EXECUTE

Start the mock OPC UA server in the background. Run the OPC UA client.

```
lifecycle/execute
```

CLEAN

Remove the temporary files created by Open Modelica and output files generated by OPC UA client.

```
lifecycle/clean
```

References

More explanation about this example is available at:


```
Royo, L., Labarías, A., Arnau, R., Gómez, A., Ezquerra, A., Cilla, I., &
Díez-Antoñazas, L. (2023). Improving energy efficiency in the tertiary
treatment of Alguazas WWTP (Spain) by means of digital twin development
based on Physics modelling . Cambridge Open Engage.
[doi:10.33774/coe-2023-1vjcw](https://doi.org/10.33774/coe-2023-1vjcw)
```

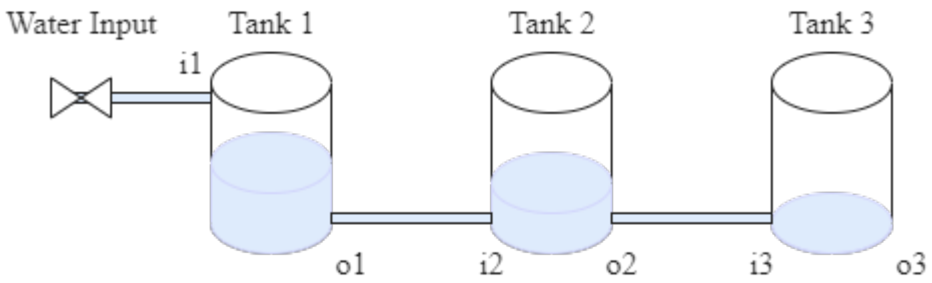

Acknowledgements

The work on this example was done in a project subsidised thanks to the support grants for Agrupación Empresarial Innovadora (AEI) of the Ministry of Industry, Trade and Tourism (MINCOTUR) with the aim of improving the competitiveness of small and medium-sized enterprises within the Recovery, Transformation and Resilience Plan (PRTR) financed by the Next Generation EU (NGEU) funds, with Grant Number: AEI-010500-2022b-196.

3.6.7 Three-Tank System Digital Twin

Overview

The three-tank system is a simple case study allows us to represent a system that is composed of three individual components that are coupled in a cascade as follows: The first tank is connected to the input of the second tank, and the output of the second tank is connected to the input of the third tank.



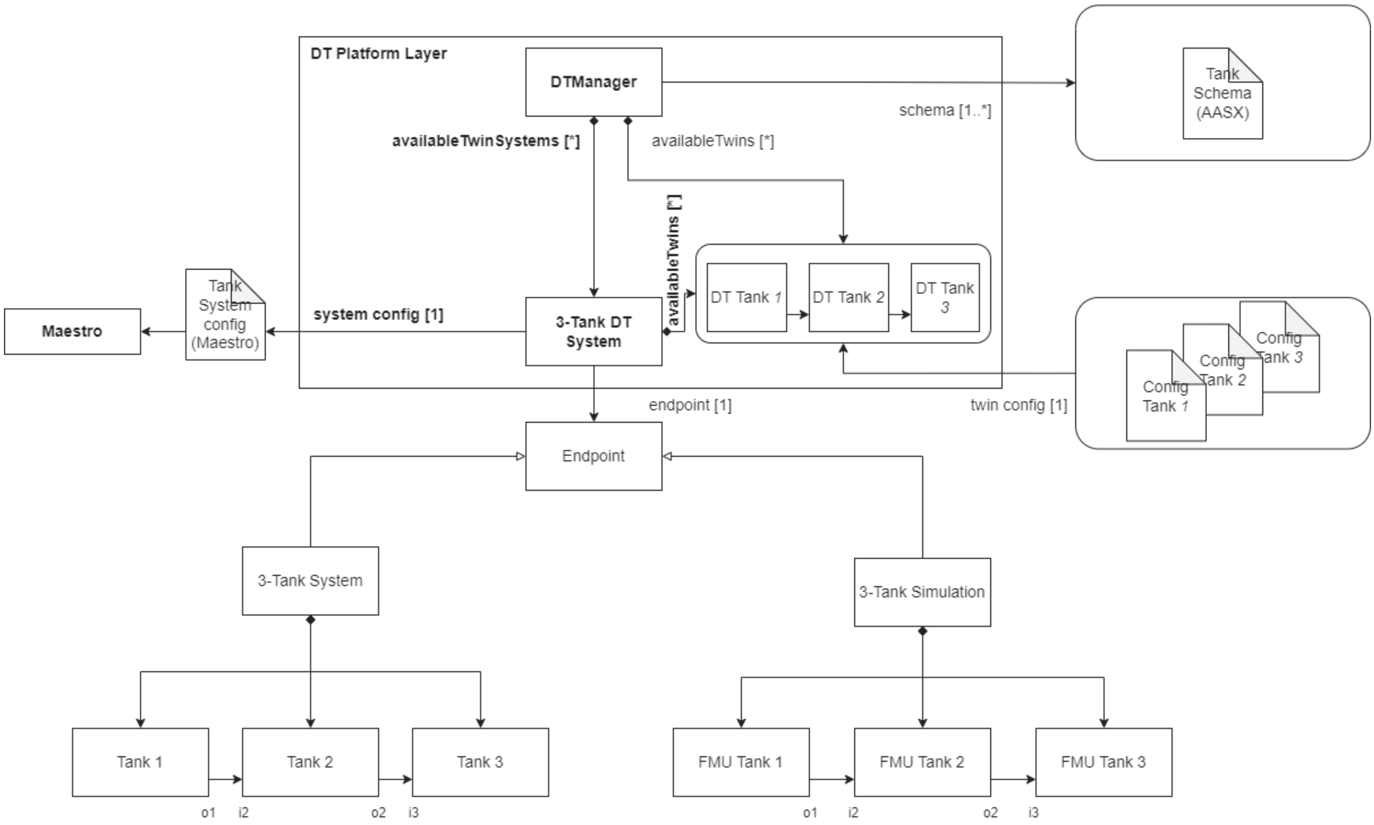
This example contains only the simulated components for demonstration purposes; therefore, there is no configuration for the connection with the physical system.

The three-tank system case study is managed using the `DTManager`, which is packed as a jar library in the tools, and run from a java main file. The `DTManager` uses Maestro as a slave for co-simulation, so it generates the output of the co-simulation.

The main file can be changed according to the application scope, i.e., the `/workspace/examples/tools/three-tank/TankMain.java` can be manipulated to get a different result.

The `/workspace/examples/models/three-tank/` folder contains the `Linear.fmu` file, which is a non-realistic model for a tank with input and output and the `TankSystem.aasx` file for the schema representation with Asset Administration Shell. The three instances use the same `.fmu` file and the same schema due to being of the same object class. The `DTManager` is in charge of reading the values from the co-simulation output.

Example Structure



Digital Twin Configuration

This example uses two models, two tools, one data, and one script. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Model	Linear.fmu	Private	No
	TankSystem.aasx	Private	No
Tool	DTManager-0.0.1-Maestro.jar (wraps Maestro)	Common	Yes
	maestro-2.3.0-jar-with-dependencies.jar (used by DTManager)	Common	Yes
	TankMain.java (main script)	Private	No
Data	outputs.csv	Private	No

This DT has multiple configuration files. The *coe.json* and *multimodel.json* are used by Maestro tool. The *tank1.conf*, *tank2.conf* and *tank3.conf* are the config files for three different instances of one model (Linear.fmu).

Lifecycle Phases

The lifecycles that are covered include:

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool
Execute	The DT Manager executes the three-tank digital twin and produces output in `data/three-tank/output` directory
Terminate	Terminating the background processes and cleaning up the output

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital twins/three-tank
```

If required, change the execute permission of lifecycle scripts you need to execute, for example:

```
chmod +x lifecycle/create
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 11 and pip dependencies. Also creates `DTManager` tool (DTManager-0.0.1-Maestro.jar) from source code.

```
lifecycle/create
```

EXECUTE

Execute the three-tank digital twin using DTManager. DTManager in-turn runs the co-simulation using Maestro. Generates the co-simulation output.csv file at `/workspace/examples/data/three-tank/output`.

```
lifecycle/execute
```

TERMINATE

Stops the Maestro running in the background. Also stops any other jvm process started during **execute** phase.

```
lifecycle/terminate
```

CLEAN

Removes the output generated during execute phase.

```
lifecycle/terminate
```

Examining the results

Executing this Digital Twin will generate a co-simulation output, but the results can also be monitored from updating the `/workspace/examples/tools/three-tank/TankMain.java` with a specific set of `getAttributeValue` commands, such as shown in the code.

That main file enables the online execution of the Digital Twin and its internal components.

The output of the co-simulation is generated to the `/workspace/examples/data/three-tank/output` folder.

In the default example, the co-simulation is run for 10 seconds in steps of 0.5 seconds. This can be modified for a longer period and different step size. The output stored in `outputs.csv` contains the level, in/out flow, and leak values.

No data from the physical twin are generated/used.

References

More information about the DT Manager is available at:

```
D. Lehner, S. Gil, P. H. Mikkelsen, P. G. Larsen and M. Wimmer,
"An Architectural Extension for Digital Twin Platforms to Leverage
Behavioral Models," 2023 IEEE 19th International Conference on
Automation Science and Engineering (CASE), Auckland, New Zealand,
2023, pp. 1-8, doi: 10.1109/CASE56687.2023.10260417.
```

3.6.8 Flex Cell Digital Twin with Two Industrial Robots

Overview

The flex-cell Digital Twin is a case study with two industrial robotic arms, a UR5e and a Kuka LBR iiwa 7, working in a cooperative setting on a manufacturing cell.

The case study focuses on the robot positioning in the discrete cartesian space of the flex-cell working space. Therefore, it is possible to send (X,Y,Z) commands to both robots, which refer to the target hole and height they want should move to.

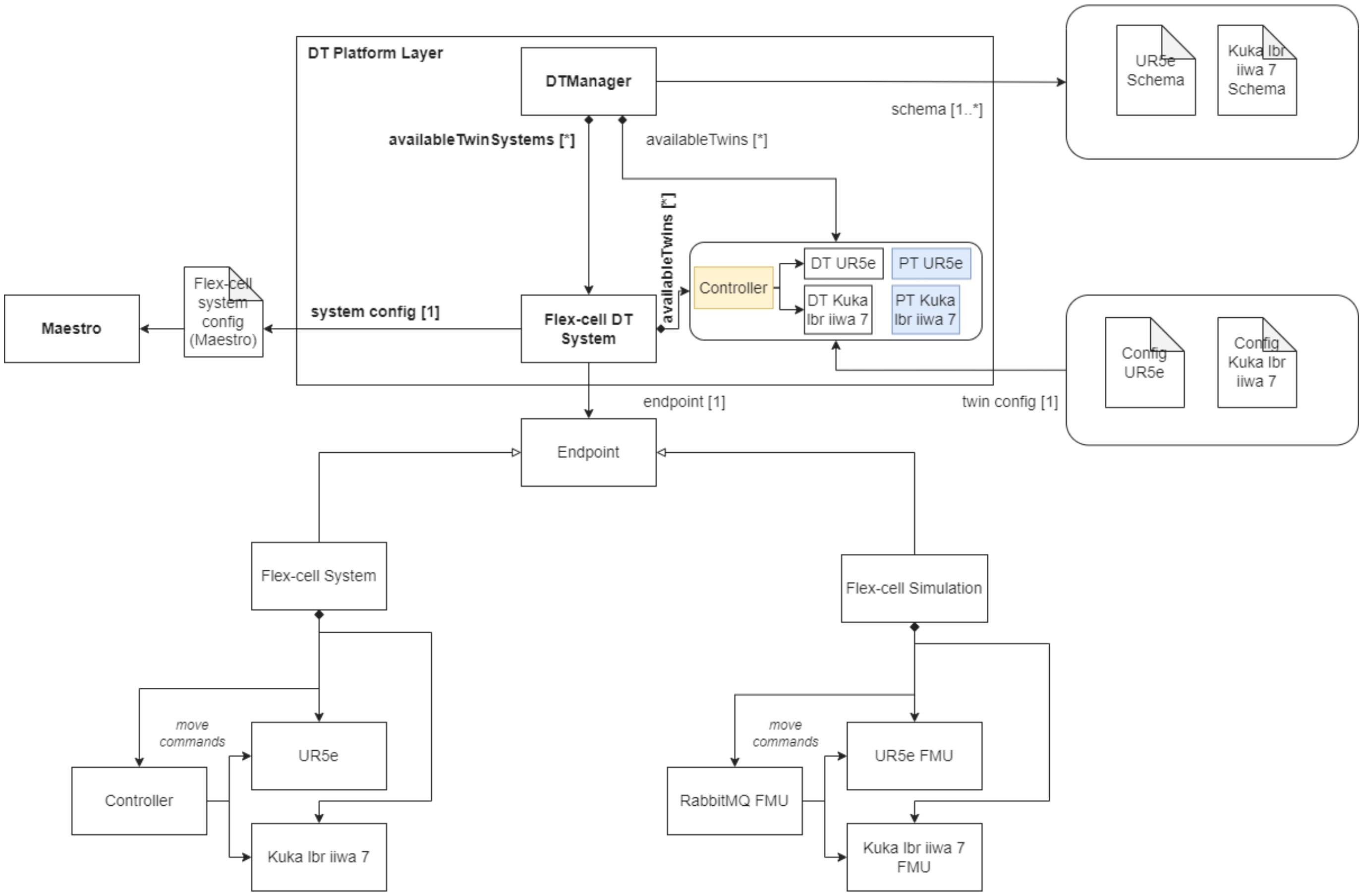
The flex-cell case study is managed using the `TwinManager` (formerly `DTManager`), which is packed as a jar library in the tools, and run from a java main file.

The `TwinManager` uses Maestro as a slave for co-simulation, so it generates the output of the co-simulation and can interact with the real robots at the same time (with the proper configuration and setup). The mainfile can be changed according to the application scope, i.e., the `/workspace/examples/tools/flex-cell/FlexCell.java` can be manipulated to get a different result.

The `/workspace/examples/models/flex-cell/` folder contains the `.fmu` files for the kinematic models of the robotic arms, the `.urdf` files for visualization (including the grippers), and the `.aasx` files for the schema representation with Asset Administration Shell.

The case study also uses RabbitMQFMU to inject values into the co-simulation, therefore, there is the rabbitmqfmu in the models folder as well. Right now, RabbitMQFMU is only used for injecting values into the co-simulation, but not the other way around. The `TwinManager` is in charge of reading the values from the co-simulation output and the current state of the physical twins.

Example Structure



Digital Twin Configuration

This example uses seven models, five tools, six data, two functions, and one script. The specific assets used are:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Model	kukalbriiwa_model.fmu	Private	No
	kuka_irw_gripper_rg6.urdf	Private	No
	kuka.aasx	Private	No
	ur5e_model.fmu	Private	No
	ur5e_gripper_2fg7.urdf	Private	No
	ur5e.aasx	Private	No
	rmq-vhost.fmu	Private	Yes
Tool	maestro-2.3.0-jar-with-dependencies.jar	Common	Yes
	TwinManagerFramework-0.0.2.jar	Private	Yes
	urinterface	Private	No
	kukalbrinterface	Private	No
	robots_flexcell	Private	No
	FlexCell.java (main script)	Private	No
Data	publisher-flexcell-physical.py	Private	No
	ur5e_mqtt_publisher.py	Private	No
	connections.conf	Private	No
	outputs.csv	Private	No
	kukalbriiwa7_actual.csv	Private	No
	ur5e_actual.csv	Private	No
Function	plots.py	Private	No
	prepare.py	Private	No

Lifecycle Phases

The lifecycles that are covered include:

1. Installation of dependencies in the create phase.
2. Preparing the credentials for connections in the prepare phase.
3. Execution of the experiment in the execution phase.
4. Saving experiments in the save phase.
5. Plotting the results of the co-simulation and the real data coming from
6. the robots in the analyze phase.
7. Terminating the background processes and cleaning up the outputs in the termination phase.

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool, Compiles source code of DTManager to create a usable jar package (used as tool)
Prepare	Takes the RabbitMQ and MQTT credentials in connections.conf file and configures different assets of DT.
Execute	The DT Manager executes the flex-cell DT and produces output in `data/flex-cell/output` directory
Save	Save the experimental results
Analyze	Uses plotting functions to generate plots of co-simulation results
Terminate	Terminating the background processes
Clean	Cleans up the output data

Run the example

To run the example, change your present directory.

```
cd /workspace/examples/digital twins/flex-cell
```

If required, change the execute permission of lifecycle scripts you need to execute, for example:

```
chmod +x lifecycle/create
```

This example requires Java 11. The **create** script installs Java 11; however if you have already installed other Java versions, your default *java* might be pointing to another version. You can check and modify the default version using the following commands.

```
java -version
update-alternatives --config java
```

Now, run the following scripts:

CREATE

Installs Open Java Development Kit 11 and pip dependencies. Also creates `DTManager` tool (DTManager-0.0.1-Maestro.jar) from source code.

```
lifecycle/create
```

PREPARE

Configure different assets of DT with these credentials. The `functions/flex-cell/prepare.py` script is used for this purpose. The only thing needed to set up the connection is to update the file `/workspace/examples/data/flex-cell/connections.conf` with the connection parameters for MQTT and RabbitMQ and then execute the `prepare` script.

```
lifecycle/prepare
```

The following files are updated with the configuration information:

1. `/workspace/examples/digital_twins/flex-cell/kuka_actual.conf`
2. `/workspace/examples/digital_twins/flex-cell/ur5e_actual.conf`
3. `/workspace/examples/tools/flex-cell/publisher-flexcell-physical.py`
4. `modelDescription.xml` for the RabbitMQFMU require special credentials to connect to the RabbitMQ and the MQTT brokers.

EXECUTE

Execute the flex-cell digital twin using DTManager. DTManager in-turn runs the co-simulation using Maestro. Generates the co-simulation output.csv file at `/workspace/examples/data/flex-cell/output`.

```
lifecycle/execute
```

SAVE

Each execution of the DT is treated as a single run. The results of one execution are saved as time-stamped co-simulation output file in The DT Manager executes the flex-cell digital twin and produces output in `data/flex-cell/output/saved_experiments` directory.

```
lifecycle/execute
```

ANALYZE

There are dedicated plotting functions in `functions/flex-cell/plots.py`. This script plots the co-simulation results against the recorded values from the two robots.

```
lifecycle/analyze
```

TERMINATE

Stops the Maestro running in the background. Also stops any other jvm process started during **execute** phase.

```
lifecycle/terminate
```

CLEAN

Removes the output generated during execute phase.

```
lifecycle/clean
```

Examining the results

Executing this Digital Twin will generate a co-simulation output, but the results can also be monitored from updating the `/workspace/examples/tools/flex-cell/FlexCell.java` with a specific set of `getAttributeValue` commands, such as shown in the code. That main file enables the online execution and comparison on Digital Twin and Physical Twin at the same time and at the same abstraction level.

The output is generated to the `/workspace/examples/data/flex-cell/output` folder. In case a specific experiments is to be saved, the `save` lifecycle script stores the co-simulation results into the `/workspace/examples/data/flex-cell/output/saved_experiments` folder.

In the default example, the co-simulation is run for 10 seconds in steps of 0.5 seconds. This can be modified for a longer period and different step size. The output stored in `outputs.csv` contains the joint position of both robotic arms and the current discrete (X,Y,Z) position of the TCP of the robot. Additional variables can be added, such as the discrete (X,Y,Z) position of the other joints.

When connected to the real robots, the tools `urinterface` and `kukalbrinterface` log their data at a higher sampling rate.

References

The RabbitMQ FMU github repository contains complete documentation and source code of the rmq-vhost.fmu.

More information about the DT Manager and the case study is available in:

1. D. Lehner, S. Gil, P. H. Mikkelsen, P. G. Larsen and M. Wimmer, "An Architectural Extension for Digital Twin Platforms to Leverage Behavioral Models," 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand, 2023, pp. 1-8, doi: 10.1109/CASE56687.2023.10260417.
2. S. Gil, P. H. Mikkelsen, D. Tola, C. Schou and P. G. Larsen, "A Modeling Approach for Composed Digital Twins in Cooperative Systems," 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 2023, pp. 1-8, doi: 10.1109/ETFA54631.2023.10275601.
3. S. Gil, C. Schou, P. H. Mikkelsen, and P. G. Larsen, “Integrating Skills into Digital Twins in Cooperative Systems,” in 2024 IEEE/SICE International Symposium on System Integration (SII), 2024, pp. 1124–1131, doi: 10.1109/SII58957.2024.10417610.

4. Frequently Asked Questions

4.1 Abreviations

Term	Full Form
DT	Digital Twin
DTaaS	Digital Twin as a Service
PT	Physical Twin

4.2 General Questions

What is DTaaS?

DTaaS is software platform on which you can create and run digital twins. Please see the features page to get a sense of the things you can do in DaaS.

What is the scope and current capabilities of DTaaS?

1. DTaaS is a web based interface to allow you to invoke various tools related to work you want to perform with one or more DTs.
2. DTaaS as such doesn't help you create DTs; it loads and runs DTs that you create using tools and services that are available via DTaaS.
3. DTaaS can help you create DTs only if your DT or DT asset authoring tools can work in user workspace containing Linux terminal / Desktop / Jupyter / VSCode services.
4. DTaaS as such won't help you to install DTs that you get from elsewhere.
5. DTs are just executables, as far as DTaaS is concerned. Users are not constrained to work with DTs in a certain way. The DTaaS suggests creation of DTs from reusable assets and provides a suggestive structure for DTs. The examples provide more insight into DTaaS way of working with DTs. But this suggested workflow is not binding on the users.
6. DTaaS doesn't help you to autorun run DTs, other than by providing user workspaces. These user workspaces are based on Ubuntu 20.04 Operating system.
7. The current user interface of DTaaS web application is heavily reliant on the use of Jupyter lab and notebook. The **Digital Twins** page has Create / Execute / Analyze sections but all point to Jupyter lab. Web interface. The functionality of these pages is still under development.

Are there any Key Performance / Capability Indicators for DTaaS?

Key Performance Indicator	Value
Processor	Two AMD EPYC 7443 24-Core Processors
Maximum Storage Capacity	4TB SSD, RAID 0 configuration
Storage Type	File System
Maximum file size	10 GB
Data transfer speed	100 Mbps
Data Security	Yes
Data Privacy	Yes
Redundancy	None
Availability	It is a matter of human resources. If you have human
resources to maintain DTaaS round the clock,	
upwards 95% is easily possible.	

Do you provide licensed software like Matlab?

The licensed software are not available on the software platform. But users have private workspaces which are based on Linux-based xfce Desktop environment. Users can install software in their workspaces. Please see a screencast of using Matlab Simulink within the DTaaS software. The licensed software installed by one user is not available to another user.

4.3 Digital Twin Models

Can DTaaS create new DT models?

DTaaS is not a model creation tool. You can put model creation tool inside DTaaS and create new models. The DTaaS itself does not create digital twin models but it can help users create digital twin models. You can run Linux desktop / terminal tools inside the DTaaS. So you can create models inside DTaaS and run them using tools that can run in Linux. The Windows only tools can not run in DTaaS.

How can DTaaS help to design geometric model? Does it support 3D modeling and simulation?

Well, DTaaS by itself does not produce any models. DTaaS only provides a platform and an ecosystem of services to facilitate digital twins to be run as services. Since each user has a Linux OS at their disposal, they can also run digital twins that have graphical interface. In summary, DTaaS is neither a modeling nor simulation tool. If you need these kinds of tools, you need to bring them onto the platform. For example, if you need Matlab for your work, you need to bring he licensed Matlab software.

Commercial DT platforms in market provide modelling and simulation alongside integration and UI. DTaas is not able to do any modelling or simulation on its own like other commercial platforms. Is this a correct understanding?

Yes, you are right

Can DTaaS support only the information models (or behavioral models) or some other kind of models?

The DTaaS as such is agnostic to the kind of models you use. DTaaS can run all kinds of models. This includes behavioral and data models. As long as you have models and the matching solvers that can run in Linux OS, you are good to go in DTaaS. In some cases, models and solvers (tools) are bundled together to form monolithic DTs. The DTaaS does not limit you from running such DTs as well. DTaaS does not provide dedicated solvers. But if you can install a solver in your workspace, then you don't need the platform to provide one.

Does it support XML-based representation and ontology representation?

Currently No. **We are looking for users needing this capability. If you have concrete requirements and an example, we can discuss a way of realizing it in DTaaS**.

4.4 Communication Between Physical Twin and Digital Twin

How would you measure a physical entity like shape, size, weight, structure, chemical attributes etc. using DTaaS? Any specific technology used in this case?

The real measurements are done at physical twin which are then communicated to the digital twin. Any digital twin platform like DTaaS can only facilitate this communication of these measurements from physical twin. The DTaaS provides InfluxDB, RabbitMQ and Mosquitto services for this purpose. These three are probably most widely used services for digital twin communication. Having said that, DTaaS allows you to utilize other communication technologies and services hosted elsewhere on the Internet.

How a real-time data can be differed from static data and what is the procedure to identify dynamic data? Is there any UI or specific tool used here?

DTaaS can not understand the static or dynamic nature of data. It can facilitate storing names, units and any other text description of interesting quantities (weight of batter, voltage output etc). It can also store the data being sent by the physical twin. The distinction between static and dynamic data needs to be made by the user. Only metadata of the data can reveal such more information about the nature of data. A tool can probably help in very specific cases, but you need metadata. If there is a human being making this distinction, then the need for metadata goes down but does not completely go away. In some of the DT platforms supported by manufacturers, there is a tight integration between data and model. In this case, the tool itself is taking care of the metadata. The DTaaS is a generic platform which can support execution of digital twins. If a tool can be executed on a Linux desktop / commandline, the tool can be supported within DTaaS. The tool (ex. Matlab) itself can take care of the metadata requirements.

How can DTaaS control the physical entity? Which technologies it uses for controlling the physical world?

At a very abstract level, there is a communication from physical entity to digital entity and back to physical entity. How this communication should happen is decided by the person designing the digital entity. The DTaaS can provide communication services that can help you do this communication with relative ease. You can use InfluxDB, RabbitMQ and Mosquitto services hosted on DTaaS for two communication between digital and physical entities.

4.5 Data Management

Does DTaaS support data collection from different sources like hardware, software and network? Is there any user interface or any tracking instruments used for data collection?

The DTaaS provids InfluxDB, RabbitMQ, MQTT and MongoDB services. Both the physical twin and digital twin can utilize these protocols for communication. The IoT (time-series) data can be collected using InfluxDB and MQTT broker services. There is a user interface for InfluxDB which can be used to analyze the data collected. Users can also manually upload their data files into DTaaS.

Which transmission protocol does DTaaS allow?

InfluxDB, RabbitMQ, MQTT and anything else that can be used from Cloud service providers.

Does DTaaS support multisource information and combined multi sensor input data? Can it provide analysis and decision-supporting inferences?

You can store information from multiple sources. The existing InfluxDB services hosted on DTaaS already has a dedicated Influx / Flux query language for doing sensor fusion, analysis and inferences.

Which kinds of visualization technologies DTaaS can support (e.g. graphical, geometry, image, VR/AR representation)?

Graphical, geometric and images. If you need specific licensed software for the visualization, you will have to bring the license for it. DTaaS does not support AR/VR.

Can DTaaS collect data directly from sensors?

Yes

Is DTaaS able to transmit data to cloud in real time?

Yes

4.6 Platform Native Services on DTaaS Platform

Is DTaaS able to detect the anomalies about-to-fail components and prescribe solutions?

This is the job of a digital twin. If you have a ready to use digital twin that does the job, DTaaS allows others to use your solution.

4.7 Comparison with other DT Platforms

All the DT platforms seem to provide different features. Is there a comparison chart?

Here is a qualitative comparison of different DT integration platforms:

Legend: high performance (**H**), mid performance (**M**) and low performance (**L**)

DT Platforms	License	DT Development Process	Connectivity	Security	Processing power, performance and Scalability	Data Storage	V
Microsoft Azure DT	Commercial Cloud	H	H	H	M	H	H
AWS IOT Greengrass	Open source commercial	H	H	H	M	H	H
Eclipse Ditto	Open source	M	H	M	H	H	L
Asset Administration Shell	Open source	H	H	L	H	M	L
PTC Thingworx	Commercial	H	H	H	H	H	M
GE Predix	Commercial	M	H	H	M	L	M
AU's DTaaS	Open source	H	H	L	L	M	M

Adopted by Tanusree Roy from Table 4 and 5 of the following paper.

Ref: Naseri, F., Gil, S., Barbu, C., Cetkin, E., Yarimca, G., Jensen, A. C., ... & Gomes, C. (2023). Digital twin of electric vehicle battery systems: Comprehensive review of the use cases, requirements, and platforms. Renewable and Sustainable Energy Reviews, 179, 113280.

All the comparisons between DT platforms seems so confusing. Why?

The fundamental confusion comes from the fact that different DT platforms (Azure DT, GE Predix) provide different kind of DT capabilities. You can run all kinds of models natively in GE Predix. In fact you can run models even next to (on) PTs using GE Predix. But you cannot natively do that in Azure DT service. You have to do the leg work of integrating with other Azure services or third-party services to get the kind of capabilities that GE Predix natively provides in one interface. The takeaway is that we pick horses for the courses.

4.8 Create Assets

Can DTaaS be used to create new DT assets?

The core feature of DTaaS software is to help users create DTs from assets already available in the library. Create Library Assets However, it is possible for users to take advantage of services available in their workspace to install asset authoring tools in their own workspace. These authoring tools can then be used to create and publish new assets. User workspaces are private and are not shared with other users. Thus any licensed software tools installed in their workspace is only available to them.

4.9 GDPR Concerns

Does your platform adhere to GDPR compliance standards? If so, how?

The DTaaS software platform does not store any personal information of users. It only stores username to identify users and these usernames do not contain enough information to deduce the true identify of users.

Which security measures are deployed? How is data encrypted (if exists)?

The default installation requires a HTTPS terminating reverse proxy server from user to the DTaaS software installation. The administrators of DTaaS software can also install HTTPS certificates into the application. The codebase can generate HTTPS application and the users also have the option of installing their own certificates obtained from certification agencies such as LetsEncrypt.

What security measures does your cloud provider offer?

The current installation of DTaaS software runs on Aarhus University servers. The university network offers firewall access control to servers so that only permitted user groups have access to the network and physical access to the server.

How is user access controlled and authenticated?

There is a two-level authorization mechanism in place in each default installation of DTaaS. The first-level is HTTP basic authorization over secure HTTPS connection. The second-level is the OAuth PKCE authorization flow for each user. The OAuth authorization is provider by a Gitlab instance. The DTaaS does not store the account and authorization information of users.

Does you platform manage personal data? How is data classified and tagged based on the sensitivity? Who has access to the critical data?

The platform does not store personal data of users.

How are identities and roles managed within the platform?

There are two roles for users on the platform. One is the administrator and the other one is user. The user roles are managed by the administrator.

5. Developer

5.1 Developers Guide

This guide is to help developers get familiar with the project. Please see developer-specific Slides, Video, and Research paper.

5.1.1 💻 Development Environment

Ideally, developers should work on Ubuntu/Linux. Other operating systems are not supported inherently and may require additional steps.

To start with, install the required software and git-hooks.

```
bash script/env.sh
bash script/configure-git-hooks.sh
```

The git-hooks will ensure that your commits are formatted correctly and that the tests pass before you push the commits to remote repositories.

You can also run the git-hooks manually before committing or pushing by using the run commands below. The autoupdate command will set the revisions of the git repos used in the .pre-commit-config.yaml up to date.

```
pre-commit run --hook-stage pre-commit # runs format and syntax checks
pre-commit run --hook-stage pre-push   # runs test
pre-commit autoupdate                  # update hooks to latests versions
```

Be aware that the tests may take a long time to run. If you want to skip the tests or formatting, you can use the `--no-verify` flag on `git commit` or `git push`. Please use this option with care.

There is a script to download all the docker containers used in the project. You can download them using

```
bash script/docker.sh
```

⚠️ The docker images are large and are likely to consume about 5GB of bandwidth and 15GB of space. You will have to download the docker images on a really good network.

5.1.2 🏗️ Development Workflow

To manage collaboration by multiple developers on the software, a development workflow is in place. Each developer should follow these steps:

1. Fork of the main repository into your github account.
2. Setup Code Climate and Codecov for your fork. The codecov does not require secret token for public repositories.
3. Install git-hooks for the project.
4. Use Fork, Branch, PR workflow.
5. Work in your fork and open a PR from your working branch to your `feature/distributed-demo` branch. The PR will run all the github actions, code climate and codecov checks.
6. Resolve all the issues identified in the previous step.
7. If you have access to the integration server, try your working branch on the integration server.
8. Once changes are verified, a PR should be made to the `feature/distributed-demo` branch of the upstream DTaaS repository.
9. The PR will be merged after checks by either the project administrators or the maintainers.

Remember that every PR should be meaningful and satisfies a well-defined user story or improve the code quality.

5.1.3 👁 Code Quality

The project code qualities are measured based on:

- Linting issues identified by Code Climate
- Test coverage report collected by Codecov
- Successful github actions

Code Climate

Code Climate performs static analysis, linting and style checks. Quality checks are performed by codeclimate are to ensure the best possible quality of code to add to our project.

While any new issues introduced in your code would be shown in the PR page itself, to address any specific issue, you can visit the issues or code section of the codeclimate page.

It is highly recommended that any code you add does not introduce new quality issues. If they are introduced, they should be fixed immediately using the appropriate suggestions from Code Climate, or in worst case, adding a ignore flag (To be used with caution).

Codecov

Codecov keeps track of the test coverage for the entire project. For information about testing and workflow related to that, please see the testing page.

Github Actions

The project has multiple github actions defined. All PRs and direct code commits must have successful status on github actions.

5.2 System

5.2.1 🏰 System Overview

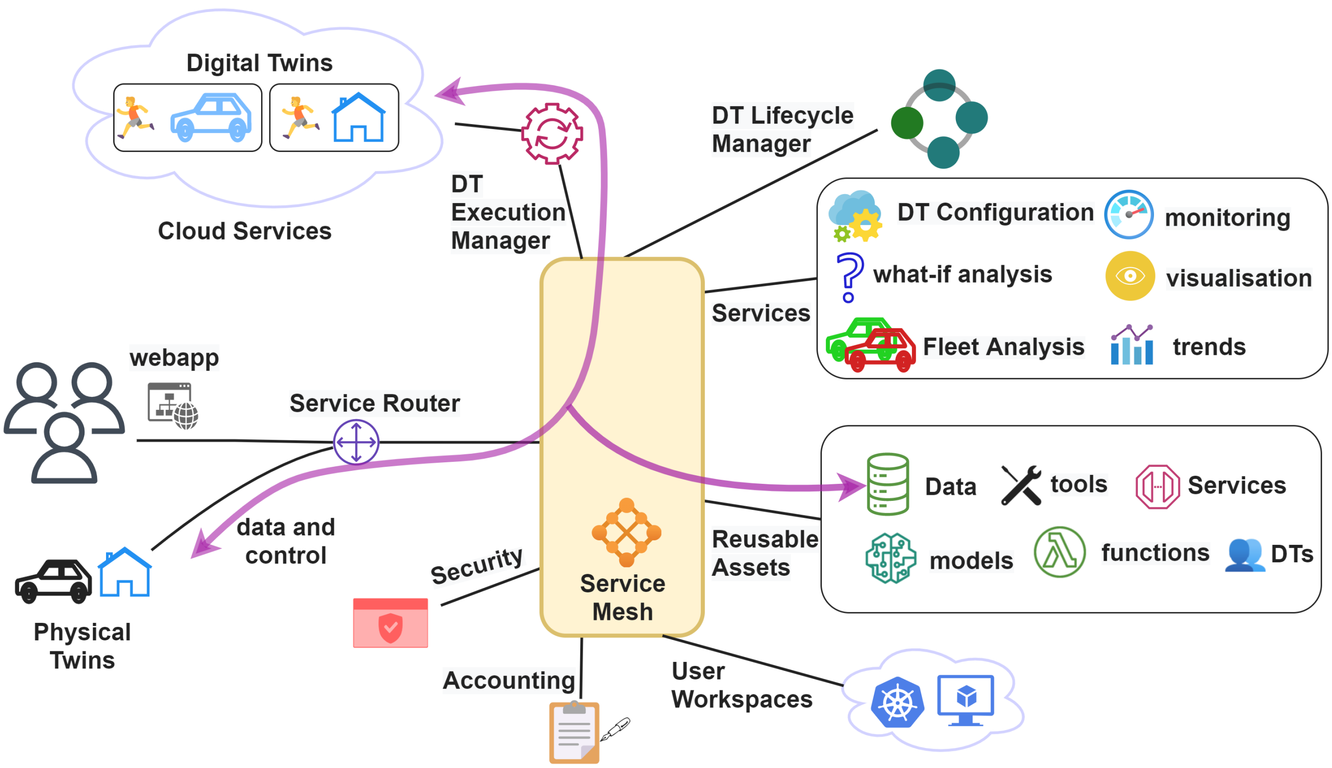
User Requirements

The DTaaS software platform users expect a single platform to support the complete DT lifecycle. To be more precise, the platform users expect the following features:

1. **Author** – create different assets of the DT on the platform itself. This step requires use of some software frameworks and tools whose sole purpose is to author DT assets.
2. **Consolidate** – consolidate the list of available DT assets and authoring tools so that user can navigate the library of reusable assets. This functionality requires support for discovery of available assets.
3. **Configure** – support selection and configuration of DTs. This functionality also requires support for validation of a given configuration.
4. **Execute** – provision computing infrastructure on demand to support execution of a DT.
5. **Explore** – interact with a DT and explore the results stored both inside and outside the platform. Exploration may lead to analytical insights.
6. **Save** – save the state of a DT that's already in the execution phase. This functionality is required for on demand saving and re-spawning of DTs.
7. **Services** – integrate DTs with on-platform or external services with which users can interact with.
8. **Share** – share a DT with other users of their organisation.

System Architecture

The figure shows the system architecture of the the DTaaS software platform.



SYSTEM COMPONENTS

The users interact with the software platform using a webapp. The service router is a single point of entry for direct access to the platform services. The service router is responsible for controlling user access to the microservice components. The service mesh enables discovery of microservices, load balancing and authorization functionalities.

In addition, there are microservices for catering to managing DT reusable assets, platform services, DT lifecycle manager, DT execution manager, accouting and security. The microservices are complementary and composable; they fulfil core requirements of the system.

The microservices responsible for satisfying the user requirements are:

1. **The security microservice** implements role-based access control (RBAC) in the platform.
2. **The accounting microservice** is responsible for keeping track of the live status of platform, DT asset and infrastructure usage. Any licensing, usage restrictions need to be enforced by the accounting microservice. Accounting is a pre-requisite to commercialisation of the platform. Due to significant use of external infrastructure and resources via the platform, the accounting microservice needs to interface with accounting systems of the external services.
3. **User Workspaces** are virtual environments in which users can perform lifecycle operations on DTs. These virtual environments are either docker containers or virtual machines which provide desktop interface to users.
4. **Reusable Assets** are assets / parts from which DTs are created. Further explation is available on the assets page
5. **Services** are dedicated services available to all the DTs and users of the DTaaS platform. Services build upon DTs and provide user interfaces to users.
6. **DT Execution Manager** provides virtual and isolated execution environments for DTs. The execution manager is also responsible for dynamic resource provisioning of cloud resources.
7. **DT Lifecycle Manager** manages the lifecycle operations on all DTs. It also directs *DT Execution Manager* to perform execute, save and terminate operations on DTs.

If you are interested, please take a look at the C4 architectural diagram.

A mapping of the architectural components to related pages in the documentation is available in the table.

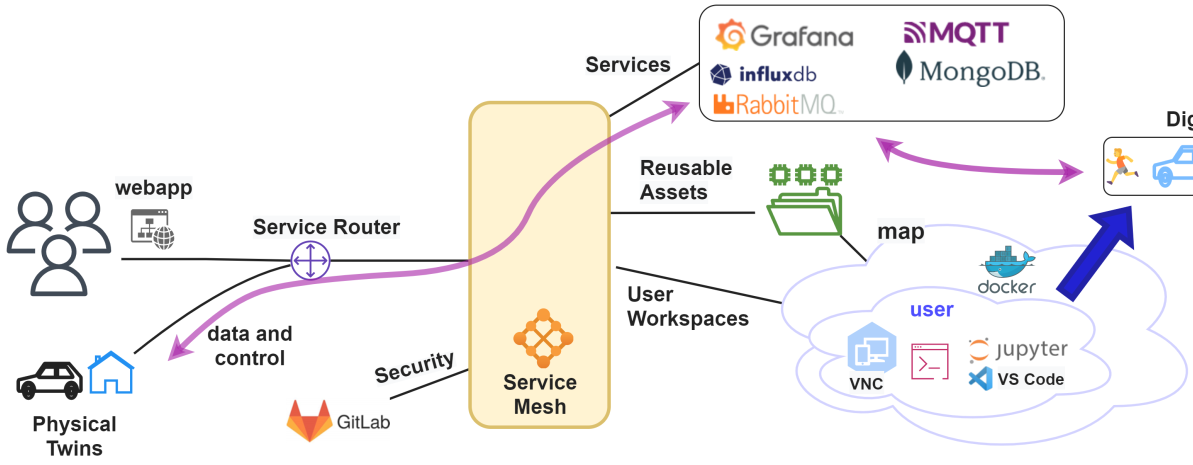
System Component	Doc Page(s)
Service Router	Traefik Gateway
Web Application	React Webapplication
Reusable Assets	Library Microservice
Services	Third-party Services (MQTT, InfluxDB, RabbitMQ, Grafana and MongoDB)
DT Lifecycle	Not available yet
Security	Gitlab OAuth
Accounting	Not available yet
Execution Manager	Not available yet

References

Font sources: fileformat

5.2.2 📋 Current Status

The DTaaS software platform is currently under development. Crucial system components are in place with ongoing development work focusing on increased automation and feature enhancement. The figure below shows the current status of the development work.



A C4 representation of the same diagram is also available.

🔒 User Security

There is authorization mechanisms in place for the react website and the Traefik gateway.

The react website component uses Gitlab for user authorization using OAuth protocol.

GATEWAY AUTHORIZATION

The Traefik gateway has HTTP basic authorization enabled by default. This authorization on top of HTTPS connection can provide a good protection against unauthorized use.

> ⚠️ **Warning**
>
> Please note that HTTP basic authorization over insecure non-TLS is insecure.

There is also a possibility of using self-signed mTLS certificates. The current security functionality is based on signed Transport Layer Security (TLS) certificates issued to users. The TLS certificate based mutual TLS (mTLS) authorization protocol provides better security than the usual username and password combination. The mTLS authorization takes place between the users browser and the platform gateway. The gateway federates all the backend services. The service discovery, load balancing, and health checks are carried by the gateway based on a dynamic reconfiguration mechanism.

> 📝 **Note**
>
> The mTLS is not enabled in the default install. Please use the scripts in `ssl/` directory to generate the required certificates for users and Traefik gateway.

🧑‍💻 User Workspaces

All users have dedicated dockerized-workspaces. These docker-images are based on container images published by mltooling group.

Thus DT experts can develop DTs from existing DT components and share them with other users. A file server has been setup to act as a DT asset repository. Each user gets space to store private DT assets and also gets access to shared DT assets. Users can synchronize their private DT assets with external git repositories. In addition, the asset repository transparently gets mapped to user workspaces within which users can perform DT lifecycle operations. There is also a library microservice which in the long-run will replace the file server.

Users can run DTs in their workspaces and also permit remote access to other users. There is already shared access to internal and external services. With these two provisions, users can treat live DTs as service components in their own software systems.

🔌 Platform Services

There are four external services integrated with the DTaaS software platform. They are: InfluxDB, Grafana, RabbitMQ, MQTT, and MongoDB.

These services can be used by DTs and PTs for communication, storing and visualization of data. There can also be monitoring services setup based on these services.

Development Priorities

The development priorities for the DTaaS software development team are:

- DT Runner (API Interface to DT)
- Multi-user and microservice security
- Increased automation of installation procedures
- Upgrade software stack of user workspaces
- DT Configuration DSL ín the form of YAML schema
- UI for DT creation

Your contributions are highly welcome.

References

Font sources: fileformat

5.3 OAuth2 in DTaaS

5.3.1 **This document must provide general explanation of OAuth**

The DTaaS uses PKCE authorization flow for React client application and web server authorization flow for traefik gateway. Please an explanation of different oauth flows.

5.3.2 Requirements

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

- 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.
- DNS name (optional, required only when the DTaaS is to be
- deployed on a web server)

5.4 Components

5.4.1 React Website

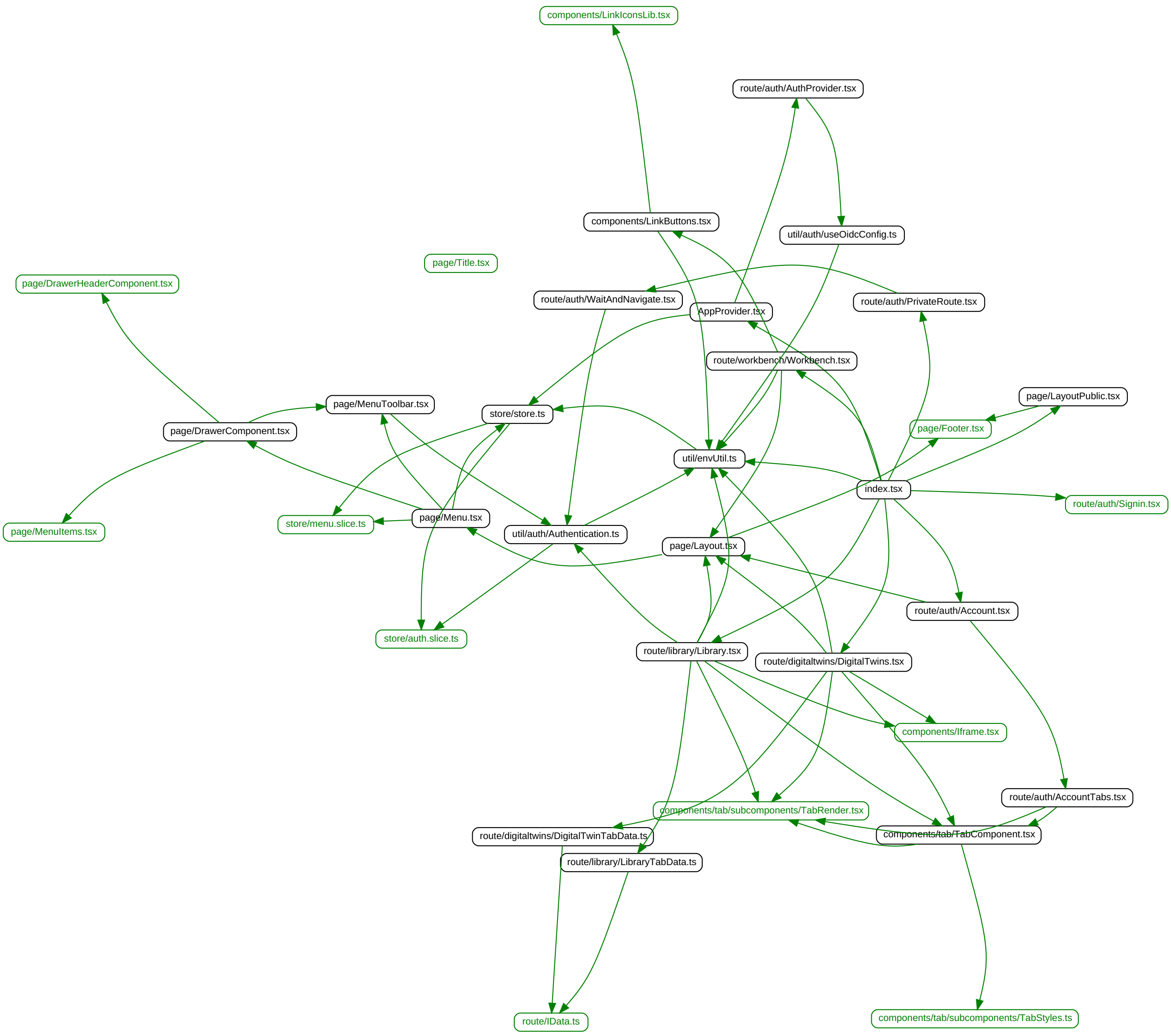
The Website is how the end-users interact with the software platform. The website is being developed as a React single page web application.

A dependency graph for the entire codebase of the react application is:

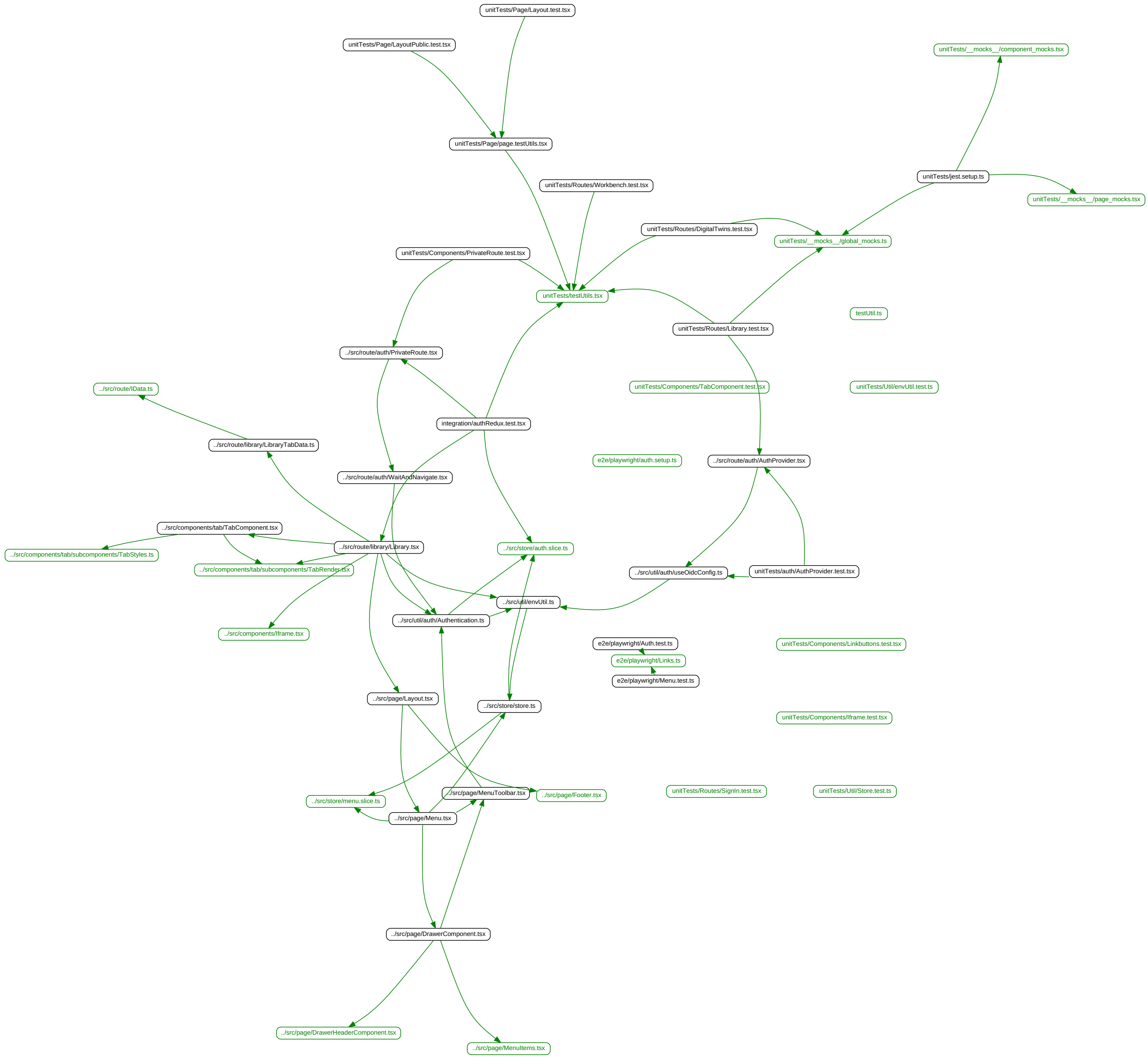
Dependency Graphs

The figures are the dependency graphs generated from the code.

SRC DIRECTORY



TEST DIRECTORY



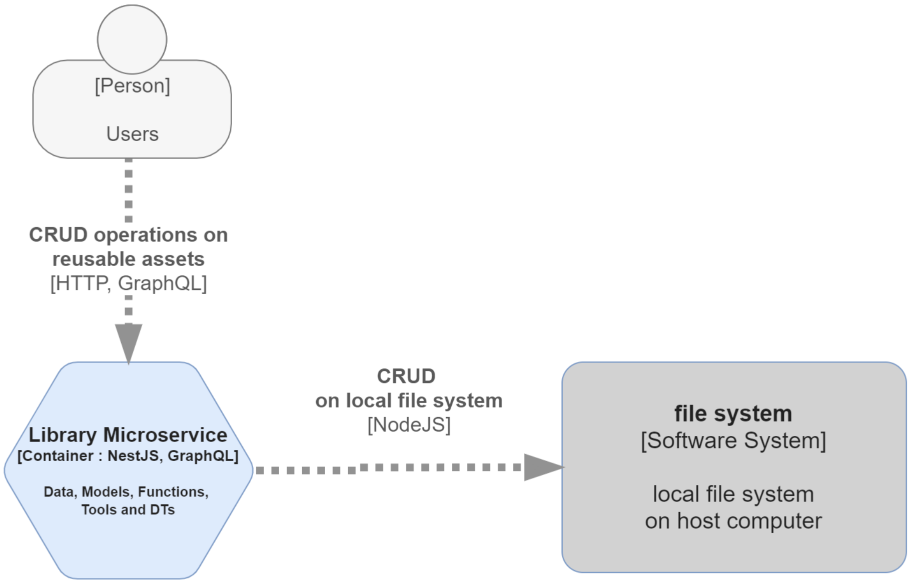
5.4.2 Library Microservice

The Library Microservices provides users with access to files in user workspaces via API. This microservice will interface with local file system and Gitlab to provide uniform Gitlab-compliant API access to files.

> **Warning**
>
> This microservice is still under heavy development. It is still not a good replacement for file server we are using now.

Architecture and Design

The C4 level 2 diagram of this microservice is:



The GraphQL API provided by the library microservice shall be compliant with the Gitlab GraphQL service.

UML Diagrams

CLASS DIAGRAM

```
classDiagram
    class FilesResolver {
    -filesService: IFilesService
    +listDirectory(path: string): Promise<Project>
    +readFile(path: string): Promise<Project>
    }

    class FilesServiceFactory {
    -configService: ConfigService
    -localFilesService: LocalFilesService
    +create(): IFilesService
    }

    class LocalFilesService {
    -configService: ConfigService
    -getFileStats(fullPath: string, file: string): Promise<Project>
    +listDirectory(path: string): Promise<Project>
    +readFile(path: string): Promise<Project>
    }

    class ConfigService {
    +get(propertyPath: string): any
    }

    class IFilesService{
    listDirectory(path: string): Promise<Project>
    readFile(path: string): Promise<Project>
```

```
    }

    IFilesService <|-- FilesResolver: uses
    IFilesService <|.. LocalFilesService: implements
    IFilesService <|-- FilesServiceFactory: creates
    ConfigService <|-- FilesServiceFactory: uses
    ConfigService <|-- LocalFilesService: uses
```

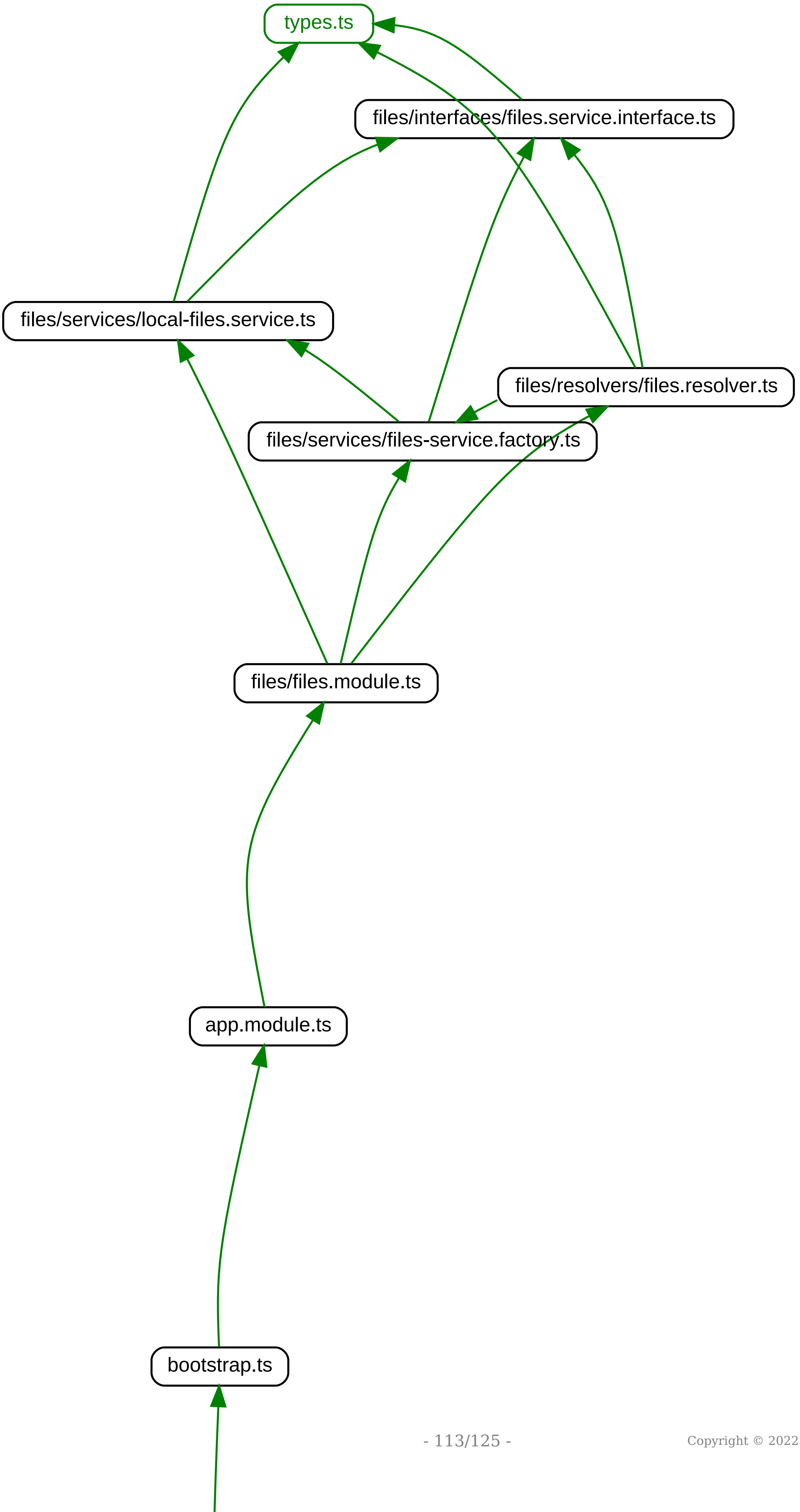
SEQUENCE DIAGRAM

```
sequenceDiagram
    actor Client
    actor Traefik

    box LightGreen Library Microservice
    participant FR as FilesResolver
    participant FSF as FilesServiceFactory
    participant CS as ConfigService
    participant IFS as IFilesService
    participant LFS as LocalFilesService
    end

    participant FS as Local File System DB

    Client ->> Traefik : HTTP request
    Traefik ->> FR : GraphQL query
    activate FR

    FR ->> FSF : create()
    activate FSF

    FSF ->> CS : getConfiguration("MODE")
    activate CS

    CS -->> FSF : return configuration value
    deactivate CS

    alt MODE = Local
    FSF ->> FR : return filesService (LFS)
    deactivate FSF

    FR ->> IFS : listDirectory(path) or readFile(path)
    activate IFS

    IFS ->> LFS : listDirectory(path) or readFile(path)
    activate LFS

    LFS ->> CS : getConfiguration("LOCAL_PATH")
    activate CS

    CS -->> LFS : return local path
    deactivate CS

    LFS ->> FS : Access filesystem
    alt Filesystem error
        FS -->> LFS : Filesystem error
        LFS ->> LFS : Throw new InternalServerErrorException
        LFS -->> IFS : Error
    else Successful file operation
        FS -->> LFS : Return filesystem data
        LFS ->> IFS : return Promise<Project>
    end
    deactivate LFS
    activate IFS
    end

    alt Error thrown
    IFS ->> FR : return Error
    deactivate IFS
    FR ->> Traefik : return Error
    Traefik ->> Client : HTTP error response
    else Successful operation
    IFS ->> FR : return Promise<Project>
    deactivate IFS
    FR ->> Traefik : return Promise<Project>
    Traefik ->> Client : HTTP response
    end

    deactivate FR
```

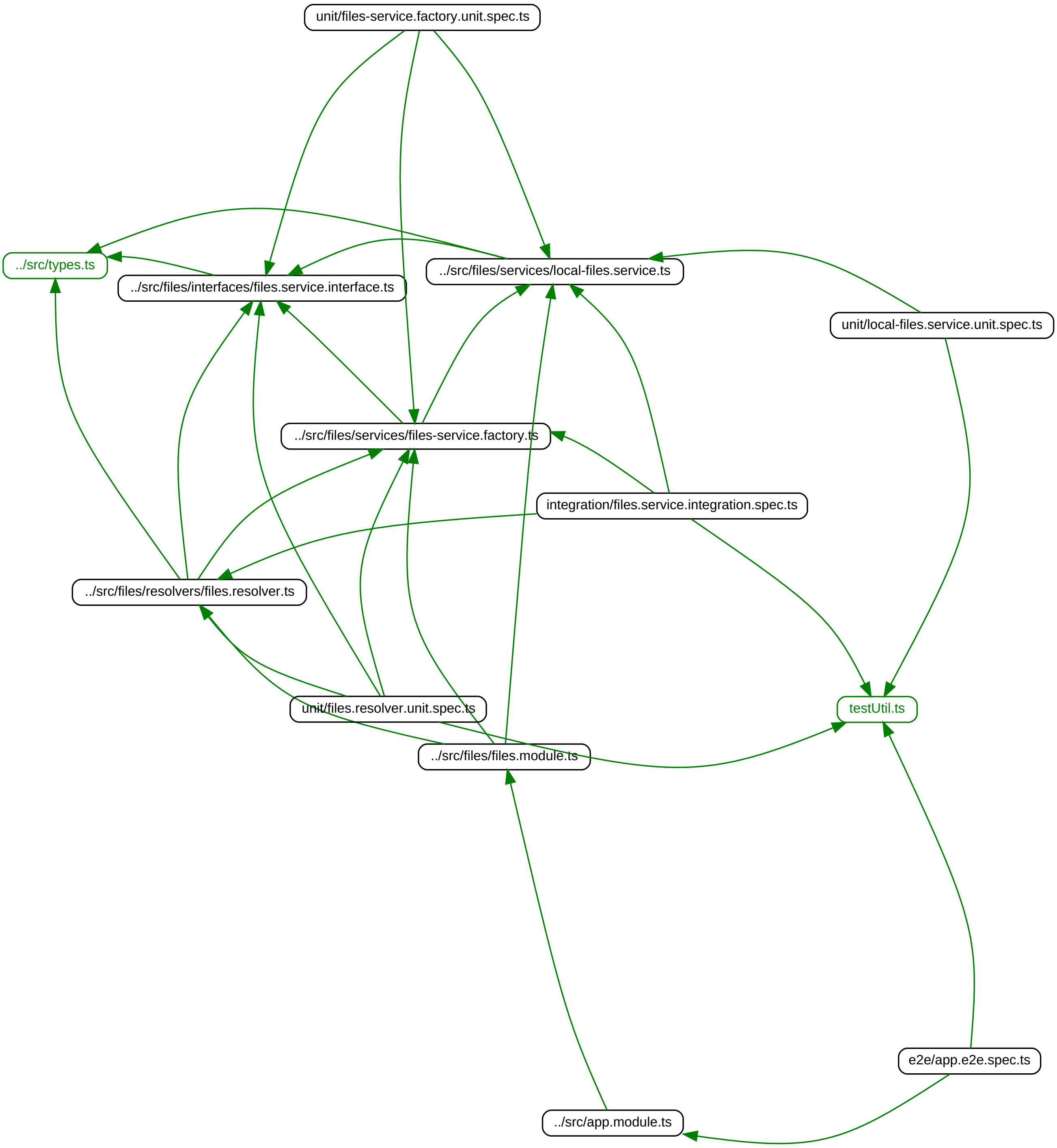
Dependency Graphs

The figures are the dependency graphs generated from the code.

SRC DIRECTORY


types.ts
files/interfaces/files.service.interface.ts
files/services/local-files.service.ts
files/resolvers/files.resolver.ts
files/services/files-service.factory.ts
files/files.module.ts
app.module.ts
bootstrap.ts

TEST DIRECTORY



5.5 Testing

5.5.1 ❓ Common Questions on Testing

What is Software Testing

Software testing is a procedure to investigate the quality of a software product in different scenarios. It can also be stated as the process of verifying and validating that a software program or application works as expected and meets the business and technical requirements that guided design and development.

Why Software Testing

Software testing is required to point out the defects and errors that were made during different development phases. Software testing also ensures that the product under test works as expected in all different cases – stronger the test suite, stronger is our confidence in the product that we have built. One important benefit of software testing is that it facilitates the developers to make incremental changes to source code and make sure that the current changes are not breaking the functionality of the previously existing code.

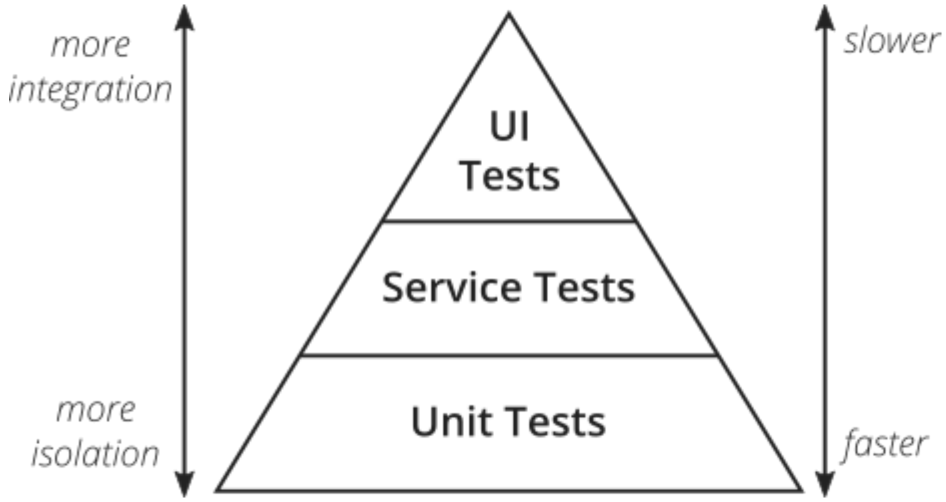
What is TDD

TDD stands for **Test Driven Development**. It is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards. The goal of TDD can be viewed as specification and not validation. In other words, it’s one way to think through your requirements or design before your write your functional code.

What is BDD

BDD stands for “Behaviour Driven Development”. It is a software development process that emerged from TDD. It includes the practice of writing tests first, but focuses on tests which describe behavior, rather than tests which test a unit of implementation. This provides software development and management teams with shared tools and a shared process to collaborate on software development. BDD is largely facilitated through the use of a simple domain-specific language (DSL) using natural language constructs (e.g., English-like sentences) that can express the behavior and the expected outcomes. Mocha and Cucumber testing libraries are built around the concepts of BDD.

5.5.2 🏗 Testing workflow



(Ref: Ham Vocke, The Practical Test Pyramid)

We follow a testing workflow in accordance with the test pyramid diagram given above, starting with isolated tests and moving towards complete integration for any new feature changes. The different types of tests (in the order that they should be performed) are explained below:

Unit Tests

Unit testing is a level of software testing where individual units/ components of a software are tested. The objective of Unit Testing is to isolate a section of code and verify its correctness.

Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, and spies can be used to assist testing a module in isolation.

BENEFITS OF UNIT TESTING

- Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change.
- If codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
- The cost, in terms of time, effort and money, of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

UNIT TESTS IN DTAAS

Each component DTaaS project uses unique technology stack. Thus the packages used for unit tests are different. Please check the `test/` directory of a component to figure out the unit test packages used.

Integration tests

Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. In DTaaS, we use an integration server for software development as well as such tests.

The existing integration tests are done at the component level. There are no integration tests between the components. This task has been postponed to future.

End-to-End tests

Testing any code changes through the end user interface of your software is essential to verify if your code has the desired effect for the user. End-to-End tests in DTaaS a functional setup.

There are end-to-end tests in the DTaaS. This task has been postponed to future.

Feature Tests

A Software feature can be defined as the changes made in the system to add new functionality or modify the existing functionality. Each feature is said to have a characteristics that is designed to be useful, intuitive and effective. It is important to test a new feature when it has been added. We also need to make sure that it does not break the functionality of already existing features. Hence feature tests prove to be useful.

The DTaaS project does not have any feature tests yet. Cucumber shall be used in future to implement feature tests.

5.5.3 References

Justin Searls and Kevin Buchanan, Contributing Tests wiki. This wiki has goog explanation of TDD and test doubles.

5.6 Publish NPM packages

The DTaaS software is developed as a monorepo with multiple npm packages.

5.6.1 Default npm registry

The default registry for npm packages is npmjs. The freely-accessible public packages are published to the **npmjs** registry. The publication step is manual.

```
npm login --registry="https://registry.npmjs.org"
cat ~/.npmrc  #shows the auth token for the registry
//registry.npmjs.org/:_authToken=xxxxxxxxxx
yarn publish --registry="https://registry.npmjs.org" \
  --no-git-tag-version --access public
```

At least one package is published to this registry for each release of DTaaS. This published package is used in the release scripts.

5.6.2 Github npm registry

The Github actions of the project publish packages. The only limitation is that the users need an access token from Github.

5.6.3 Private Registry

Setup private npm registry

Since publishing to npmjs is irrevocable and public, developers are encouraged to setup their own private npm registry for local development. A private npm registry will help with local publish and unpublish steps.

We recommend using verdaccio for this task. The following commands help you create a working private npm registry for development.

```
docker run -d --name verdaccio -p 4873:4873 verdaccio/verdaccio
npm adduser --registry http://localhost:4873 #create a user on the verdaccio registry
npm set registry http://localhost:4873/
yarn config set registry "http://localhost:4873"
yarn login --registry "http://localhost:4873" #login with the credentials for yarn utility
npm login #login with the credentials for npm utility
```

You can open `http://localhost:4873` in your browser, login with the user credentials to see the packages published.

PUBLISH TO PRIVATE NPM REGISTRY

To publish a package to your local registry, do:

```
yarn install
yarn build #the dist/ directory is needed for publishing step
yarn publish --no-git-tag-version #increments version in package.json, publishes to registry
yarn publish #increments version in package.json, publishes to registry and adds a git tag
```

The package version in package.json gets updated as well. You can open `http://localhost:4873` in your browser, login with the user credentials to see the packages published. Please see verdaccio docs for more information.

If there is a need to unpublish a package, ex: `@dtaas/runner@0.0.2`, do:

```
npm unpublish  --registry http://localhost:4873/ @dtaas/runner@0.0.2
```

To install / uninstall this utility for all users, do:

```
sudo npm install  --registry http://localhost:4873 -g @dtaas/runner
sudo npm list -g # should list @dtaas/runner in the packages
sudo npm remove --global @dtaas/runner
```

🚀 Use the packages

The packages available in private npm registry can be used like the regular npm packages installed from npmjs.

For example, to use `@dtaas/runner@0.0.2` package, do:

```
sudo npm install  --registry http://localhost:4873 -g @dtaas/runner
runner # launch the digital twin runner
```

6. Few issues in the Software

6.1 Third-Party Software

- We use third-party software which have certain known issues. Some of the issues are listed below.

6.1.1 ML Workspace

- the docker container slows down a bit after ten days. The only known solution is to restart the docker container. You don't need to restart the complete DTaaS platform, restart of the docker container of ml-workspace is sufficient.
- the terminal tool doesn't seem to have the ability to refresh itself. If there is an issue, the only solution is to close and reopen the terminal from "open tools" drop down of notebook
- terminal app does not show at all after some time: terminal always comes if it is open from drop-down menu of Jupyter Notebook, but not as a direct link.

6.2 Gitlab

- The gilab oauth authorization service does not have a way to sign out of a third-party application. Even if you sign out of DTaaS, the gitlab still shows user as signed in. The next time you click on the sign in button on the DTaaS page, user is not shown the login page. Instead user is directly taken to the **Library** page. So close the brower window after you are done. Another way to overcome this limitation is to open your gitlab instance (`https://gitlab.foo.com`) and signout from there. Thus user needs to sign out of two places, namely DTaaS and gitlab, in order to completely exit the DTaaS application.

7. Contributors

code contributors

7.1 Users

Cláudio Ângelo Gonçalves Gomes, Dmitri Tcherniak, Elif Ecem Bas, Giuseppe Abbiati, Hao Feng, Henrik Ejersbo, Tanusree Roy, Farshid Naseri

7.2 Documentation

1. Talasila, P., Gomes, C., Mikkelsen, P. H., Arboleda, S. G., Kamburjan, E., & Larsen, P. G. (2023). Digital Twin as a Service (DTaaS): A Platform for Digital Twin Developers and Users arXiv preprint arXiv:2305.07244.
2. Astitva Sehgal for developer and example documentation.
3. Tanusree Roy and Farshid Naseri for asking interesting questions that ended up in FAQs.

8. License

8.1 License

--- Start of Definition of INTO-CPS Association Public License ---

/*

- This file is part of the INTO-CPS Association.
- Copyright (c) 2017-CurrentYear, INTO-CPS Association (ICA),
- c/o Peter Gorm Larsen, Aarhus University, Department of Engineering,
- Finlandsgade 22, 8200 Aarhus N, Denmark.
- All rights reserved.
- THIS PROGRAM IS PROVIDED UNDER THE TERMS OF GPL VERSION 3 LICENSE OR
- THIS INTO-CPS ASSOCIATION PUBLIC LICENSE (ICAPL) VERSION 1.0.
- ANY USE, REPRODUCTION OR DISTRIBUTION OF THIS PROGRAM CONSTITUTES
- RECIPIENT'S ACCEPTANCE OF THE INTO-CPS ASSOCIATION PUBLIC LICENSE OR
- THE GPL VERSION 3, ACCORDING TO RECIPIENTS CHOICE.
- The INTO-CPS tool suite software and the INTO-CPS Association
- Public License (ICAPL) are obtained from the INTO-CPS Association, either
- from the above address, from the URLs: http://www.into-cps.org or
- in the INTO-CPS tool suite distribution.
- GNU version 3 is obtained from:
- http://www.gnu.org/copyleft/gpl.html.
- This program is distributed WITHOUT ANY WARRANTY; without
- even the implied warranty of MERCHANTABILITY or FITNESS
- FOR A PARTICULAR PURPOSE, EXCEPT AS EXPRESSLY SET FORTH
- IN THE BY RECIPIENT SELECTED SUBSIDIARY LICENSE CONDITIONS OF
- THE INTO-CPS ASSOCIATION PUBLIC LICENSE.
- See the full ICAPL conditions for more details.

*/

--- End of INTO-CPS Association Public License Header ---

The ICAPL is a public license for the INTO-CPS tool suite with three modes/alternatives (GPL, ICA-Internal-EPL, ICA-External-EPL) for use and redistribution, in source and/or binary/object-code form:

- GPL. Any party (member or non-member of the INTO-CPS Association) may use and redistribute INTO-CPS tool suite under GPL version 3.
- Silver Level members of the INTO-CPS Association may also use and redistribute the INTO-CPS tool suite under ICA-Internal-EPL conditions.
- Gold Level members of the INTO-CPS Association may also use and redistribute The INTO-CPS tool suite under ICA-Internal-EPL or ICA-External-EPL conditions.

Definitions of the INTO-CPS Association Public license modes:

- GPL = GPL version 3.
- ICA-Internal-EPL = These INTO-CPA Association Public license conditions together with Internally restricted EPL, i.e., EPL version 1.0 with the Additional Condition that use and redistribution by a member of the INTO-CPS Association is only allowed within the INTO-CPS Association member's own organization (i.e., its own legal entity), or for a member of the INTO-CPS Association paying a membership fee corresponding to the size of the organization including all its affiliates, use and redistribution is allowed within/between its affiliates.
- ICA-External-EPL = These INTO-CPA Association Public license conditions together with Externally restricted EPL, i.e., EPL version 1.0 with the Additional Condition that use and redistribution by a member of the INTO-CPS Association, or by a Licensed Third Party Distributor having a redistribution agreement with that member, to parties external to the INTO-CPS Association member’s own organization (i.e., its own legal entity) is only allowed in binary/object-code form, except the case of redistribution to other members the INTO-CPS Association to which source is also allowed to be distributed.

[This has the consequence that an external party who wishes to use the INTO-CPS Association in source form together with its own proprietary software in all cases must be a member of the INTO-CPS Association].

In all cases of usage and redistribution by recipients, the following conditions also apply:

a) Redistributions of source code must retain the above copyright notice, all definitions, and conditions. It is sufficient if the ICAPL Header is present in each source file, if the full ICAPL is available in a prominent and easily located place in the redistribution.

b) Redistributions in binary/object-code form must reproduce the above copyright notice, all definitions, and conditions. It is sufficient if the ICAPL Header and the location in the redistribution of the full ICAPL are present in the documentation and/or other materials provided with the redistribution, if the full ICAPL is available in a prominent and easily located place in the redistribution.

c) A recipient must clearly indicate its chosen usage mode of ICAPL, in accompanying documentation and in a text file ICA-USAGE-MODE.txt, provided with the distribution.

d) Contributor(s) making a Contribution to the INTO-CPS Association thereby also makes a Transfer of Contribution Copyright. In return, upon the effective date of the transfer, ICA grants the Contributor(s) a Contribution License of the Contribution. ICA has the right to accept or refuse Contributions.

Definitions:

"Subsidiary license conditions" means:

The additional license conditions depending on the by the recipient chosen mode of ICAPL, defined by GPL version 3.0 for GPL, and by EPL for ICA-Internal-EPL and ICA-External-EPL.

"ICAPL" means:

INTO-CPS Association Public License version 1.0, i.e., the license defined here (the text between "--- Start of Definition of INTO-CPS Association Public License ---" and "--- End of Definition of INTO-CPS Association Public License ---", or later versions thereof.

"ICAPL Header" means:

INTO-CPS Association Public License Header version 1.2, i.e., the text between "--- Start of Definition of INTO-CPS Association Public License ---" and "--- End of INTO-CPS Association Public License Header ---, or later versions thereof.

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under ICAPL, and

b) in the case of each subsequent Contributor: i) changes to the INTO-CPS tool suite, and ii) additions to the INTO-CPS tool suite;

where such changes and/or additions to the INTO-CPS tool suite originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the INTO-CPS tool suite by such Contributor itself or anyone acting on such Contributor's behalf.

For Contributors licensing the INTO-CPS tool suite under ICA-Internal-EPL or ICA-External-EPL conditions, the following conditions also hold:

Contributions do not include additions to the distributed Program which: (i) are separate modules of software distributed in conjunction with the INTO-CPS tool suite under their own license agreement, (ii) are separate modules which are not derivative works of the INTO-CPS tool suite, and (iii) are separate modules of software distributed in conjunction with the INTO-CPS tool suite under their own license agreement where these separate modules are merged with (weaved together with) modules of The INTO-CPS tool suite to form new modules that are distributed as object code or source code under their own license agreement, as allowed under the Additional Condition of internal distribution according to ICA-Internal-EPL and/or Additional Condition for external distribution according to ICA-External-EPL.

"Transfer of Contribution Copyright" means that the Contributors of a Contribution transfer the ownership and the copyright of the Contribution to the INTO-CPS Association, the INTO-CPS Association Copyright owner, for inclusion in the INTO-CPS tool suite. The transfer takes place upon the effective date when the Contribution is made available on the INTO-CPS Association web site under ICAPL, by such Contributors themselves or anyone acting on such Contributors' behalf. The transfer is free of charge. If the Contributors or the INTO-CPS Association so wish, an optional Copyright transfer agreement can be signed between the INTO-CPS Association and the Contributors.

"Contribution License" means a license from the INTO-CPS Association to the Contributors of the Contribution, effective on the date of the Transfer of Contribution Copyright, where the INTO-CPS Association grants the Contributors a non-exclusive, world-wide, transferable, free of charge, perpetual license, including sublicensing rights, to use, have used, modify, have modified, reproduce and or have reproduced the contributed material, for business and other purposes, including but not limited to evaluation, development, testing, integration and merging with other software and distribution. The warranty and liability disclaimers of ICAPL apply to this license.

"Contributor" means any person or entity that distributes (part of) the INTO-CPS tool chain.

"The Program" means the Contributions distributed in accordance with ICAPL.

"The INTO-CPS tool chain" means the Contributions distributed in accordance with ICAPL.

"Recipient" means anyone who receives the INTO-CPS tool chain under ICAPL, including all Contributors.

"Licensed Third Party Distributor" means a reseller/distributor having signed a redistribution/resale agreement in accordance with ICAPL and the INTO-CPS Association Bylaws, with a Gold Level organizational member which is not an Affiliate of the reseller/distributor, for distributing a product containing part(s) of the INTO-CPS tool suite. The Licensed Third Party Distributor shall only be allowed further redistribution to other resellers if the Gold Level member is granting such a right to it in the redistribution/resale agreement between the Gold Level member and the Licensed Third Party Distributor.

"Affiliate" shall mean any legal entity, directly or indirectly, through one or more intermediaries, controlling or controlled by or under common control with any other legal entity, as the case may be. For purposes of this definition, the term "control" (including the terms "controlling," "controlled by" and "under common control with") means the possession, direct or indirect, of the power to direct or cause the direction of the management and policies of a legal entity, whether through the ownership of voting securities, by contract or otherwise.

NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THE BY RECIPIENT SELECTED SUBSIDIARY LICENSE CONDITIONS OF ICAPL, THE INTO-CPS ASSOCIATION IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the INTO-CPS tool suite and assumes all risks associated with its exercise of rights under ICAPL , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THE BY RECIPIENT SELECTED SUBSIDIARY LICENSE CONDITIONS OF ICAPL, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE INTO-CPS TOOL SUITE OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

A Contributor licensing the INTO-CPS tool suite under ICA-Internal-EPL or ICA-External-EPL may choose to distribute (parts of) the INTO-CPS tool suite in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of ICAPL; or for the case of redistribution of the INTO-CPS tool suite together with proprietary code it is a dual license where the INTO-CPS tool suite parts are distributed under ICAPL compatible conditions and the proprietary code is distributed under proprietary license conditions; and

b) its license agreement: i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose; ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits; iii) states that any provisions which differ from ICAPL are offered by that Contributor alone and not by any other party; and iv) states from where the source code for the INTO-CPS tool suite is available, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the INTO-CPS tool suite is made available in source code form:

a) it must be made available under ICAPL; and

b) a copy of ICAPL must be included with each copy of the INTO-CPS tool suite.

c) a copy of the subsidiary license associated with the selected mode of ICAPL must be included with each copy of the INTO-CPS tool suite.

Contributors may not remove or alter any copyright notices contained within The INTO-CPS tool suite.

If there is a conflict between ICAPL and the subsidiary license conditions, ICAPL has priority.

This Agreement is governed by the laws of Denmark. The place of jurisdiction for all disagreements related to this Agreement, is Aarhus, Denmark.

The EPL 1.0 license definition has been obtained from: http://www.eclipse.org/legal/epl-v10.html. It is also reproduced in the INTO-CPS distribution.

The GPL Version 3 license definition has been obtained from http://www.gnu.org/copyleft/gpl.html. It is also reproduced in the INTO-CPS distribution.

--- End of Definition of INTO-CPS Association Public License ---

8.2 Third Party Software

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

8.2.1 User Installations

The list of software included with DTaaS installation scripts are:

Software Package	Usage	License
docker v24.0	mandatory	Apache 2.0
ml-workspace-minimal v0.13	mandatory	Apache 2.0
nodejs v20.10	mandatory	Custom - Modified MIT
npm v10.2	mandatory	Artistic License 2.0
serve	mandatory	MIT
Træfik v2.10	mandatory	MIT
yarn v1.22	mandatory	BSD 2-Clause
eclipse-mosquitto v2	optional	Eclipse Public License-2.0
gitlab-ce v16.4	optional	MIT
Grafana v10.1	optional	GNU Affero General Public (AGPL) License v3.0
InfluxDB v2.7	optional	Apache2, MIT
Mongodb v7.0	optional	AGPL License and Server Side Public License (SSPL) v1
Tabbitmq v3-management	optional	Mozilla Public License
Telegraf v1.28	optional	MIT

8.2.2 Development Environments

Inaddition to all the software included in user installations, the DTaaS development environments may use the following additional software packages.

Software Package	Usage	License
Material for mkdocs	mandatory	MIT
Docker-compose v2.20	optional	Apache 2.0
Jupyter Lab	optional	3-Clause BSD
Microk8s v1.27	optional	Apache 2.0
Openssl	optional	Custom License

8.2.3 Package Dependencies

There are specific software packages included in the development of client, library microservice and runner microservice. These packages can be seen in the **package.json** file of the matching directories.

The plugins of *material for mkdocs* might have their own licenses. The list of plugins used are in **requirements.txt** file.