



# INTO-CPS

## **Digital Twin as a Service (DTaaS)**

---

*DTaaS Development Team*

*Copyright © 2022 - 2023 The INTO-CPS Association*

## Table of contents

---

1. Home	3
1.1 License	3
2. Admin	4
2.1 Host Install	4
2.2 Vagrant	7
2.3 Separate Packages	11
3. User	15
3.1 Features	15
3.2 Website	17
3.3 Library	26
3.4 Digital Twins	33
4. FAQ	38
4.1 Abbreviations	38
4.2 General Questions	38
4.3 Digital Twin Models	39
4.4 Communication Between Physical Twin and Digital Twin	39
4.5 Data Management	40
4.6 Platform Native Services on DTaaS Platform	41
5. Bugs	42
5.1 Some limitations	42
5.2 Third-Party Software	42
5.3 Gitlab	42
6. Thanks	43
6.1 Contributors	43
6.2 Users	43
6.3 Documentation	43

# 1. Home

---

The Digital Twin as a Service (DTaaS) software is useful to create and run digital twins. The digital twins that are running can be used as service by other users. These users need not be members of the DTaaS software platform itself.

There is an overview of the software available for:

- General users - [overview slides](#) and [overview video](#), [feature walkthrough](#)
- Developers - [slides](#) and [video](#).

There is also a [research paper draft](#) if you are interested in reading the scientific roadmap for this software.

## 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 [Traefik](#), [ML Workspace](#), [Grafana](#), [InfluxDB](#) and [RabbitMQ](#) open-source components. These software components have their own licenses.

## 2. Admin

---

### 2.1 Host Install

---

#### 2.1.1 Trial Installation

---

A single step install script is helpful in performing a trial run of the software. This script installs DTaaS software with default credentials and users on a Ubuntu server OS. You can use it to check a test installation of DTaaS.

```
1 wget https://github.com/INTO-CPS-Association/DTaaS/releases/download/v0.2.0/single-script-install.sh
2 bash single-script-install.sh
```



This test installation has default credentials and is thus highly insecure.

## 2.1.2 DTaaS on Linux Operating System

These are installation instructions for running DTaaS application on a Ubuntu Server 20.04 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.

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**.



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 usernames. The traefik gateway configuration has a template for two users. You can modify the usernames in the template to the usernames chosen by you.

#### THE TRAEFIK GATEWAY SERVER

You can run the Run the Traefik gateway server in both and HTTPS 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.



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

#### Authentication

The dummy username is `foo` and the password is `bar`. Please change this before starting the gateway.

```
1 rm deploy/config/gateway/auth
2 touch deploy/config/gateway/auth
3 httpasswd deploy/config/gateway/auth <first_username>
4 password: <your password>
```

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

### Configure lib microservice

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

Update the *deploy/config/lib* of the library microservice. The simplest possibility is to use `local` mode with the following example. The filepath is the absolute filepath to `files/` directory.

```
1 PORT='4001'
2 MODE='local'
3 LOCAL_PATH='filepath'
4 LOG_LEVEL='debug'
5 APOLLO_PATH='/Lib'
6 GRAPHQL_PLAYGROUND='true'
```

### Configure react website

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

```
1  window.env = {  
2    REACT_APP_ENVIRONMENT: 'prod',  
3    REACT_APP_URL: 'https://foo.com/',  
4    REACT_APP_URL_BASENAME: '',  
5    REACT_APP_URL_DTLINK: '/lab',  
6    REACT_APP_URL_LIINK: '',  
7    REACT_APP_WORKBENCHLINK_TERMINAL: '/terminals/main',  
8    REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',  
9    REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',  
10   REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/lab',  
11   REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',  
12  };
```

### 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

```
1  source deploy/install.sh
```

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

### Access the application

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

## 2.2 Vagrant

### 2.2.1 Vagrant Box for DTaaS

There are some good vagrant boxes on [vagrant website](#). But these boxes require too many installations that take a long time and network bandwidth. So it is efficient to create one local vagrant box for DTaaS application and reuse the same in all installations.

#### Installed Software

This base DTaaS vagrant box, when it is successfully created, has the following software:

- docker
- nodejs and yarn
- jupyter
- microk8s
- containers
- mltooling/ml-workspace:0.13.2
- traefik2.5
- influxdb2.4
- grafana
- telegraf
- gitlab

#### Create the vagrant box

Publish a base virtualbox package to be used by vagrant to publish all other virtualbox packages

```

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

## 2.2.2 DTaaS on Single Vagrant Machine

---

These are installation instructions for running DTaaS application 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.

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

Please follow these steps to make this work in your local environment.

1. Create **dtaas Vagrant box**. Copy *vagrant* SSH private key into *deploy/vagrant/single-machine*. This shall be useful for logging into the vagrant machine created for single-machine deployment. 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.
2. Update the **Vagrantfile**. Fields to update are:
  - a. Hostname ( `node.vm.hostname = "foo.com"` )
  - b. MAC address ( `:mac => "xxxxxxx"` ). This change is required if you have a DHCP server assigning domain names based on MAC address. Otherwise, you can leave this field unchanged.
  - c. Other adjustments are optional.
3. Execute the following commands from terminal

```
1  vagrant up
2  vagrant ssh
```

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

```
1  wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.2/deploy/vagrant/route.sh
2  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.



## 2.2.3 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 and RabbitMQ 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/single-machine` ). This shall be useful for logging into the vagrant machines created for two-machine deployment.

### 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"` )
2. `hostname` - hostname of server1 ( `"name" = "foo.com"` )
3. MAC address ( `:mac => "xxxxxxx"` ). 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 => "xxxxxxx"` ). 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.

### Launch platform default services

RabbitMQ, Grafana and InfluxDB services are provisioned on this server. InfluxDB webUI will be available at: *services.foo.com*. The RabbitMQ service and its management interface shall be available at 5672 and 15672 TCP ports respectively. The Grafana service shall be available at TCP port 3000.

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.

Execute the following commands from terminal to start the machine.

```
1 vagrant up --provision services
2 vagrant ssh services
3 wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.2/deploy/vagrant/two-machine/services.sh
4 bash services.sh
5 wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.2/deploy/vagrant/route.sh
6 sudo bash route.sh
```

After the server is up and running, you can see the following services active within server2.

service	external url
Influx visualization service	services.foo.com
Grafana visualization service	services.foo.com:3000
RabbitMQ communication service	services.foo.com:5672
RabbitMQ management service	services.foo.com:15672

### Launch DTaaS application

Execute the following commands from terminal

```
1 vagrant up --provision dtaas
2 vagrant ssh dtaas
3 wget https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/release-v0.2/deploy/vagrant/route.sh
4 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.

## 2.3 Separate Packages

### 2.3.1 Host the DTaaS Client Website

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.
- Locate the file `site/env.js` and replace the example values to match your infrastructure. See the definitions below:

```

1  window.env = {
2    REACT_APP_ENVIRONMENT: "prod | dev",
3    REACT_APP_URL: "URL for the gateway",
4    REACT_APP_URL_BASENAME: "Base URL for the client website"(optional),
5    REACT_APP_URL_DTLINK: "Endpoint for the Digital Twin",
6    REACT_APP_URL_LTLINK: "Endpoint for the Library Assets",
7    REACT_APP_WORKBENCHLINK_TERMINAL: "Endpoint for the terminal link",
8    REACT_APP_WORKBENCHLINK_VNCDESKTOP: "Endpoint for the VNC Desktop link",
9    REACT_APP_WORKBENCHLINK_VSCODE: "Endpoint for the VS Code link",
10   REACT_APP_WORKBENCHLINK_JUPYTERLAB: "Endpoint for the Jupyter Lab link",
11   REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK:
12     "Endpoint for the Jupyter Notebook link",
13 };
14
15 // Example values with no base URL. Trailing and ending slashes are optional.
16 window.env = {
17   REACT_APP_ENVIRONMENT: 'prod',
18   REACT_APP_URL: 'https://foo.com/',
19   REACT_APP_URL_BASENAME: '',
20   REACT_APP_URL_DTLINK: '/Lab',
21   REACT_APP_URL_LTLINK: '',
22   REACT_APP_WORKBENCHLINK_TERMINAL: '/terminals/main',
23   REACT_APP_WORKBENCHLINK_VNCDESKTOP: '/tools/vnc/?password=vncpassword',
24   REACT_APP_WORKBENCHLINK_VSCODE: '/tools/vscode/',
25   REACT_APP_WORKBENCHLINK_JUPYTERLAB: '/Lab',
26   REACT_APP_WORKBENCHLINK_JUPYTERNOTEBOOK: '',
27 };

```

- 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 at least one **ML Workspace** serving the following routes.

```

1  https://foo.com/<username>/Lab
2  https://foo.com/<username>/terminals/main
3  https://foo.com/<username>/tools/vnc/?password=vncpassword
4  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.

```

1  mkdir -p files/alice files/bob files/common
2
3  printf "\n\n start the user workspaces"
4  docker run -d \
5  -p 8090:8080 \
6  --name "ml-workspace-alice" \
7  -v "$(pwd)/files/alice:/workspace" \
8  -v "$(pwd)/files/common:/workspace/common" \
9  --env AUTHENTICATE_VIA_JUPYTER="" \
10 --env WORKSPACE_BASE_URL="alice" \
11 --shm-size 512m \
12 --restart always \
13 mltooling/ml-workspace:0.13.2
14
15
16
17 docker run -d \
18 -p 8091:8080 \
19 --name "ml-workspace-bob" \
20 -v "$(pwd)/files/bob:/workspace" \
21 -v "$(pwd)/files/common:/workspace/common" \
22 --env AUTHENTICATE_VIA_JUPYTER="" \
23 --env WORKSPACE_BASE_URL="bob" \
24 --shm-size 512m \
25 --restart always \
26 mltooling/ml-workspace: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.

## 2.3.2 Host Library Microservice

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

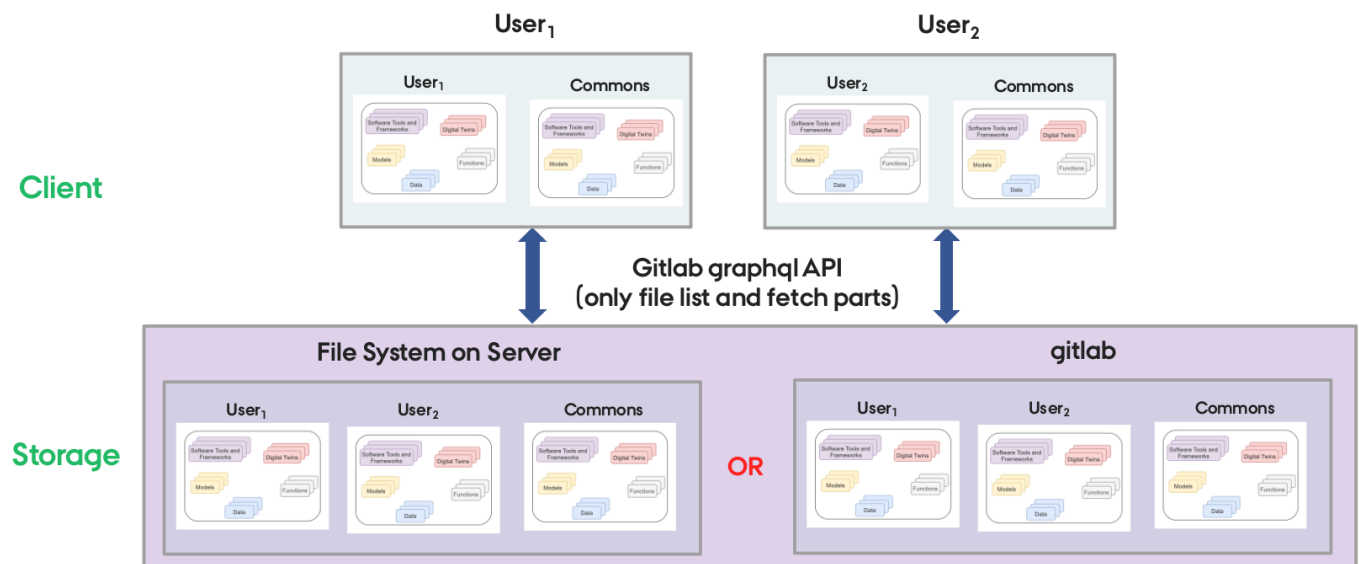
- provide a listing of directory contents.
- transfer a file to user.
- Source files can either come from local file system or from a gitlab instance.

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.

### Gitlab setup (optional)

For this microservice to be functional, a certain directory or gitlab project structure is expected. The microservice expects that the gitlab consisting of one group, DTaaS, and within that group, all of the projects be located, **user1**, **user2**, ... , as well as a **commons** project. Each project corresponds to files of one user. A sample file structure can be seen in [gitlab dtaas group](#). You can visit the gitlab documentation on [groups](#) for help on the management of gitlab groups.

You can clone the git repositories from the [dtaas](#) group to get a sample file system structure for the lib microservice.

### Setup Microservice

To set up the lib microservice, follow these steps:

Download the **lib-microservice.zip** from the [releases page](#).

### Configuration setup

The microservices uses `.env` environment files to receive configuration.

To set up the environment variables for the lib microservice, create a new file named `.env` in the `lib-ms` directory. Then, add the following variables and their respective values. Below you can see and how, with included examples:

```
1  PORT='4001'
2  MODE='local' or 'gitlab'
3  LOCAL_PATH='/Users/<Username>/DTaaS/files'
4  GITLAB_GROUP='dtaas'
5  GITLAB_URL='https://gitlab.com/api/graphql'
6  TOKEN='123-sample-token'
7  LOG_LEVEL='debug'
8  TEST_PATH='/Users/<Username>/DTaaS/servers/lib/test/data/test_assets'
9  APOLLO_PATH='/lib' or ''
10 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.

The `GITLAB_URL`, `GITLAB_GROUP` and `TOKEN` are only relevant for `gitlab` mode. The `TOKEN` should be set to your GitLab Group access API token. For more information on how to create and use your access token, [gitlab page](#).

Once you've generated a token, copy it and replace the value of `TOKEN` with your token for the gitlab group, can be found.

Replace the default values the appropriate values for your setup.

#### NOTE:

1. When `MODE=local`, only `LOCAL_PATH` is used. Other environment variables are unused.
2. When `MODE=gitlab`, `GITLAB_URL`, `TOKEN`, and `GITLAB_GROUP` are used; `LOCAL_PATH` is unused.

#### START MICROSERVICE

```
1  yarn install # Install dependencies for the microservice
2  yarn build   # build the application
3  yarn start   # start the application
```

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

```
1  nohup yarn start & disown
```

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

Users can access the library microservice at URL: `http://localhost:<PORT>/lib`.

#### Developer Commands

```
1  yarn install # Install dependencies for the microservice
2  yarn syntax  # analyzes source code for potential errors, style violations, and other issues,
3  yarn build   # compile ES6 files into ES5 javascript files and copy all JS files into build/ directory
4  yarn test -a # run all tests
5  yarn test -e # run end-to-end tests
6  yarn test -i # run integration tests
7  yarn test -u # run unit tests
8  yarn start   # start the application
9  yarn clean   # deletes directories "build", "coverage", and "dist"
```

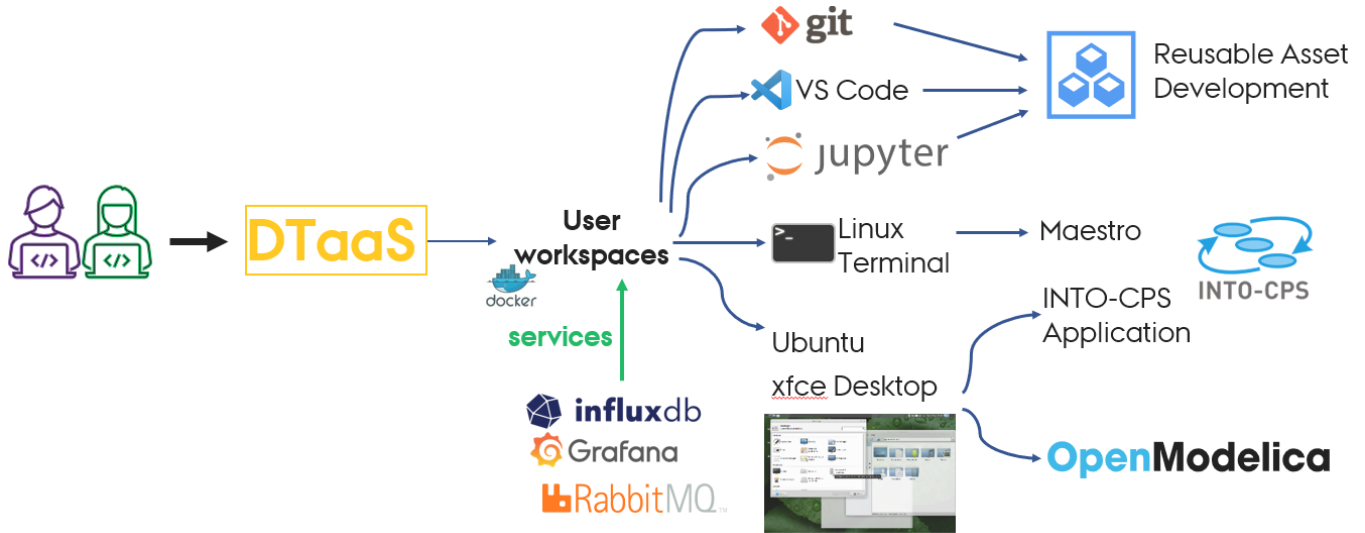
#### Service Endpoint

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

## 3. User

### 3.1 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 primarily 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	Users can create visualization dashboards for their digital twins.

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

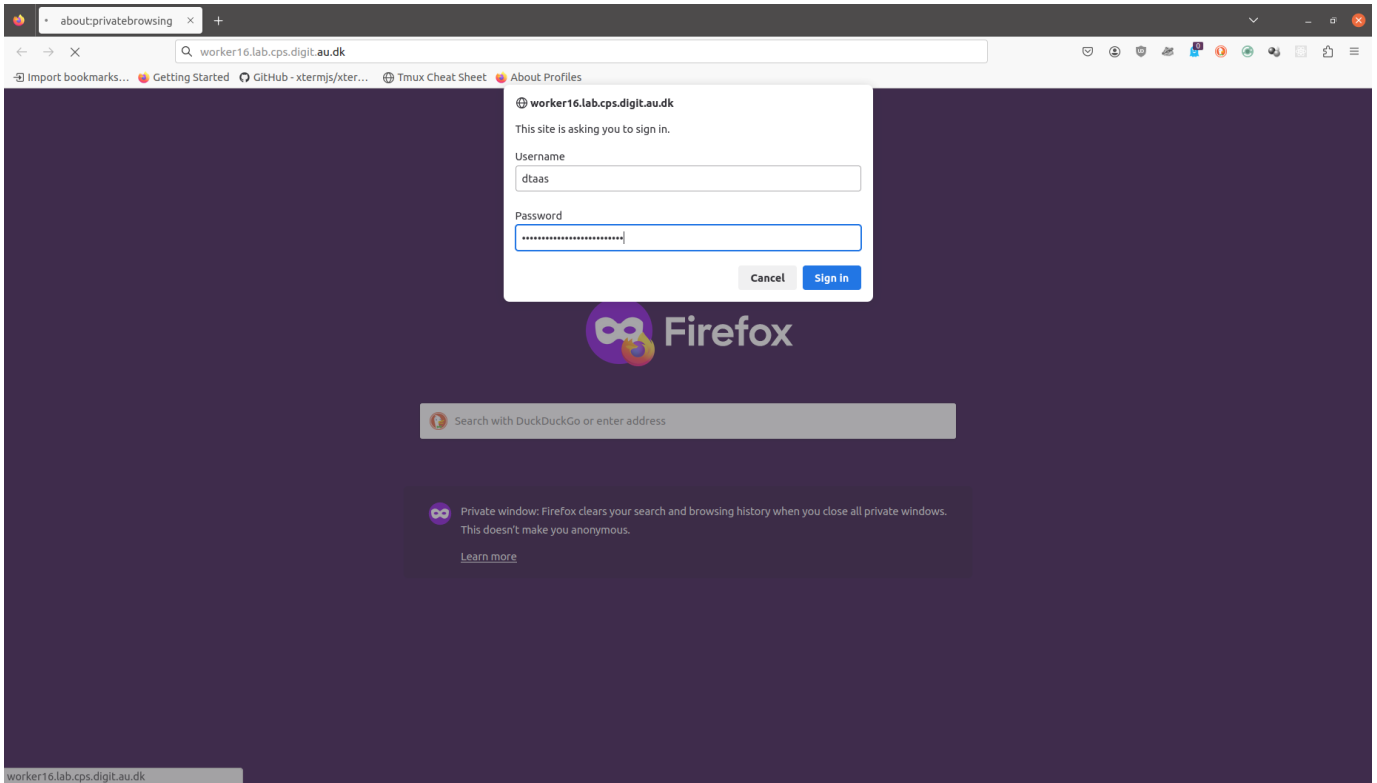
The users can publish and reuse the digital twin assets available on the platform. In addition, users can run their digital twins and make these live digital twins available as services to their clients. The clients need not be users of the DTaaS software installation.



## 3.2 Website

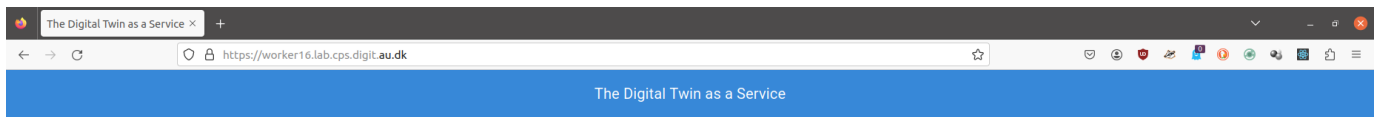
This page contains a screenshot driven preview of the website.

### Login to enter the DTaaS software platform



The screen presents with HTTP authentication form. You can enter the user credentials. You will be using HTTPS secure communication so the username and password are secure.

### Enter username again




Copyright © The INTO-CPS Association 2023.

Thanks to Material-UI for the [Dashboard template](#), which was the basis for our React app.

You are now logged into the server. You can enter the same username again to log into your workspace.

### 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.

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.

**Workspace**

Files | Running | IPython Clusters | Nbextensions

Select items to perform actions on them.

	Name	Last Modified	File size
<input type="checkbox"/>	/		
<input type="checkbox"/>	common	a month ago	
<input type="checkbox"/>	data	22 minutes ago	
<input type="checkbox"/>	digital twins	22 minutes ago	
<input type="checkbox"/>	functions	22 minutes ago	
<input type="checkbox"/>	models	22 minutes ago	
<input type="checkbox"/>	tools	22 minutes ago	
<input type="checkbox"/>	user2	22 minutes ago	

Copyright © The INTO-CPS Association 2023.  
Thanks to Material-UI for the [Dashboard template](#), which was the basis for our React app.

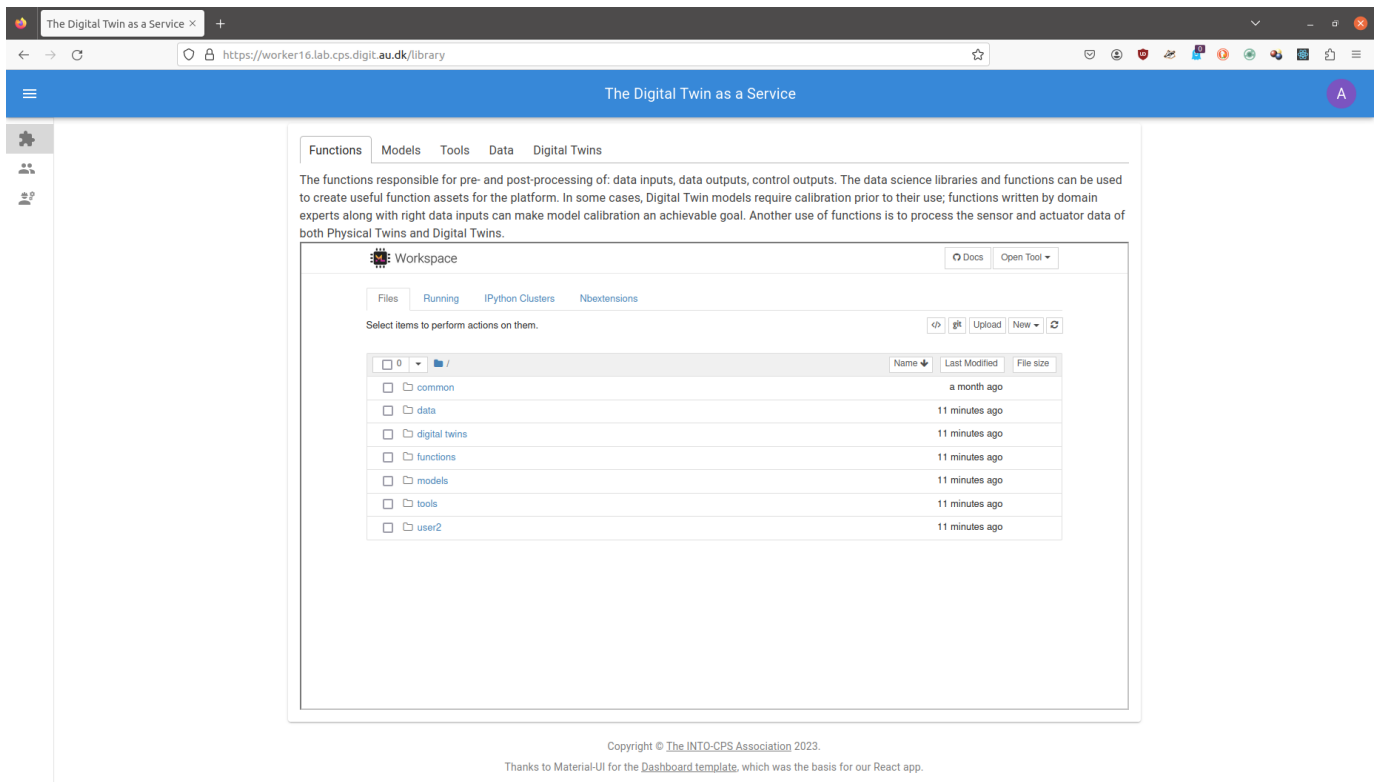
There are three menu items:

**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 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.

## 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:

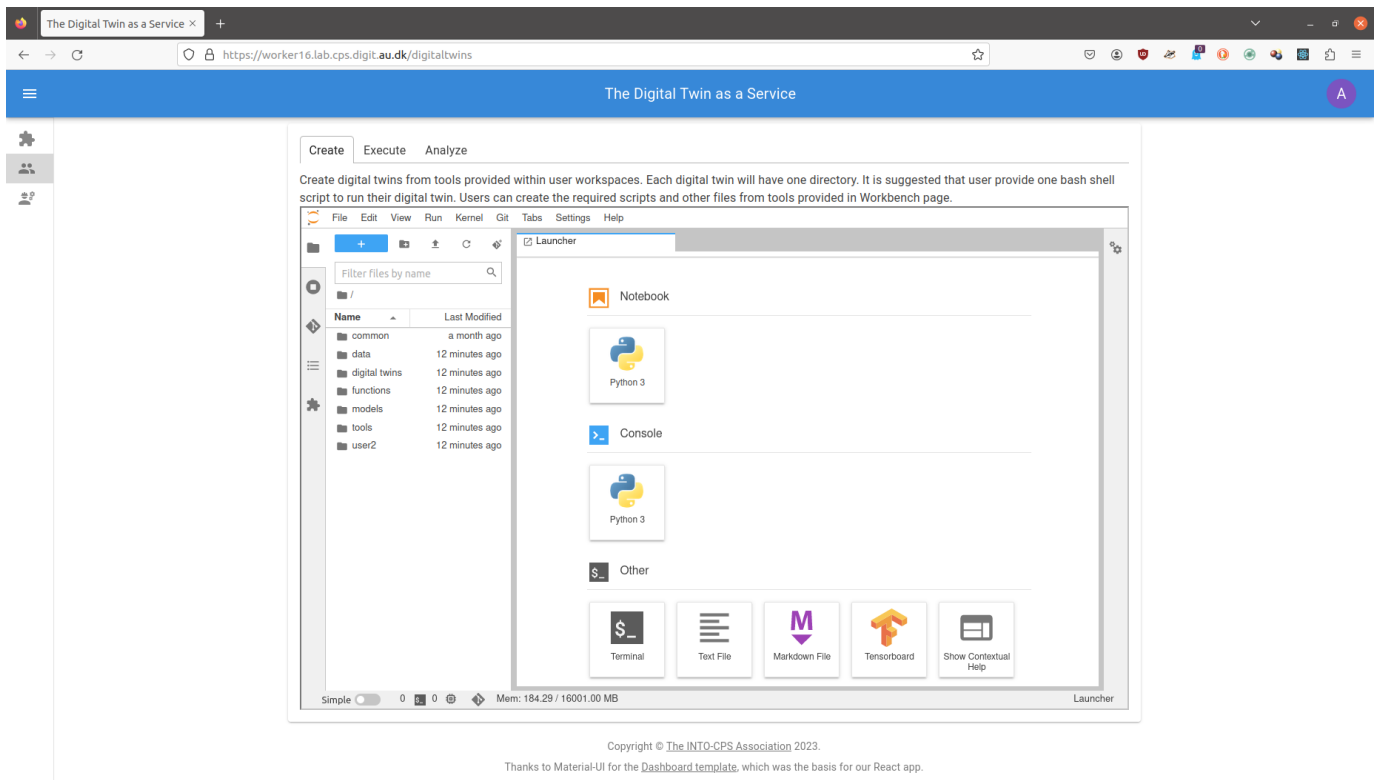
- data
- functions
- models
- tools
- digital twins

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.

## 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.

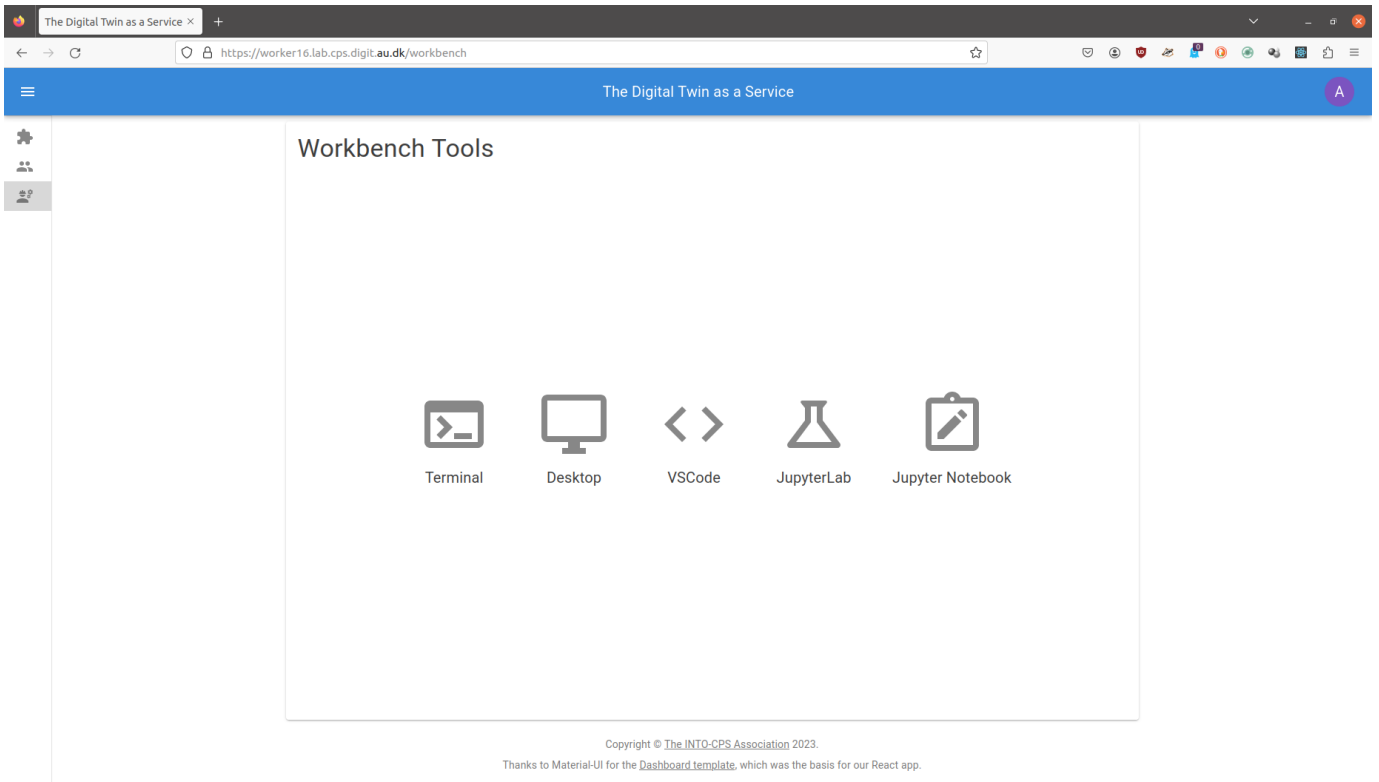
### Analyze

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.

**i** 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.

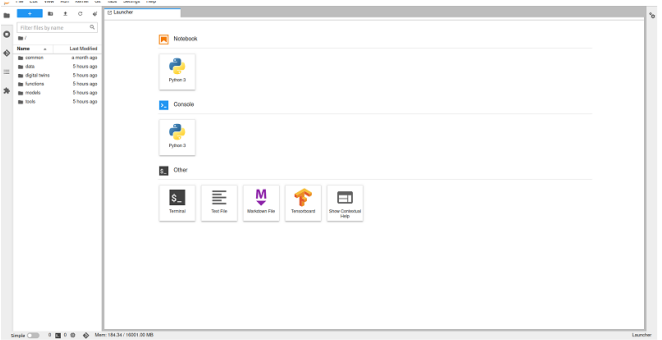
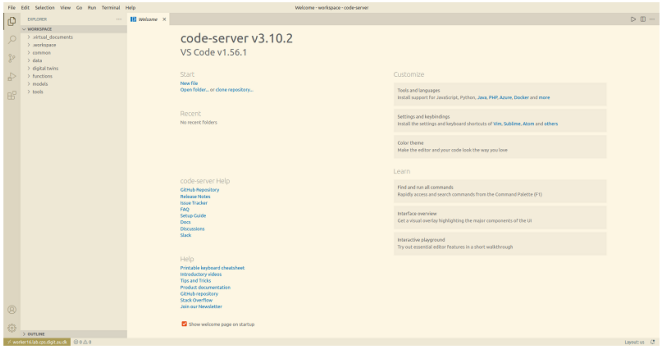
## Workbench

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:

```
(base)
/workspace
└─ ls
    common data 'digital twins' functions models tools
(base)
/workspace
└─
```

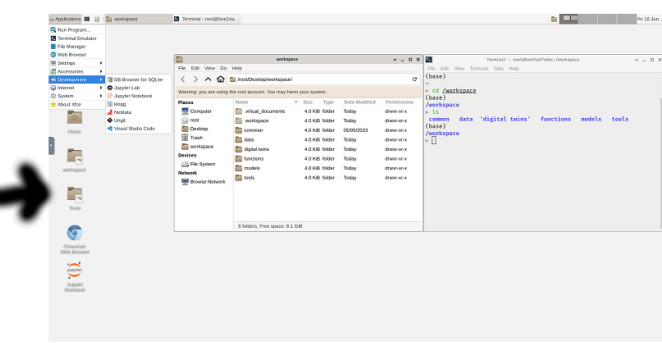


Workspace

Files Running IPython Clusters Nbextensions

Select items to perform actions on them.

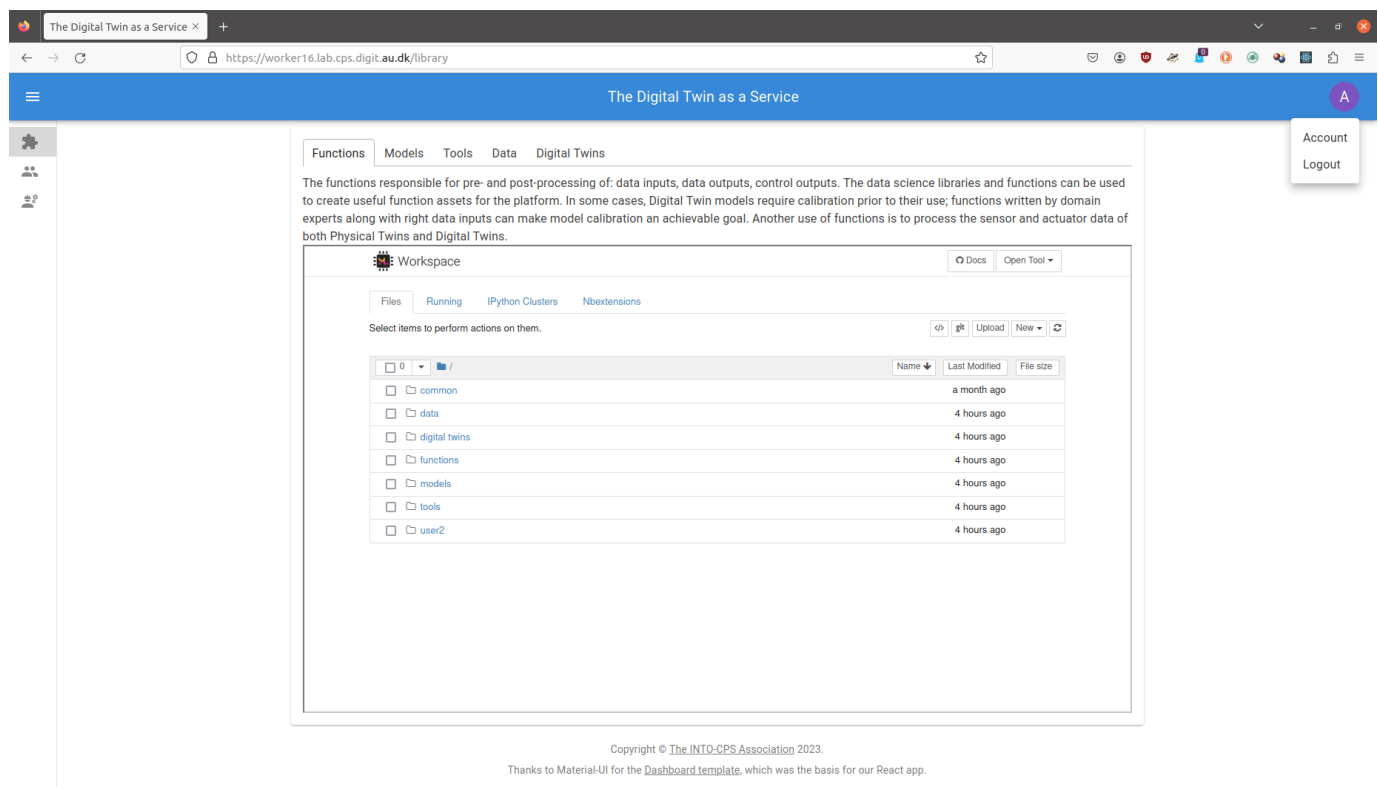
	Name	Last Modified	File size
<input type="checkbox"/>	common	a month ago	
<input type="checkbox"/>	data	23 days ago	
<input type="checkbox"/>	digital twins	12 days ago	
<input type="checkbox"/>	functions	a month ago	
<input type="checkbox"/>	models	23 days ago	
<input type="checkbox"/>	tools	a month ago	
<input type="checkbox"/>	prasad tar.gz	11 days ago	56.9 MB



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



Finally logout



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

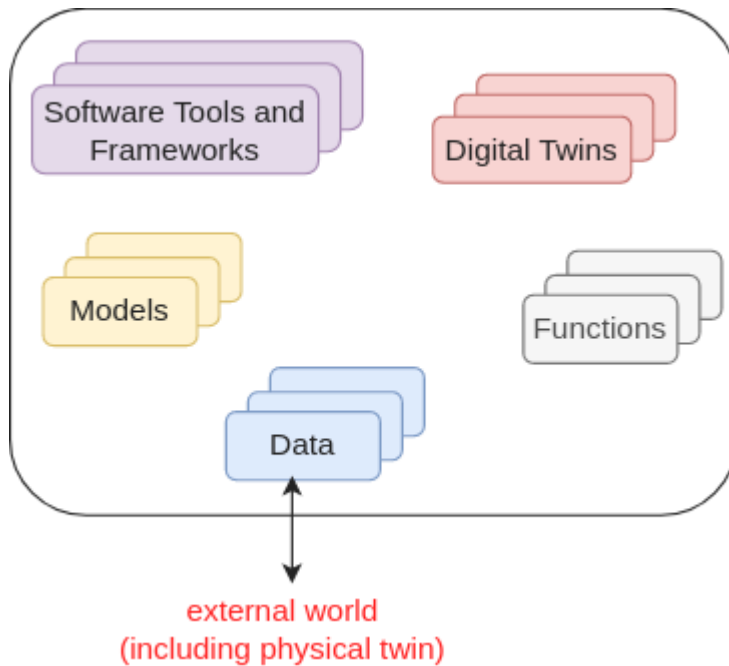
## 3.3 Library

### 3.3.1 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 five categories:



#### 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 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:

```

1  workspace/
2    data/
3      data1/ (ex: sensor)
4        filename (ex: sensor.csv)
5        README.md
6      data2/ (ex: turbine)
7        README.md (remote source; no local file)
8      ...
9    digital twins/
10     digital twin-1/ (ex: incubator)
11       code and config
12       README.md (usage instructions)
13     digital twin-2/ (ex: mass spring damper)
14       code and config
15       README.md (usage instructions)
16     digital twin-3/ (ex: model swap)
17       code and config
18       README.md (usage instructions)
19     ...
20   functions/
21     function1/ (ex: graphs)
22       filename (ex: graphs.py)
23       README.md
24     function2/ (ex: statistics)
25       filename (ex: statistics.py)
26       README.md
27     ...
28   models/
29     model1/ (ex: spring)
30       filename (ex: spring.fmu)
31       README.md
32     model2/ (ex: building)
33       filename (ex: building.skp)
34       README.md
35     model3/ (ex: rabbitmq)
36       filename (ex: rabbitmq.fmu)
37       README.md
38     ...
39   tools/
40     tool1/ (ex: maestro)
41       filename (ex: maestro.jar)
42       README.md
43     ...
44   common/
45     data/
46     functions/
47     models/
48     tools/

```

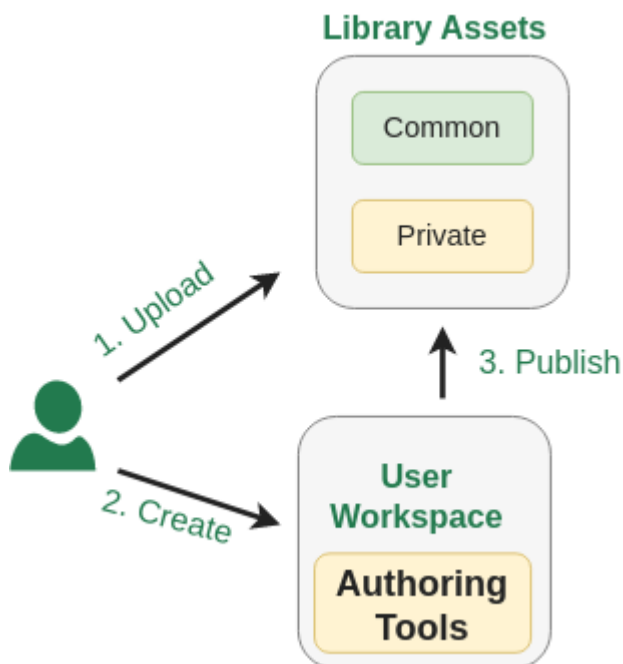


The DTaaS is agnostic to the format of your assets. The only requirement is that they are files which can be uploaded on the Library page. Any directories can be compressed as one file and uploaded. You can decompress the file into a directory from a Terminal or xfce Desktop available on the Workbench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

A recommended file system structure for storing assets is also available in [DTaaS examples](#).

## Create Assets

The DTaaS software allows users to create new library assets on the platform.



Users can 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.

## Upload Assets

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

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.

**Workspace** Docs Open Tool

Files Running IPython Clusters Nbextensions

Select items to perform actions on them.

<input type="checkbox"/>	Name	Last Modified	File size
<input type="checkbox"/>	common	a month ago	
<input type="checkbox"/>	data	11 minutes ago	
<input type="checkbox"/>	digital twins	11 minutes ago	
<input type="checkbox"/>	functions	11 minutes ago	
<input type="checkbox"/>	models	11 minutes ago	
<input type="checkbox"/>	tools	11 minutes ago	
<input type="checkbox"/>	user2	11 minutes ago	

Copyright © The INTO-CPS Association 2023.  
Thanks to Material-UI for the [Dashboard template](#), which was the basis for our React app.

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.

### 3.3.2 Library Microservice

---

**i** 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.

#### The GraphQL Queries

The library microservice services two graphql requests:

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

The format of the accepted queries are:

PROVIDE LIST OF CONTENTS FOR A DIRECTORY

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

#### GraphQL Query      GraphQL Response

```

1  query {
2    listDirectory(path: "user1") {
3      repository {
4        tree {
5          blobs {
6            edges {
7              node {
8                name
9                type
10             }
11           }
12         }
13       }
14     }
15   }
16 }

```

```

1  {
2    "data": {
3      "listDirectory": {
4        "repository": {
5          "tree": {
6            "blobs": {
7              "edges": []
8            },
9            "trees": {
10             "edges": [
11               {
12                 "node": {
13                   "name": "common",
14                   "type": "tree"
15                 }
16               },
17               {
18                 "node": {
19                   "name": "data",
20                   "type": "tree"
21                 }
22               },
23               {
24                 "node": {
25                   "name": "digital twins",
26                   "type": "tree"
27                 }
28               },
29               {
30                 "node": {
31                   "name": "functions",
32                   "type": "tree"
33                 }
34               },
35               {
36                 "node": {
37                   "name": "models",
38                   "type": "tree"
39                 }
40               },
41               {
42                 "node": {
43                   "name": "tools",
44                   "type": "tree"
45                 }
46               }
47             ]
48           }
49         }
50       }
51     }
52   }
53 }

```

## FETCH A FILE FROM THE AVAILABLE FILES

## GraphQL Request      GraphQL Response

```

1  query {
2    readFile(path: "user2/data/sample.txt") {
3      repository {
4        blobs {
5          nodes {
6            name
7            rawBlob
8            rawTextBlob
9          }
10         }
11       }
12     }
13   }

```

```

1  {
2    "data": {
3      "readFile": {
4        "repository": {
5          "blobs": {
6            "nodes": [
7              {
8                "name": "sample.txt",
9                "rawBlob": "hello world",
10               "rawTextBlob": "hello world"
11             }
12           ]
13         }
14       }
15     }
16   }
17 }

```

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.

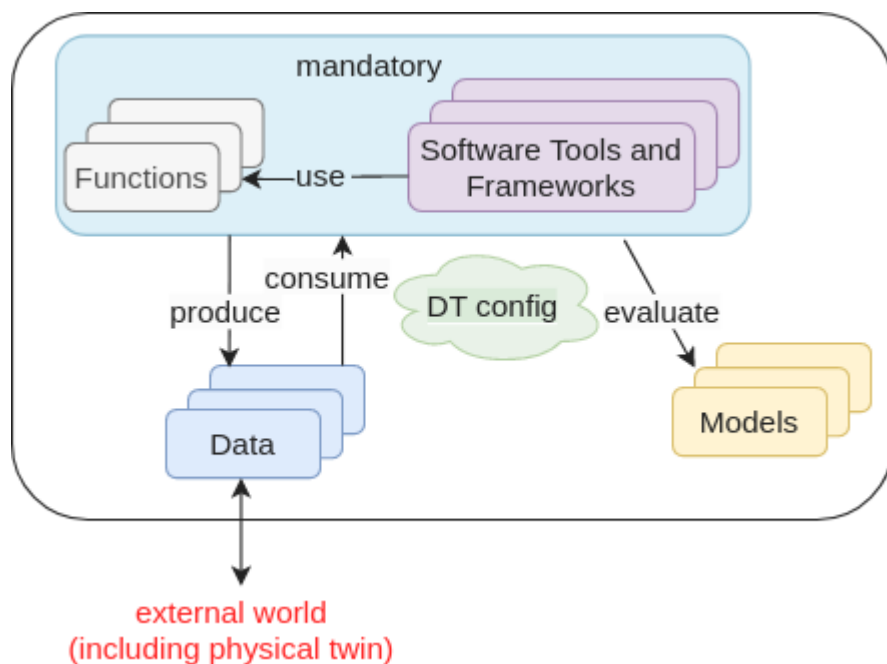


## 3.4 Digital Twins

### 3.4.1 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.



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. The complete details on this example are available on [github](#).

This example illustrates the potential of using co-simulation for digital twins.

The file system contents for this example are:

```

1  workspace/
2    data/
3      mass-spring-damper
4        input/
5        output/
6
7    digital twins/
8      mass-spring-damper/
9        cosim.json
10       time.json
11       lifecycle/
12         analyze
13         clean
14         evolve
15         execute
16         save
17         terminate
18       README.md
19
20     functions/
21     models/
22       MassSpringDamper1.fmu
23       MassSpringDamper2.fmu
24
25     tools/
26
27     common/
28       data/
29       functions/
30       models/
31       tools/
32       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](#)

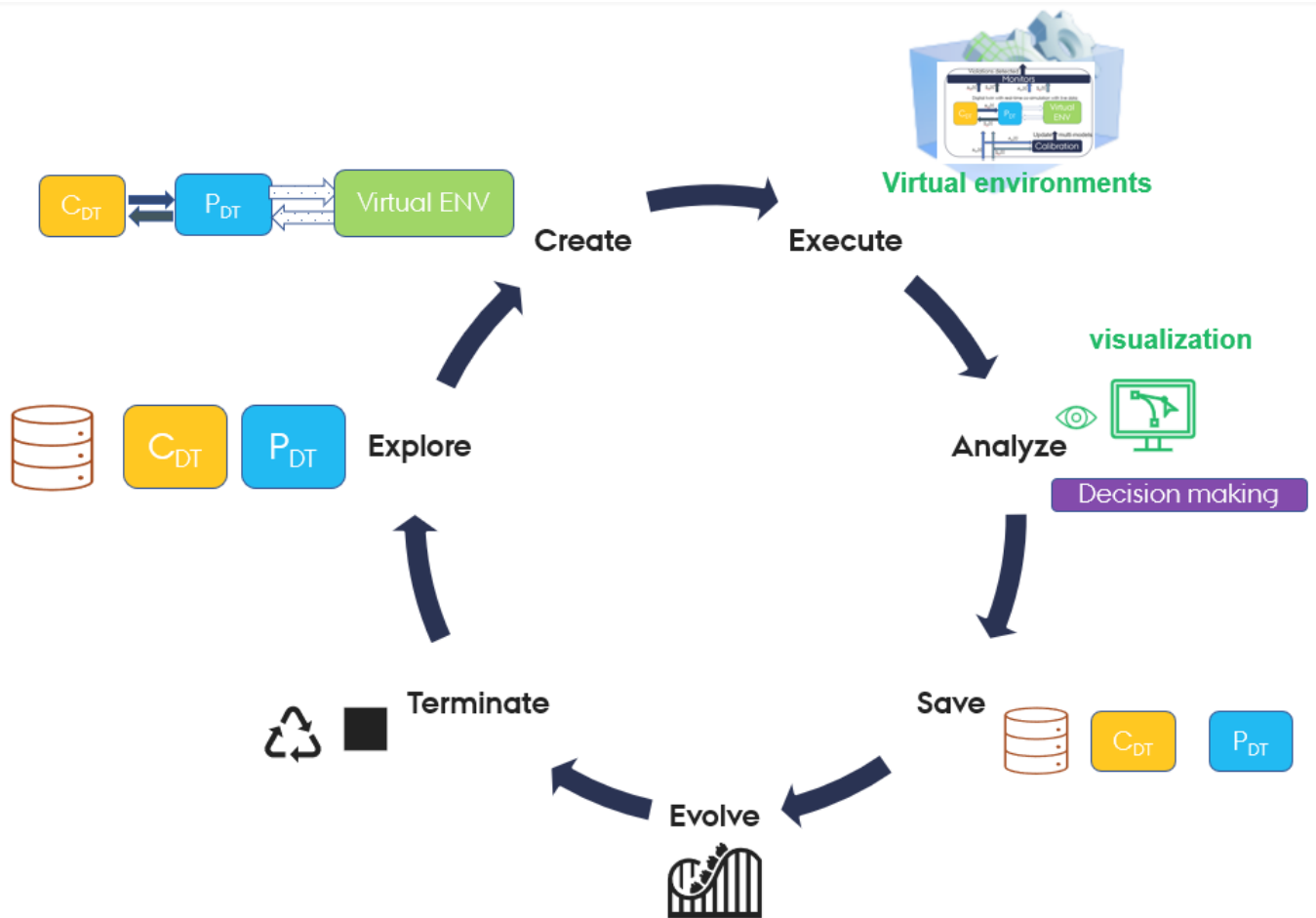
```
1 $wget https://github.com/INTO-CPS-Association/DTaaS-examples/archive/refs/heads/main.zip
2 $unzip main.zip
```

5. Open a file browser and copy the files from this uncompressed folder into your workspace folder ( `/workspace` ). Make sure that the file placement matches the one given above.
6. Go to the digital twin directory and run

```
1 $cd /workspace/digital\ twins/mass-spring-damper
2 $lifecycle/execute
```

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

## 3.4.2 Lifecycle



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

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

A complete digital twin will support all the phases but it is not mandatory.

Even though not mandatory, having a coding structure makes it easy to manage DT lifecycle phases. It is recommended to have the following structure

```
1 workspace/
2   digital twins/
3     digital-twin-1/
4       lifecycle/
5         analyze
6         clean
7         evolve
8         execute
9         save
10        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.

## Examples

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

```
1 #!/bin/bash
2 mkdir -p /workspace/data/mass-spring-damper/output
3 #cd ..
4 java -jar /workspace/common/tools/maestro-2.3.0-jar-with-dependencies.jar \
5   import -output /workspace/data/mass-spring-damper/output \
6   --dump-intermediate sgl cosim.json time.json -i -vi FMI2 \
7   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.

```
1 #!/bin/bash
2 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.

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

## 4. FAQ

### 4.1 Abbreviations

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.

#### 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. 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 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 the licensed Matlab software.

### **DTaaS is not able to do any modelling or simulation in this case, like other platforms in market provide modelling and simulation alongside integration and UI. Is this a correct understanding?**

Yes, you are right

### **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 provides InfluxDB, RabbitMQ, MQTT 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.

## 5. Bugs

---

### 5.1 Some limitations

---

- The complete DTaaS software requires multiple docker containers and one client website. All of these can work together only on a server with a proper domain name. The complete application does not work on localhost.

### 5.2 Third-Party Software

---

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

#### 5.2.1 ML Workspace

---

- the docker container loses network connectivity after three 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.

### 5.3 Gitlab

---

- The gitlab oauth authentication 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 browser 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.

## 6. Thanks

---

### 6.1 Contributors

---

[code contributors](#)

### 6.2 Users

---

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

### 6.3 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.