

Digital Twin as a Service

Digital Twin as a Service (DTaaS)

DTaaS Development Team

Copyright © 2022 - 2026 The INTO-CPS Association

Table of contents

1. What is the DTaaS Platform? 4
1.1 License 4
2. Media Playlist 5
2.1 Current GIF previews 5
2.2 Referenced slides and videos 6
3. User 8
3.1 DTaaS for Users 8
3.2 Overview 9
3.3 Website 11
3.4 Reusable Assets 48
3.5 Workflow Logging 57
3.6 Digital Twins 60
3.7 Working with GitLab 88
3.8 Runner 92
3.9 Examples 96
4. Admin 192
4.1 Installation Steps 192
4.2 Auto Installation 195
4.3 Manual Installation (Advanced) 220
4.4 Platform Services 264
4.5 GitLab 270
4.6 Independent Packages 281
4.7 Guides 288
5. Frequently Asked Questions 300
5.1 Abbreviations 300
5.2 General Questions 300
5.3 Digital Twin Assets 301
5.4 Digital Twin Models 301
5.5 Communication Between Physical Twin and Digital Twin 301
5.6 Digital Twin DevOps Automation 302
5.7 Data Management 302
5.8 Platform Native Services on the DTaaS Platform 303
5.9 Comparison with other DT Platforms 304
5.10 GDPR Concerns 304

Copyright © 2022 - 2026 The INTO-CPS Association

6. Developer 306
6.1 Guide 306
6.2 System 308
6.3 Technical Concepts 330
6.4 Codebase 347
6.5 Contributions 370
7. Known Issues in the Software 386
7.1 Third-Party Software 386
7.2 GitLab 386
8. Thanks 387
8.1 Funding Sources 387
8.2 Contributions 387
8.3 Developers 388
8.4 Documentation 388
9. License 389
9.1 License 389
9.2 Third Party Software 393

Copyright © 2022 - 2026 The INTO-CPS Association

1. What is the DTaaS Platform?

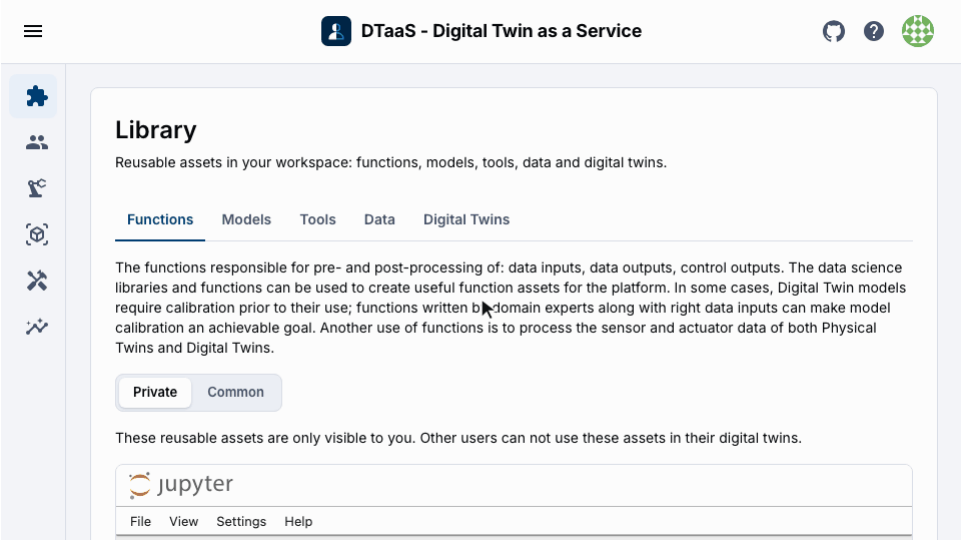
The Digital Twin as a Service (DTaaS) software platform is used to **Build, Use and Share** digital twins (DTs).

Build: construct DTs on DTaaS from the reusable DT assets available on the platform.

Use: run DTs on the DTaaS platform.

Share: share ready-to-use DTs with other users, or share the services that one DT offers.

This walkthrough shows the DTaaS v1.1 interface and workflow:



An overview of the DTaaS platform is available as a short introduction video and slides.

1.1 License

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

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

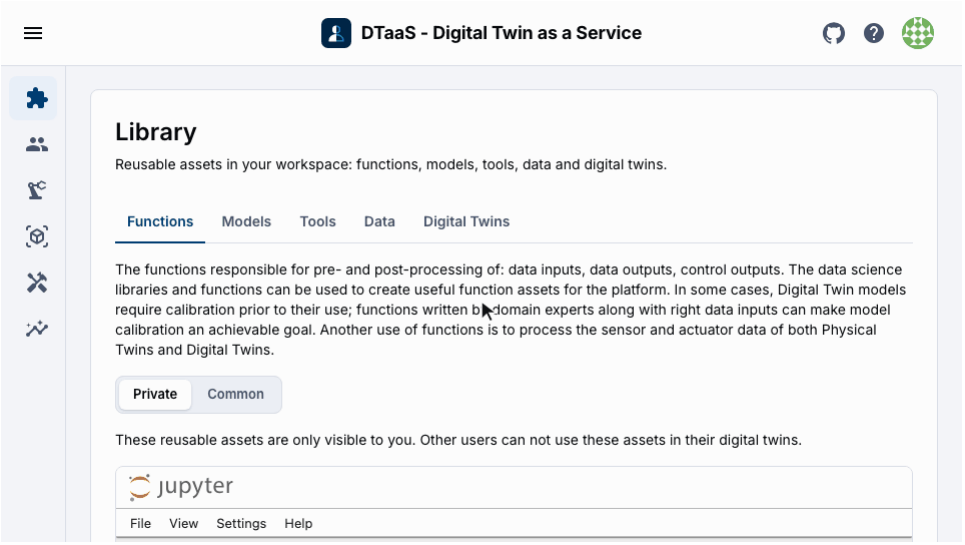
Copyright © 2022 - 2026 The INTO-CPS Association

2. Media Playlist

This page lists the slide decks and videos that the DTaaS documentation links to. GIF previews are embedded where a preview file exists. When both a standard and a `4x` version exist, this page shows the `4x` version.

2.1 Current GIF previews

2.1.1 DTaaS v1.1 feature walkthrough



Referenced from index and user/website/dtaas.

Copyright © 2022 - 2026 The INTO-CPS Association

2.2 Referenced slides and videos

Copyright © 2022 - 2026 The INTO-CPS Association

Topic	Slides/PDF	Video	Referenced From
DTaaS short introduction	Slides	Video	index
DTaaS v1.1 user walkthrough		Video	index
User presentation	Slides	Video	user/motivation
Developer presentation	Slides	Video	developer/index
DTaaS DevOps workflows	----------	Video	developer/index
DTaaS and DevOps	----------	Video	developer/system/research-overview
Federated DTaaS	----------	Video	developer/system/research-overview
Automation layer demo	----------	Video	user/digital-twins/devops/ui
Examples overview	Slides	Video	user/examples/index
CP-SENS project demo	Slides	Video	user/examples/index
Wind turbine blade testing demo	----------	Video	user/examples/index
Python package demo (CP-SENS)	----------	Video	user/examples/index
DAQ part 1	----------	Video	user/examples/index
DAQ part 2	----------	Video	user/examples/index
Operational model analysis	----------	Video	user/examples/index
Model update demo	----------	Video	user/examples/index
Incubator demo	----------	Video	user/examples/index
MATLAB Simulink screencast	----------	Video	FAQ

Copyright © 2022 - 2026 The INTO-CPS Association

3. User

3.1 DTaaS for Users

3.1.1 User Guide

This guide is for users of the DTaaS platform. You need access to a live installation of the DTaaS platform. The simplest option is the workspace localhost installation scenario.

These Slides and Video explain the concepts behind the DTaaS platform.

3.1.2 Motivation

A central question for Digital Twin (DT) software platforms is how to support collaborative activities:

- Building digital twins (DTs)
- Using DTs independently
- Sharing DTs with other users
- Providing existing DTs as a service to other users

DT software platforms must also offer:

- Support for DT lifecycle management
- Scalability through flexible convention over configuration

3.1.3 Existing Approaches

Recent literature proposes several solutions to these challenges, including:

- Focus on data from Physical Twins (PTs) for analysis, diagnosis, and planning
- Sharing DT assets across upstream and downstream stakeholders
- Evaluating different models of PT
- DevOps methodologies for Cyber Physical Systems (CPS)
- Scaling DT execution and ensemble management of related DTs
- Support for PT product lifecycle

3.1.4 The DTaaS Approach

The DTaaS platform follows these principles[1]:

- Support for transition from existing workflows to DT frameworks
- Creation of DTs from reusable assets
- Enabling users to share DT assets
- Offering DTs as a Service
- Integration of DTs with external software systems
- Separation of configurations for independent DT components

3.1.5 References

[1]: Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

Copyright © 2022 - 2026 The INTO-CPS Association

3.2 Overview

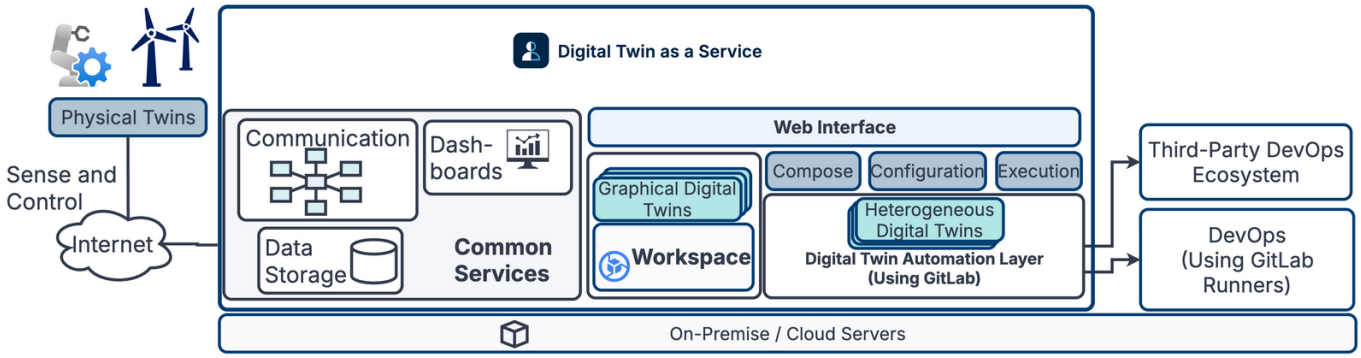
3.2.1 Advantages

The DTaaS software platform offers:

- Support for heterogeneous Digital Twin implementations
- CFD, Simulink, co-simulation, FEM, ROM, ML, and other paradigms
- Integration with existing Digital Twin frameworks
- Digital Twins offered as a service
- Collaboration and reuse of assets
- Private workspaces for verifying reusable assets and trying out DT executions
- Lower cost through shared infrastructure

3.2.2 Software Features

Each installation of the DTaaS platform includes the features shown in the following diagram.



Every user gets a dedicated workspace. These workspaces are containerized Linux desktops. The user desktops are isolated, so software installed and customisations made in one workspace do not affect other user workspaces. Graphical digital twins can be executed in these private workspaces.

Each user workspace comes with pre-installed development tools, which are accessed through a web browser. The following tools are currently available:

Tool	Advantage
Jupyter Lab	Create and use digital twins and their components in a web browser. All native JupyterLab use cases are supported.
Jupyter Notebook	Manage files and library assets in a web browser.
VS Code in the browser	A widely used IDE for software development. Digital twin assets can be developed in this environment.
ungit	An interactive git client for managing repositories in a web browser.

In addition, an xfce-based remote desktop is accessible through a VNC client that runs in the web browser. Desktop software supported by xfce can also be executed in the workspace.

The workspaces have Internet access, so Digital Twins running in a workspace can interact with both internal and external services.

Copyright © 2022 - 2026 The INTO-CPS Association

A DT automation layer manages DT automation tasks. It creates, modifies and executes DTs on both on-premise infrastructure and commercial cloud (DevOps) service providers.

The web client also has a Building Models page that draws IFC building models from the user's library and colours them by the readings of their sensors.

The DTaaS software platform includes several pre-installed services. The currently available services are:

Service	Advantage
InfluxDB	Internet of Things (IoT) device management and data visualisation platform. This service stores data for digital twins and provides alerting capabilities.
RabbitMQ	Communication broker for message exchange between physical and digital twins.
Grafana	Visualisation dashboard service for presenting digital twin data.
MQTT	Lightweight data transfer broker for IoT devices and physical twins providing data to digital twins.
MongoDB	NoSQL document database for storing metadata from physical twins.
PostgreSQL	SQL database server for storing historical and time-series data.
ThingsBoard	Internet of Things (IoT) device management and data visualisation platform.

Users can publish and reuse the digital twin assets available on the platform. Digital twins can also be executed and offered as services to external clients. These clients do not need to be registered users of the DTaaS installation.

3.2.3 References

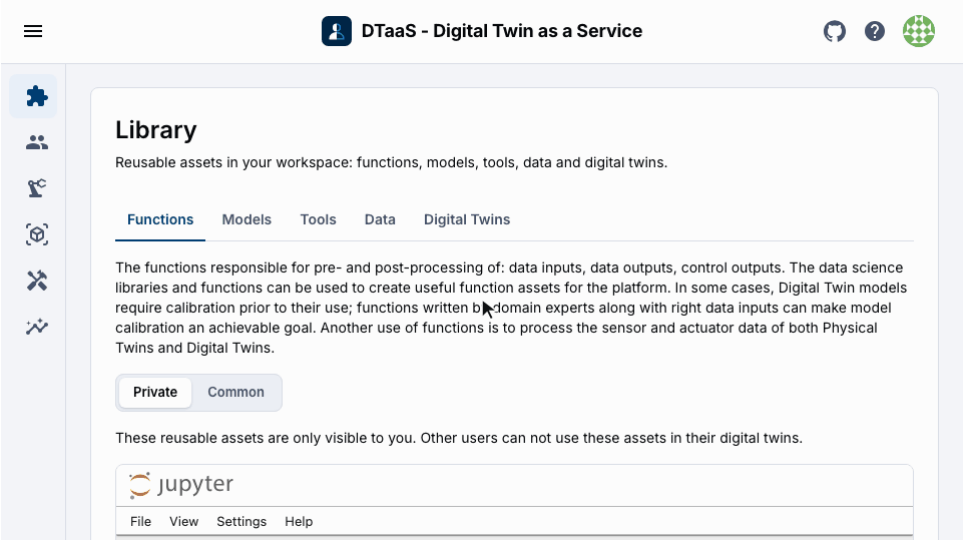
Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

Copyright © 2022 - 2026 The INTO-CPS Association

3.3 Website

3.3.1 DTaaS Website Screenshots

This page walks through the DTaaS website with screenshots. A short screen recording is also available.

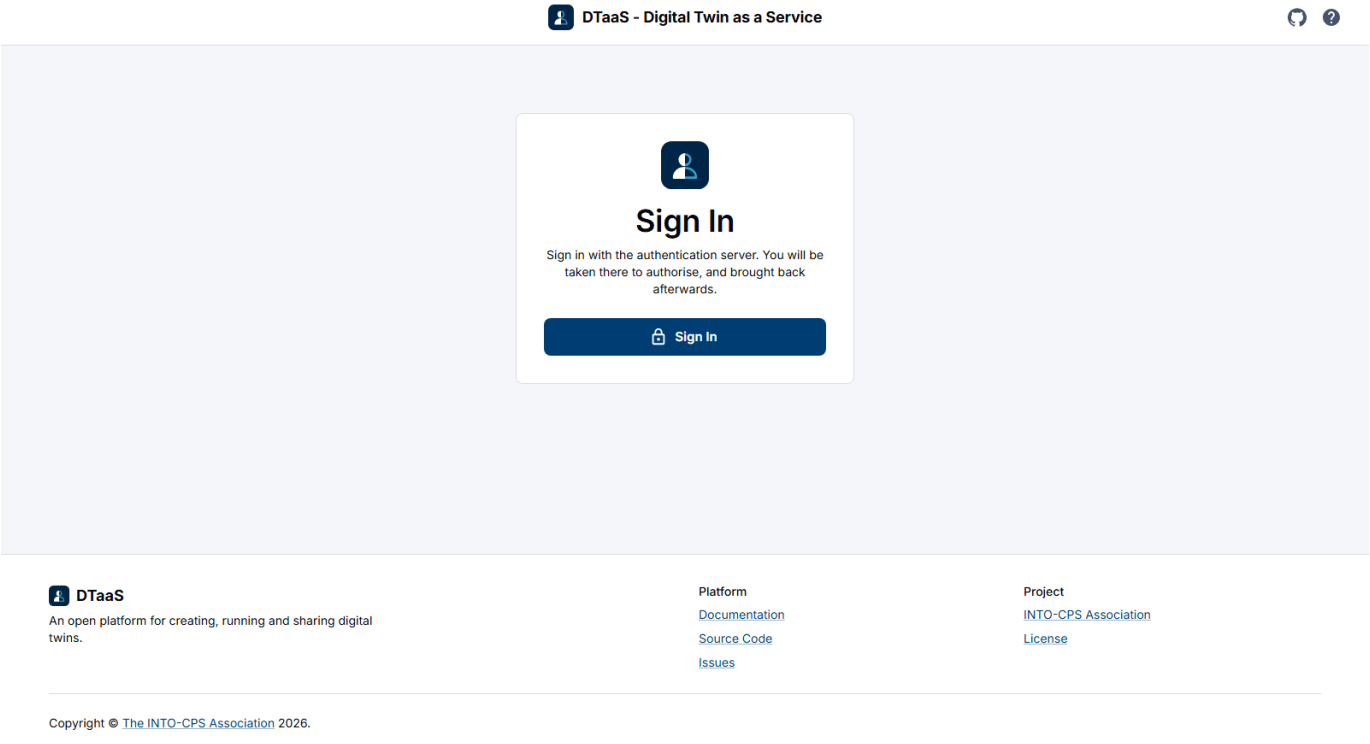


This tutorial uses `https://intocps.org` as an example base URL. Replace it with the URL of your installation.

Visit the DTaaS Installation

URL: `https://intocps.org` (specified in `config/client.js` file)

Start by visiting the website of the DTaaS installation you are registered with.

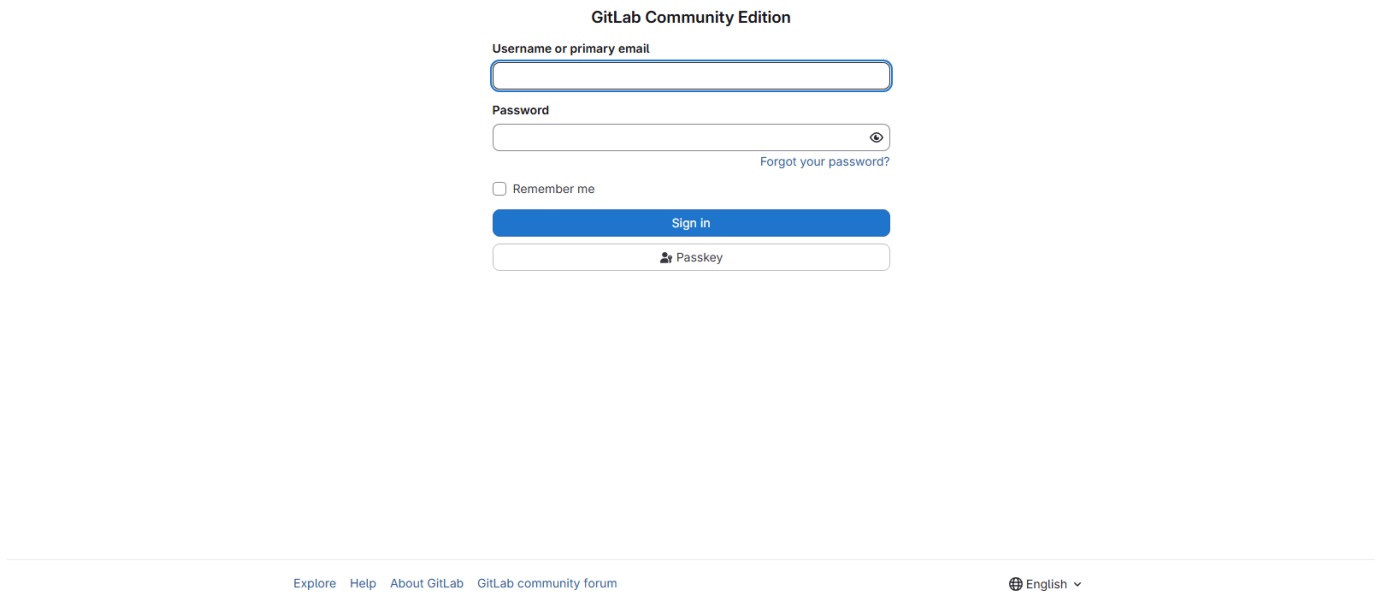


Copyright © 2022 - 2026 The INTO-CPS Association

Redirected to Authorization Provider

URL: `https://gitlab.com/xxxx` (specified in `config/.env` file)

The browser redirects to the GitLab authorization page for the DTaaS.



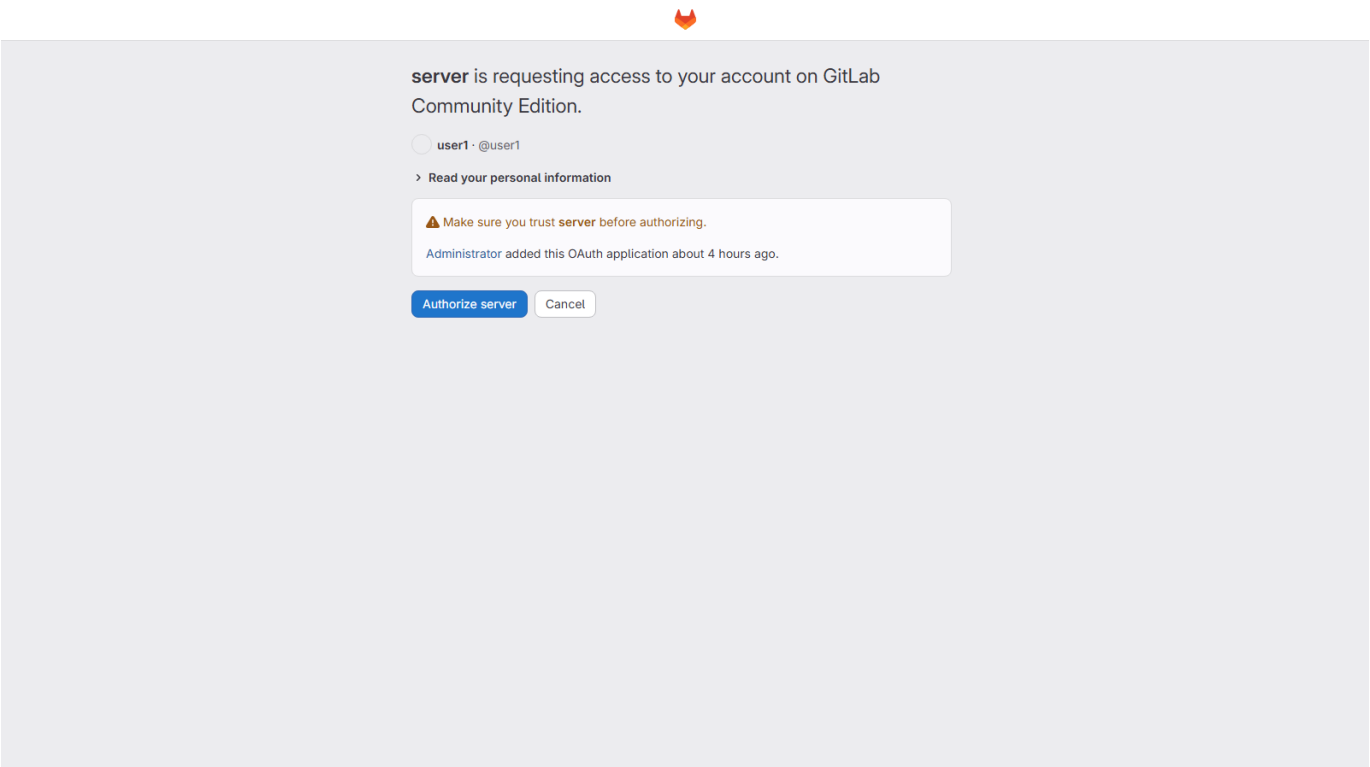
Enter the email or username and password of the GitLab account whose email ID is registered with the DTaaS.

The browser then redirects to the OAuth 2.0 application page.

Permit DTaaS Server to Use GitLab

URL: `https://gitlab.com/xxxx`

Copyright © 2022 - 2026 The INTO-CPS Association

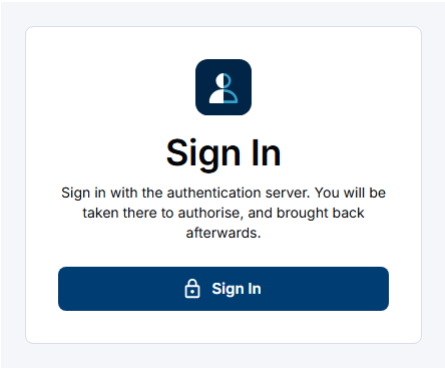


Clicking **Authorize** lets the OAuth 2.0 application access the information associated with the GitLab account. This step is required.

After successful authentication, the browser returns to the sign-in page of the DTaaS website.

URL: `https://intocps.org`

The DTaaS website adds a second layer of security with the OAuth 2.0 third-party authorization protocol. Only users with an active account at the selected OAuth 2.0 provider can access the installation. This installation also uses GitLab as the OAuth 2.0 provider.



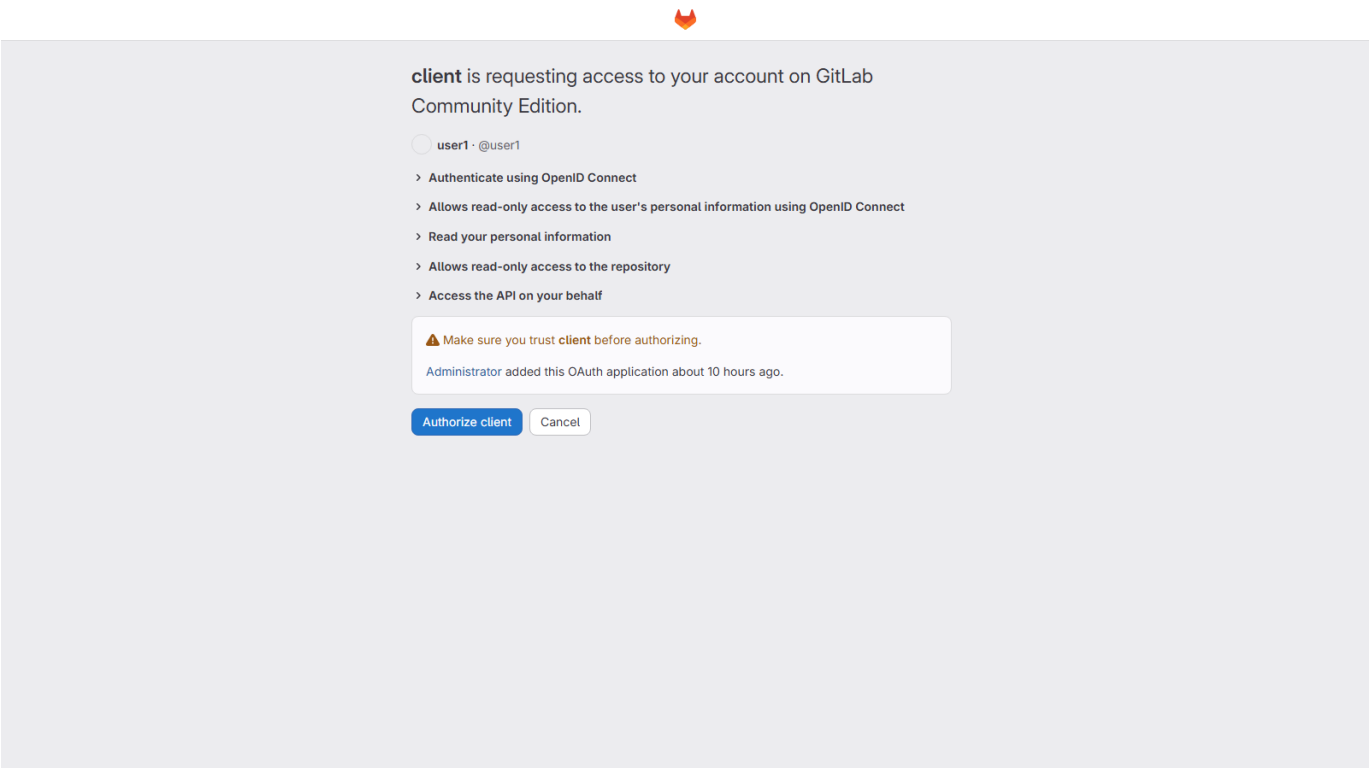
The sign-in page shows a **Sign In** button. Clicking it redirects to the GitLab instance that authorizes users for the DTaaS. You do not need to sign in to GitLab again unless you have explicitly logged out of your GitLab account.

Permit DTaaS Website to Use GitLab

URL: `https://gitlab.com/xxxx`

The DTaaS website needs permission to use the GitLab account for authorization. Click the **Authorize** button.

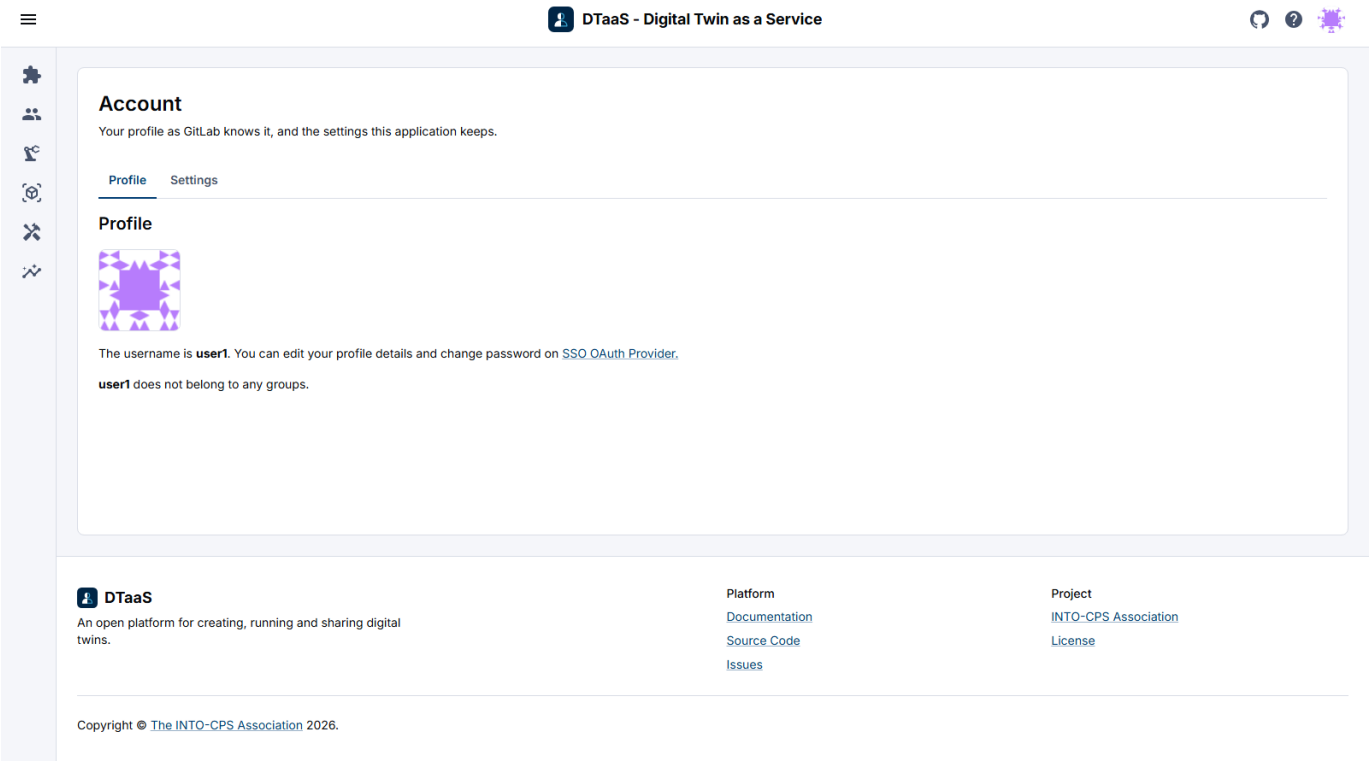
Copyright © 2022 - 2026 The INTO-CPS Association



After successful authorization, the browser opens the **Library** page of the DTaaS website.

URL: `https://intocps.org/library/xxxx`

The application bar at the top of every page shows the DTaaS logo and title in the middle. On the right are three buttons. The GitHub icon opens the DTaaS code repository and the question-mark icon opens this documentation. The round avatar, which shows your profile picture or the first letter of your username, opens a menu with **Account** and **Logout**. The **Account** page has a **Profile** tab and a **Settings** tab where you can change settings.

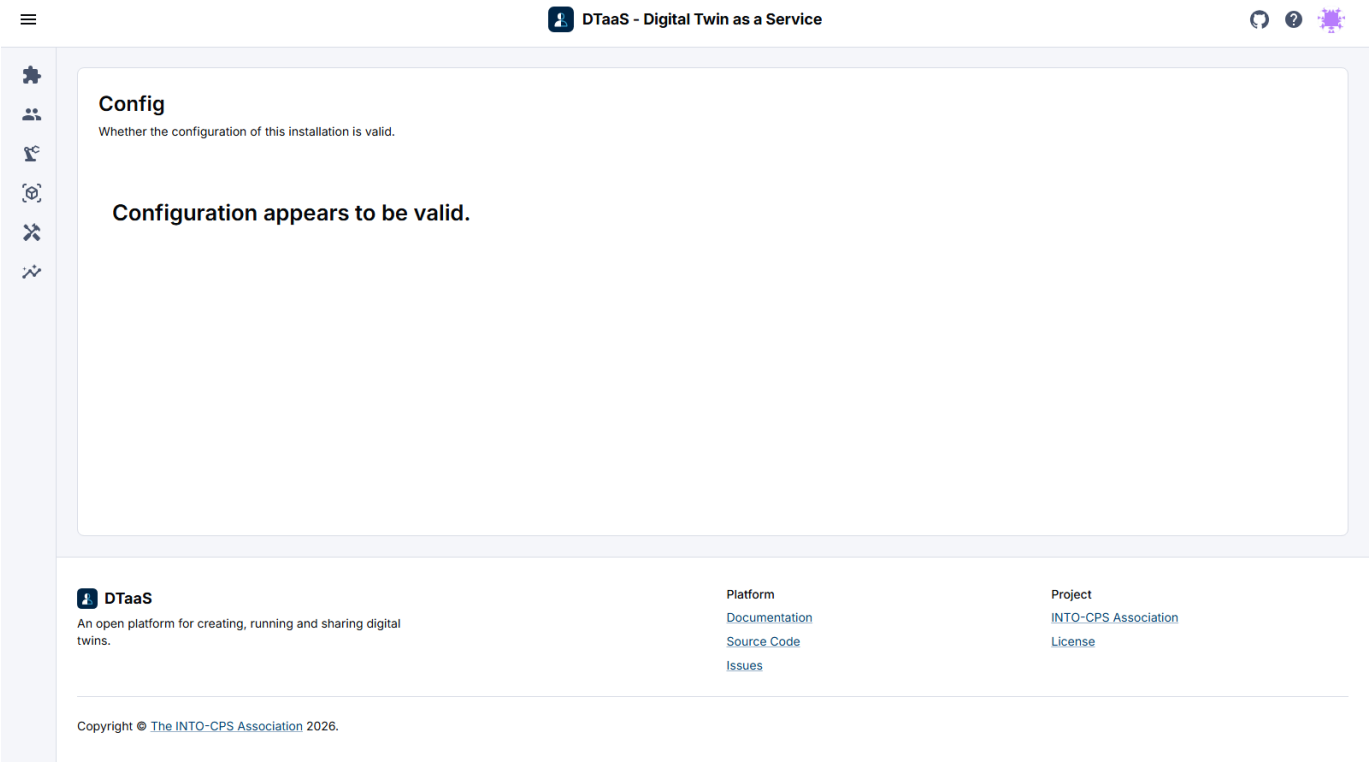


Copyright © 2022 - 2026 The INTO-CPS Association

The footer at the bottom of every page links to the documentation, the source code, the issue tracker, the INTO-CPS Association and the license.

Check Website Access

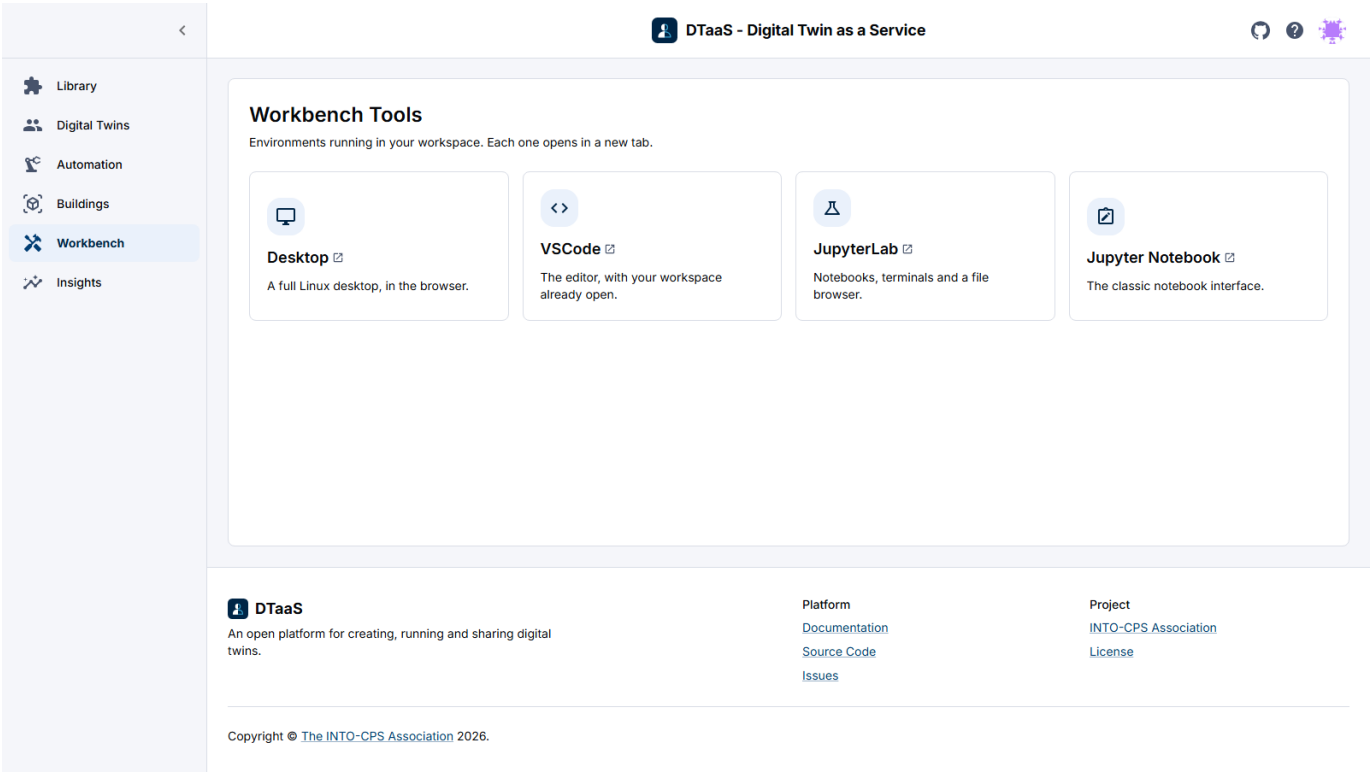
To troubleshoot sign-in problems, open https://intocps.org/config/user to check the website configuration. The page below means the application is configured correctly. A signed-in user can run the same check from **Insights > Config**, which shows it inside the application. Its link to the full list of configuration values also stays inside the application.



Menu Items

The menu is a sidebar on the left of the page. By default it is collapsed to a narrow rail that shows only the icons, and hovering over an icon shows its name. Click the **three horizontal bars** icon in the top-left corner to expand the sidebar with the names of the pages. The arrow at the top of the expanded sidebar collapses it again.

Copyright © 2022 - 2026 The INTO-CPS Association



The menu has six items:

Library: Manage reusable library assets. You can upload, download, create and modify files on this page.

Digital Twins: Manage digital twins. The page embeds Jupyter Lab, from which digital twins can be executed.

Automation: Links to the Library Page, the Digital Twins Page and the Measurement page. These pages show the integration of the DTaaS with GitLab CI/CD and are experimental.

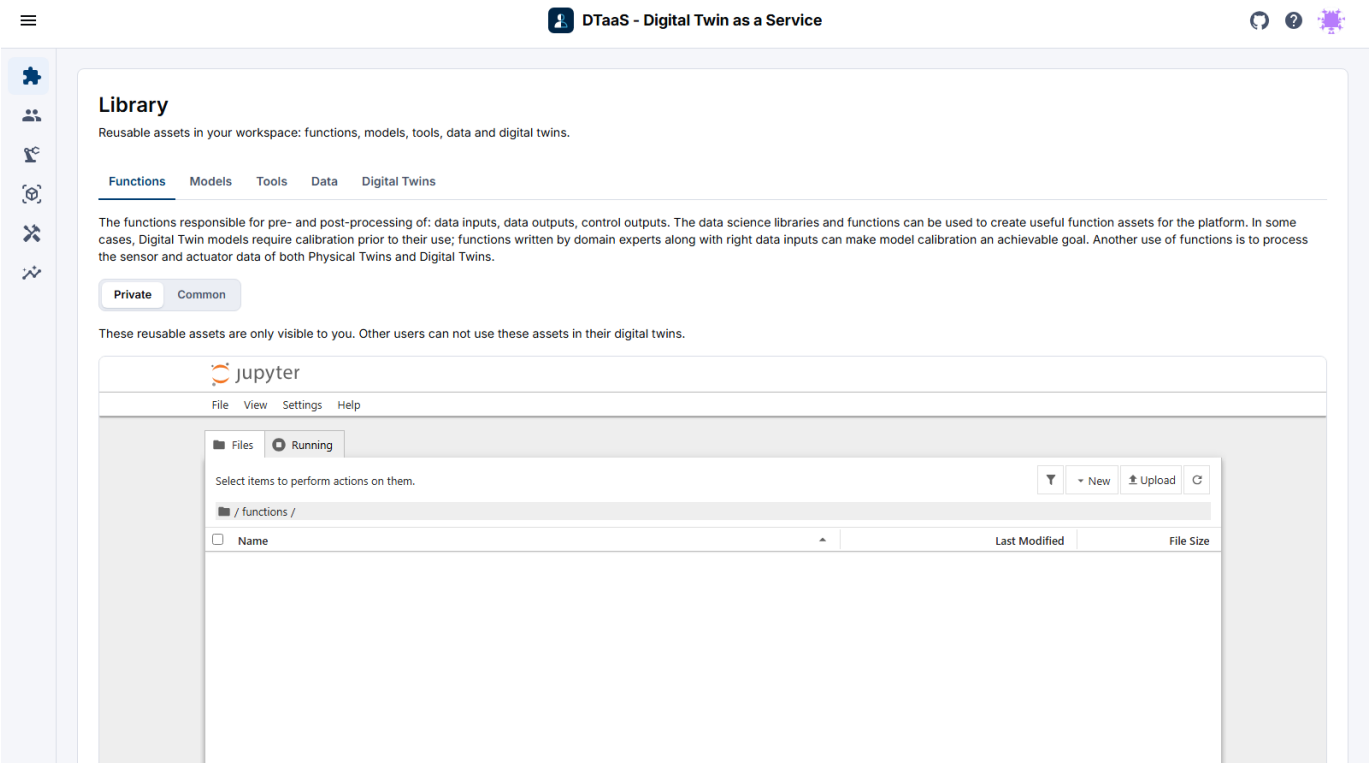
Buildings: Opens the Building Models page, which draws the IFC building models in your library and colours them by the readings their sensors report.

Workbench: Not every digital twin can be managed in Jupyter Lab. This page links to the other tools in your workspace.

Insights: Information the platform can give you about itself: the Logs of this website and the configuration check described above.

Copyright © 2022 - 2026 The INTO-CPS Association

Library Page



The page has five tabs, one for each type of digital twin asset. Each tab has a description of its asset type.

Functions

The functions responsible for pre- and post-processing of data inputs, data outputs and 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 the right data inputs can make model calibration achievable. Functions can also process the sensor and actuator data of both Physical Twins and Digital Twins.

Models

The model assets 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 models that are useful for tracking the wear and tear of parts. The same robot can have behavioural models describing the safety guarantees provided by the robot manufacturer. The same robot can also have functional models describing the part manufacturing capabilities of the robot.

Tools

The software tool assets are software used to create, evaluate and analyse models. These tools run on top of a computing platform, i.e., an operating system, a virtual machine such as the Java virtual machine, or inside docker containers. Tools tend to be platform specific, which makes them less reusable than models. A tool can be packaged to run on a local or distributed virtual machine environment, so the most suitable execution environment can be selected for a Digital Twin. Most models require tools to evaluate them in the context of data inputs. In some cases executable packages are run as binaries in a computing environment. Each of these packages is a pre-packaged combination of models and tools that makes a ready-to-use Digital Twin.

Copyright © 2022 - 2026 The INTO-CPS Association

Data

The data sources and sinks available to 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 visualisation software, external users and data storage services. There are also 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.

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.

Each asset tab has two sub-tabs: **Private** and **Common**. Private library assets are visible only to the signed-in user. Common library assets are available to all users.

The assets page explains where to place reusable assets of each type and the directory structure on the server.

Note

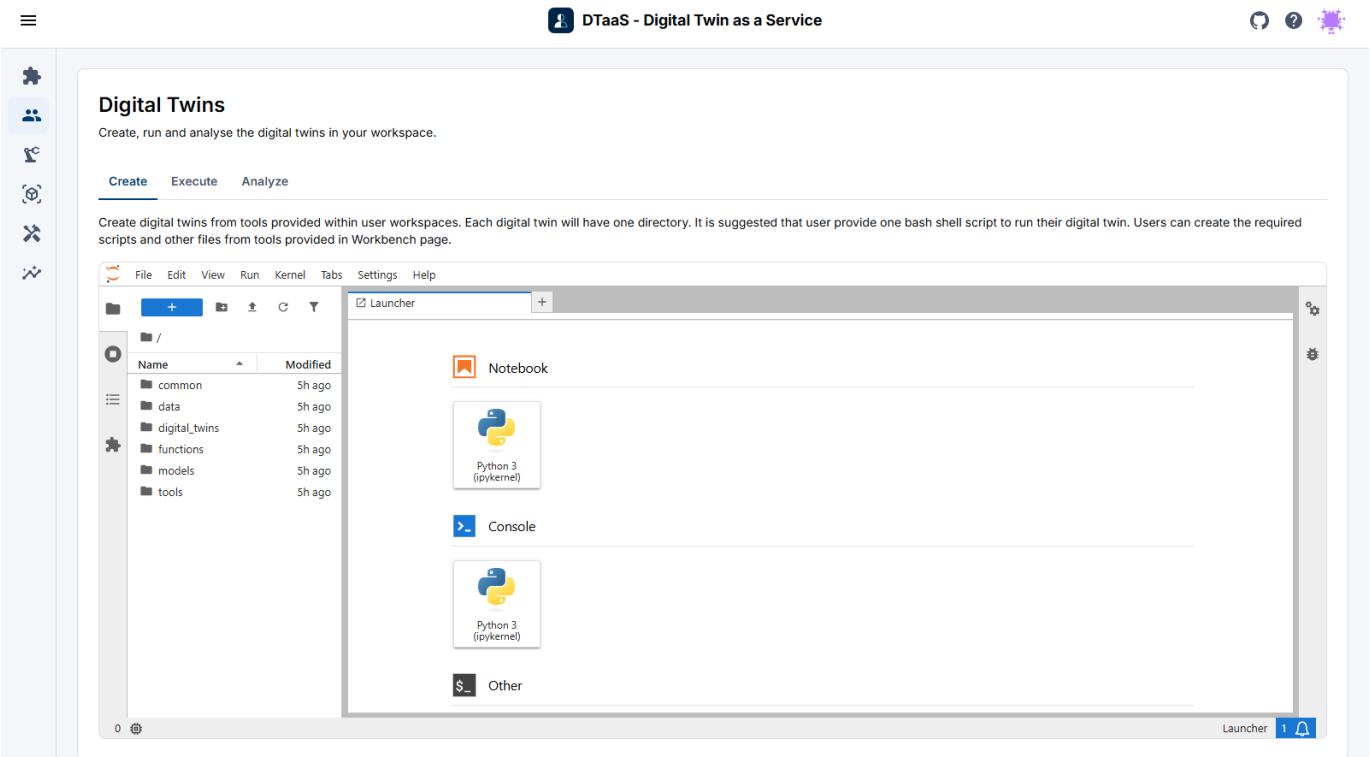
Assets (files) can be uploaded using the **upload** button.

Note

The file manager is based on Jupyter Notebook, and all tasks available in Jupyter Notebook can be performed here.

Digital Twins Page

URL: `https://intocps.org/digitaltwins`



Copyright © 2022 - 2026 The INTO-CPS Association

The Digital Twins page has three tabs, **Create**, **Execute** and **Analyze**, and each tab shows Jupyter Lab below its description. The descriptions suggest tasks for these lifecycle phases of a digital twin. The lifecycle phases of digital twin page explains them further.

Create

Create digital twins from tools provided within user workspaces. Each digital twin has one directory. Users are advised to provide one bash shell script to run their digital twin. Users can create the required scripts and other files with the tools on the Workbench page.

Execute

Digital twins are executed from within user workspaces. The given bash script is executed from the 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

Analysing a digital twin requires running the digital twin script from the user workspace. Analysis scripts process the execution results placed in the data directory and put their results back in the data directory. These scripts can be executed from VSCode, or from VNC GUI when they produce graphical results.

Note

The reusable assets (files) shown in the file manager are also available in Jupyter Lab. Jupyter Lab also has a git plugin for linking files with external git repositories.

Workbench

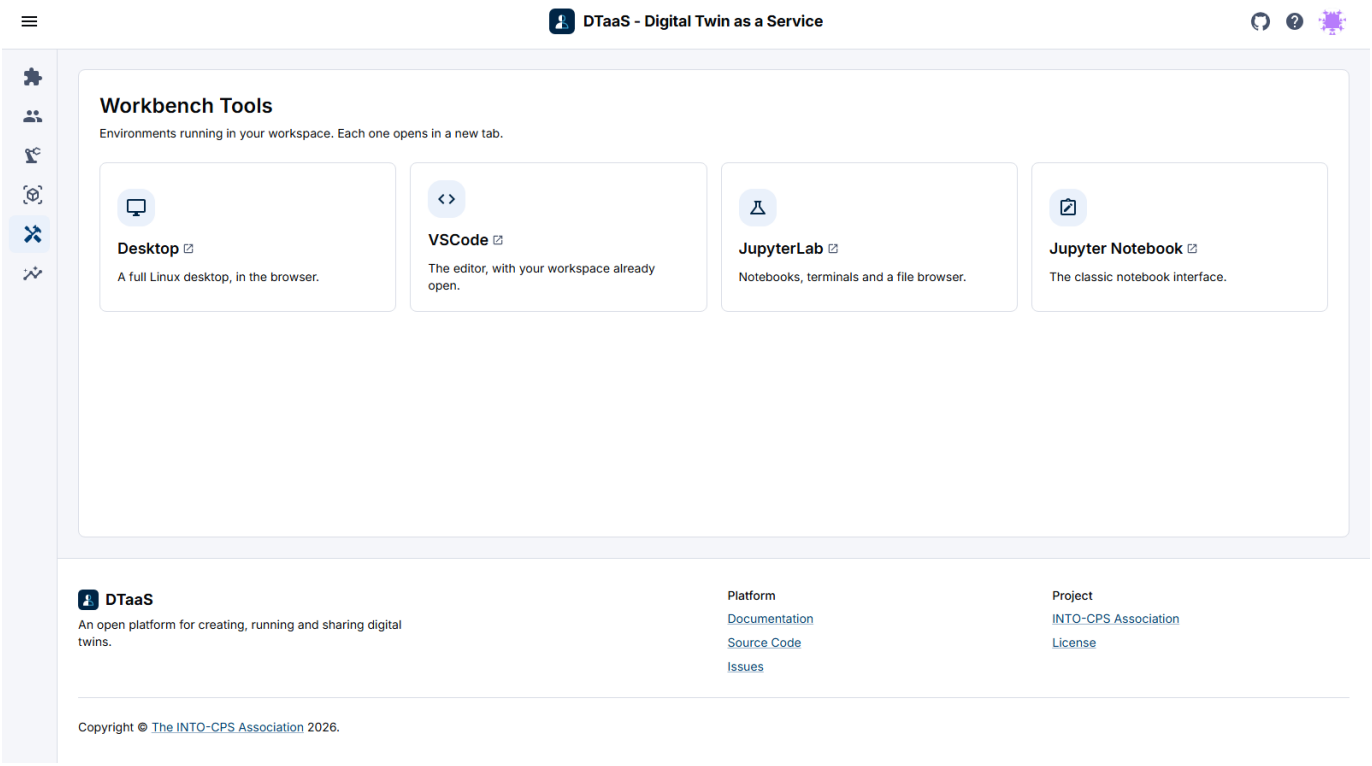
URL: `https://intocps.org/workbench`

The **Workbench Tools** page shows a card for each tool running in your workspace. A card has the tool's name, a one-line description and the address it opens. A standard workspace has four tools:

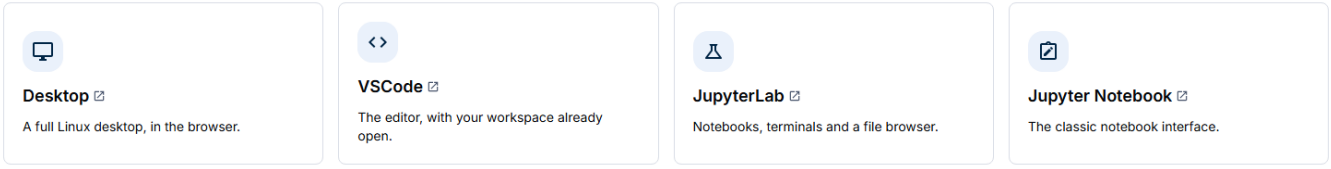
- Desktop
- VSCode
- JupyterLab
- Jupyter Notebook

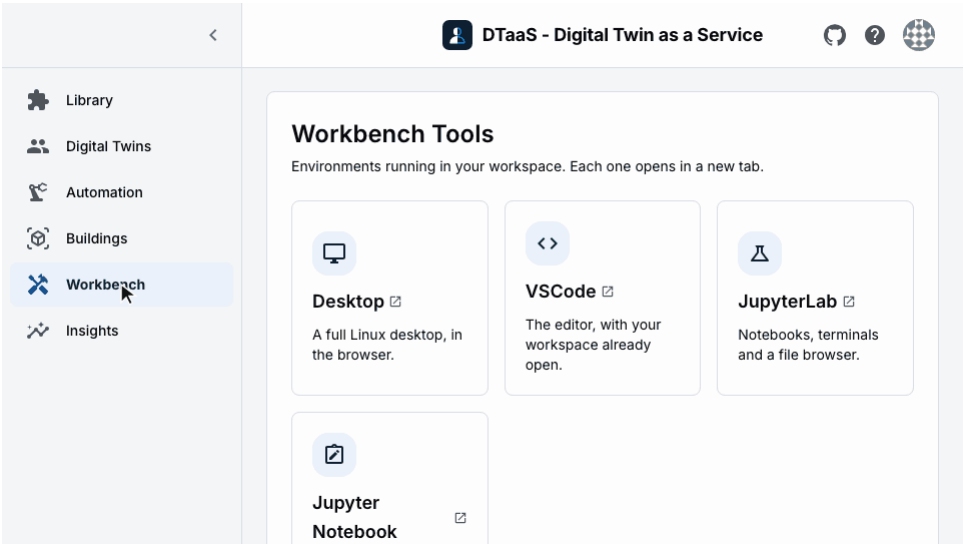
The page lists only the tools your workspace reports. If it reports none, the page says so instead of showing cards.

Copyright © 2022 - 2026 The INTO-CPS Association



Each card opens its tool in a new browser tab. The Workspace page shows what each tool looks like.





Automation

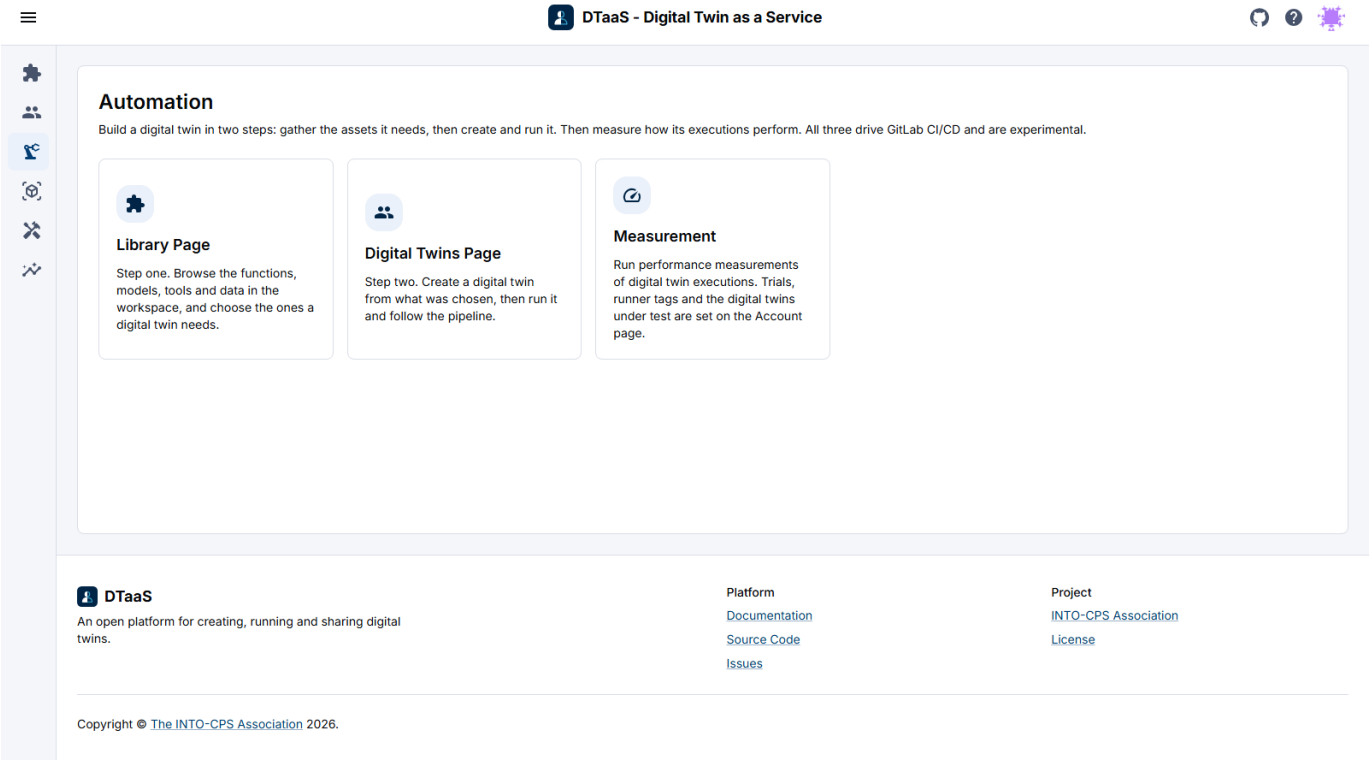
URL: `https://intocps.org/automation`

Copyright © 2022 - 2026 The INTO-CPS Association

The **Automation** page has three cards for the DevOps-based implementation of composable digital twins. Each card opens its page in the same browser tab.

- Library Page
- Digital Twins Page
- Measurement

The trials, runner tags and digital twins that the Measurement page uses are set on the Settings tab of the Account page.

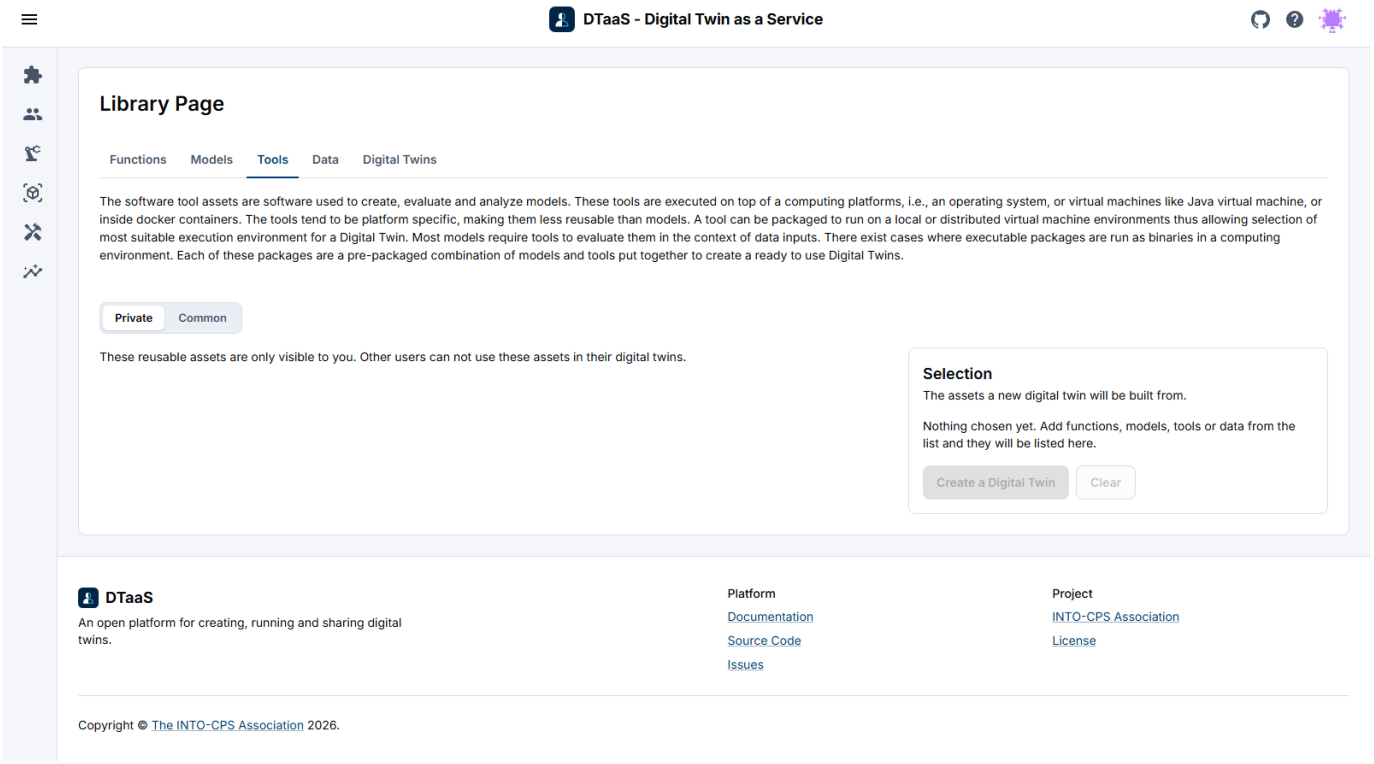


LIBRARY PREVIEW PAGE

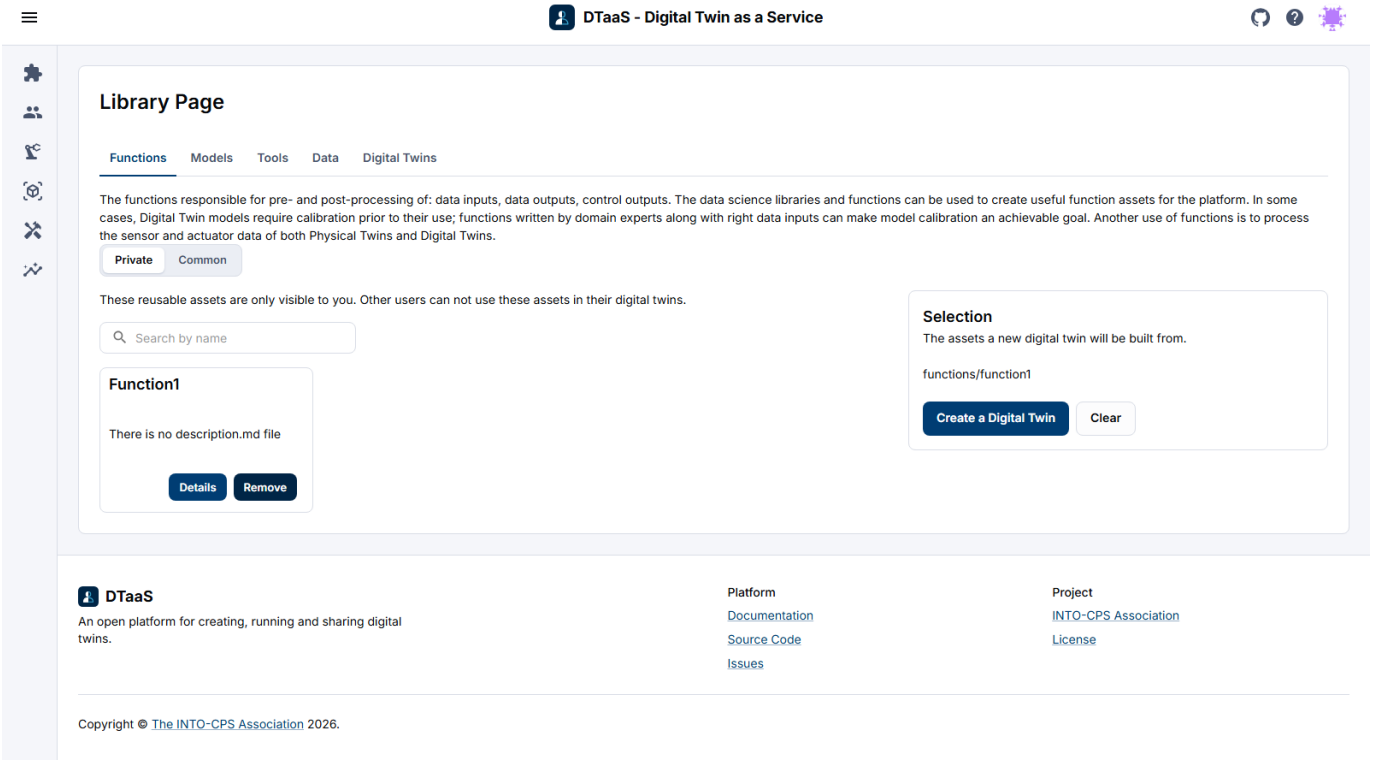
URL: `https://intocps.org/preview/library`

This page, titled **Library Page**, follows the same idea as the Library page and has a similar interface.

Copyright © 2022 - 2026 The INTO-CPS Association



Unlike the Library page, it uses digital twin assets stored in a GitLab repository. You compose a new digital twin by selecting the library assets it needs. The selected assets are listed on the page.



Clicking the **Create a Digital Twin** button opens the **Create** tab of the Digital Twins Page. The **Clear** button empties the selection. Both buttons are disabled until at least one asset is selected.

Copyright © 2022 - 2026 The INTO-CPS Association

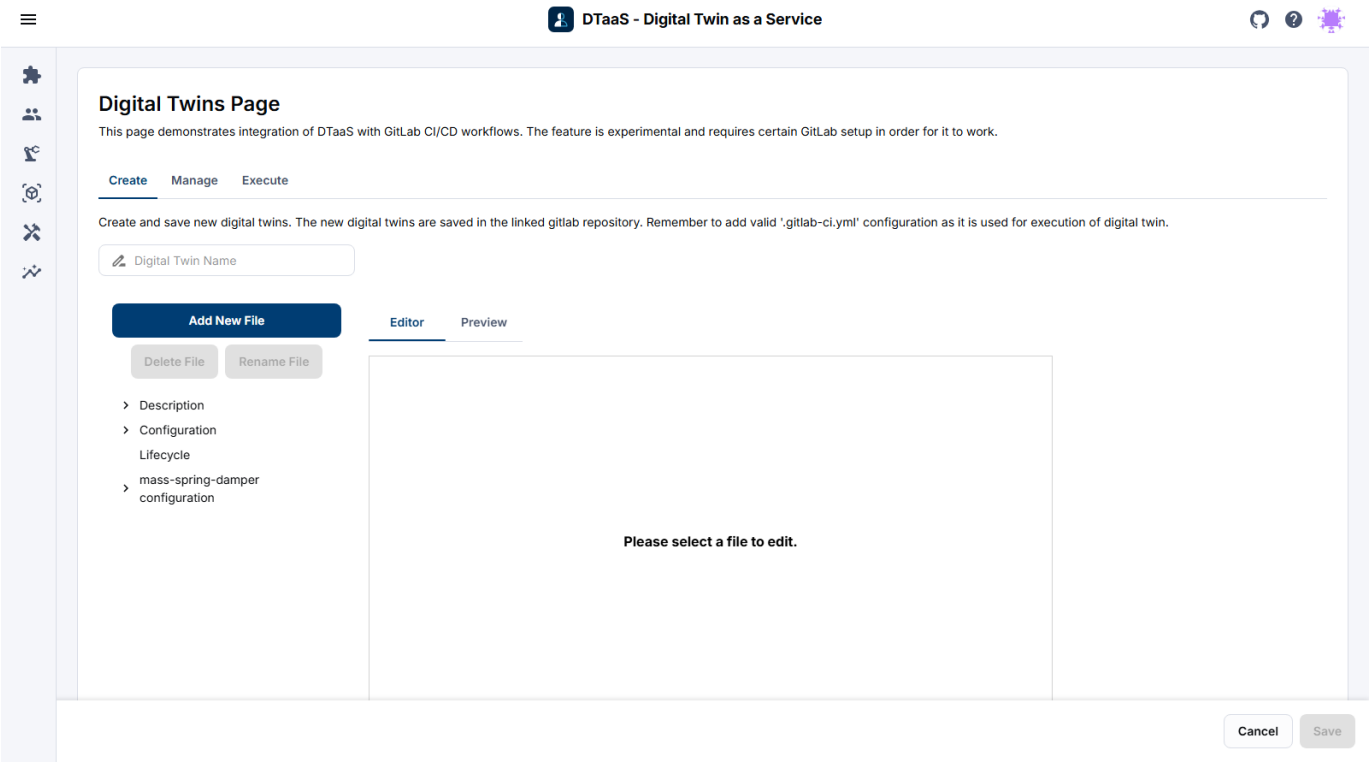
Digital Twins Preview Page

URL: `https://intocps.org/preview/digitaltwins`

This page, titled **Digital Twins Page**, manages digital twins with the DevOps methodology. It has three tabs: **Create**, **Manage** and **Execute**.

CREATE TAB

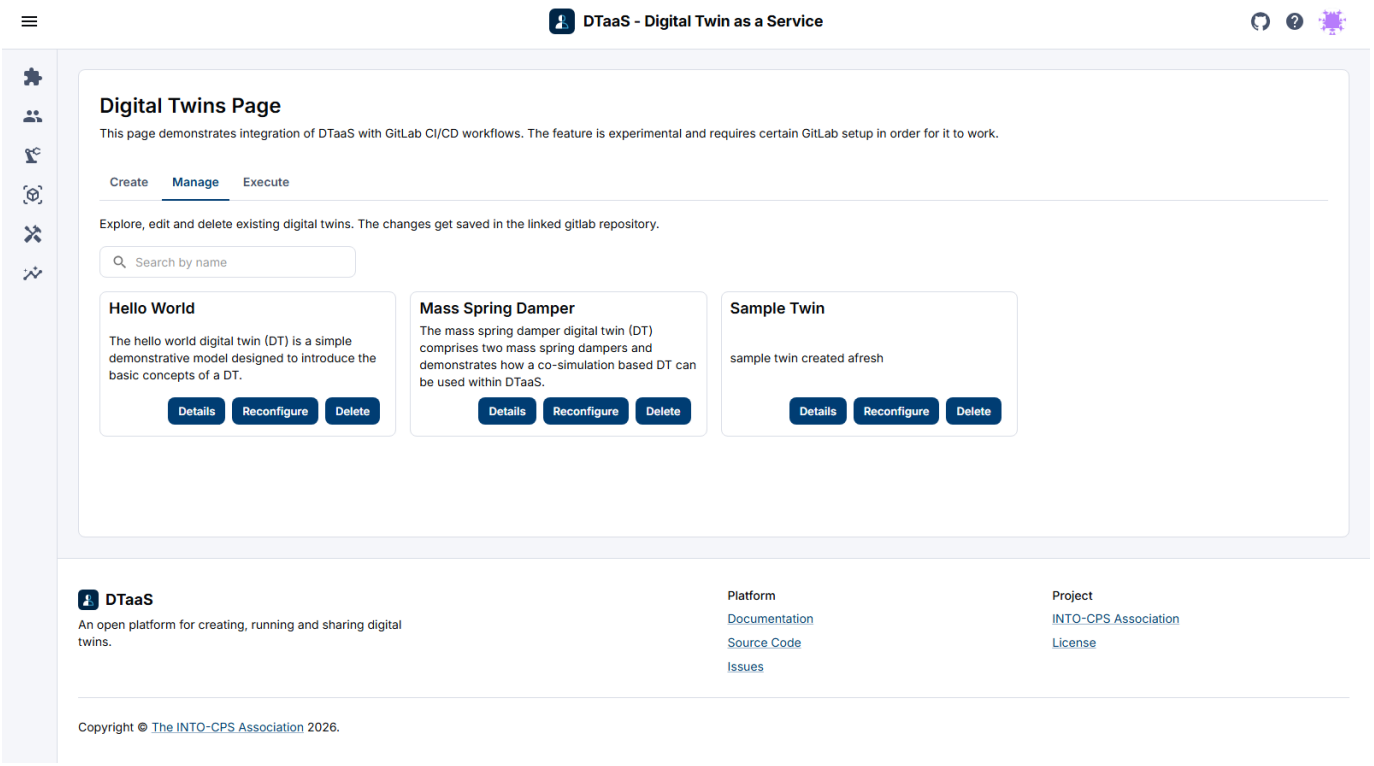
The **Create** tab uses the selected library assets to create new digital twins. New digital twins are saved in the linked GitLab repository. Remember to add a valid `.gitlab-ci.yml` configuration, because it is used to execute the digital twin.



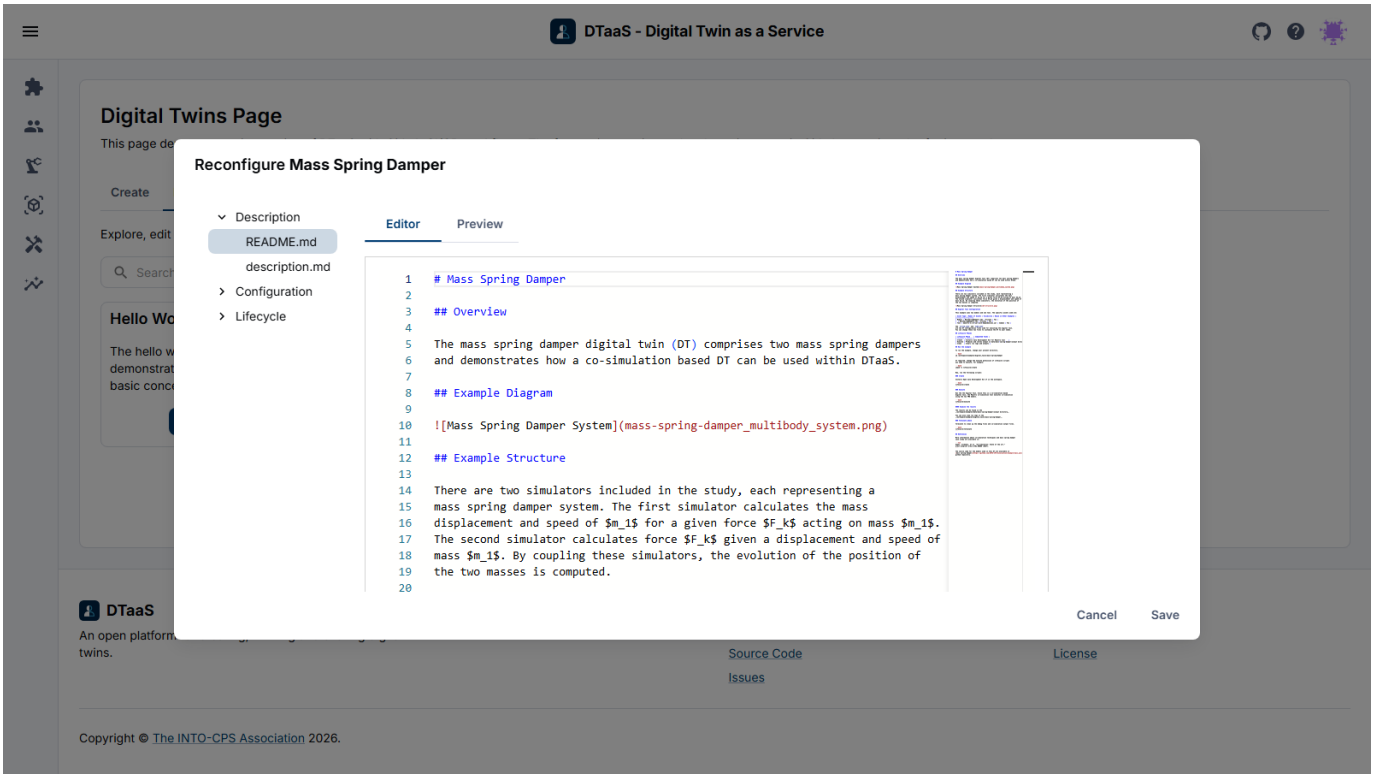
MANAGE TAB

The **Details** button shows the full description of a digital twin.

Copyright © 2022 - 2026 The INTO-CPS Association



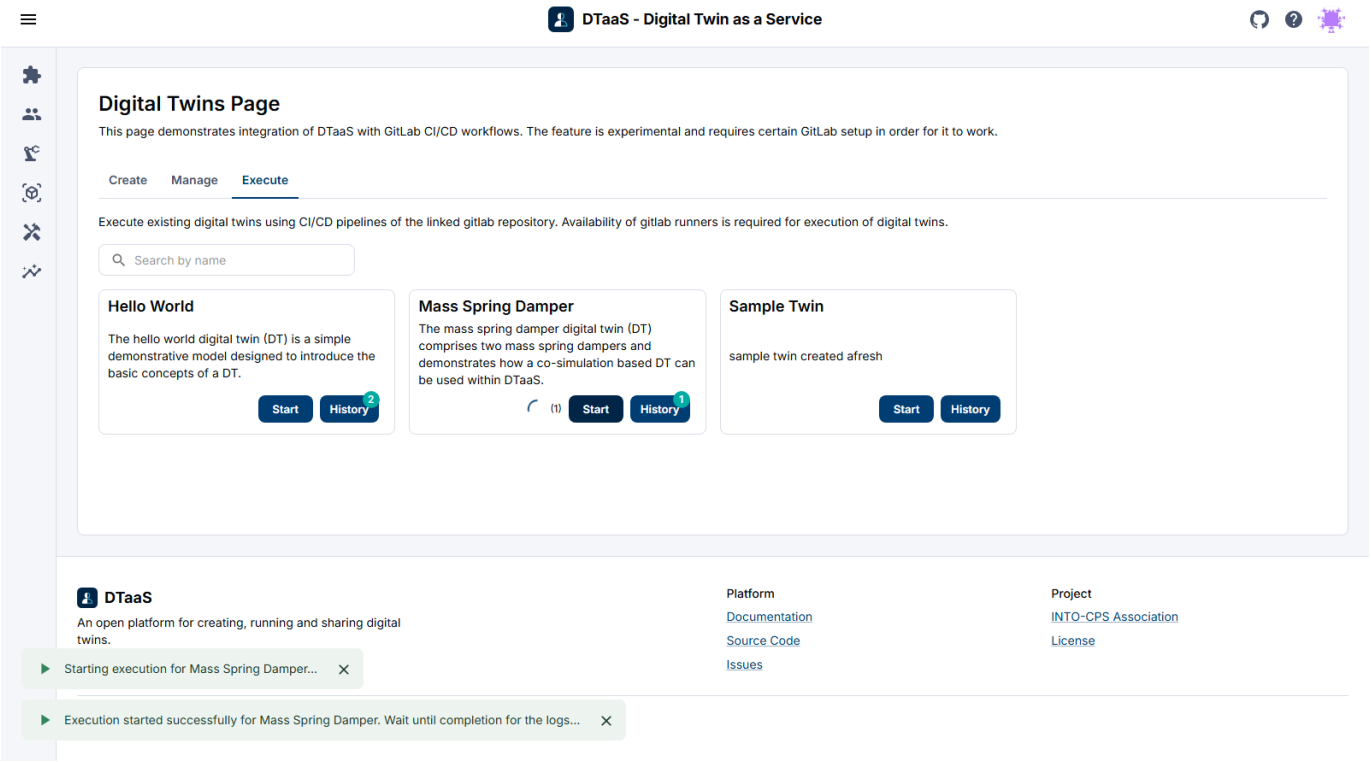
The **Delete** button removes a digital twin from the workspace along with all its data. The **Reconfigure** button changes the configuration of a digital twin.



EXECUTE TAB

Digital twins are executed with GitLab CI/CD workflows. Several digital twins can run at the same time.

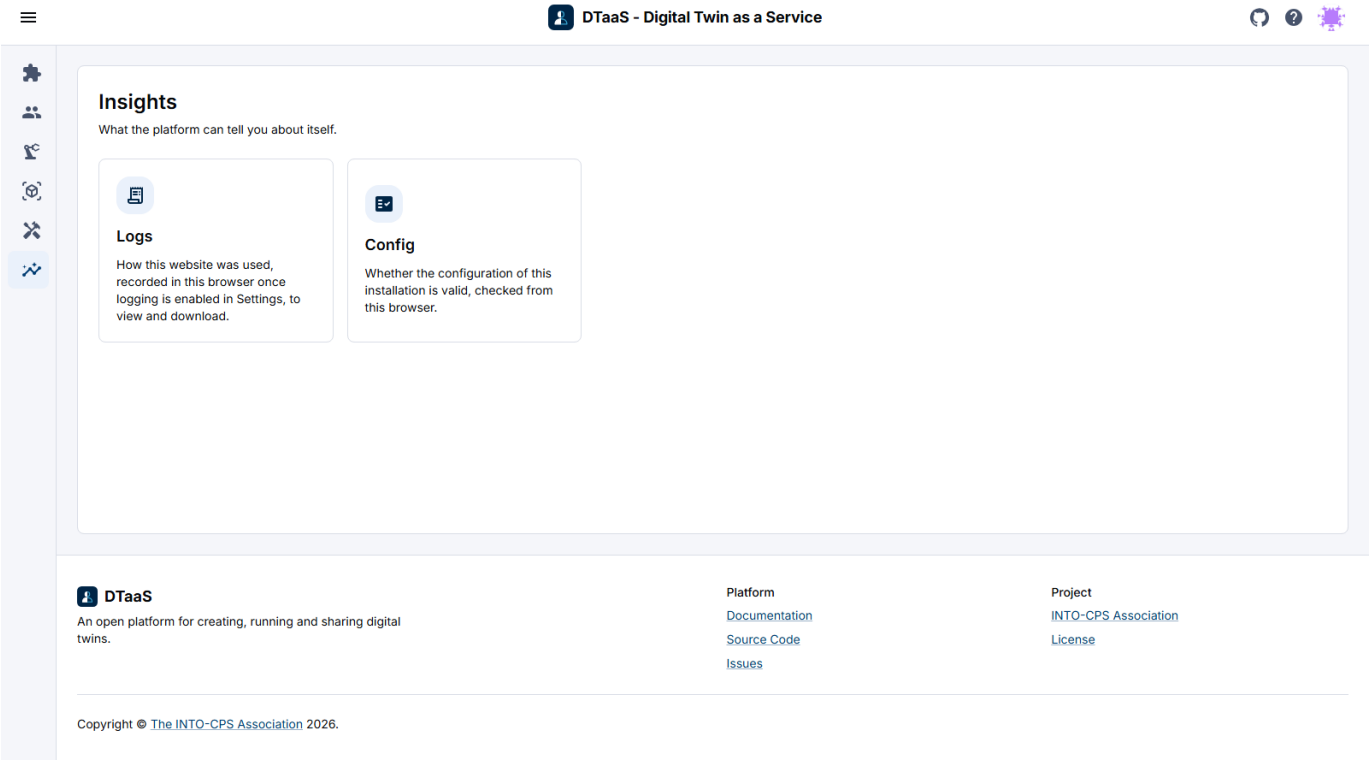
Copyright © 2022 - 2026 The INTO-CPS Association



Insights

URL: `https://intocps.org/insights`

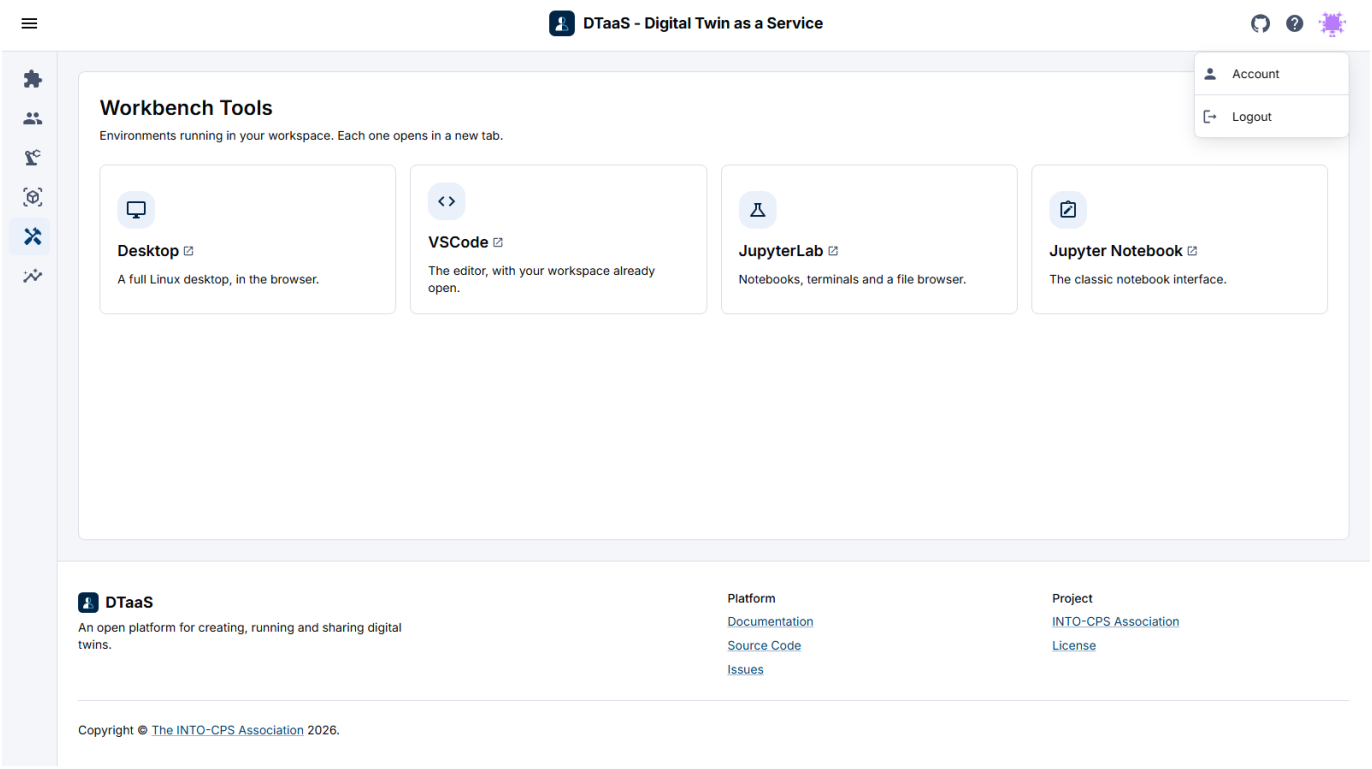
The **Insights** page has two cards. **Logs** opens the activity log this browser keeps once logging is enabled, and **Config** runs the configuration check.



Copyright © 2022 - 2026 The INTO-CPS Association

Finally logout

Click the avatar in the top-right corner and choose **Logout**.

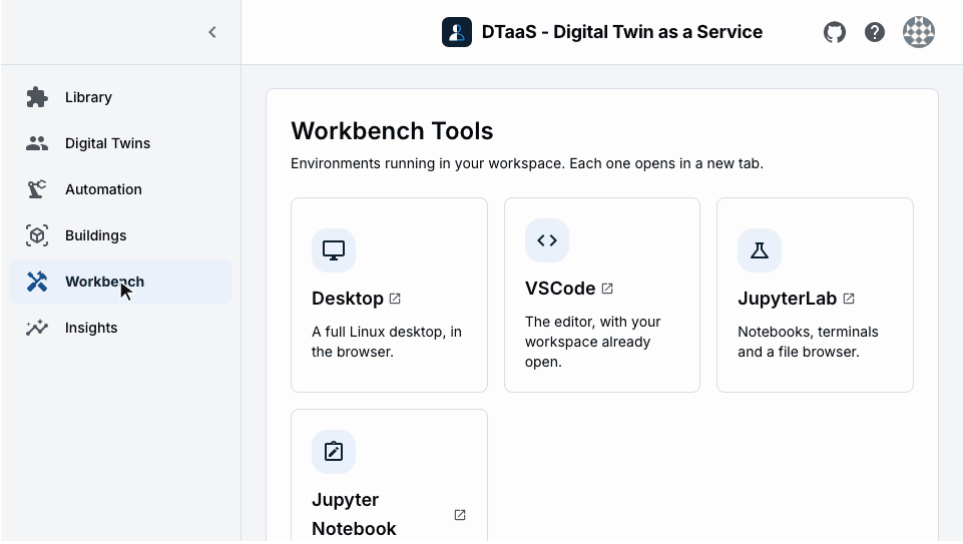


Close the browser to exit the DTaaS software platform completely.

Copyright © 2022 - 2026 The INTO-CPS Association

3.3.2 Workspace

This page walks through the website of the workspace software platform with screenshots. A short screen recording is also available.

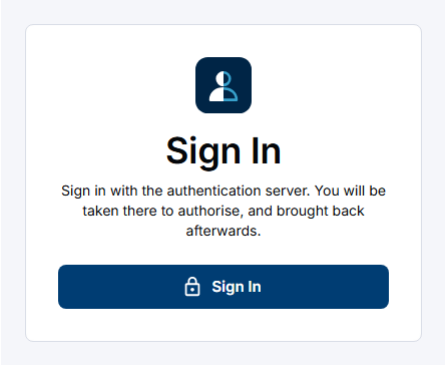


The user workspace in DTaaS is a personal Linux environment given to each registered user. This page uses `https://intocps.org` as an example base URL. Replace it with the URL of your installation.

Visit the Workspace Installation

URL: `https://intocps.org` (specified in `config/client.js` file)

Start by visiting the website of the DTaaS installation you are registered with.

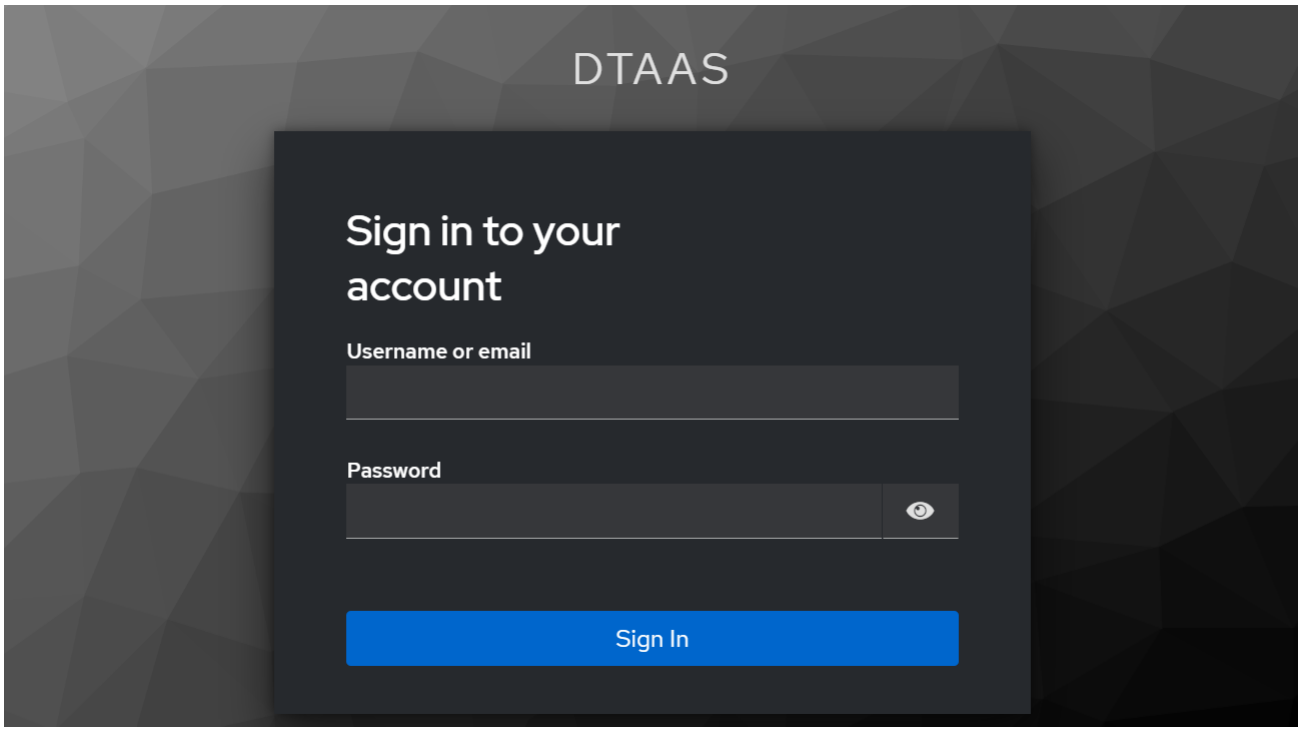


Redirected to Authorization Provider

URL: `https://intocps.org/auth/realms/dtaas/xxxx` (specified in `config/.env` file)

The browser redirects to the Keycloak authorization page for the DTaaS Workspace.

Copyright © 2022 - 2026 The INTO-CPS Association



Enter your email or username and password.

The browser then redirects to the application page.

After successful authentication, the browser returns to the sign-in page of the DTaaS website.

URL: `https://intocps.org`

The DTaaS website adds a second layer of security with the OIDC third-party authorization protocol. Only users with an active account at the selected OIDC provider can access the installation. This installation uses Keycloak as the OIDC provider. The **Sign In** button on the sign-in page starts the authorization.

After successful authorization, the browser opens the **Library** page of the DTaaS website.

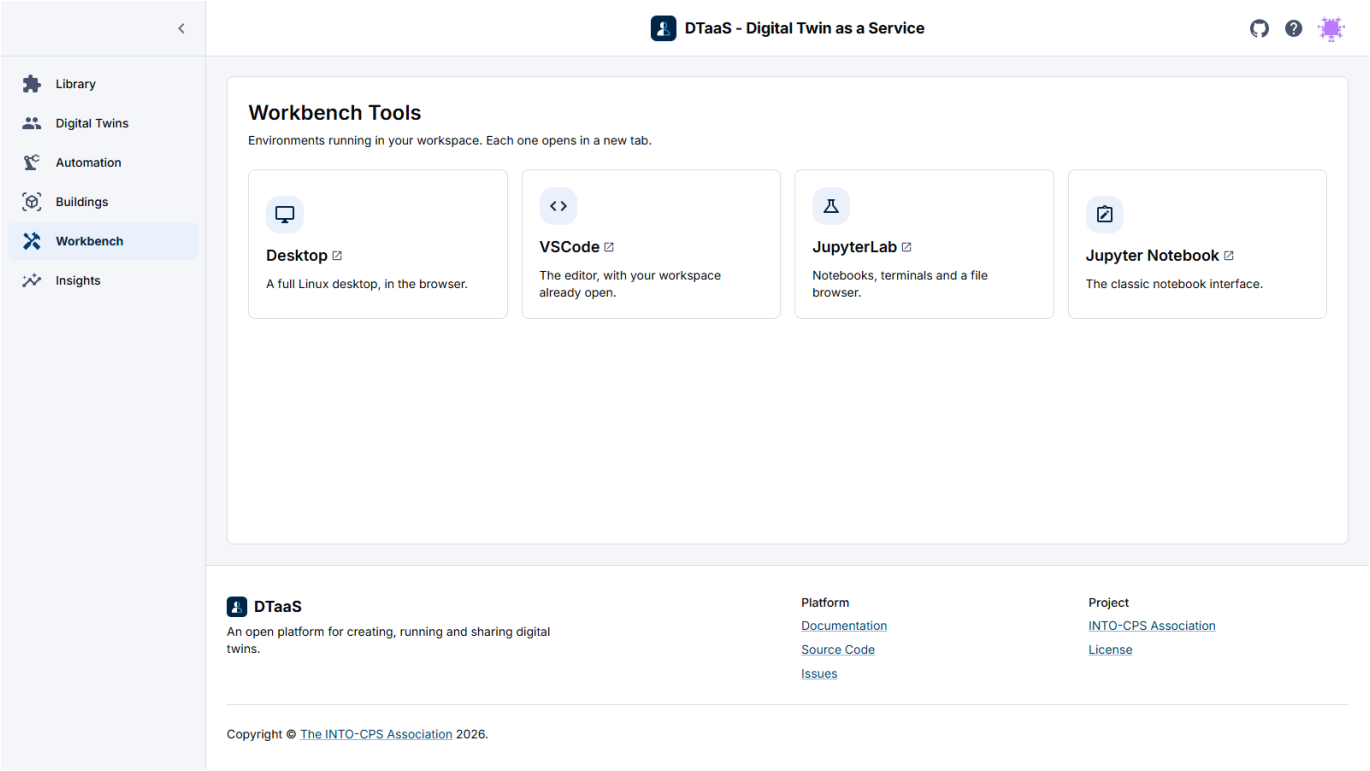
URL: `https://intocps.org/library/xxxx`

The application bar at the top of every page shows the DTaaS logo and title in the middle. On the right are three buttons. The GitHub icon opens the DTaaS code repository and the question-mark icon opens this documentation. The round avatar, which shows your profile picture or the first letter of your username, opens a menu with **Account** and **Logout**. The **Account** page has a **Settings** tab where you can change settings.

Menu Items

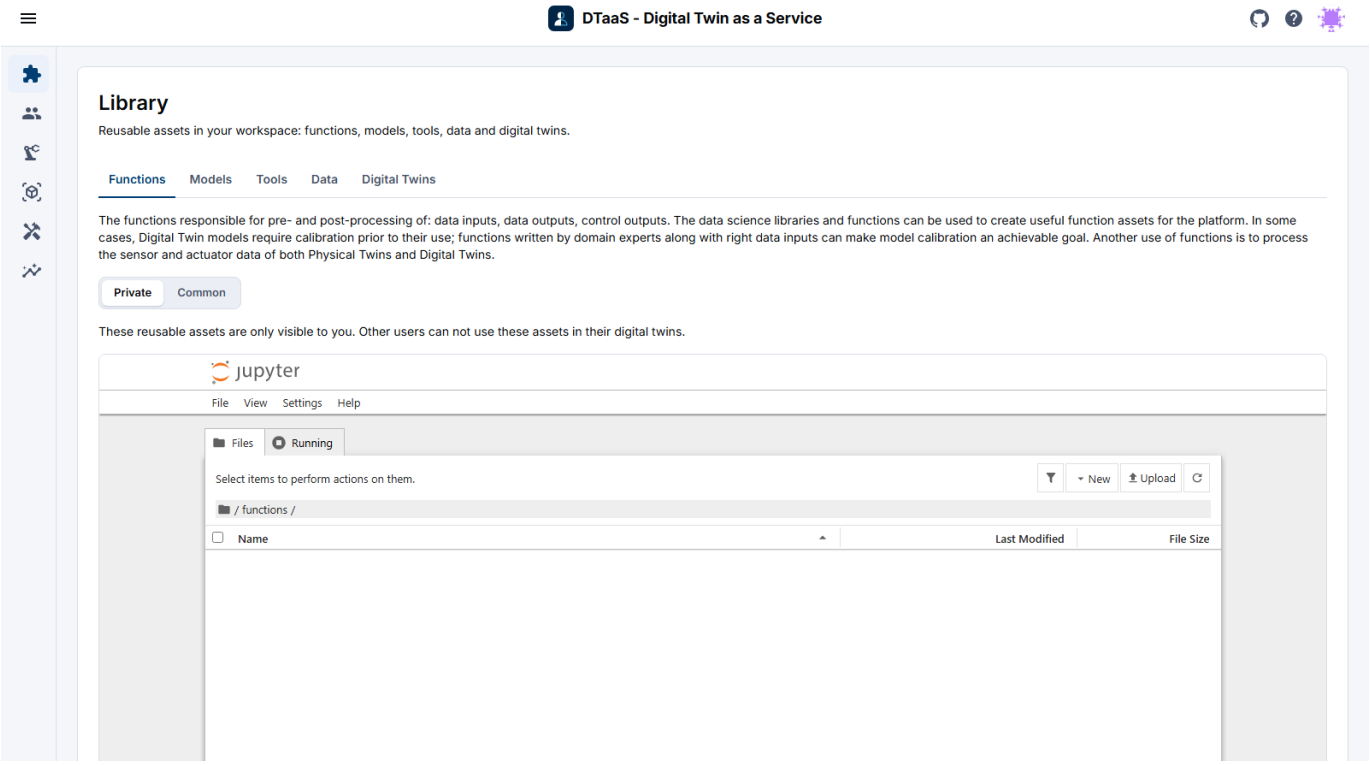
The menu is a sidebar on the left of the page. By default it is collapsed to a narrow rail that shows only the icons. Click the **three horizontal bars** icon in the top-left corner to expand it.

Copyright © 2022 - 2026 The INTO-CPS Association



The menu items are described on the website page.

Library Page



The page has five tabs, one for each type of digital twin asset. Each tab has a description of its asset type.

Copyright © 2022 - 2026 The INTO-CPS Association

Functions

The functions responsible for pre- and post-processing of data inputs, data outputs and 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 the right data inputs can make model calibration achievable. Functions can also process the sensor and actuator data of both Physical Twins and Digital Twins.

Models

The model assets 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 models that are useful for tracking the wear and tear of parts. The same robot can have behavioural models describing the safety guarantees provided by the robot manufacturer. The same robot can also have functional models describing the part manufacturing capabilities of the robot.

Tools

The software tool assets are software used to create, evaluate and analyse models. These tools run on top of a computing platform, i.e., an operating system, a virtual machine such as the Java virtual machine, or inside docker containers. Tools tend to be platform specific, which makes them less reusable than models. A tool can be packaged to run on a local or distributed virtual machine environment, so the most suitable execution environment can be selected for a Digital Twin. Most models require tools to evaluate them in the context of data inputs. In some cases executable packages are run as binaries in a computing environment. Each of these packages is a pre-packaged combination of models and tools that makes a ready-to-use Digital Twin.

Data

The data sources and sinks available to 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 visualisation software, external users and data storage services. There are also 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.

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.

Each asset tab has two sub-tabs: **Private** and **Common**. Private library assets are visible only to the signed-in user. Common library assets are available to all users.

The assets page explains where to place reusable assets of each type and the directory structure on the server.

Note

Assets (files) can be uploaded using the **upload** button.

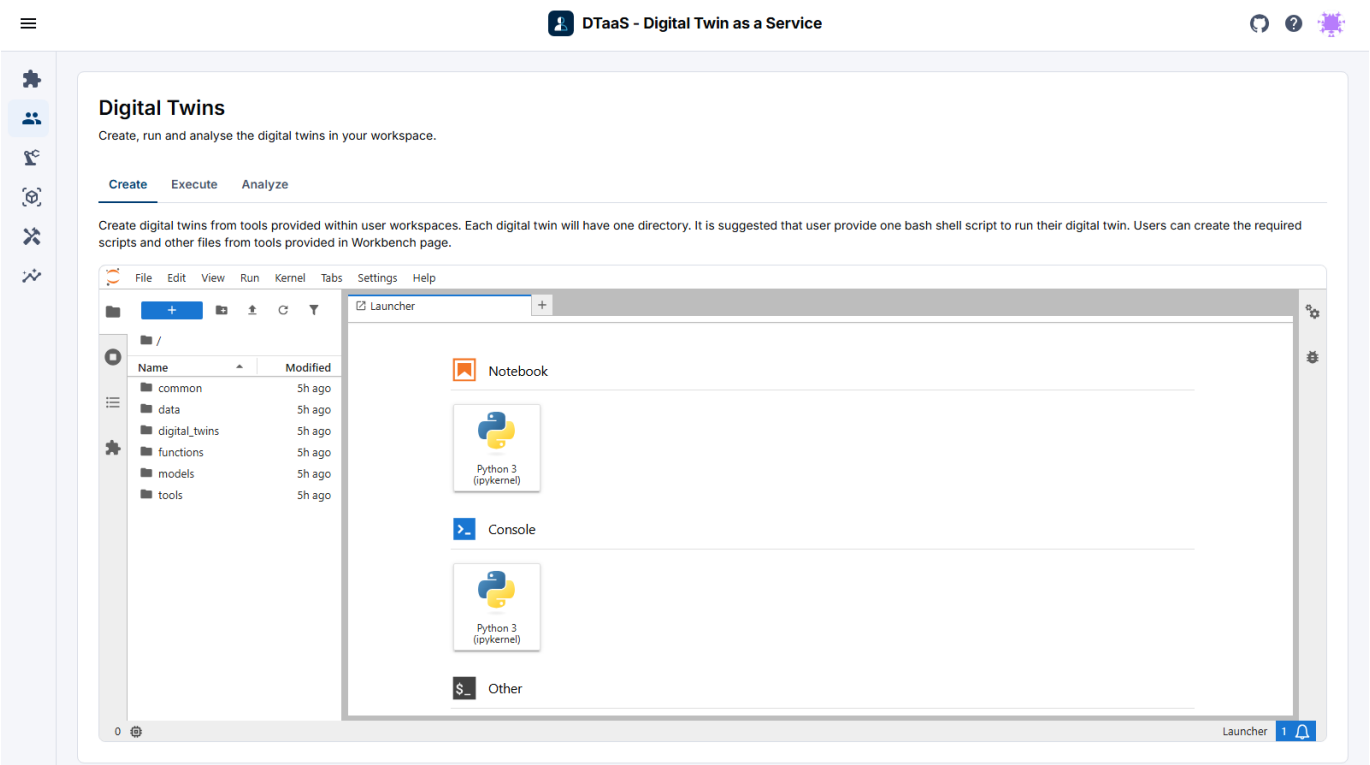
Note

The file manager is based on Jupyter Notebook, and all tasks available in Jupyter Notebook can be performed here.

Copyright © 2022 - 2026 The INTO-CPS Association

Digital Twins Page

URL: `https://intocps.org/digitaltwins`



The Digital Twins page has three tabs, **Create**, **Execute** and **Analyze**, and each tab shows Jupyter Lab below its description. The descriptions suggest tasks for these lifecycle phases of a digital twin. The lifecycle phases of digital twin page explains them further.

Create

Create digital twins from tools provided within user workspaces. Each digital twin has one directory. Users are advised to provide one bash shell script to run their digital twin. Users can create the required scripts and other files with the tools on the Workbench page.

Execute

Digital twins are executed from within user workspaces. The given bash script is executed from the 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

Analysing a digital twin requires running the digital twin script from the user workspace. Analysis scripts process the execution results placed in the data directory and put their results back in the data directory. These scripts can be executed from VSCode, or from VNC GUI when they produce graphical results.

Copyright © 2022 - 2026 The INTO-CPS Association

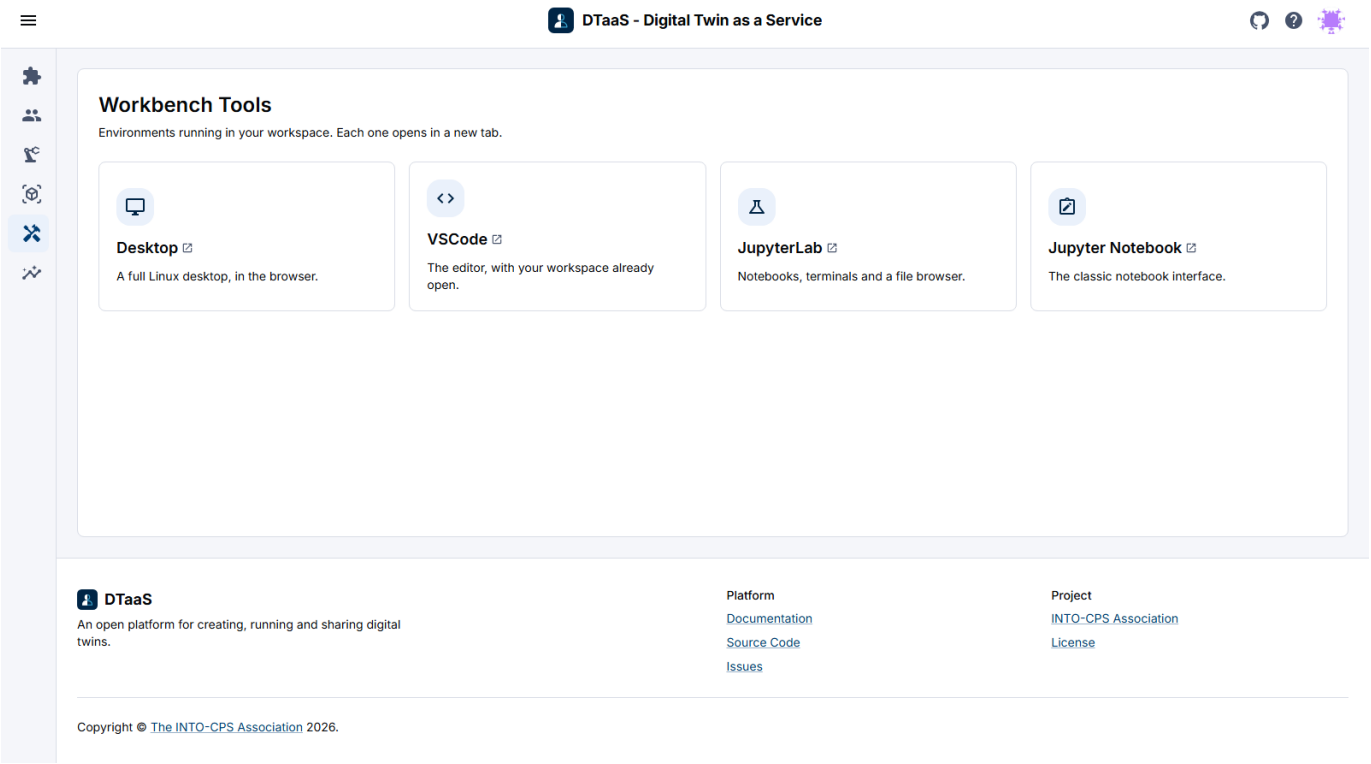
> **Note**
>
> The reusable assets (files) shown in the file manager are also available in Jupyter Lab. Jupyter Lab also has a git plugin for linking files with external git repositories.

Workbench

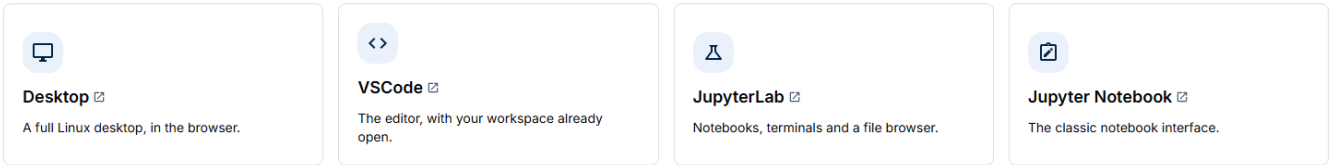
URL: `https://intocps.org/workbench`

The **Workbench Tools** page shows a card for each tool running in your workspace. A standard workspace has four tools:

Tool	Description
Desktop	XFCE-based Linux desktop environment in the browser
VSCode	Browser-based VS Code editor (code-server)
JupyterLab	Interactive notebook environment
Jupyter Notebook	Classic Jupyter interface



Each card opens its tool in a new browser tab. The sections below show the page each tool opens.

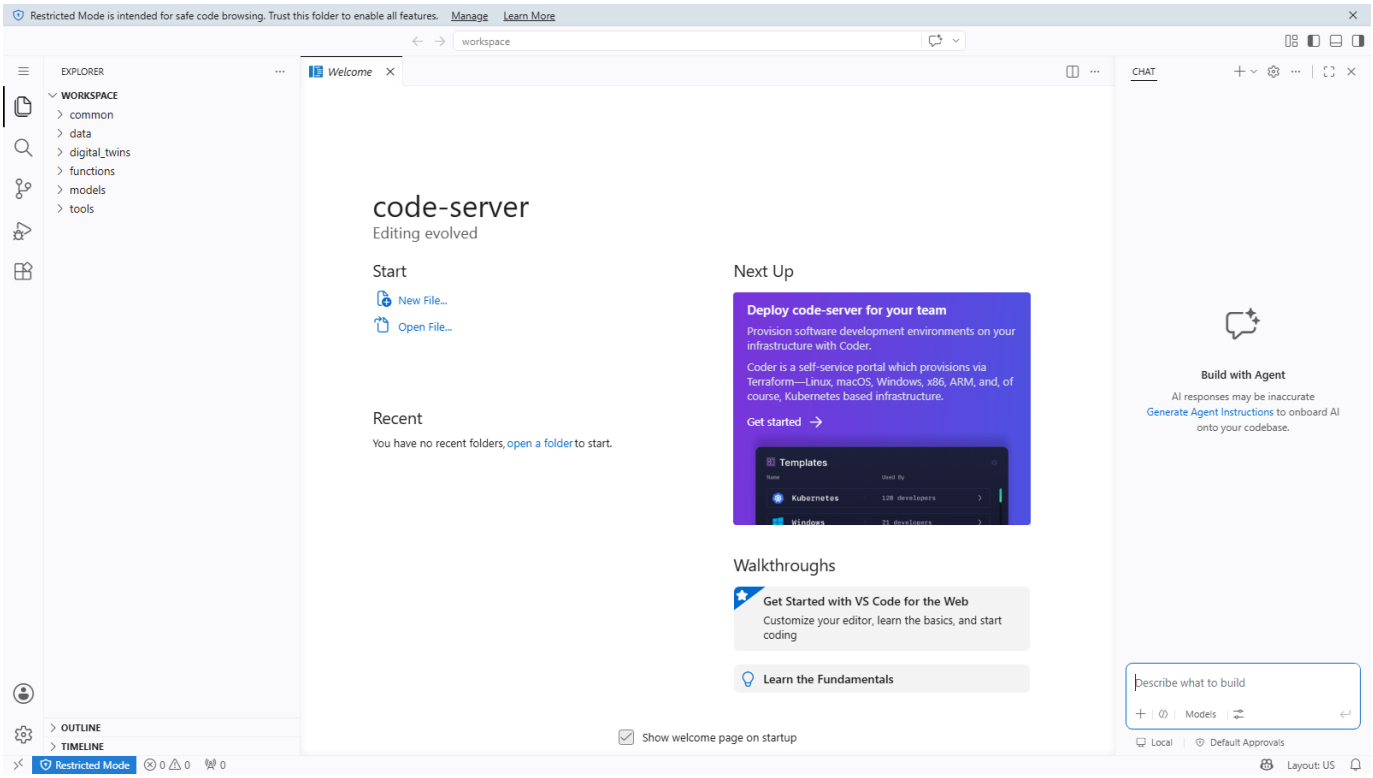


Copyright © 2022 - 2026 The INTO-CPS Association

VS Code Workspace

The VS Code interface has:

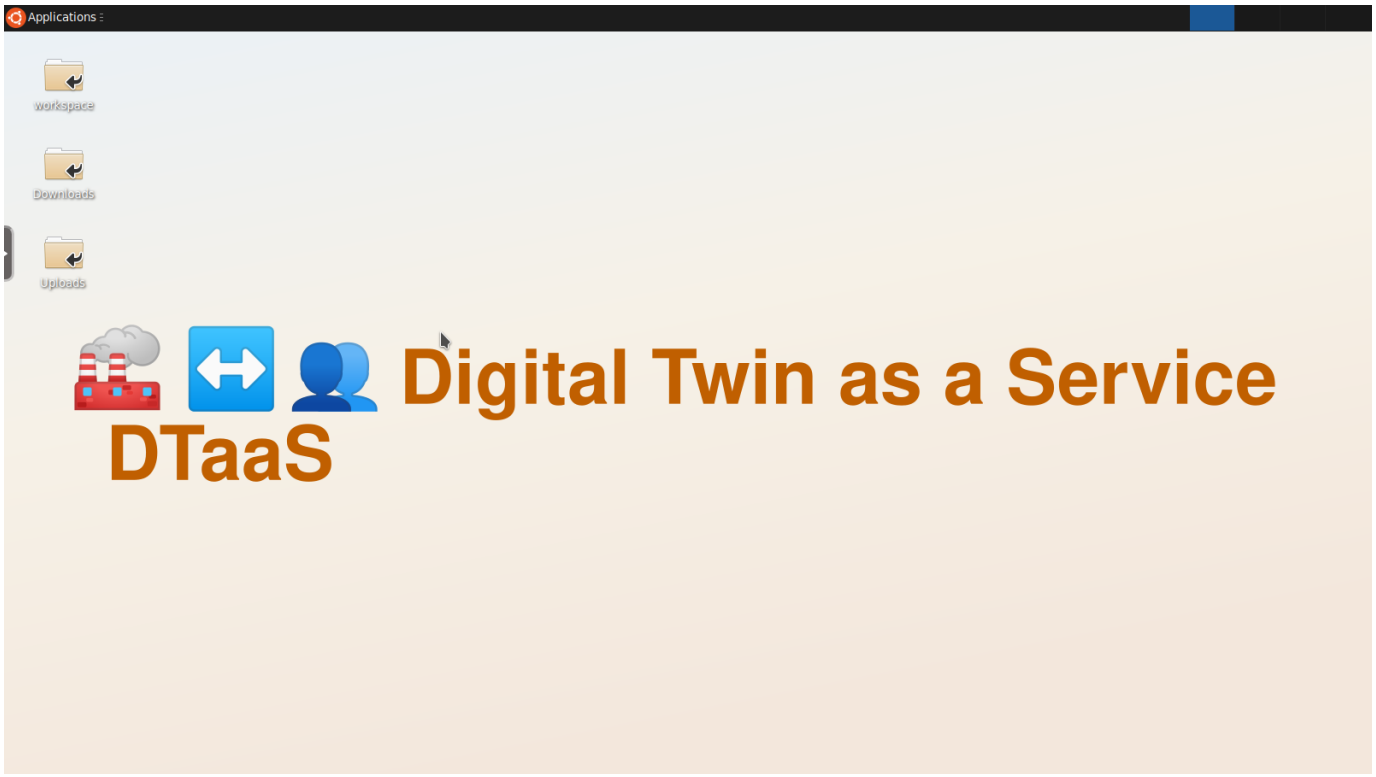
- File explorer for navigating the workspace
- Integrated terminal for running commands
- Extensions for Python, Markdown, and other languages



XFCE Desktop

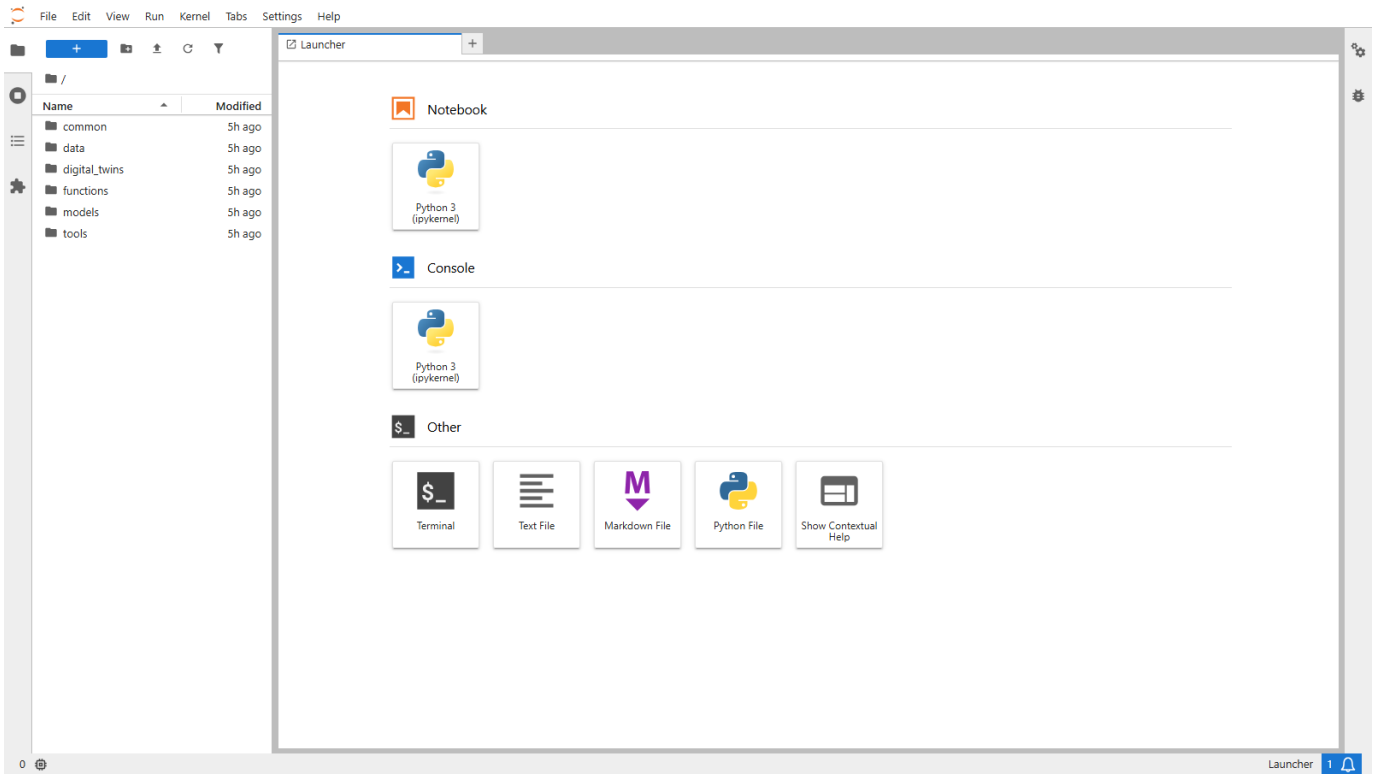
The XFCE desktop is a full graphical Linux environment in the browser. Use it to run GUI-based tools that cannot be used from a terminal alone.

Copyright © 2022 - 2026 The INTO-CPS Association



JupyterLab

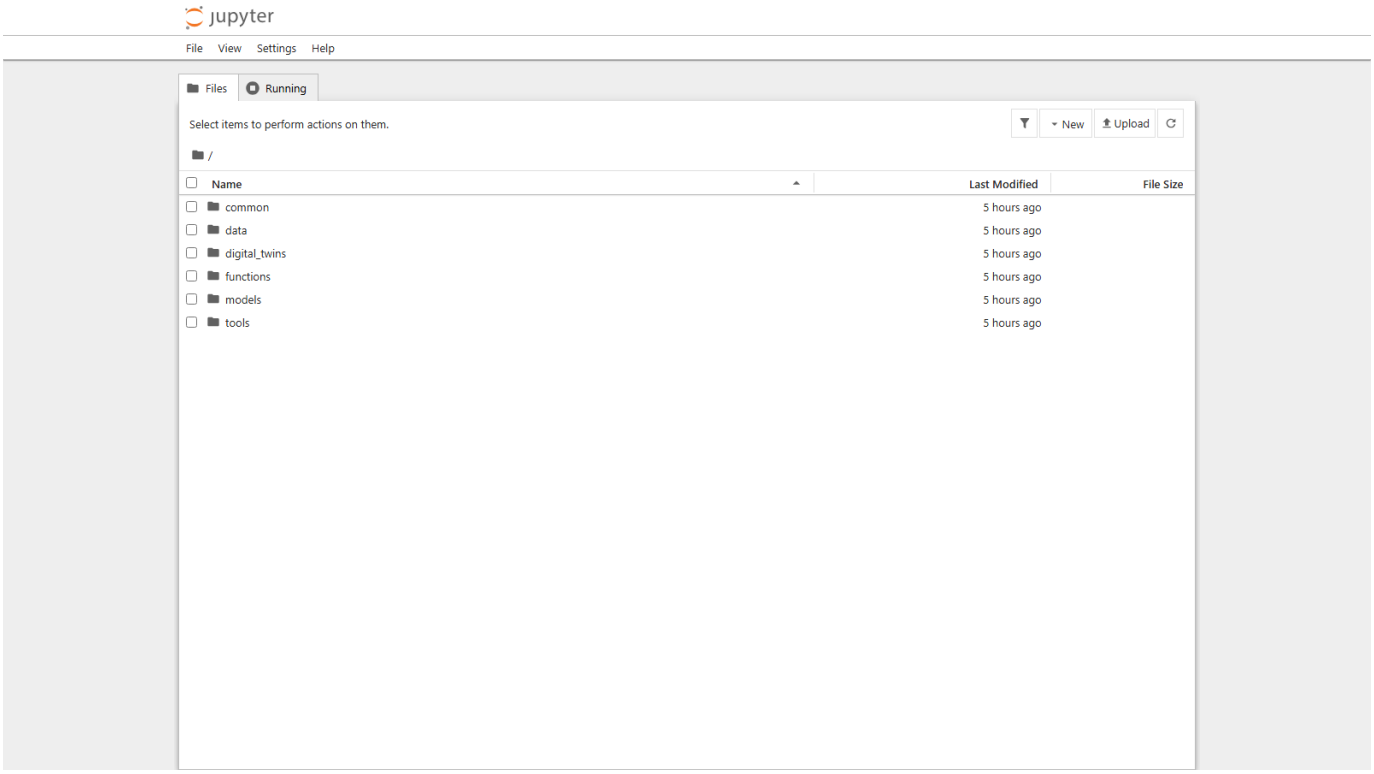
JupyterLab combines notebooks, terminals and a file browser in one window. The **Digital Twins** page embeds the same interface.



Copyright © 2022 - 2026 The INTO-CPS Association

Jupyter Notebook

The classic notebook interface opens on the file list of the workspace. The **Library** page embeds the same interface.



Workspace File System

Each user's workspace container mounts two directories:

```
/workspace/           # Private user files (read-write)
├── data/
├── digital_twins/
├── functions/
├── models/
└── tools/

/workspace/common/    # Shared library assets
├── data/
├── digital_twins/
├── functions/
├── models/
└── tools/
```

Files in `/workspace/` are private to each user. Files in `/workspace/common/` are shared across all users. Administrators can mount `/workspace/common` as read-only in the compose configuration.

Installing Software

Users can install more software in their workspace from the XFCE Desktop, Jupyter Lab or VSCode:

```
sudo apt-get install <package>
pip install <package>
```

Installed software is private to each user's workspace and persists across sessions.

Copyright © 2022 - 2026 The INTO-CPS Association

Running Digital Twins

To execute a digital twin stored in `/workspace/digital_twins/`, run its lifecycle scripts in the terminal:

```
cd /workspace/digital_twins/<dt-name>
lifecycle/create
lifecycle/execute
```

Summary

The DTaaS workspace gives each user a private, persistent Linux environment with a desktop, a terminal and development tools in the browser. Users can install software, run digital twins and develop new DT assets in their workspace.

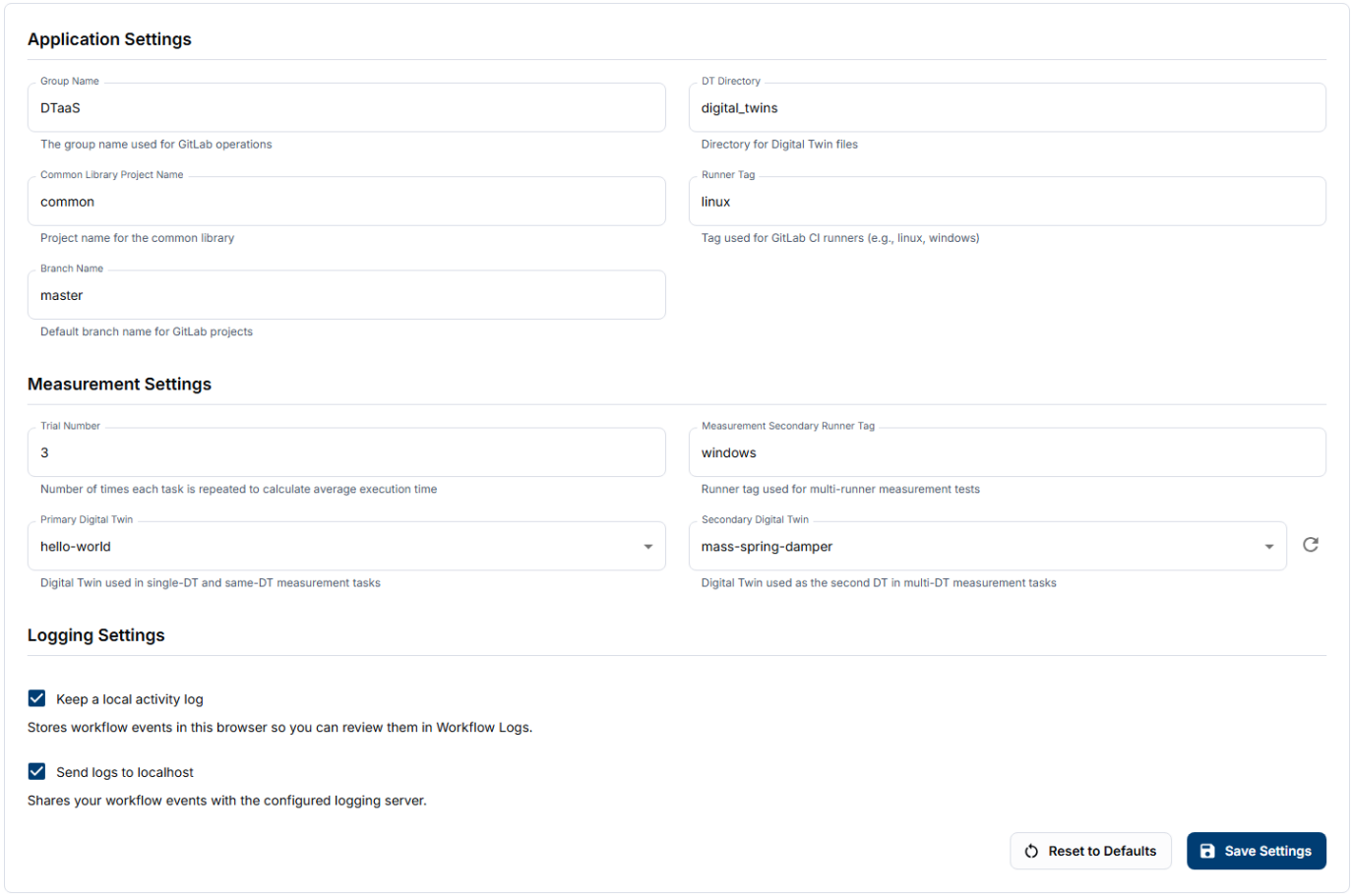
Copyright © 2022 - 2026 The INTO-CPS Association

3.3.3 Settings

You need the settings when you first set up the DTaaS and again whenever you work with a different configuration. The main settings are collected on one page for both cases.

Changing Settings

The parameters for sending information to and receiving it from the storage and execution services (e.g., GitLab) may need to be adjusted to match your infrastructure. Click the avatar in the top-right corner, choose **Account**, and open the **Settings** tab to see the adjustable parameters:



The settings are in three sections: **Application Settings**, **Measurement Settings** and **Logging Settings**. Application settings are used by the DevOps features. Measurement settings are used to benchmark the DevOps features. Logging settings control the activity log described on the Logs page.

Each parameter is described below.

APPLICATION SETTINGS

Group Name

The Group Name is the group on the storage service, the highest level of organisation there. The DTaaS requires a GitLab group. The group holds projects whose names must match the usernames of system users. See the file organisation for details. Set this parameter to the name of the group; it is case-insensitive.

Default: DTaaS

Copyright © 2022 - 2026 The INTO-CPS Association

Common Library Project Name

One project in the group is the Digital Twins *Library*. All users can access the files in the Library through the DTaaS and copy them to their own projects as needed. This parameter must match the project name of the Library.

Default: common

DT Directory

In the common library and in user projects, files related to Digital Twins are stored in a designated folder. This parameter is the name of that folder.

Default: digital_twins

Branch Name

This parameter sets the branch that is searched for data (Twins, Functions, etc.) in user and library projects. Digital Twins are also executed from this branch.

Default: master

Runner Tag

The (GitLab) runners that execute Digital Twin code must have a tag. Only one tag can be given, and it **cannot** be left blank; otherwise the job that runs the twin is not processed. This is a limitation of the DTaaS.

Default: linux

MEASUREMENT SETTINGS

Trial Number

The number of times each task is repeated during a benchmark run. The execution times are averaged over the trials. The value must be at least 1.

Default: 3

Measurement Secondary Runner Tag

Some benchmark runs require two GitLab runners. The *Runner Tag* in **Application Settings** points to the first runner, and the *Measurement Secondary Runner Tag* points to the second runner.

Default: windows

Primary Digital Twin

Select the first digital twin from the drop-down list of available digital twins. It is used in the single digital twin and same digital twin measurement tasks.

Default: `hello-world`, or the first digital twin in the list if `hello-world` is not available

Secondary Digital Twin

Select the second digital twin from the drop-down list of available digital twins. It is used as the second digital twin in the multiple digital twin measurement tasks. The refresh button next to the list reloads the available digital twins using the **Branch Name** and **DT Directory** entered above.

Default: `mass-spring-damper`, or the first digital twin in the list if `mass-spring-damper` is not available

Logging Settings

KEEP A LOCAL ACTIVITY LOG

The **Keep a local activity log** checkbox controls whether the DTaaS records workflow events in your browser (stored locally in IndexedDB). It is off by default.

Copyright © 2022 - 2026 The INTO-CPS Association

If the installation is configured with a remote logger, a second checkbox, **Send logs to** followed by the name of the logging server, also sends the events to that server. It is off by default as well. See Logs for viewing and downloading logs at `/insights/log`.

Saving and Resetting Changes

When you are satisfied with the changes, press **Save Settings** so that they persist after you leave the Settings page. To undo a mistake, press **Reset to Defaults** to restore the default values. The reset values are saved automatically, so you do not need to save them again. The two buttons are shown below:



The default values are defined in the code and can be changed there if needed.

You can now return to the DevOps pages (i.e., the Digital Twins Preview page). When you change the Group Name, Common Library Project Name, DT Directory or Branch Name, saving fetches the digital twins again from the remote repository. The Digital Twins should then be visible and ready to be executed, edited and shared.

Summary

This page described how to edit the settings, both to initialise the DTaaS for a project and during continuous use (for example, to change the Runner Tag or Branch Name). It also covered saving changes and restoring the default values after a mistake.

Copyright © 2022 - 2026 The INTO-CPS Association

3.3.4 Logs

The DTaaS website can record how you use it: page navigation, button and link clicks, form changes and notifications. The events are captured in the browser and can be reviewed on the **Workflow Logs** page.

Enabling Logging

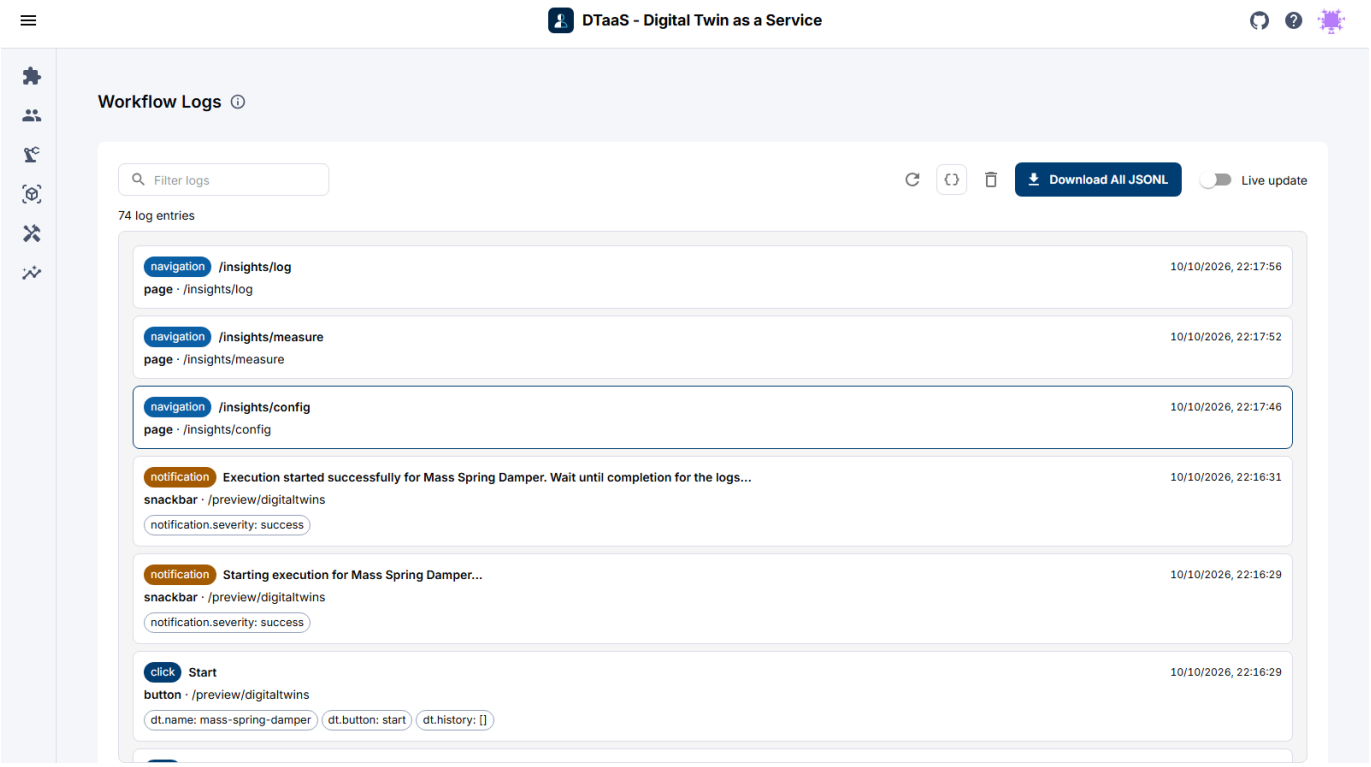
Logging is off by default. To turn it on, select the **Keep a local activity log** checkbox in the **Logging Settings** section of the Settings page (**Account > Settings**). Nothing is stored in the browser until this checkbox is selected, so the pages described below stay empty until then.

If the DTaaS installation is configured to stream events to a remote logging server, the Settings page shows a second checkbox, **Send logs to** followed by the server name. It is also off by default and works independently of the local log. Keep remote logging on only with the consent of the person whose activity is being recorded.

Viewing the Logs

URL: `https://intocps.org/insights/log` or `http://localhost:4000/insights/log`

Open the logs from **Insights** in the menu and then the **Logs** card, or go to `/insights/log` on the DTaaS installation. The page is titled **Workflow Logs**. The ℹ icon next to the title shows a short description of the page.



Copyright © 2022 - 2026 The INTO-CPS Association

The toolbar above the log list has these controls:

Control	Description
Filter logs	Shows only the entries that match the search text (see Filtering).
Refresh	Reloads the log entries from local storage.
Raw view {}	Switches between the card view and the raw JSON Lines view.
Clear logs	Permanently deletes all stored log entries, after confirmation.
Download All JSONL	Exports the displayed entries as a `.jsonl` file.
Live update	Streams new events into the view as they are recorded.

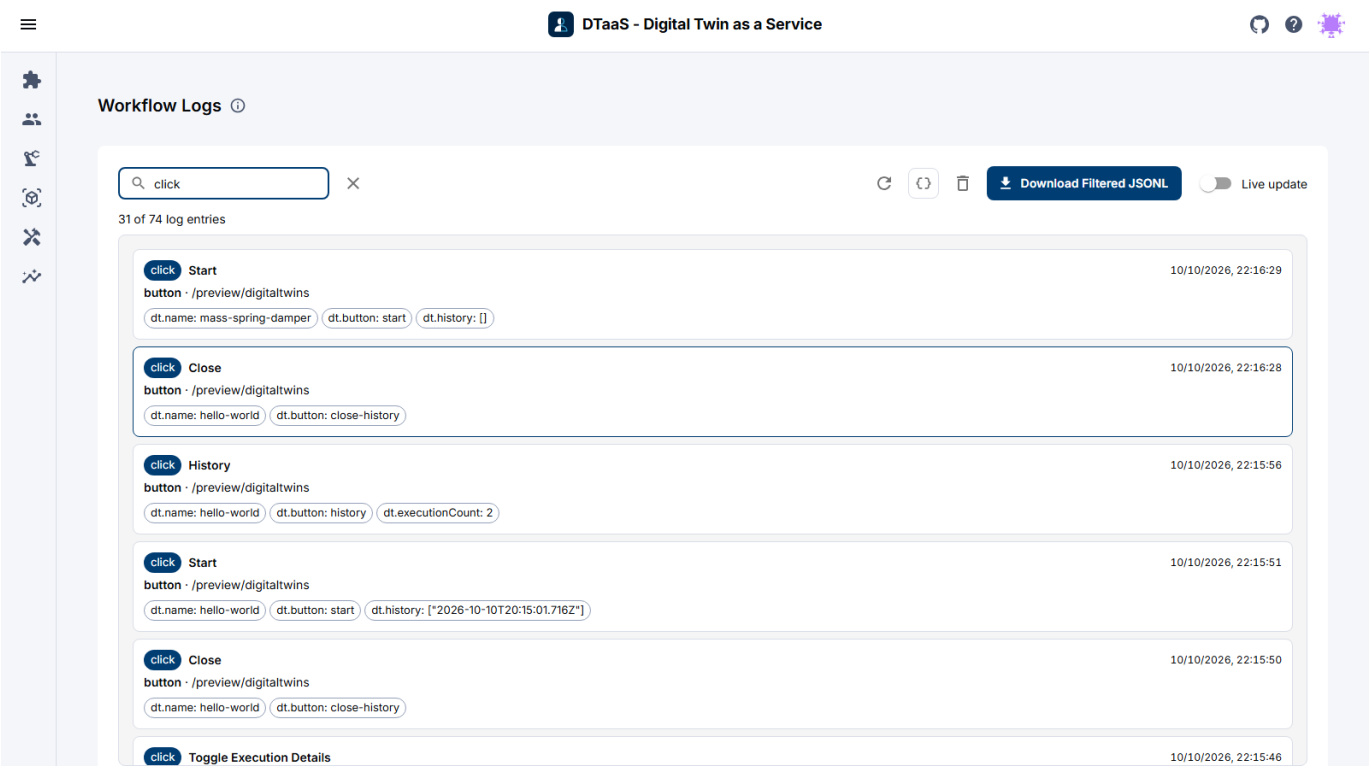
Log Entries (Card View)

Each entry is shown as a card with:

- A coloured chip for the event type: `click`, `change`, `navigation`, `notification` or `dismiss`.
- The label of the element that triggered the event.
- The timestamp of the event.
- The element type and the path of the page where the event occurred.
- Any extra context recorded for the event, shown as small chips (for example a link's target URL or the new value of a form field).

Filtering Logs

Type into the **Filter logs** box to show only the entries whose event type, label, element, page or context contains the search text. The count above the log list shows how many of the entries match, for example *3 of 10 log entries*.



While a filter is active, the **Download All JSONL** button becomes **Download Filtered JSONL** and exports only the entries shown.

Copyright © 2022 - 2026 The INTO-CPS Association

Raw View

Raw view shows the same entries as pretty-printed JSON Lines instead of cards and adds a **Copy to clipboard** button. To keep the page responsive, both views render at most the newest 500 entries and show a note when more exist. The download button and **Copy to clipboard** ignore this limit and always cover every displayed entry (that is, every entry that matches the filter, if there is one).

Each line is a single JSON object. Entries are shown newest first, so a click on **Digital Twins** in the menu followed by the page navigation it causes appears as:


```
{
  "timestamp": "2026-07-22T20:19:02.736Z",
  "event": "navigation",
  "label": "/digitaltwins",
  "element": "page",
  "page": "/digitaltwins",
  "page_transition": {
    "src": "/library",
    "target": "/digitaltwins"
  },
  "context": {},
  "sessionId": "eaba3642-324e-4f3b-bd21-4204ddd10ebe",
  "userHash": "dee55bf8057abd8211bf80d6513577ff260e739d627c20762d9d617a393a89db"
}

{
  "timestamp": "2026-07-22T20:19:01.245Z",
  "event": "click",
  "label": "Digital Twins",
  "element": "nav-link",
  "page": "/library",
  "context": {
    "nav": {
      "link": "/digitaltwins",
      "active": false
    }
  },
  "sessionId": "eaba3642-324e-4f3b-bd21-4204ddd10ebe",
  "userHash": "dee55bf8057abd8211bf80d6513577ff260e739d627c20762d9d617a393a89db"
}
```


LOG EVENT FIELDS

Field	Description
`timestamp`	UTC time the event was recorded, in ISO 8601 format.
`event`	The event type: `click`, `change`, `navigation`, `notification` or `dismiss`.
`label`	A human-readable label for the element involved.
`element`	The type of UI element involved, e.g. `nav-link`, `button`, `link`, `page`.
`page`	The URL path of the page on which the event occurred.
`page_transition`	Present on `navigation` events after the first; records the source and target page.
`context`	Extra metadata specific to the event, e.g. a link's target URL.
`sessionId`	A random identifier generated for the current browser session.
`userHash`	A SHA-256 hash of the username, which links events without storing the name in plain text.

Live Updates

Live update subscribes the page to new log events as they are recorded elsewhere in the browser (including other open tabs) and reloads the list automatically. It is off by default, because reloading a large log store continuously is unnecessary when you are only reviewing past entries.

Clearing Logs

The icon opens a confirmation dialog, and **Clear Logs** in that dialog permanently deletes all stored log entries. This cannot be undone. It only clears the entries stored in the browser and has no effect on copies already sent to a remote logging server.

Copyright © 2022 - 2026 The INTO-CPS Association

Privacy and Storage

- Log entries are stored locally in the browser (IndexedDB) and never leave the device, unless the DTaaS installation is configured with a remote logging server and remote logging is turned on. Then the entries are also sent to that server.
- The local log store is limited to about 25 MB. The oldest entries are removed automatically once the limit is reached.
- The `userHash` field is a hash of the username, not the username itself. It pseudonymizes the user while still allowing their events to be linked together.
- The `sessionId` field is a random value generated per browser session and is not linked to the user's identity.

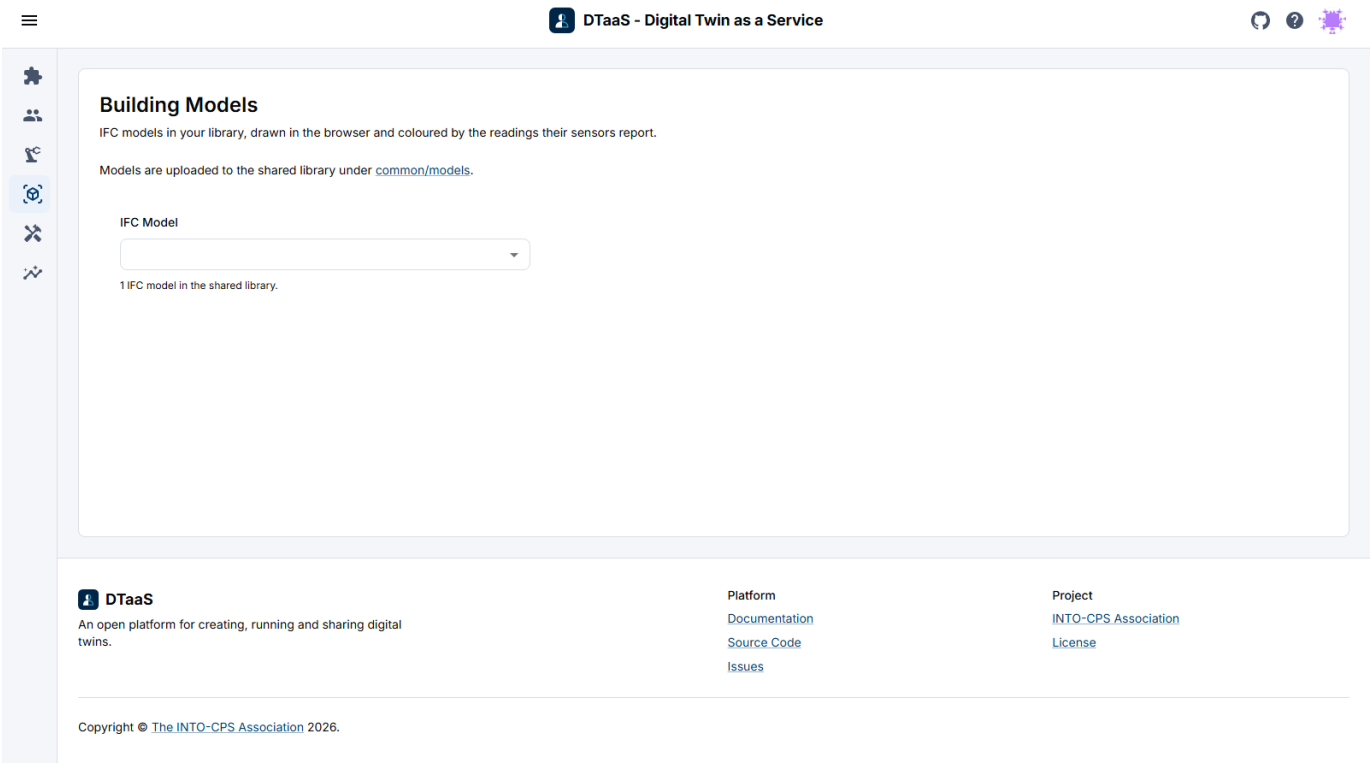
Summary

This page described the logging feature of the DTaaS website: turning it on in Settings, viewing and filtering events on the Workflow Logs page, switching between the card and raw JSON Lines views, downloading and clearing logs, and how the stored data protects privacy.

Copyright © 2022 - 2026 The INTO-CPS Association

3.3.5 Building Models

The Building Models page draws a building from the user's own library and colours it with the readings its sensors report, so you can see, for example, which of forty rooms is the warm one. A time-series dashboard cannot show that. Open it from **Buildings** in the menu.



This page describes the asset layout, the binding manifest and the conversion workflow. It is the user documentation for DTaaS issue 1762. The developer documentation, covering what the route contains and what it leaves out, is in `client/src/route/bim/README.md`.

What the Page Needs

The page reads up to four files per model, all in the user's library under `common/models`. The page links to that folder above the viewer.

File	Required	What it holds
`model.ifc`	yes	The building, as the authoring tool exported it
`model.glb`	no	The geometry converted from the IFC file. The page writes it the first time the model is converted
`model.json`	no	The property tree: every object, its IFC class, its storey, its property sets
`model.manifest.json`	no	The binding manifest, mapping each sensor to the object it measures

An IFC file on its own is enough to see the building. With the geometry file, the next visit loads the model instead of converting it again. The property tree adds the panel that opens when you click an object, and the manifest adds the readings. Nothing fails when the last three files are missing.

Copyright © 2022 - 2026 The INTO-CPS Association

How a Model Is Named

The IFC Model menu names each model by what its own file says the building is called: the project's long name, then its name, then the building's. Only the first 64 KB of each file is read for this. The menu skips template text that an authoring tool leaves behind, such as `Project Name`, and names that two files share. Those models are shown by their file name, with underscores read as spaces.

To give a name to a file that has none, use `ifc-set-name` from the ifc-utils repository. It writes the name into the file and changes nothing else. DTaaS stores nothing about the names.

The Geometry Format

The model is converted to glTF binary, `.glb`. The issue offered OBJ as an alternative. OBJ was rejected because it cannot carry a stable identifier per object, which is the property the issue itself relies on.

The binding key is the IFC `GlobalId`, a 22-character identifier that the schema expects to stay the same across exports of the same model. glTF carries arbitrary data per node in its `extras` field, so the converter writes the `GlobalId` there and the viewer resolves a binding by reading it back. OBJ has no such field. A marker on an OBJ model can only be placed by node name, and names are not unique. Two walls are often both called `Wall`, and the resolver refuses that case instead of guessing which one the reading belongs to. A model without a stable identity per object cannot carry a binding that someone can review.

The Binding Manifest

The manifest is plain JSON beside the model, and it is validated with `zod` before anything is drawn. People write it by hand and it will often contain mistakes, so every error names the binding and the field that caused it, for example `bindings[3].display.ramp`, instead of only reporting that the file is invalid.

```
{
  "model": {
    "source": "substation.ifc",
    "source_sha256": "60b1b94ee88a2b087080200c3c347edaf2347e768c1accb4b3608c6a129fbb09",
    "converter": "ifc_explorer.to_manifest 0.1.0"
  },
  "bindings": [
    {
      "selector": { "globalId": "0_sgz7bzz4Jh2ckU1ehFe$" },
      "label": "HX-1 temperature TS-01",
      "source": {
        "live": { "transport": "mqtt", "topic": "swim/B1/temperature/TS-01" },
        "history": { "bucket": "swim", "measurement": "temperature" }
      },
      "display": { "unit": "°C", "ramp": [10, 50] }
    }
  ]
}
```

Four design decisions shape the format:

- `selector` is a small union of `globalId`, `nodeName` and `expressId`, so a non-IFC asset can use the same manifest instead of a second schema. A selector names exactly one of the three. If it named two, the resolver would have to choose between them, and the result would depend on that choice.
- `live` and `history` are separate. MQTT supplies the current value shown on the model, and the time-series database supplies the panel that opens on a click. The 3D view does not reimplement charting.
- `source_sha256` is required. Without it there is no way to tell whether a derived artifact still describes the file beside it. The literal `unknown` is accepted, for a generator that never read the source, and anything else has to be a SHA-256.
- `display.ramp` is the range the colour scale spans, and the viewer also reads it as the range the sensor is expected to stay inside. A value outside it raises an alert on the sensor card.

A binding whose object is missing from the geometry is reported and counted. This happens when an object is deleted and drawn again, or when the exporting tool does not keep the `GlobalId` and writes a new one. If the manifest asks for six markers and only four objects are found, the viewer says which two are missing instead of silently drawing four.

Copyright © 2022 - 2026 The INTO-CPS Association

Conversion

Conversion runs in the browser of whoever opens the model. Putting an IFC file in the library is the whole workflow. You do not need a terminal or a separate service, and there is no step to repeat when the file changes.

WHY THE PARSER SHIPS INSIDE THE PACKAGE

The issue asked that the WebAssembly parser and its workers be served from the DTaaS origin, so the feature works in a localhost install with no external network, and it proposed a `postinstall` copy step to achieve that.

The converter package meets the same requirement differently. The `web-ifc` WebAssembly binary is embedded in the JavaScript as base64 at build time, so the published package contains it and there is no separate `.wasm` file to serve. A production build of the client contains no `.wasm` files and fetches none.

This was chosen over the copy step for three reasons.

First, there is no extra step to forget. A `postinstall` copy sits outside the dependency graph. It has to be present in the host's build and survive every change to that build. When it is missing, the failure shows up at runtime in somebody else's deployment as a parser that cannot load. An embedded binary either installs or does not.

Second, it saves a request and a content type. Serving `.wasm` requires the right MIME type from the reverse proxy in front of the deployment, and a Content Security Policy that permits `wasm-unsafe-eval`. Both are configured in files this project does not own. A base64 payload inside a JavaScript chunk is served as JavaScript, which every deployment already handles correctly.

Third, the cost is paid only when needed. Base64 is one third larger than the binary it encodes. That extra size stays out of the main bundle: the parser is in a chunk that is fetched the first time a model has to be converted, and a user who never opens the page never downloads it. The entry chunk of a production build has no reference to the renderer or the parser.

The result is a larger download for the one page that needs it, in exchange for a deployment step that cannot be done wrong. DTaaS is installed on localhost and in air-gapped environments, where the second matters more.

CONVERTING ONCE INSTEAD OF ON EVERY VISIT

When a model has no `.glb` beside it, the page converts it in the browser and writes the result back into the library, next to the `.ifc`. The next visit finds the file and loads it, so only the first person to open the model waits for the conversion.

The write goes through the workspace's own Jupyter Contents API, the same interface the Library page reads through. It writes in pieces of 512 KB, because the server in front of Jupyter refuses a request body of a megabyte. The write has limits: the destination has to be a single file directly inside `common/models`, and an address that already holds a file is left alone, so a geometry produced outside the browser is never replaced. When the page cannot tell whether a file is there, it does not write.

The bytes are written to `model.glb.part`, and the file takes its real name only when the last piece has landed. A write that fails, or that stops because the person left the page, deletes its `model.glb.part`. If the `.part` file could not be deleted because the tab was closed, it stays visible in JupyterLab beside the model. It is safe to delete, and the next conversion of that model replaces it anyway.

If the server in front of the workspace refuses a piece as too large, the write starts over once in pieces of half the size.

A failed write shows nothing on the page, by design: a model that is not stored is simply converted again on the next visit. One case is worth knowing about. On a workspace with Jupyter XSRF protection enabled, the token is in an `HttpOnly` cookie that scripts cannot read, so every write is refused. The failure is written to the browser console.

For a large model it is still better to produce the `.glb` outside the browser with `ifc-to-glb` from the ifc-utils repository, as the administrator documentation recommends. The browser conversion covers the case where nobody has done that.

RECORDING WHERE AN ARTIFACT CAME FROM

The manifest records the hash of the source it was made from and the name and version of the converter that made it, and the schema refuses a manifest without them. This makes it possible to detect a stale manifest. A `.glb` does not carry that record yet, whether `ifc-to-glb` wrote it or the browser did. It names the tool that generated it and says nothing about the source.

Copyright © 2022 - 2026 The INTO-CPS Association

Live Readings

The viewer draws whatever readings it is given and does not subscribe to anything. It knows neither the broker nor the topic scheme, so the same component can serve a demonstration fed by a local publisher and a deployment fed by the platform.

How readings reach the client is not decided yet. There are two options, and they are not equivalent:

- The client subscribes to MQTT over WebSocket against the platform's RabbitMQ, through the `rabbitmq_web_mqtt` plugin, and passes the latest value to the viewer.
- A platform service, the MQTT Agent described in the DTaaS material, subscribes, validates, stores, and serves the latest value to the client.

The second option separates concerns better, but it depends on that service existing and running, which has not been confirmed. Until this is settled the route shows models without readings. That is already the correct behaviour for a model that declares no sensors, and most architectural models declare none.

What Is Not Built Yet

- History on click: the manifest already carries the `history` block, with bucket, measurement and tags, so the Grafana address can be built from it. Opening that panel from a marker is not implemented.
- Federation: viewing several discipline models together as one building is not supported. Each model is drawn on its own.

Copyright © 2022 - 2026 The INTO-CPS Association

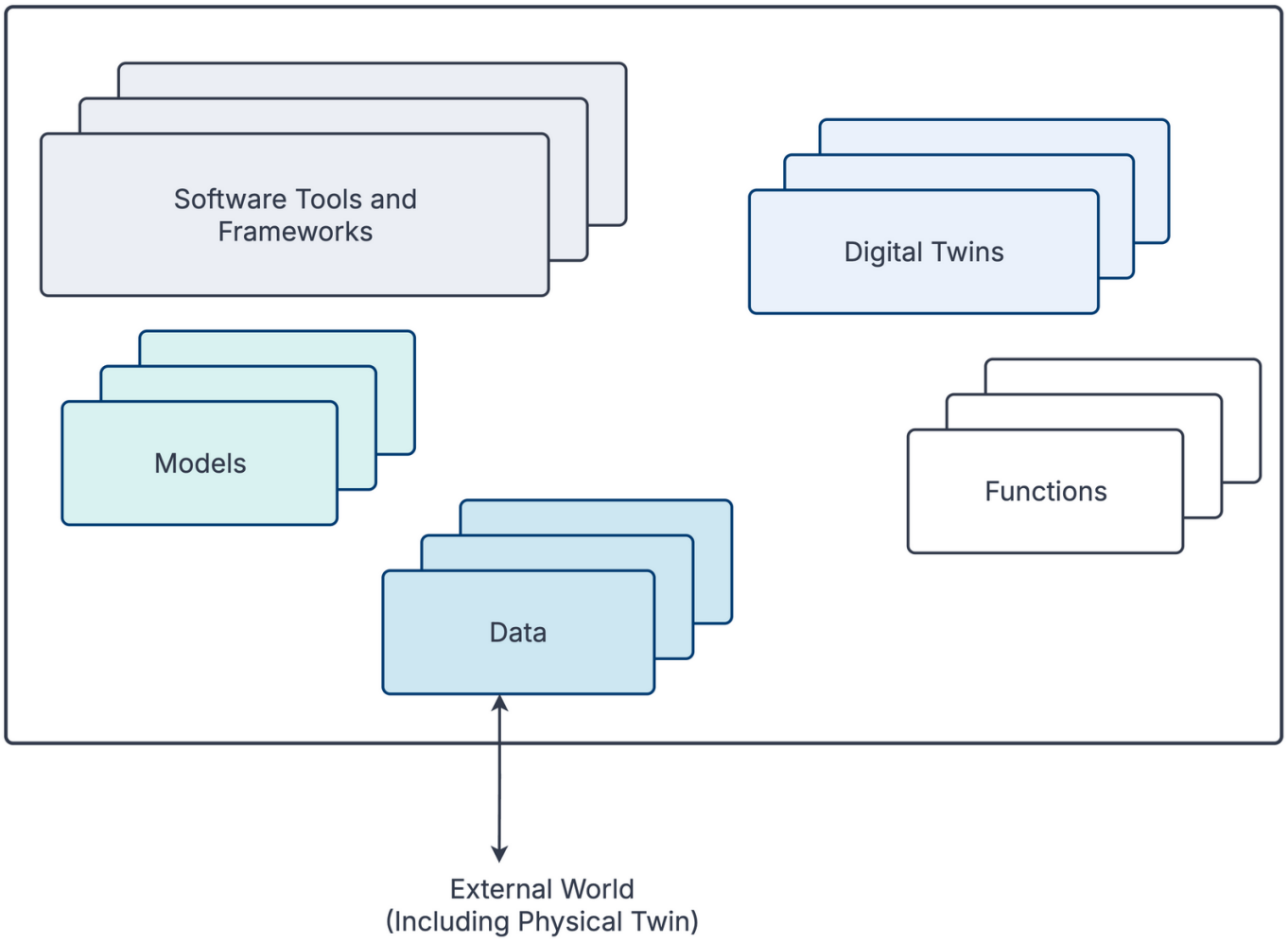
3.4 Reusable Assets

3.4.1 Reusable Assets

Reusable assets make it easier to work with digital twins, and reusability of assets is a fundamental feature of the platform[1].

Kinds of Reusable Assets

The DTaaS platform sorts all reusable library assets into five categories:



DATA

The data sources and sinks available to a digital twin. Typical data sources are sensor measurements from Physical Twins and test data provided by manufacturers for calibrating models. Typical data sinks are visualisation software, external users and data storage services. Some outputs, such as events and commands, 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

Model assets describe different aspects of Physical Twins and their environment, at different levels of abstraction, so the same Physical Twin can have several models. For example, a flexible robot used in a car production plant may have structural models that help track the wear and tear of parts. The same robot can have behavioural models describing the safety guarantees provided by the robot manufacturer, and functional models describing what parts the robot can manufacture.

Copyright © 2022 - 2026 The INTO-CPS Association

TOOLS

Tool assets are software used to create, evaluate and analyse models. These tools run on a computing platform, such as an operating system, a virtual machine like the Java virtual machine, or a docker container. Tools tend to be platform specific, which makes them less reusable than models. A tool can be packaged to run on a local or distributed virtual machine, so the most suitable execution environment can be chosen for a Digital Twin. Most models need tools to evaluate them against data inputs. In some cases, executable packages run as binaries in a computing environment. Each of these packages combines models and tools into a ready-to-use Digital Twin.

FUNCTIONS

Functions pre-process and post-process data inputs, data outputs and control outputs. Data science libraries and functions can be used to create function assets for the platform. Some Digital Twin models need calibration before use; functions written by domain experts, together with the right data inputs, can calibrate them. Functions can also process the sensor and actuator data of both Physical Twins and Digital Twins.

DIGITAL TWINS

Ready-to-use digital twins created by one or more users. They can be reconfigured later for specific use cases.

File System Structure

Each user's assets are stored in five directories, one for each category above. In addition, common library assets are available to all users. A simplified example of the structure is:

```
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)
      config (yaml and json)
      README.md (usage instructions)
      description.md (short summary of digital twin)
      lifecycle/ (directory containing lifecycle scripts)
    digital_twin-2/ (ex: mass spring damper)
      config (yaml and json)
      README.md (usage instructions)
      description.md (short summary of digital twin)
      lifecycle/ (directory containing lifecycle scripts)
    digital_twin-3/ (ex: model swap)
      config (yaml and json)
      README.md (usage instructions)
      description.md (short summary of digital twin)
      lifecycle/ (directory containing lifecycle scripts)
    ...
  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/
```


Copyright © 2022 - 2026 The INTO-CPS Association

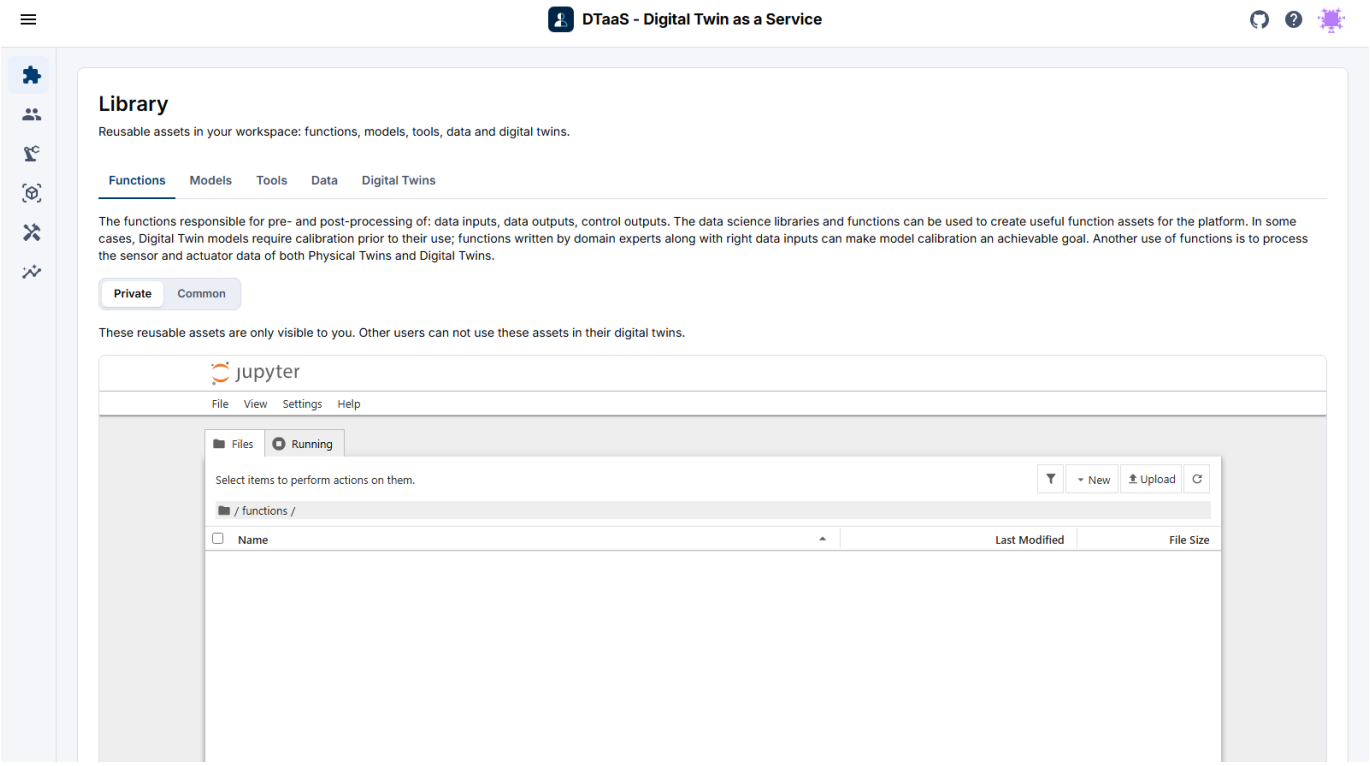
Tip

The DTaaS is agnostic to the format of assets. The only requirement is that they are files that can be uploaded on the Library page. Directories can be compressed into single files and uploaded, then decompressed from a terminal or the 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 from the Library page of the website.



Open a directory and click the **upload** button to upload files or directories into the workspace. The uploaded assets are then available in all workbench tools. The **new** dropdown menu creates new text-based files on the page. Other files must be uploaded with the **upload** button.

The user workbench provides the following services:

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

Users can also bring DT assets into their workspaces from external sources with any of these services. Developers using *git* repositories can clone from and push to remote git servers. Users can also use common file transfer protocols such as FTP and SCP to bring DT assets into their workspaces.

References

[1]: Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

Copyright © 2022 - 2026 The INTO-CPS Association

3.4.2 Library Microservice

The lib microservice serves the library assets of the DTaaS platform. Clients use its API endpoints to list and fetch these assets.

This page explains how to use the library microservice.

The assets page describes the suggested storage conventions for library assets.

Assets stored in the library are available in the user workspace.

Application Programming Interface (API)

The lib microservice provides services at two endpoints:

GraphQL API Endpoint: `http://intocps.org/lib`

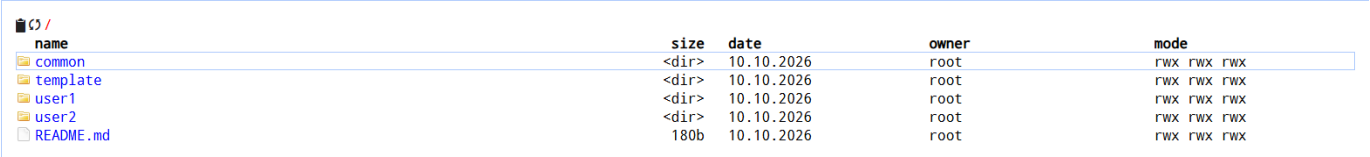
HTTP Endpoint: `http://intocps.org/lib/files`

HTTP PROTOCOL

Endpoint: `localhost:PORT/lib/files`

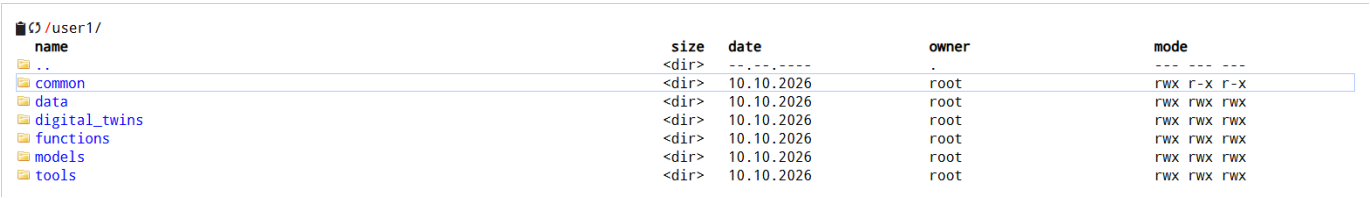
Enable this endpoint with the `-H http.json` flag. It provides the usual file upload and download options.

Sample screenshots:



/

name	size	date	owner	mode
common	<dir>	10.10.2026	root	rwx rwx rwx
template	<dir>	10.10.2026	root	rwx rwx rwx
user1	<dir>	10.10.2026	root	rwx rwx rwx
user2	<dir>	10.10.2026	root	rwx rwx rwx
README.md	180b	10.10.2026	root	rwx rwx rwx



/user1/

name	size	date	owner	mode
..	<dir>	--.--.----	.	--- --- ---
common	<dir>	10.10.2026	root	rwx r-x r-x
data	<dir>	10.10.2026	root	rwx rwx rwx
digital_twins	<dir>	10.10.2026	root	rwx rwx rwx
functions	<dir>	10.10.2026	root	rwx rwx rwx
models	<dir>	10.10.2026	root	rwx rwx rwx
tools	<dir>	10.10.2026	root	rwx rwx rwx

GRAPHQL PROTOCOL

Endpoint: `localhost:PORT/lib`

The `http://intocps.org/lib` URL opens GraphiQL, a GraphQL web page.

GraphiQL shows the query schema and lets you test sample queries. It sends the queries with your DTaaS sign-in, so they work on an installation that requires one. Outside GraphiQL, send GraphQL queries as HTTP POST requests.

The library microservice provides two API calls:

- List the contents of a directory
- Fetch a file

Both calls can be sent from GraphiQL or as plain HTTP POST requests. The accepted queries are shown below.

PROVIDE LIST OF CONTENTS FOR A DIRECTORY

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

Replace the `path` value with the directory path.

Send requests to: https://intocps.org/lib

Copyright © 2022 - 2026 The INTO-CPS Association

GraphQL query for list of contents

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

GraphQL response for list of contents

```
{
  "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"
                }
              }
            ]
          }
        }
      }
    }
  }
}
```


Copyright © 2022 - 2026 The INTO-CPS Association

HTTP request for list of contents

```
POST /lib HTTP/1.1
Host: intocps.org
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 response for list of contents

```
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 takes a file path and returns the contents of the file.

To try this query, create a file `files/user2/data/welcome.txt` containing `hello world`.

GraphQL query for fetch a file

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

GraphQL response for fetch a file

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

HTTP request for fetch a file

```
POST /lib HTTP/1.1
Host: intocps.org
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}"
}
```


Copyright © 2022 - 2026 The INTO-CPS Association

HTTP response for fetch a file

```
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* is the file path to look at. For example, *user1* looks at the files of **user1**, and *user1/functions* looks at the contents of the *functions/* directory.

Copyright © 2022 - 2026 The INTO-CPS Association

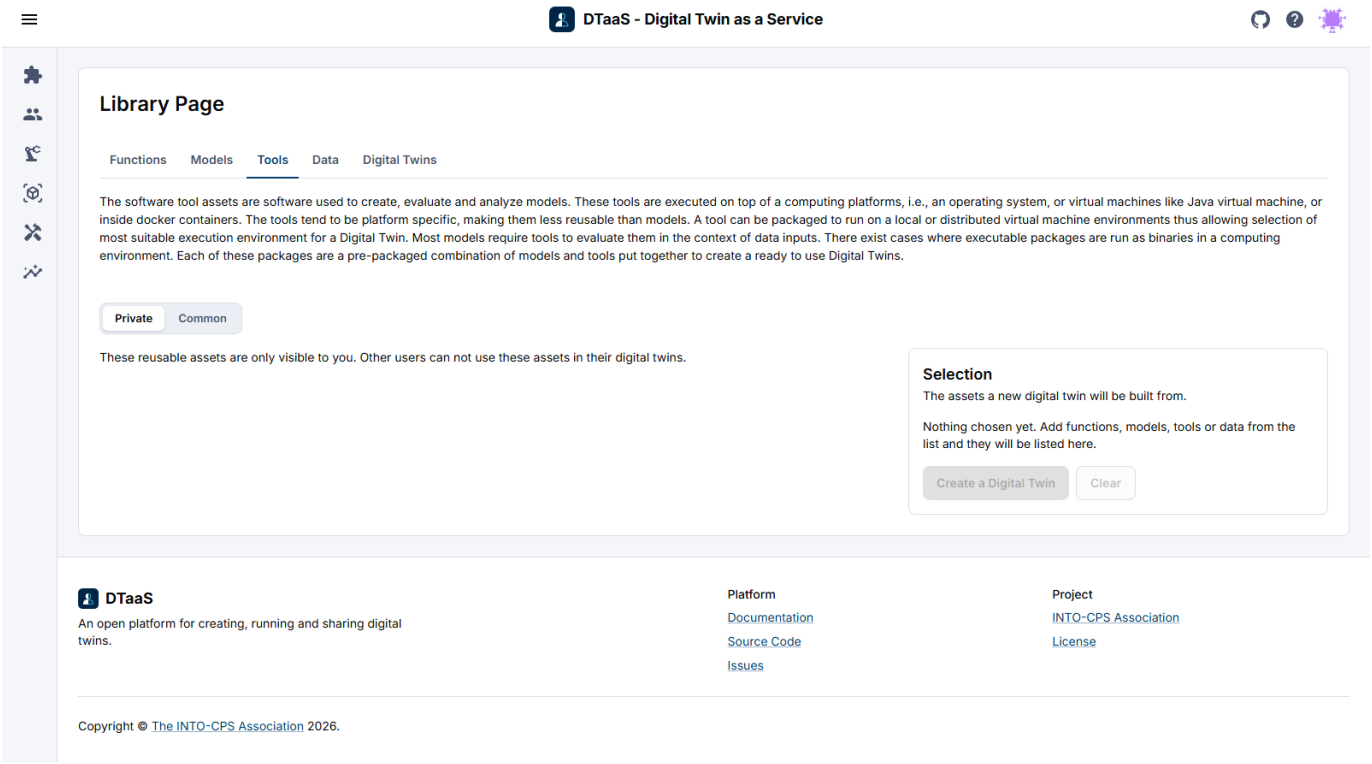
3.4.3 Reusable Assets and DevOps

DevOps is a well-established software development practice. The DTaaS platform has an experimental feature that brings DevOps to digital twins.

This feature requires a specific installation setup:

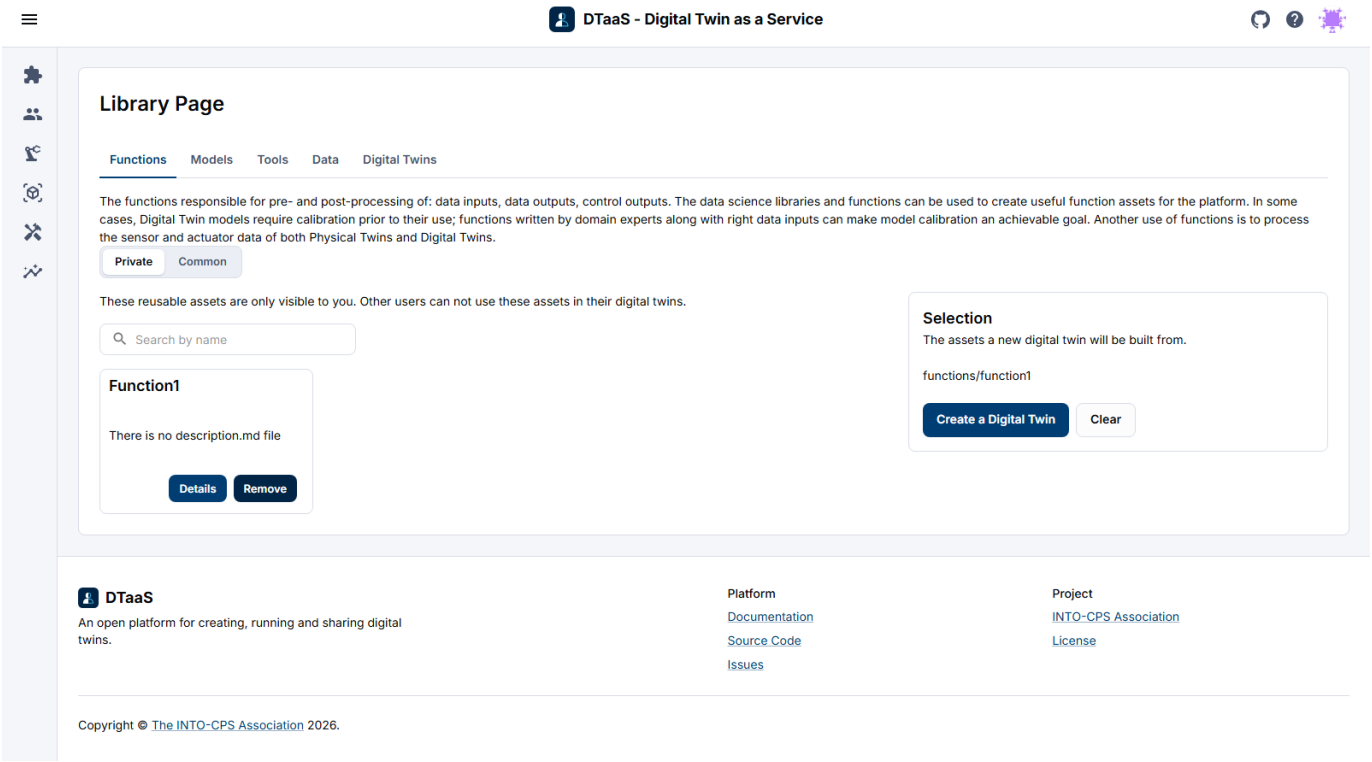
1. Integrated GitLab installation
2. A valid GitLab repository for the logged-in user. See an example repository. You can clone this repository and customise it as needed.
3. A GitLab Runner linked to the user's GitLab repository.

Once these requirements are met, open the **Library Page** by clicking **Automation** in the menu and then the **Library Page** card. The page shows all the reusable assets stored in the linked GitLab repository. A category with no assets shows an empty list.



Otherwise the page lists the existing assets. The following figure shows all the available DTs.

Copyright © 2022 - 2026 The INTO-CPS Association



Click **Add** on an asset to add it to the **Selection** pane on the right. After selecting all the required assets, click the **Create a Digital Twin** button to go to the DT create stage.

Copyright © 2022 - 2026 The INTO-CPS Association

3.5 Workflow Logging

3.5.1 Logger Microservice

The logger microservice receives workflow interaction events from the DTaaS client and stores them as JSON Lines: one JSON object per line, one line per event. The format suits command-line tools such as `jq` and data analysis libraries alike.

How Events Reach the Logger

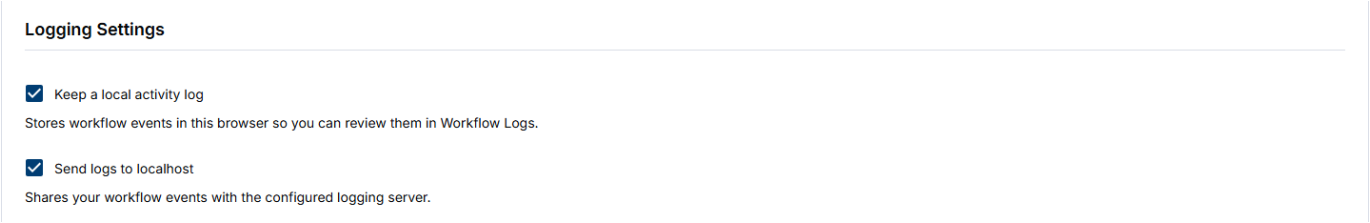
The website records page navigation, button and tab clicks, form changes and notifications. When remote logging is on, the browser sends each event to the logger with the Beacon API, which does not wait for a reply, so logging never slows the website down. The logger checks each event and appends it to its log file.

```
flowchart LR
    A[DTaaS website<br>in the browser] -- "POST /logger<br>one event" --> B[Logger<br>microservice]
    B -- "appends a line" --> C[(workflow-logs.jsonl)]
    A -. "local activity log" .-> D[(Browser storage)]
```

The local activity log is separate. It stays in the browser and is shown on the Logs page.

Enabling Remote Logging

Remote logging is off by default. When the installation has a logger, the **Logging Settings** section on the **Settings** tab of the **Account** page shows a **Send logs to ...** checkbox, followed by the logger's host name. Select it and click **Save Settings**.



The two checkboxes are independent. **Keep a local activity log** stores the events in the browser, and **Send logs to ...** sends them to the logger. Keep remote logging on only with the consent of the person whose activity is being recorded.

What a Log File Contains

The following lines were recorded while a user opened the **Digital Twins Page** from **Automation**, switched to the **Execute** tab, opened and closed the execution history of **Hello World**, and then went to the **Logs** page. Download them as workflow-logs-sample.jsonl.


```
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:34.686Z","event":"navigation","pa
account","element":"page","label":"/account","context":{}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:36.233Z","event":"navigation","pa
automation","page_transition":{"src":"/account","target":"/automation"},"element":"page","label":"/automation","context":{}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:40.379Z","event":"navigation","pa
preview/digitaltwins","page_transition":{"src":"/automation","target":"/preview/digitaltwins"},"element":"page","label":"/preview/digitaltwins","context":{}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:44.317Z","event":"click","page":"
preview/digitaltwins","element":"tab","label":"Execute","context":{"dt":{"tab":"execute"}}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:48.369Z","event":"click","page":"
preview/digitaltwins","element":"button","label":"History","context":{"dt":{"name":"hello-world","button":"history","executionCount":0}}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:50.435Z","event":"click","page":"
preview/digitaltwins","element":"button","label":"Close","context":{"dt":{"name":"hello-world","button":"close-history"}}}
{"sessionId":"70327d05-9dbc-454a-
a358-6b8914fb37b9","userHash":"0a041b9462caa4a31bac3567e0b6e6fd9100787db2ab433d96f6d178cabfce90","timestamp":"2026-10-10T20:29:52.481Z","event":"navigation","pa
insights/log","page_transition":{"src":"/preview/digitaltwins","target":"/insights/log"},"element":"page","label":"/insights/log","context":{}}
```


Copyright © 2022 - 2026 The INTO-CPS Association

Each line has the same fields:

Field	Meaning in the sample
`sessionId`	A random identifier for one browser session. All seven events share it
`userHash`	The SHA-256 hash of the username, computed in the browser. The logger never receives the username itself
`timestamp`	When the event happened, in UTC
`event`	`navigation` for a page change and `click` for a tab or button. The other types are `change`, `notification` and `dismiss`
`page`	The page on which the event happened
`page_transition`	On navigation events only: the page the user came from (`src`) and went to (`target`)
`element` and `label`	What was used, such as the `button` labelled `History`
`context`	Details that depend on the page. On the Digital Twins Page, `dt` names the digital twin and the action

The hash is the same every time a user signs in, so events can still be grouped by user across sessions. It is a pseudonym, not anonymity.

Analysing a Log File

The logger writes to `logs/workflow-logs.jsonl` unless it is configured otherwise. These `jq` commands work on that file and on the sample.

Count the events of each type:

```
jq -s -c 'group_by(.event) | map({event: .[0].event, count: length})' \
  workflow-logs.jsonl
```

```
[{"event":"click","count":3},{"event":"navigation","count":4}]
```

Follow the path a user took through the website:

```
jq -r 'select(.page_transition)
| "\(.timestamp) \(.page_transition.src) -> \(.page_transition.target)"' \ |
  workflow-logs.jsonl
```

```
2026-10-10T20:29:36.233Z /account -> /automation
2026-10-10T20:29:40.379Z /automation -> /preview/digitaltwins
2026-10-10T20:29:52.481Z /preview/digitaltwins -> /insights/log
```

List what was done to digital twins:

```
jq -c 'select(.context.dt) | {timestamp, label, dt: .context.dt}' \
  workflow-logs.jsonl
```

```
{"timestamp":"2026-10-10T20:29:44.317Z","label":"Execute","dt":{"tab":"execute"}}
{"timestamp":"2026-10-10T20:29:48.369Z","label":"History","dt":{"name":"hello-world","button":"history","executionCount":0}}
{"timestamp":"2026-10-10T20:29:50.435Z","label":"Close","dt":{"name":"hello-world","button":"close-history"}}
```

When the active file reaches its size limit, the logger renames it and starts a new one, keeping a fixed number of old files. To analyse everything, pass all of them to `jq`.

Copyright © 2022 - 2026 The INTO-CPS Association

Configuring the Website

The client sends events to the URL configured in `LOGGER_URL`. Include the `/logger` route prefix, for example:

```
LOGGER_URL: "https://example.com/logger";
```

For non-Docker local development, use the logger's direct port:

```
LOGGER_URL: "http://localhost:4003/logger";
```

The API is specified in `servers/logger/API.md`. Installing and configuring the logger is covered in the administrator guide.

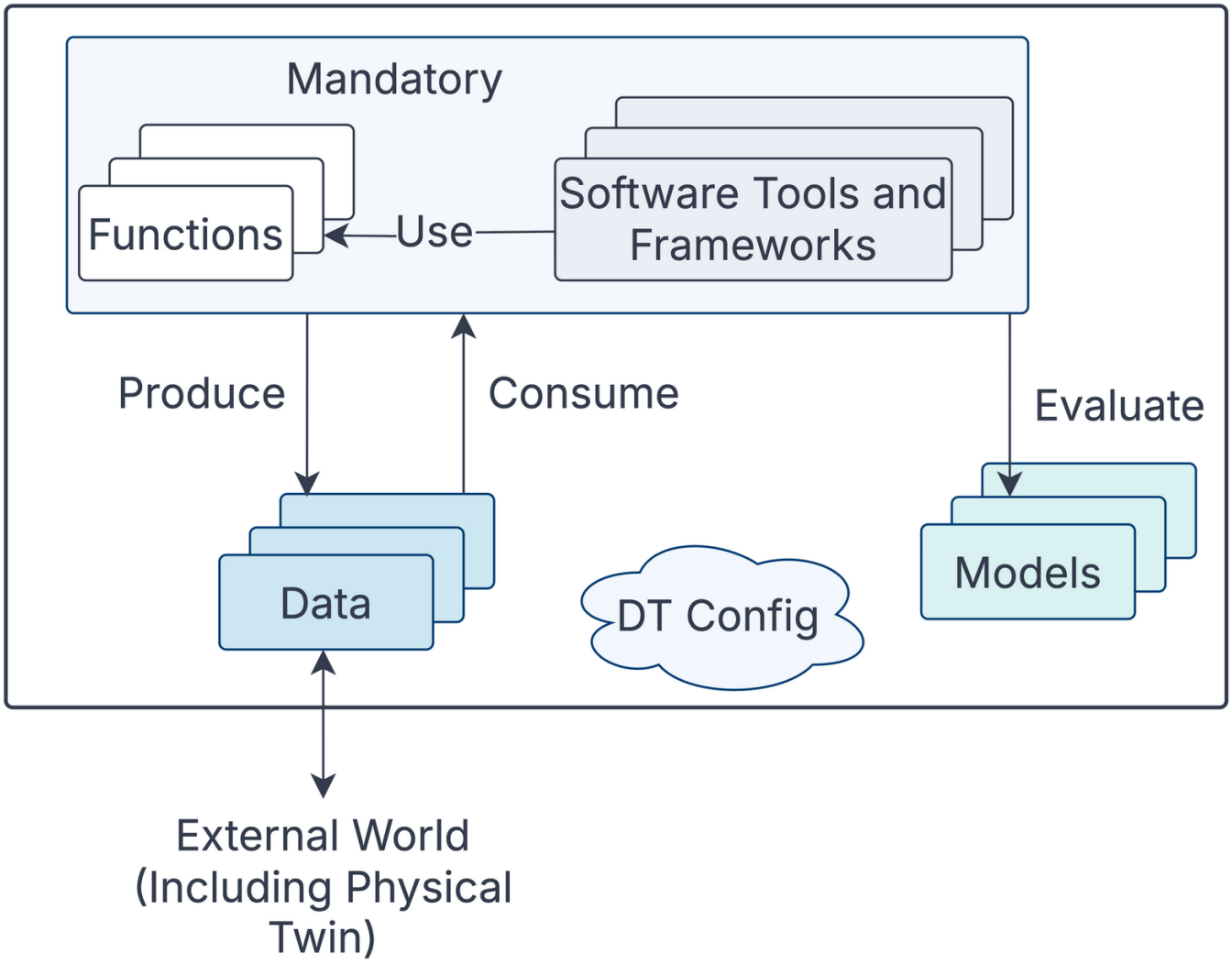
Copyright © 2022 - 2026 The INTO-CPS Association

3.6 Digital Twins

3.6.1 Create a Digital Twin

A digital twin is built from the assets available in the workspace. To bring assets or files from a local computer into the DTaaS workspace, follow the instructions in library assets.

The library assets depend on one another as shown below.



A digital twin is created by linking assets in a meaningful way. The relationship can be written as:

$$ D_t: \{ D{^*},M^{*},(FT)^{+} \}C_{dt} $$

where D denotes data, M denotes models, F denotes functions, T denotes tools, C_{dt} denotes DT configuration, and D_t is a symbolic notation for a digital twin itself. The $\{ D{^*},M^{*},(FT)^{+} \}C_{dt}$ expression denotes composition of a DT from D, M, T, and F assets. The $*$ indicates zero or more instances of an asset, and $+$ indicates one or more instances of an asset.

The DT configuration specifies which assets to use and any parameters to set for them. When a DT requires RabbitMQ, InfluxDB, or similar services supported by the platform, the DT configuration must include access credentials for these services.

This generic DT definition is based on DT examples observed in practice, and a DT may deviate from it. The only requirement is that the DT can be executed from either the command line or a graphical desktop environment.

Copyright © 2022 - 2026 The INTO-CPS Association

Tip

If you are new to digital twins and do not have separate digital twin assets, only a single directory containing all components, upload this monolithic digital twin into the **digital_twins/your_digital_twin_name** directory.

Example

The Examples repository contains a co-simulation setup for a mass-spring-damper system. The example shows how co-simulation can be used in a digital twin.

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

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

The co-simulation digital twin requires the Maestro co-simulation orchestrator. Maestro is reusable across all co-simulation-based DTs, so it is placed in the `common/tools/` directory.

The digital twin configuration is in the `digital_twins/mass-spring-damper` directory. The co-simulation configuration is defined in two JSON files: `cosim.json` and `time.json`. Documentation for the digital twin goes in `digital_twins/mass-spring-damper/README.md`.

The launch program for this digital twin is `digital_twins/mass-spring-damper/lifecycle/execute`. It runs the co-simulation until completion and then exits. The programs in `digital_twins/mass-spring-damper/lifecycle` manage the lifecycle of this digital twin. The lifecycle page explains these programs.

Execution of a Digital Twin

A frequent question concerns the run time of a digital twin. The natural intuition is 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? Yes.** The Industry 4.0 use cases seen among SMEs have digital twins that run for a finite time. Users often run these digital twins when they choose to.

Copyright © 2022 - 2026 The INTO-CPS Association

To execute this digital twin:

1. Open the **Workbench** page of the DTaaS website and click the **Desktop** tool. The desktop opens in a new browser tab.
2. The tab shows the VNC Desktop with a connect button. Click **Connect** to connect to the Linux desktop of the workspace.
3. Open a terminal (the black rectangular icon in the top left region of the tab) and enter the commands below.
4. Download the example files by following the instructions in the 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`.

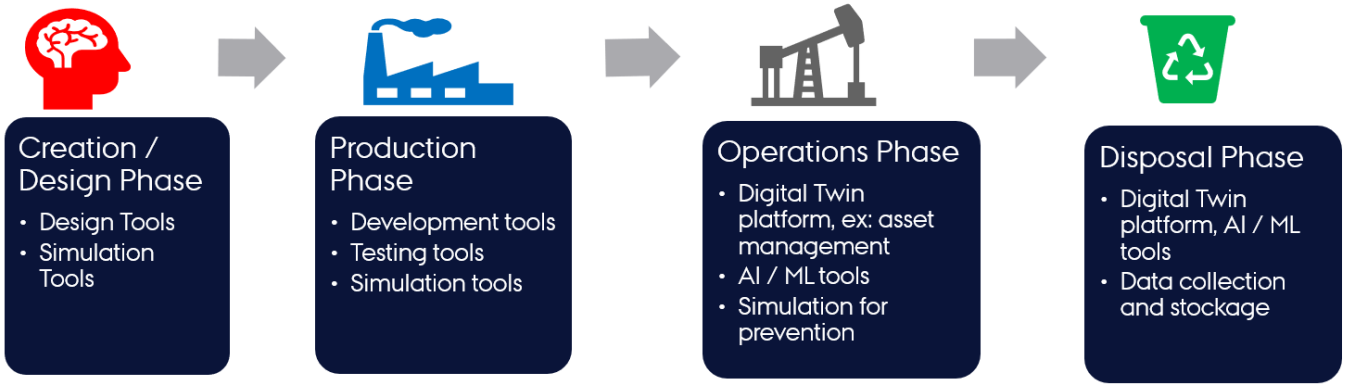
Copyright © 2022 - 2026 The INTO-CPS Association

3.6.2 Digital Twin Lifecycle

Physical products in the real world go through a product lifecycle. A simplified four-stage product lifecycle is shown here.

A digital twin that tracks physical products (twins) must evolve together with the corresponding physical twin.

The figure shows the activities that can take place in each lifecycle phase.



(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 seven phases. This section lists the phase names and the typical activities in each phase[1].

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

Phase	Main Activities
explore	Selection of suitable assets based on user requirements and verification of their compatibility for DT creation.
create	Specification of DT configuration. For existing DTs, no creation phase is required at the time of reuse.
execute	Automated or manual execution of a DT based on its configuration. The DT configuration must be verified before starting execution.
analyse	Examination of DT outputs and decision-making. Outputs may include text files or visual dashboards.
evolve	Reconfiguration of DT primarily based on analysis results.
save	Preservation of DT state to enable future recovery.
terminate	Cessation of DT execution.

A digital twin that faithfully tracks the physical twin lifecycle must support all the phases. Digital twin engineers may also add their own phases, so the DTaaS platform is designed to accommodate different DTs.

The phases do not have to run in a fixed order. A digital twin can repeat or skip phases as needed.

Copyright © 2022 - 2026 The INTO-CPS Association

A code structure with one program per phase is not mandatory, but it makes DT creation and management within the DTaaS platform easier. The recommended structure is:

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

Each phase of the DT lifecycle has its own program. A program can be as simple as a script that launches other programs or sends messages to a live digital twin.

> **Note**
>
> **The recommended way to implement lifecycle phases within DTaaS is with scripts, for example shell scripts.**

Example Lifecycle Scripts

The following example scripts manage three phases in the lifecycle of the **mass-spring-damper DT**.

lifecycle/execute

```
#!/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 phase uses the DT configuration, the FMU models and the Maestro tool to execute the digital twin. The script stores the co-simulation output in `/workspace/data/mass-spring-damper/output`.

A DT may not support every lifecycle phase. An unsupported phase can be an empty script, with a message if needed.

lifecycle/analyze

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

The lifecycle programs can invoke other programs in the codebase. For example, the `lifecycle/terminate` program calls another script to do its work.

lifecycle/terminate

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

References

[1]: Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

Copyright © 2022 - 2026 The INTO-CPS Association

3.6.3 DevOps

Digital Twin File Structure in GitLab

GitLab is the file store for DevOps on digital twins. The user interface page is a front end for this GitLab-backed file storage.

Each DTaaS installation comes with an integrated GitLab. It must have a GitLab group named **dtaas** and a GitLab repository for each user, named after the username. For example, if a DTaaS installation has two users, *user1* and *user2*, the following repositories must exist on the linked GitLab installation.

```
https://intocps.org/gitlab/dtaas/common.git
https://intocps.org/gitlab/dtaas/user1.git
https://intocps.org/gitlab/dtaas/user2.git
```

> **Warning**
>
> By default, the assets shown on the Library Page and the Digital Twins Page come from the `master` branch of the backing GitLab project. Create a branch named `master` and make it the default branch in all the user repositories, including the common repository. A different branch can be set in the **Branch Name** field of the settings.

Each user repository must have the following structure.

```
<username>/
├── common/
├── data/
├── digital_twins/
├── functions/
├── models/
├── tools/
├── .gitlab-ci.yml
└── README.md
```

This is the same structure the user sees on the **Library** page.

DIGITAL TWIN STRUCTURE

The `digital_twins` folder contains DTs that one or more users have built. They are meant to be flexible enough to be reconfigured for specific use cases.

For an example of such a configuration, see the dtaas/user1 repository on gitlab.com. Its `digital_twins` directory contains a `hello_world` example with this file structure:

```
hello_world/
├── lifecycle/ (at least one lifecycle script)
│   ├── clean
│   ├── create
│   ├── execute
│   └── terminate
├── .gitlab-ci.yml (GitLab DevOps config for executing lifecycle scripts)
└── description.md (optional but is recommended)
└── README.md (optional but is recommended)
```

The `lifecycle` directory here contains four files, `clean`, `create`, `execute` and `terminate`, which are simple BASH scripts. They correspond to stages in a digital twin's lifecycle. The lifecycle stages page explains them further.

The `.gitlab-ci.yml` at the top of the user repository contains one trigger job per digital twin. When the website executes a digital twin, the matching trigger job runs the pipeline defined in that digital twin's own `.gitlab-ci.yml`. Saving a new digital twin on the **Create** tab adds its trigger job, and deleting a digital twin removes it.

Copyright © 2022 - 2026 The INTO-CPS Association

Digital Twins and DevOps

DevOps is a well-established software development practice. The DTaaS has an experimental feature that brings DevOps to digital twins.

Recordings of the latest workflows:

- DTaaS DevOps workflows
- Automation layer demo

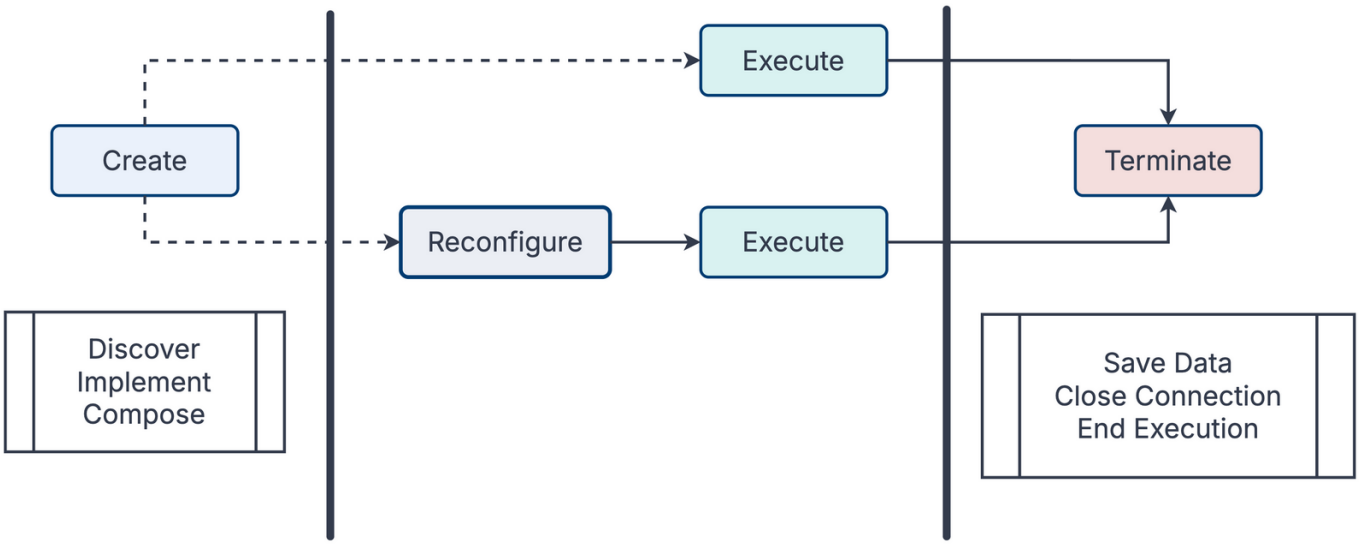
This feature requires a specific installation setup:

1. Integrated GitLab installation
2. A valid GitLab repository for the logged-in user. See an example repository. You can clone this repository and customise it as needed.
3. A GitLab Runner linked to the user's GitLab repository.

To open the feature, click **Automation** in the menu and then the **Digital Twins Page** card.

DT LIFECYCLE

The **Digital Twins Page** implements the **Create**, **Manage** and **Execute** stages of a DT lifecycle. The suggested order of the lifecycle stages is:

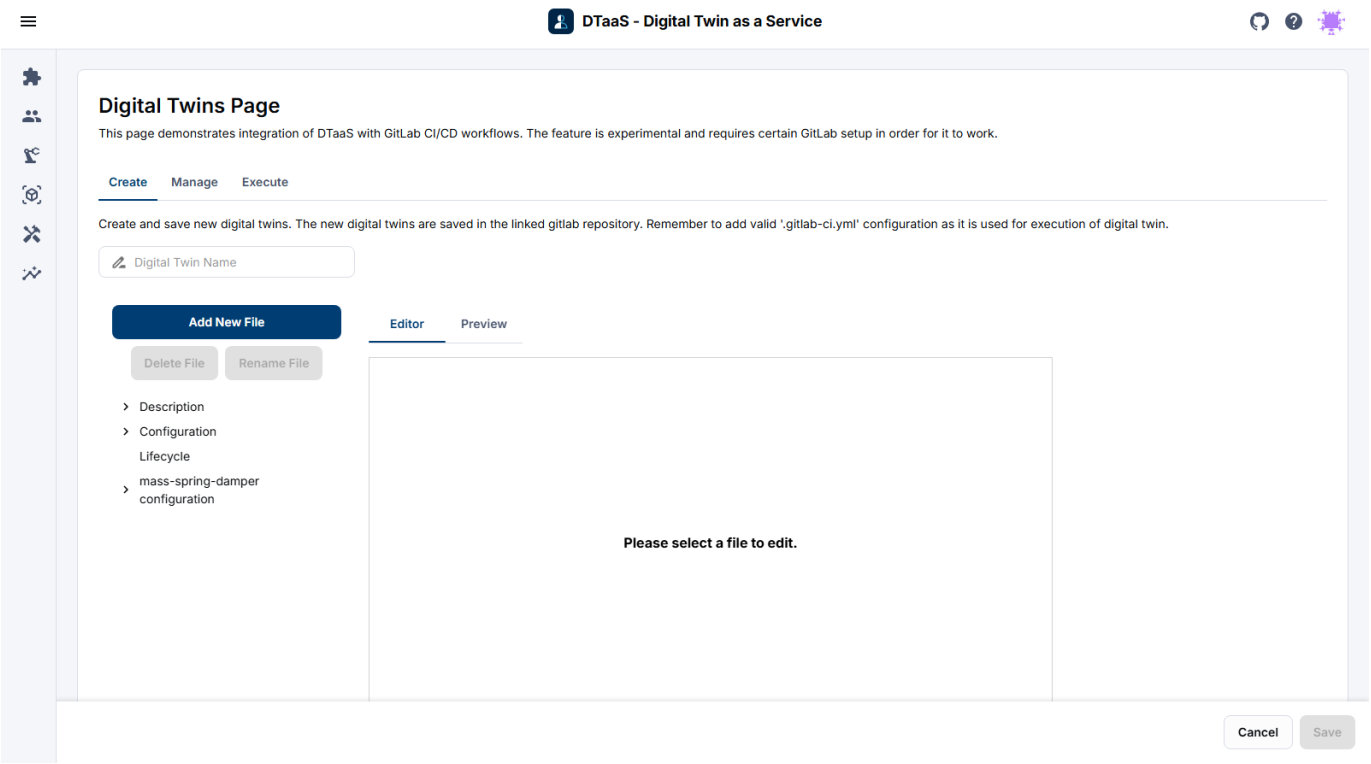


Each of the **Create**, **Manage** and **Execute** stages has its own tab. The DT assets for the Create stage are selected on the Library Page. The Manage tab reconfigures digital twins. Execution and termination happen on the Execute tab.

CREATE TAB

Users select reusable DT assets on the Library Page and then arrive on the Create tab. The following figure shows the Create tab after *mass-spring-damper* was selected as a reusable asset for the new DT.

Copyright © 2022 - 2026 The INTO-CPS Association



The menu on the left lets you create the necessary structure and elements of a new DT. Each reusable asset selected for the new DT appears in this menu, and its configuration can be updated.

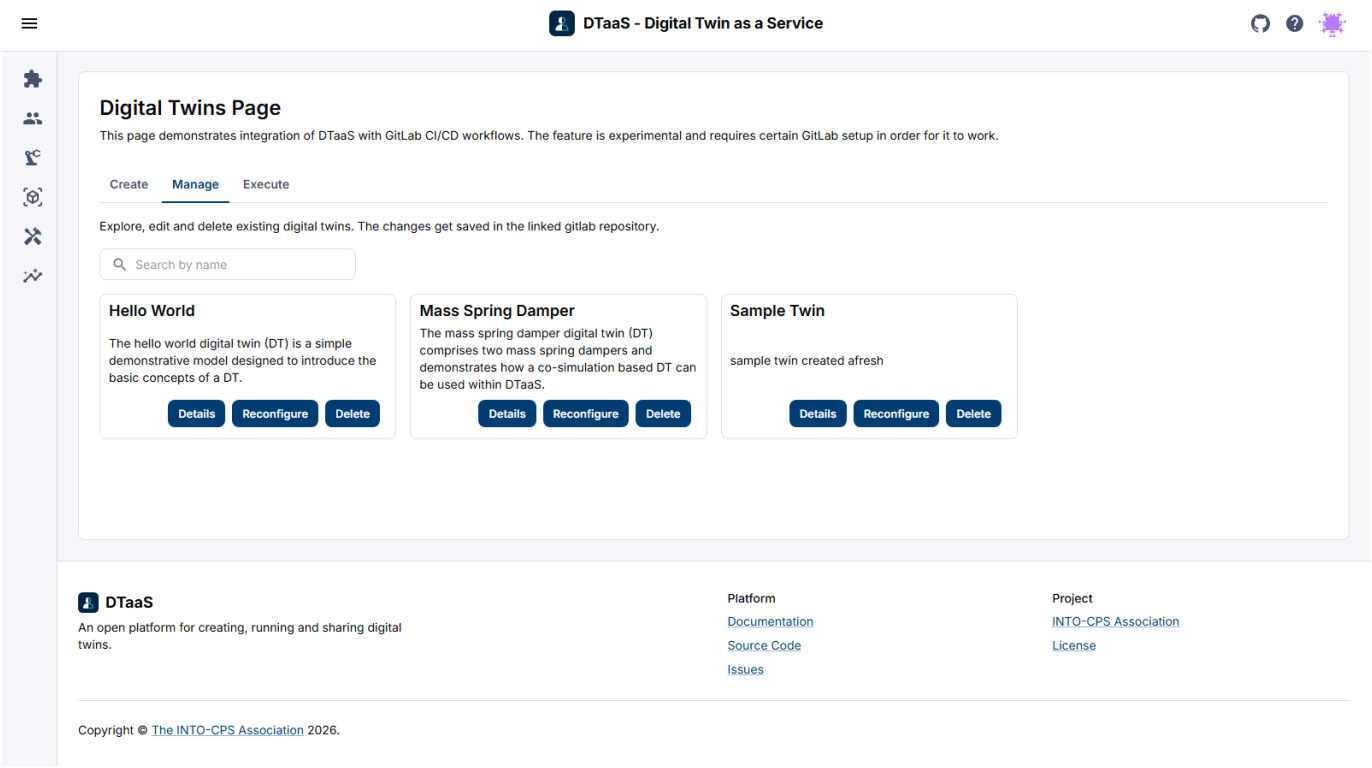
The files in the left menu fall into three categories.

- **Description**: contains *README.md*, a full description of the DT, and *description.md*, a brief description. The brief description is shown on the DT cards. The *Details* button on the Manage tab shows the complete README.md.
- **Configuration**: contains a *.gitlab-ci.yml* for running the lifecycle scripts and other operations of the DT. You can add more JSON and YAML files to configure the new DT.
- **Lifecycle**: the DT lifecycle scripts.

The *Add New File* button adds new files in any of the three categories. Click the *Save* button to save the new DT. Newly created DTs appear on the **Manage** and **Execute** tabs straight away.

Copyright © 2022 - 2026 The INTO-CPS Association

MANAGE TAB



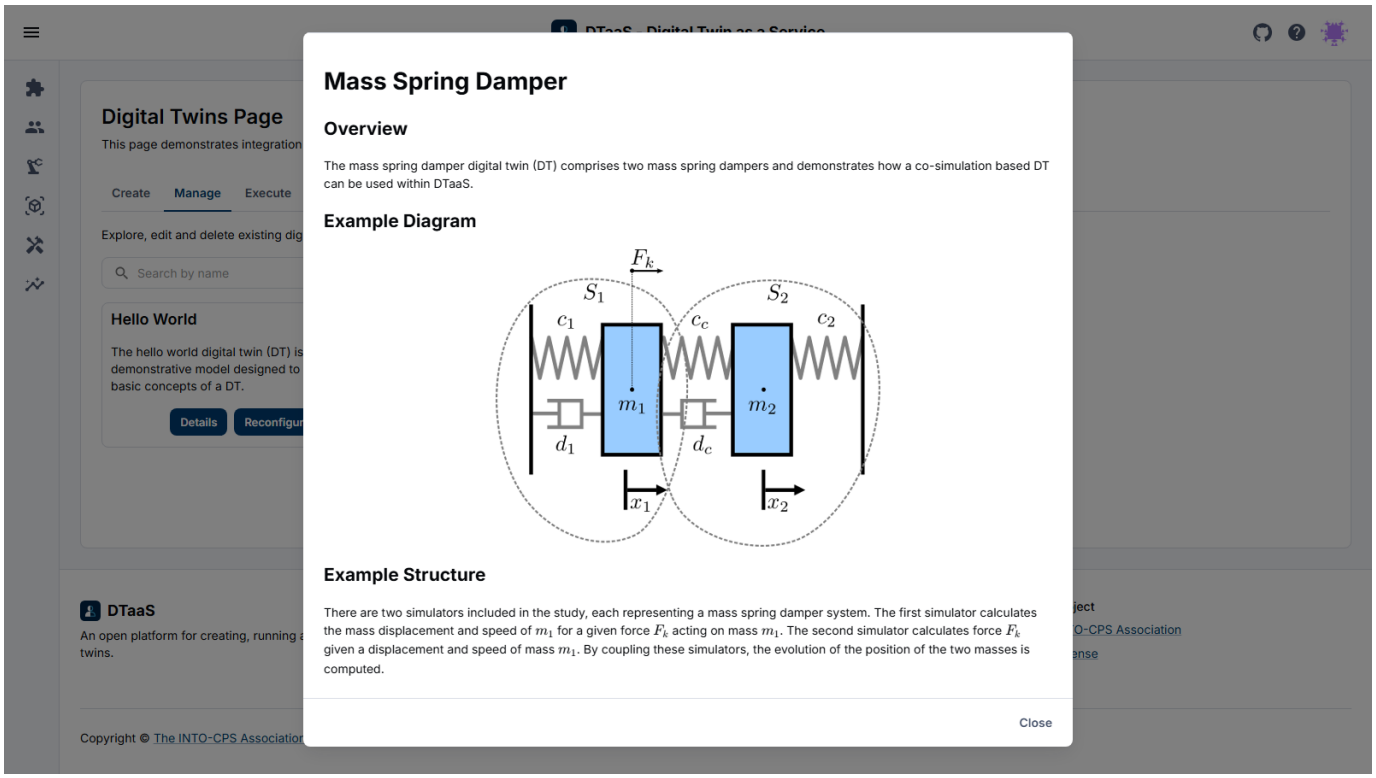
The Manage tab offers these operations on a digital twin:

- Check the details (**Details** button)
- Delete (**Delete** button)
- Modify or reconfigure (**Reconfigure** button)

A digital twin placed in the DTaaS has a recommended structure. See the assets page for an explanation and this example.

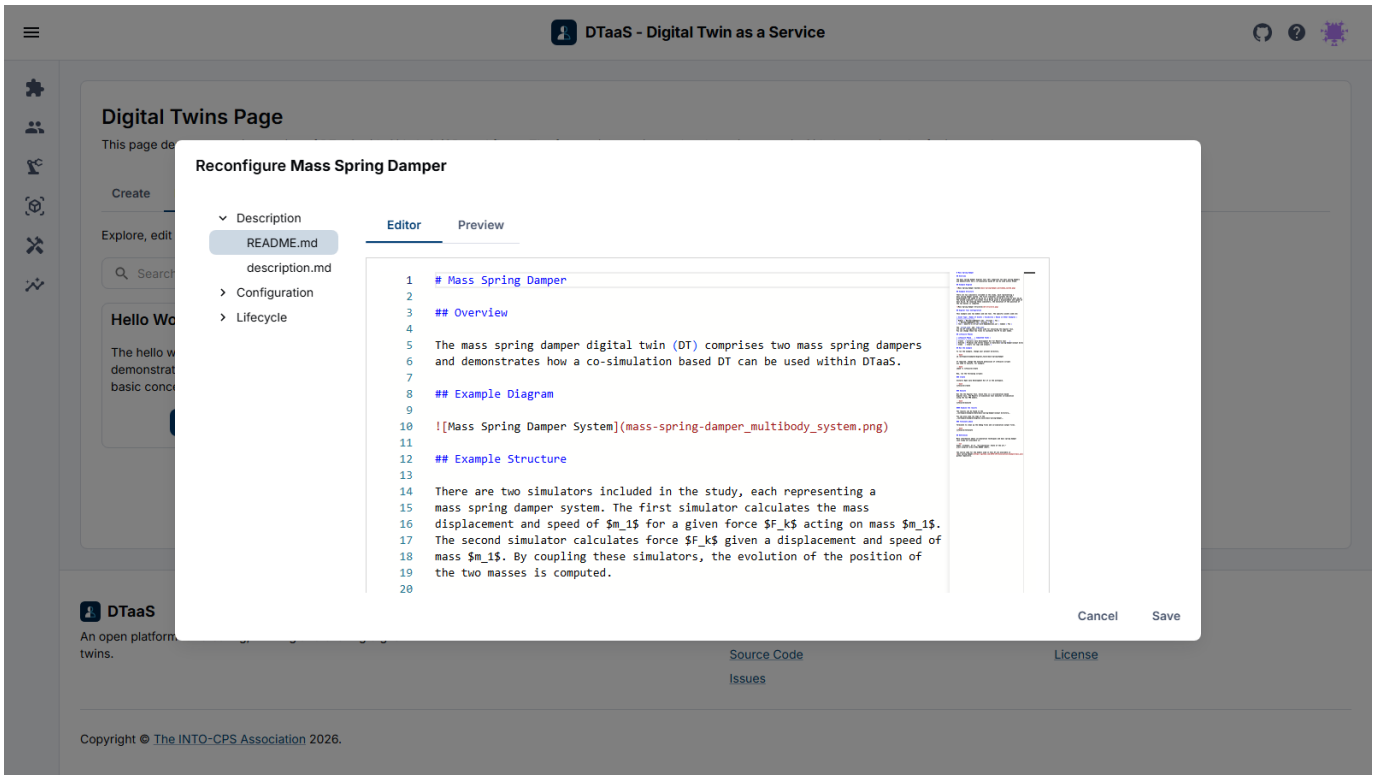
The Details button shows the README.md stored in the digital twin directory.

Copyright © 2022 - 2026 The INTO-CPS Association



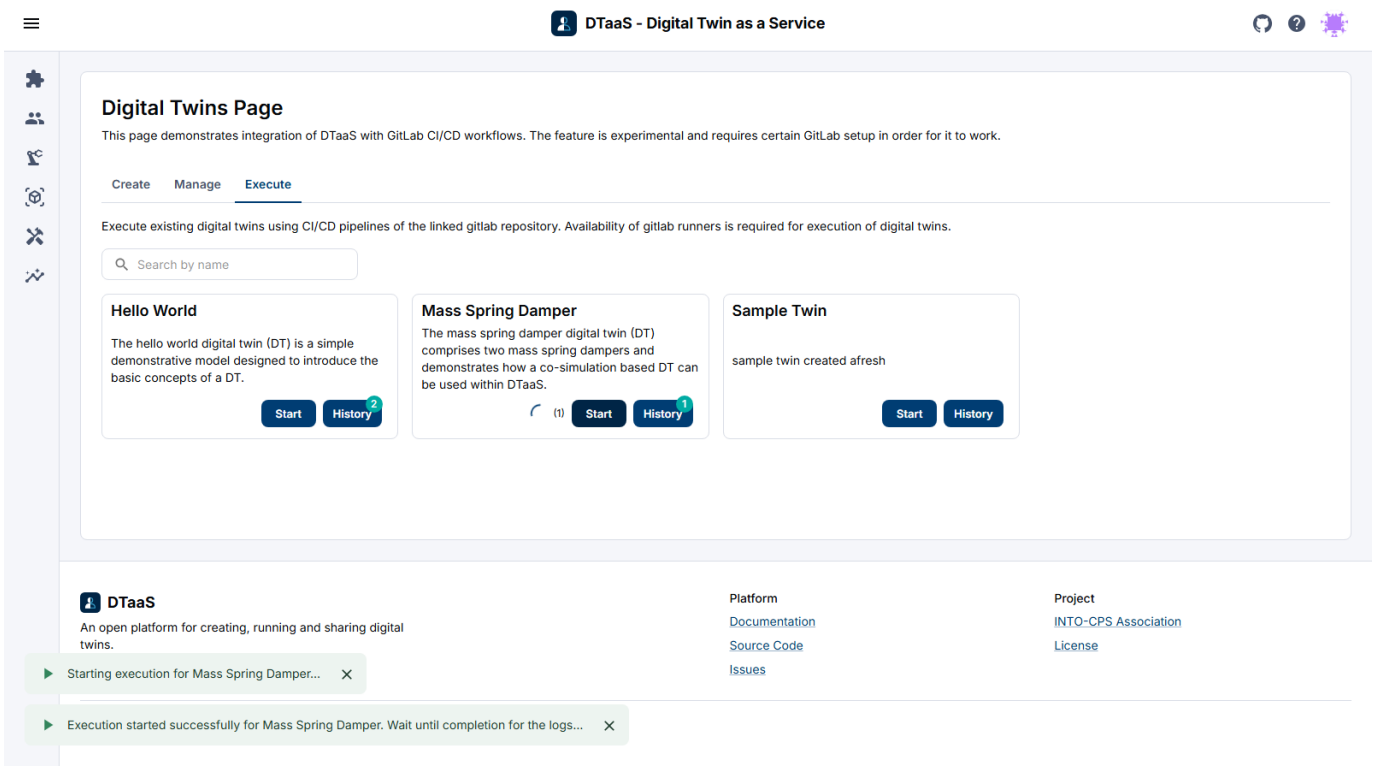
The Reconfigure button opens an editor with all the files of a digital twin, and all of them can be updated. The files fall into the same three categories.

- **Description**
- **Configuration**
- **Lifecycle**

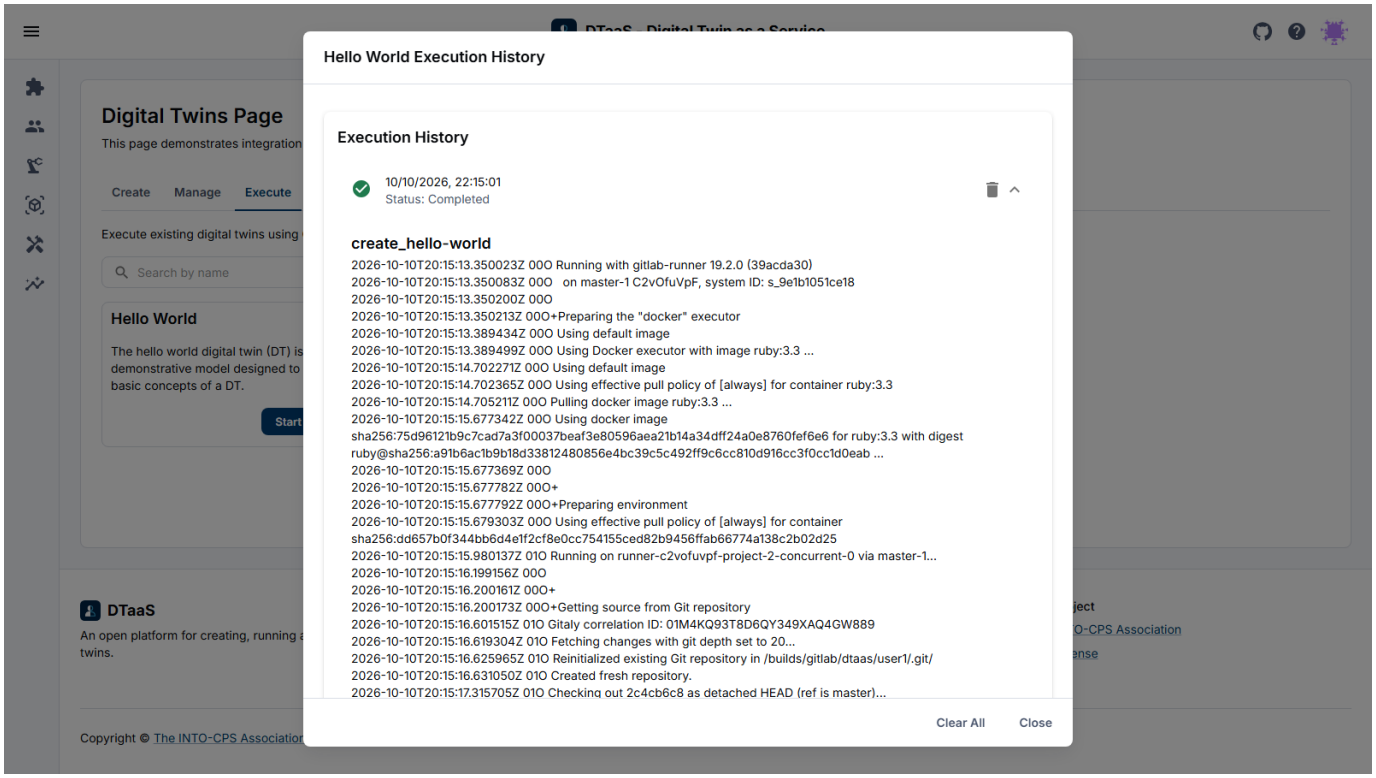


Copyright © 2022 - 2026 The INTO-CPS Association

EXECUTE TAB



The Execute tab can run several digital twins at once. When an execution of a digital twin finishes, its execution log is available.



Copyright © 2022 - 2026 The INTO-CPS Association

Setting Allowed Values

Some setting values cause problems. This page lists the values that are allowed and those that are not, with the errors to expect and troubleshooting steps.

The settings are on the **Settings** tab of the **Account** page. The settings page explains what each parameter does.

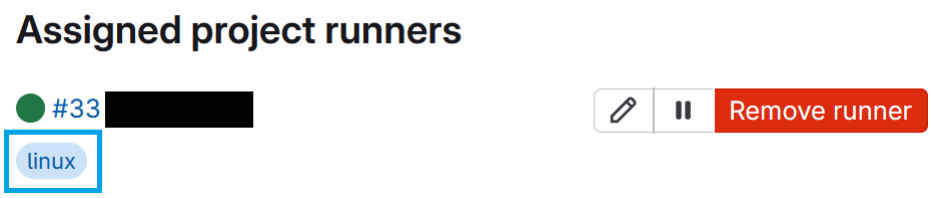
Note: the user must have access rights to all the resources connected to the application.

The Settings form rejects an empty value in any of these fields and shows a "... is required" message under the field.

RUNNER TAG

- Allowed values

This parameter is a text string such as "linux", "ubuntu" or "2", and it must match a tag of a runner on the execution service:



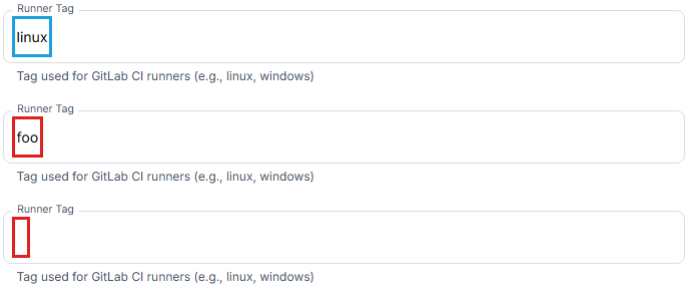
Multiple values, such as "linux, windows", are not supported and are treated as one tag.

- Not allowed values

If the **Runner Tag** field *does not* match a runner, no runner picks up the job and it times out. If the twins time out unexpectedly, check the spelling of the tag.

Some services, such as GitLab, allow jobs without a tag, which *any* runner configured to pick up untagged jobs can run. The DTaaS requires a tag, so the field cannot be left blank.

- Visual examples



BRANCH NAME

- Allowed values

The **Branch Name** field must be a text string matching a branch in the user *and* common library repositories. For example: "main" or "master":

Copyright © 2022 - 2026 The INTO-CPS Association



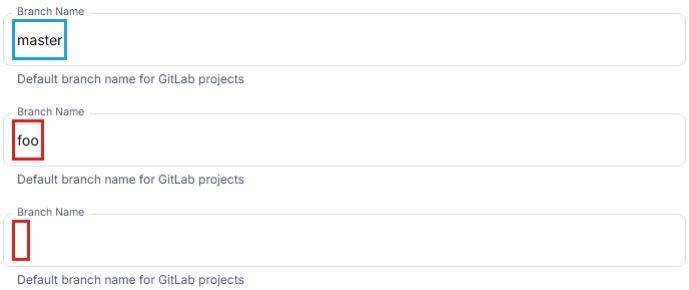
- Not allowed values

The field cannot be left empty. Do not set it to a value that does not match a branch in both repositories: the form accepts such a value, but it causes errors or unexpected behaviour.

Expected error:

An error occurred while fetching assets: GitbeakerRequestError: 404 Tree Not Found

- Visual examples



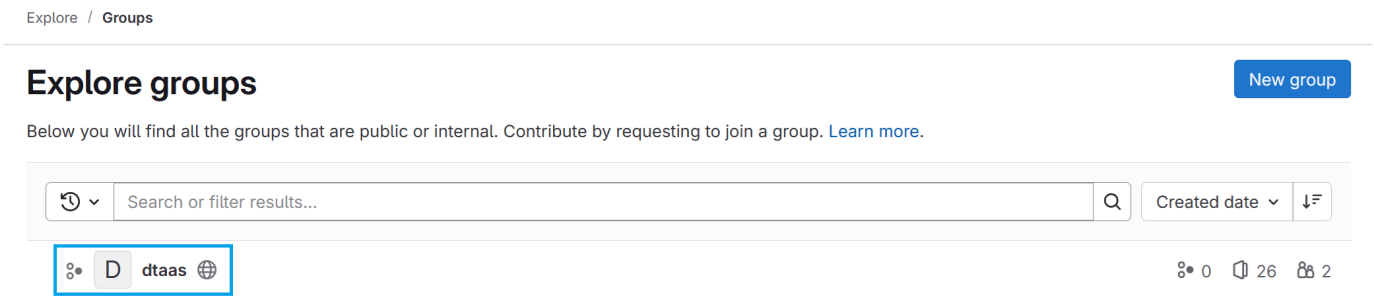
GROUP NAME

- Allowed values

The **Group Name** field is a text string that must match a group in the storage service (for example, GitLab).

Example:

DTaaS



Group names are case insensitive on GitLab, but matching the case exactly is recommended for consistency and to rule it out as a source of error.

- Not allowed values

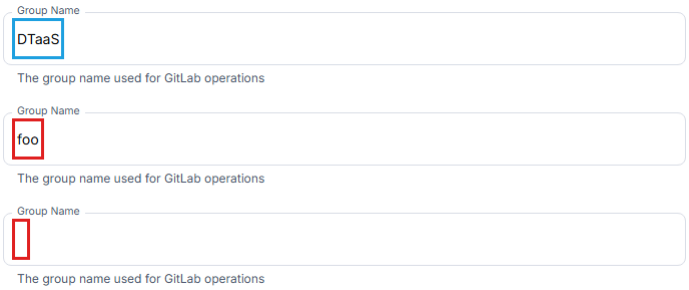
If the group does not exist, errors occur when digital twins and other assets are loaded.

Copyright © 2022 - 2026 The INTO-CPS Association

Expected error:

An error occurred while fetching assets: GitbeakerRequestError: 404 Group Not Found

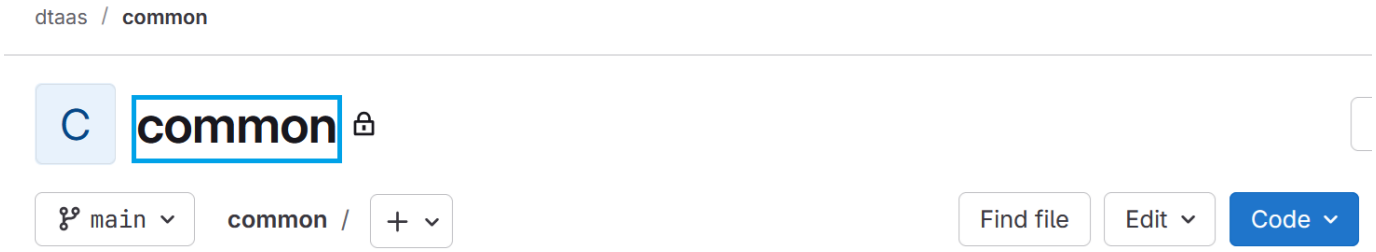
- Visual examples



COMMON LIBRARY PROJECT NAME

- Allowed values

The **Common Library Project Name** parameter is a text string. It must match the repository that holds the shared resources. Examples: "common" and "library":



If it does not match a repository, the Digital Twins Page and the Library Page cannot load their assets.

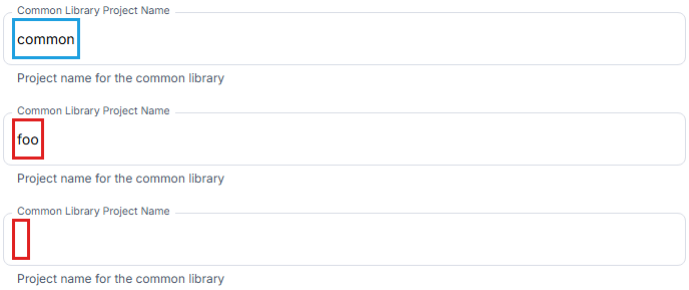
Expected error:

An error occurred while fetching assets: Error: Common project `<name>` not found

- Not allowed values

Do not set this field to a value that does not match a repository. It should not match a user repository either.

- Visual examples

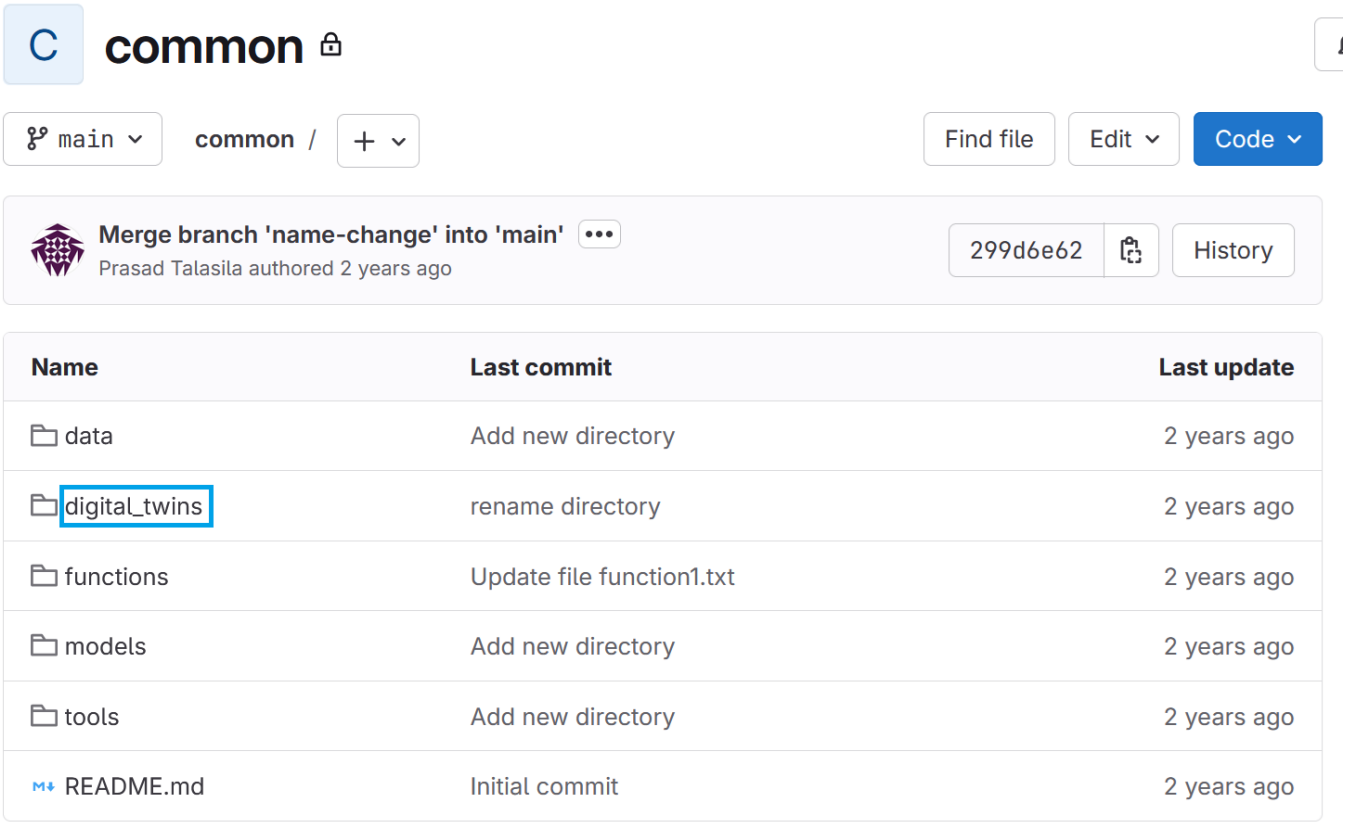


DT DIRECTORY

- Allowed values

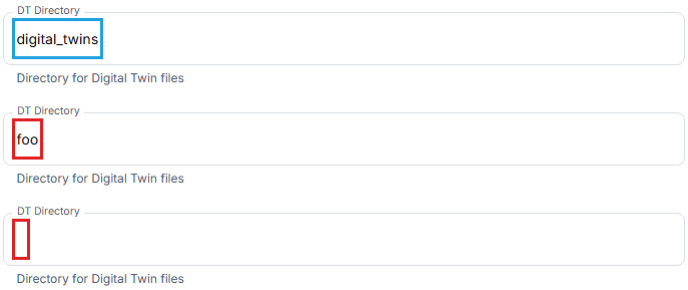
This is also a text string. It must match the name of the folder in the common library and, above all, in the user repository where the digital twins are stored.

Copyright © 2022 - 2026 The INTO-CPS Association



- Not allowed values

Do not set this to a folder that does not exist. It fails silently: the digital twins do not load, or digital twins from a previous setup show up.



TROUBLESHOOTING

The digital twins are not showing up:

Check that Group Name, Branch Name, Common Library Project Name and DT Directory match the backend exactly.

The digital twins are timing out:

Check that the Runner Tag is a single tag, not a list, and that a runner with that tag is active.

SUMMARY

Every field must be a text string that matches the corresponding repository, group, branch or runner tag on the backend. The troubleshooting steps above cover the most common problems.

Copyright © 2022 - 2026 The INTO-CPS Association

Capabilities

Besides creating and editing digital twins, you can run them and adjust their execution parameters. The DTaaS can run several digital twins at the same time, and you can change settings while they run without fetching the data manually: the data loads again when you return to the branch and group used for the execution. The settings cover both repository and execution options. All twins can be run and checked from the **Execute** tab of the **Digital Twins Page**.

Read more about Settings, Setting Values and Concurrent Execution on their own pages.

The following sections describe what each of these features can do.

Copyright © 2022 - 2026 The INTO-CPS Association

SETTINGS

The table below lists the test setups (valid, invalid, and so on), the behaviour that was expected and the behaviour that was observed. Each table is followed by a summary and a list of problems found.

Copyright © 2022 - 2026 The INTO-CPS Association

Setting	Expected behaviour	Observed behaviour	Test method
Runner tag **valid**	Job is picked up by runner with relevant tag and completes without error.	Pass: Same as expected.	Go to the Settings tab of the Account page. Change Runner Tag. Go back one page. Execute Hello world twin. Verify runner name based on tag (repeat with 2nd runner)
Runner tag **invalid**	Job is never picked up. Runner times out after 10 minutes.	Pass: Same as expected.	Change Runner Tag to "foo" (nonexistent). Run Hello World twin.
Runner Tag **no value**	Configuration saves. Running a twin succeeds if appropriate runner exists.	Fail: Not picked up, times out.	Set `run_untagged = false` in local gitlab runner config. Mark "Run untagged jobs" as true in gitlab instance (https://dtaas-digitaltwin.com/gitlab). Change Runner Tag to the empty string on application. Run Hello world twin.
Branch **valid**	Job runs with the correct branch and completes without error.	Pass: Same as expected.	Make new branch in GitLab instance project. Change Branch Name to "master-2". Execute Hello world twin. Verify ref matches branch in execution log.
Branch **invalid**	Execute tab gracefully displays no twins as branch does not exist.	Fail: IF twins are not cached: Throws an error displayed to user: An error occurred while fetching assets: GitbeakerRequestError. IF twins are cached: Job fails. Snackbar says "Execution error for [twin name]". No log available in execution history.	Change Branch Name to "master-1" (nonexistent). IF twins are cached: Execute Hello world twin.
Group name **valid**	The twin is runnable.	Pass: Same as expected.	Click "Reset to Defaults". Run twin.
Group name **invalid**	Execute tab gracefully displays no twins as group does not exist.	Fail: Same as Branch invalid case.	Change Group Name to "Foo" (nonexistent). IF twins are cached: Execute Hello world twin.
Common name **valid**	Library twins are visible.	Pass: Same as expected. *Private twins also show up in Common twins.	Click "Reset to Defaults". Go to the Library Page. Go to common twins. Inspect twin visibility.
Common name **invalid**	Library twins gracefully not visible.	Fail: Displays error: An error occurred while fetching assets: Error: Common project foo not found	Change Common Library Project Name to "Foo" (nonexistent). Go to common twins. Inspect visibility.
DT directory **valid**	Twins are visible.	Pass: Same as expected.	Click "Reset to Defaults". Go to the Library Page. Go to the Execute tab. Inspect twin visibility.
DT directory name **invalid**	Twins gracefully not visible.	Fail: Depends on the cache. Displays error if there is none: An	Change DT Directory to "Foo" (nonexistent). Go to the

Copyright © 2022 - 2026 The INTO-CPS Association

Setting	Expected behaviour	Observed behaviour	Test method
		error occurred while fetching assets: GitbeakerRequestError	Execute tab. Inspect twin visibility.
Group Name, DT Directory, Common Library Project Name, Branch name **no value**	Invalid value caught early. Cannot save, appropriate feedback of invalid form fill out.	Fail: Same as Branch invalid case.	Change Group Name, DT Directory, Common Library Project Name, Branch Name to the empty string.

Summary: Changing to valid settings works. At best, invalid settings display an error in the page (no cache); otherwise the page shows stale twins.

Problems:

- Stale state is kept after settings are updated unless the page is refreshed.
- Technical errors are shown to the user when the settings are invalid.
- Invalid values such as empty strings are accepted when a value is required.
- Runners without tags may not work, though this may be a local problem.
- Private twins show up as common twins.

> **Note**
>
> These tests were run against an earlier version of the website. The Settings form now rejects an empty value in any of these fields and shows a "... is required" message, so the **no value** cases can no longer be saved.

CONCURRENT EXECUTION

Expected behaviour	Observed behaviour	Test method
Both twins run successfully simultaneously.	Pass: Same as expected.	Run the same twin twice at the same time.
All twins run successfully simultaneously.	Pass: Same as expected.	Run different twins at the same time.

Concurrent Execution Across Different Runners

Expected behaviour	Observed behaviour	Test method
Both twins run successfully simultaneously. They report the correct runner name in the Execution logs.	Pass: Same as expected.	Set up 2 ubuntu GitLab runners with different tags. Change Runner Tag to first runner's tag. Execute twin. Change Runner Tag to second runner tag. Execute another twin.
History stays after editing, it is still executing.	Pass: Same as expected.	Execute → Edit settings → Execute. Check: is history still there? Does it look correct?

Summary:

All functionality works according to the tests. The logs remain after settings change, and any combination of twins and runners works (only Ubuntu runners were tested).

Copyright © 2022 - 2026 The INTO-CPS Association

IMPLEMENTING BACKENDS

By default the DTaaS uses GitLab as its execution and storage backend, but other combinations may suit some installations better. The code base is designed so that a new backend does not require a rewrite from scratch. Future versions of the DTaaS may include more backends by default, such as GitHub and Azure.

SUMMARY

Digital twins can be queued as needed, and changing settings while they run does not lose data, so several setups can be tested at the same time. Developers can also add new backends.

Copyright © 2022 - 2026 The INTO-CPS Association

Running Multiple Digital Twins at the Same Time

The DTaaS platform can execute several digital twins at once. This saves time with long-running digital twins or when they run on different resources.

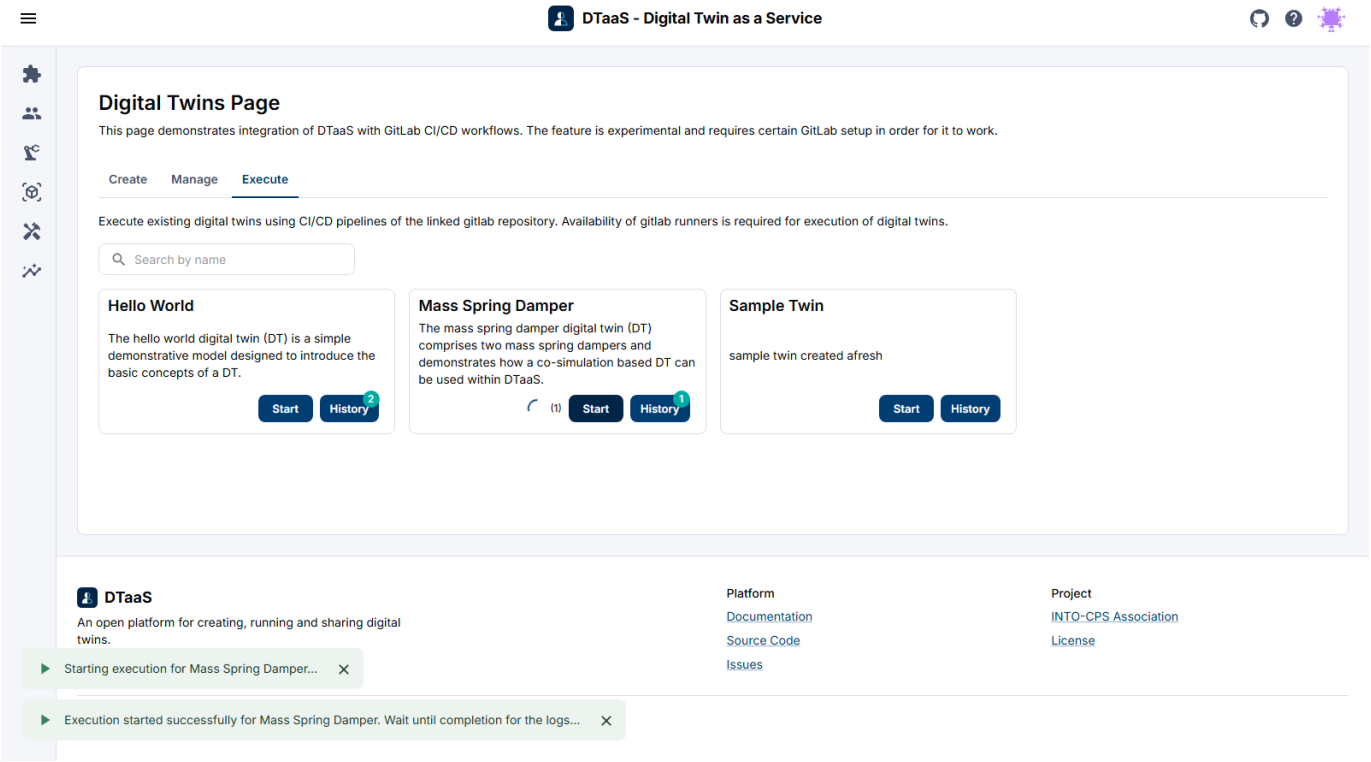
RUNNING DIGITAL TWINS

To run a digital twin, open the **Execute** tab of the **Digital Twins Page**. Click **Automation** in the menu on the left, then click the **Digital Twins Page** card. The page opens in the same browser tab.

The **Execute** tab shows a card for each digital twin, with its name, a short description and the **Start** and **History** buttons. **Start** sends a *job* to the backend, which runs the simulation defined by the digital twin's lifecycle files. You can inspect and change these files with the **Reconfigure** button on the **Manage** tab.

Running Multiple Twins

Pressing **Start** several times queues several simulations. You can also queue different digital twins.



The deployment service (for example, GitLab) balances the load across the available runners, so it can be configured once and then left to run.

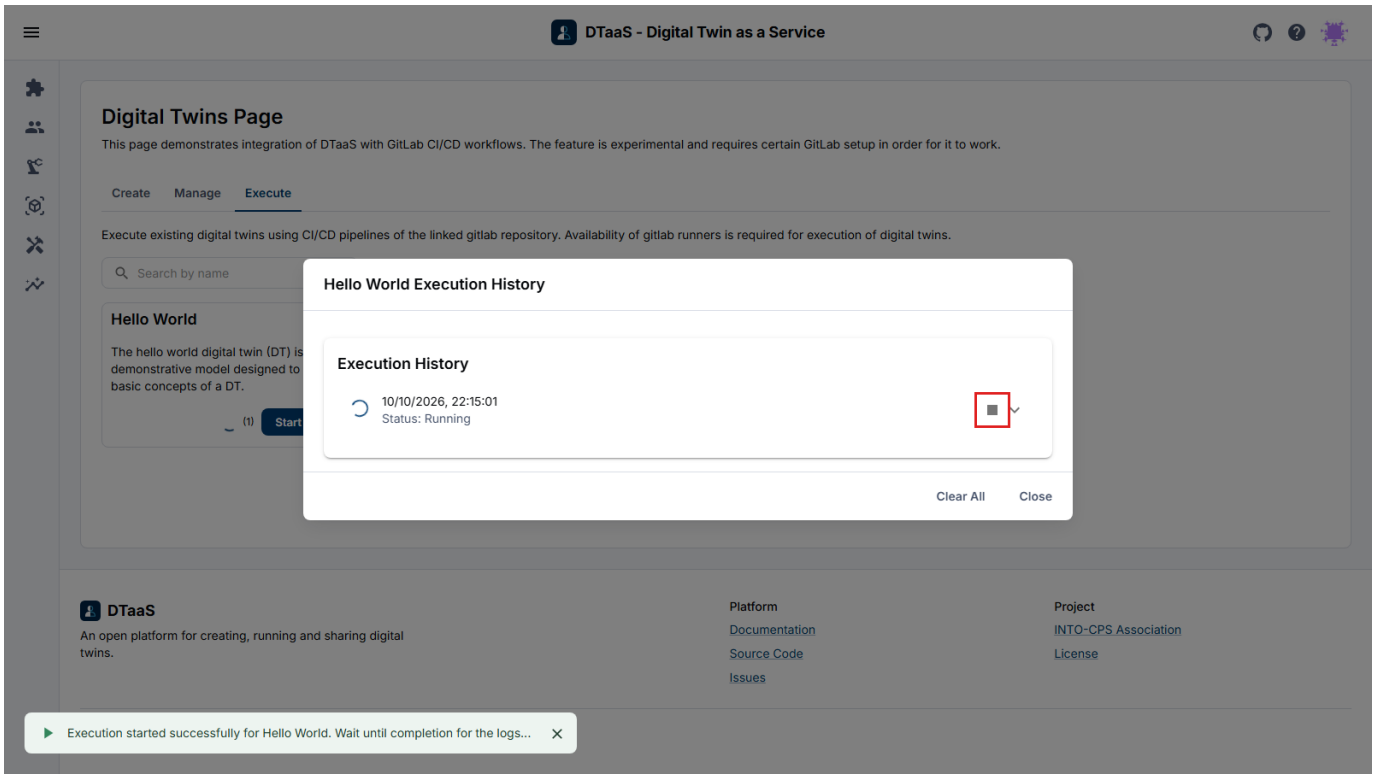
THE EXECUTION LOG

History opens a list of all current and past runs, showing the time of execution and the status (Running, Completed, Failed, Canceled or Timed out). To review the results of a run, click an entry (or the arrow on the right of the entry) to expand it. The log of each job in the pipeline is then shown.

To delete an entry, click the trash icon and confirm the deletion in the dialog. The **Clear All** button deletes all entries of a digital twin.

Click the stop icon of a running entry to stop that execution:

Copyright © 2022 - 2026 The INTO-CPS Association



CHANGING RUNNERS

The runners that pick up jobs can be changed in the settings. See these and other settings.

SUMMARY

This page described the **Execute** tab: running several digital twins at once, and inspecting and deleting their logs.

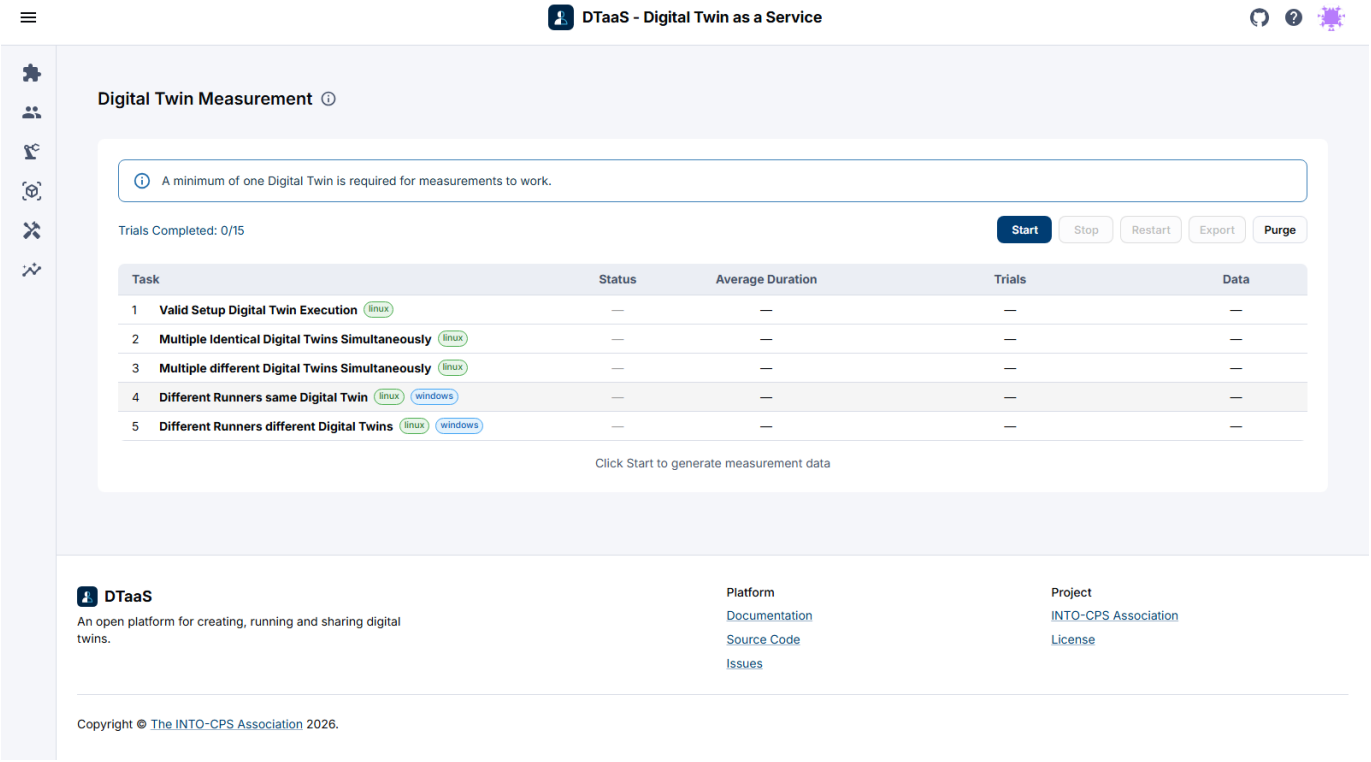
Copyright © 2022 - 2026 The INTO-CPS Association

Performance Measurement

The DTaaS has a built-in tool that measures the execution time of Digital Twin features. Developers can use it to understand performance and to record baselines for later optimization.

ACCESSING THE MEASUREMENT PAGE

Open the measurement page from the **Measurement** card on the **Automation** page, or at `http://localhost:4000/insights/measure` or `https://intocps.org/insights/measure`. Only signed-in users can open this page.



FEATURES

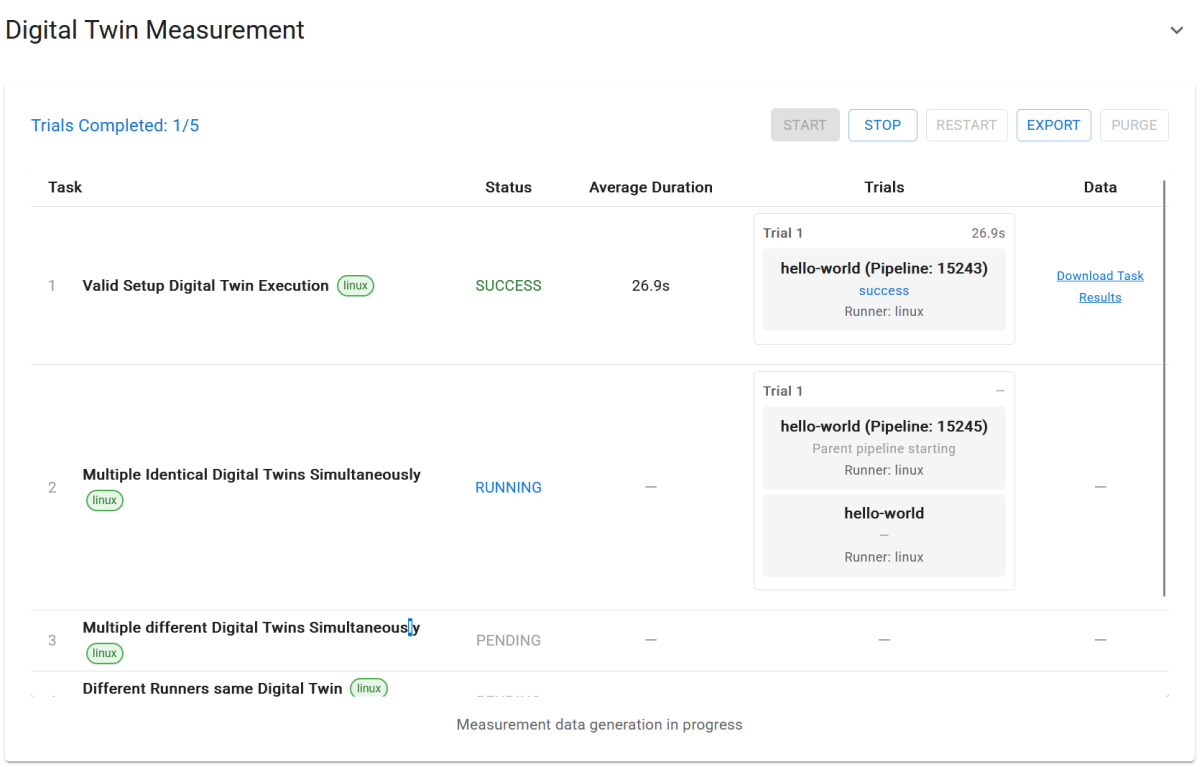
Measurement Tasks

The measurement suite includes the following tasks, each measuring a different aspect of Digital Twin execution:

Task	Description
Valid Setup Digital Twin Execution	Runs the primary Digital Twin with current setup
Multiple Identical Digital Twins Simultaneously	Runs the primary Digital Twin twice at once
Multiple different Digital Twins Simultaneously	Runs the primary and secondary Digital Twins at once
Different Runners same Digital Twin	Runs the primary Digital Twin twice with 2 different runners
Different Runners different Digital Twins	Runs the primary and secondary Digital Twins with 2 different runners

By default, the primary Digital Twin is `hello-world` and the secondary is `mass-spring-damper`. Both can be changed in the measurement settings.

Copyright © 2022 - 2026 The INTO-CPS Association

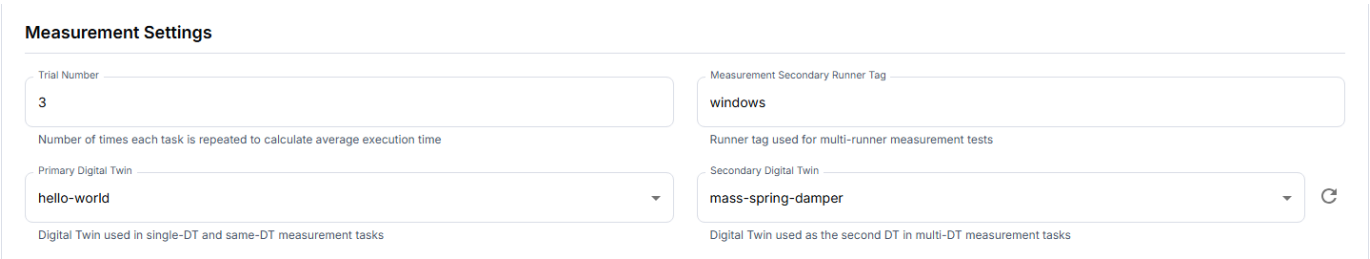


Configuration Options

The configuration is in the **Measurement Settings** section of the **Settings** tab on the **Account** page.

- **Trial Number**: number of times each task is repeated to calculate the average execution time (default: 3).
- **Measurement Secondary Runner Tag**: the runner tag used for multi-runner measurement tests. The primary runner tag is the **Runner Tag** field of the same settings tab.
- **Primary Digital Twin**: the Digital Twin used in single-DT and same-DT measurement tasks (default: `hello-world`).
- **Secondary Digital Twin**: the Digital Twin used as the second DT in multi-DT measurement tasks (default: `mass-spring-damper`).

Note: the first time the settings page is visited in a session, the **Primary Digital Twin** and **Secondary Digital Twin** dropdowns load the list of available Digital Twins. They stay disabled until loading completes.



Controls

- **Start**: begin the measurement suite from the first task
- **Stop**: cancel all running executions (asks for confirmation)
- **Restart**: discard all current results and start again (asks for confirmation)
- **Export**: download all measurement results as a JSON file (available when results exist)
- **Purge**: delete all stored measurement data from the database

Copyright © 2022 - 2026 The INTO-CPS Association

UNDERSTANDING RESULTS

Status Values

Status	Meaning
PENDING	Task has not started yet
RUNNING	Task is currently executing
SUCCESS	All trials completed successfully
FAILURE	One or more trials failed
STOPPED	Task was interrupted by user

Metrics

- **Average Duration**: mean execution time across all trials of a task
- **Total Time**: time for the complete measurement suite (shown after completion)

Measurement Table Columns

The measurement results table has the following columns:

Column	Description
Task	Task number, name, and description
Status	Current execution status (PENDING, RUNNING, SUCCESS, FAILURE, or STOPPED); a dash before the task is queued
Average Duration	Mean execution time across all completed trials, displayed in seconds
Trials	Visual cards showing each trial's execution details and status. Each trial represents one iteration of the task
Data	Download button to export individual task results as JSON

Click a row to expand it and show a toggle next to the task description. A disabled task is skipped during execution.

DATA STORAGE

Measurement data is stored in the browser's IndexedDB, separately from the regular execution history. As a result:

- measurements do not mix with normal DT execution logs
- measurement data can be purged without affecting execution history
- results can be downloaded as JSON for further analysis

DOWNLOADING RESULTS

Per-Task Download

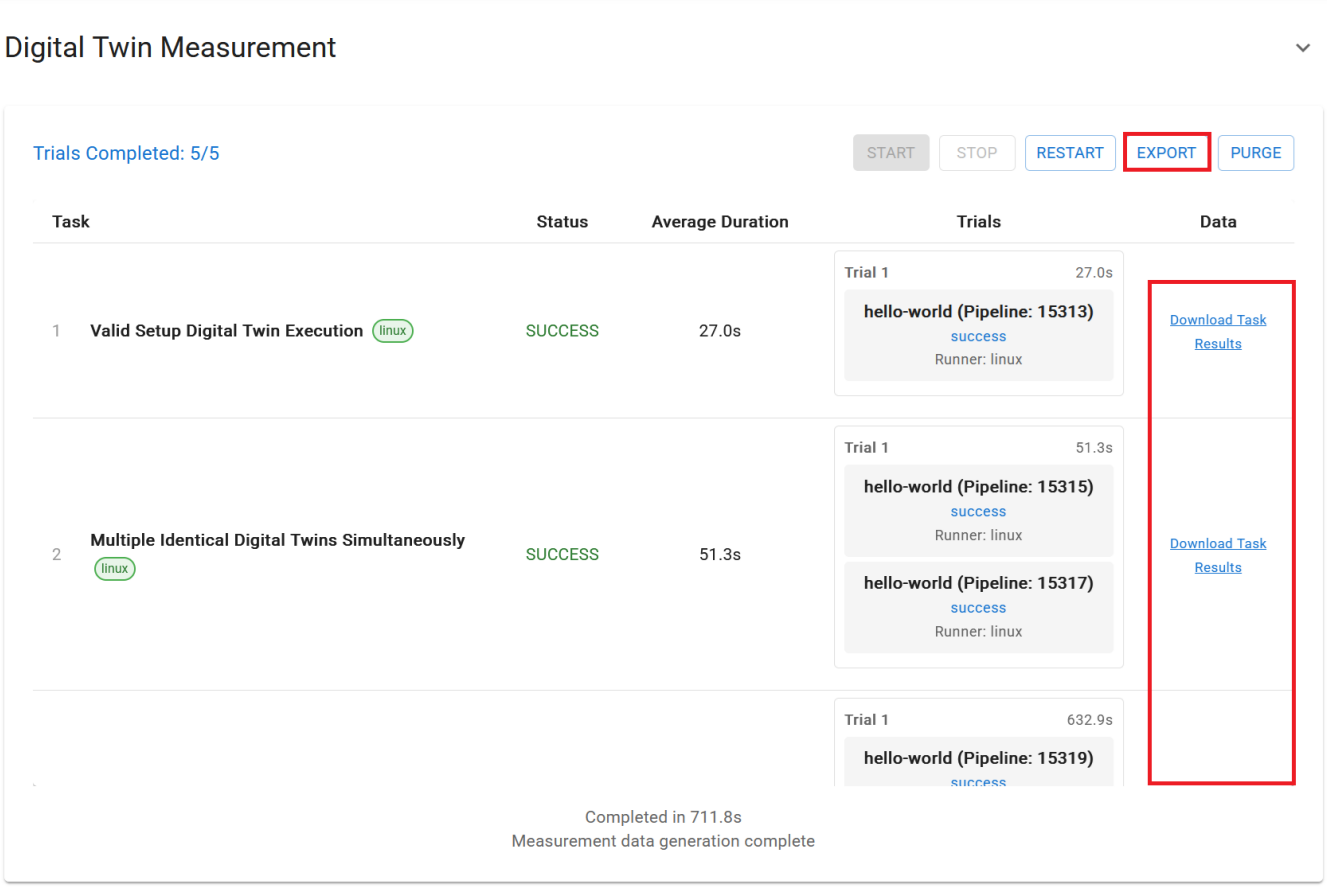
After a task completes successfully, click **Download Task Results** in the Data column to export that task's measurements as JSON.

Full Results Download

The **Export** button in the controls toolbar downloads all measurement results as JSON. It is available whenever results exist.

The figure below shows both options after a completed measurement run.

Copyright © 2022 - 2026 The INTO-CPS Association



JSON Format

The exported JSON describes each task and its trials. Each trial is one iteration of a task and contains an `executions` array with the pipeline execution details. The configuration is shared at the task level, and each execution includes its runner tag. The task records its own start and end times. A trial carries an `Error` field only when it records an error.

Copyright © 2022 - 2026 The INTO-CPS Association

The exported JSON has the following structure:

```
{
  "totalTimeSeconds": number,
  "tasks": [
    {
      "Task Name": string,
      "Description": string,
      "config": {
        "Branch name": string,
        "Group name": string,
        "Common Library project name": string,
        "DT directory": string
      },
      "trials": [
        {
          "Time Start": "ISO8601 timestamp",
          "Time End": "ISO8601 timestamp",
          "Status": "PENDING | RUNNING | SUCCESS | FAILURE | STOPPED",
          "Error": { "message": string, "error": object },
          "executions": [
            {
              "dtName": string,
              "pipelineId": number,
              "status": string,
              "Runner tag": string
            }
          ]
        }
      ],
      "Time Start": "ISO8601 timestamp",
      "Time End": "ISO8601 timestamp",
      "Average Time (s)": number,
      "Status": string
    }
  ]
}
```

TIMING BEHAVIOUR

The measurement system waits 750ms between some requests so that the GitLab instance does not receive them all at once:

- **Between trials**: a 750ms delay is applied before each trial *except the first*. Trial 0 starts immediately; trials 1, 2, and so on each wait 750ms after the previous trial completes.
- **Between executions within a trial**: when a task runs several Digital Twins (for example, "Multiple different Digital Twins Simultaneously"), each execution is staggered by 750ms. The first execution starts immediately, the second after 750ms, the third after 1500ms, and so on.

After starting each pipeline, the measurement also waits 10 seconds and checks that GitLab accepted it. A pipeline that failed or was canceled in that time is started again.

The **Average Duration** of a task includes the stagger delay between executions and the 10-second acceptance check. It does *not* include the delay between trials, because each trial's timer starts after that delay.

BEST PRACTICES

1. Run measurements when the system is lightly used, to get consistent results.
2. Use several trials (3-5) for more reliable averages.
3. Make sure runners are available before starting the measurement.
4. You can navigate to other pages while a measurement runs; execution continues in the background. However, **changing the URL, refreshing or closing the tab stops the measurement**. Notifications tell you when the measurement completes or stops.

TROUBLESHOOTING

Issue	Solution
Task appears stuck, old pipelines appear	Reauthenticate the app by refreshing the tab
Tasks time out	Verify runners are online, use the tag and are accessible

RELATED DOCUMENTATION

- Concurrent Execution

Copyright © 2022 - 2026 The INTO-CPS Association

- Execution Settings
- Capabilities

Copyright © 2022 - 2026 The INTO-CPS Association

3.7 Working with GitLab

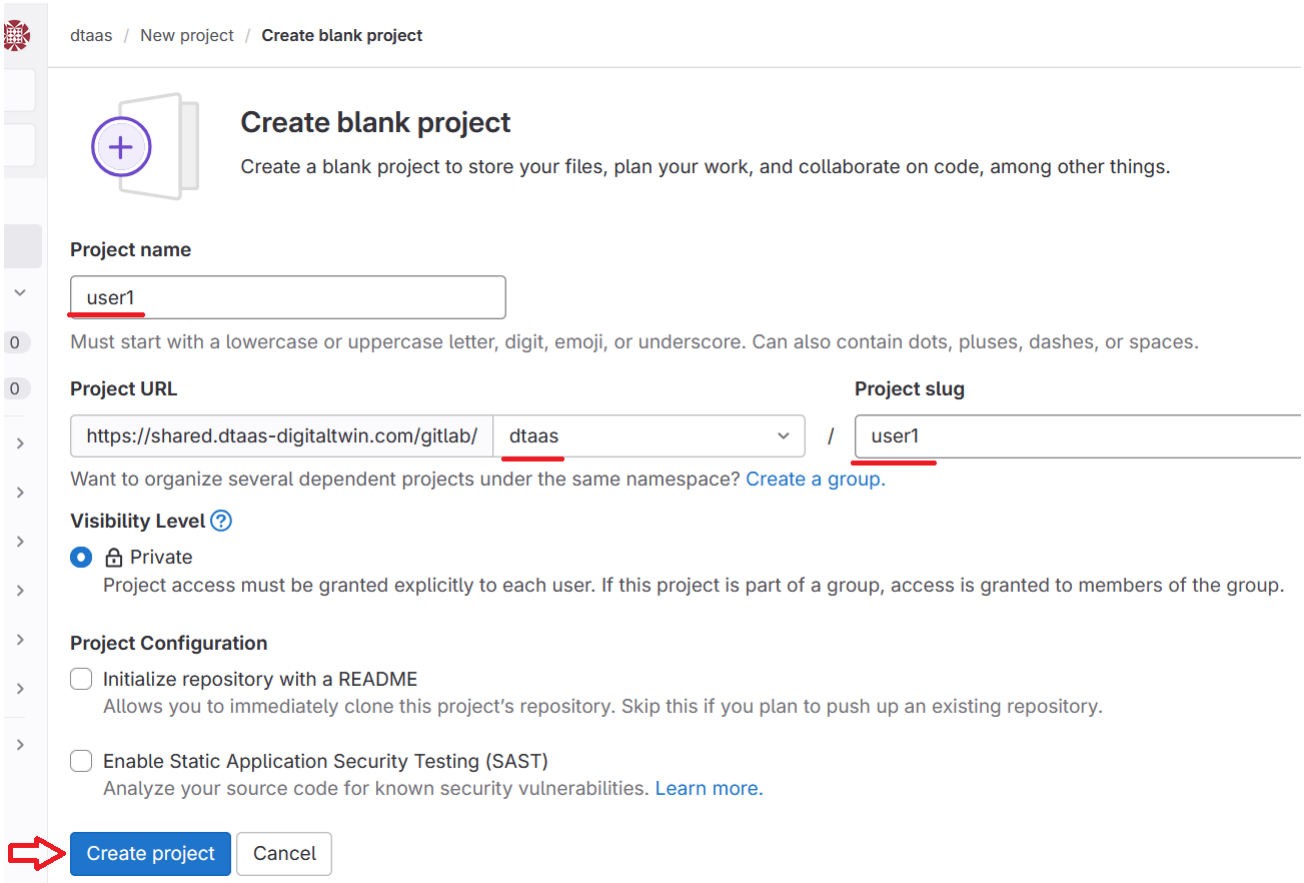
The DTaaS platform uses GitLab for two purposes:

1. OAuth 2.0 authorization service
2. DevOps service

The admin documentation covers the OAuth 2.0 authorization configuration. This guide covers the git commands and project structure for the GitLab DevOps service within the DTaaS.

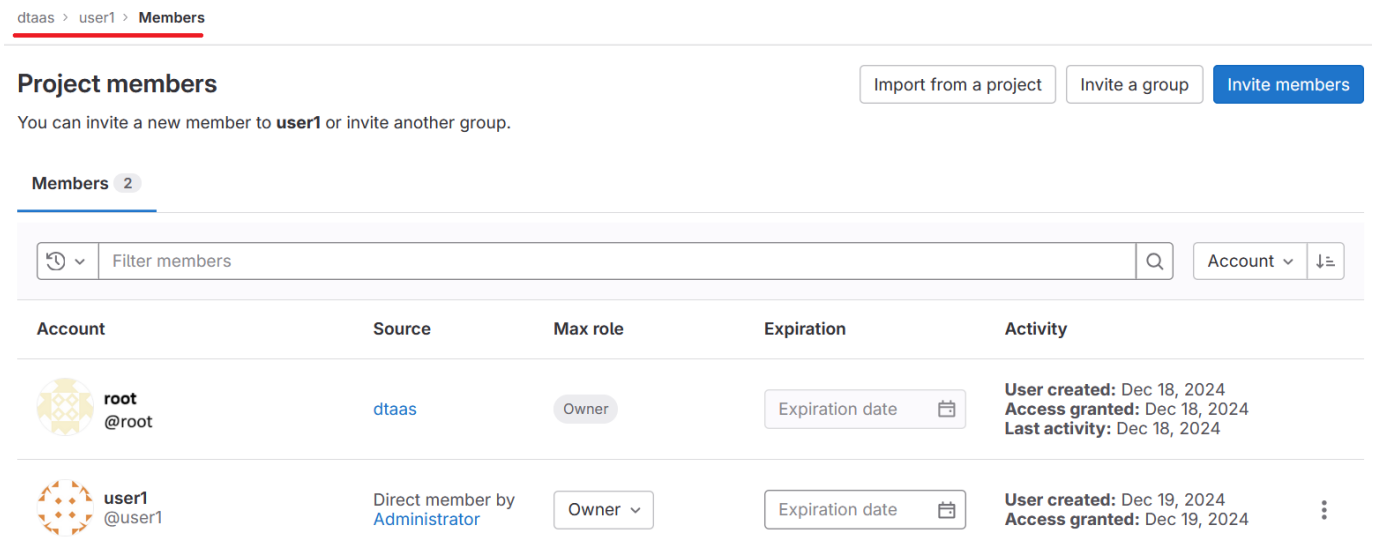
3.7.1 Preparation

First, create a GitLab project named after your *username* in the GitLab group named *dtaas*.



The user must have ownership permissions over the project.

Copyright © 2022 - 2026 The INTO-CPS Association



Warning

By default, the DTaaS website expects a branch named `master` to exist and performs all git operations on it. A different branch can be set in the **Branch Name** field of the settings page.

3.7.2 Git commands

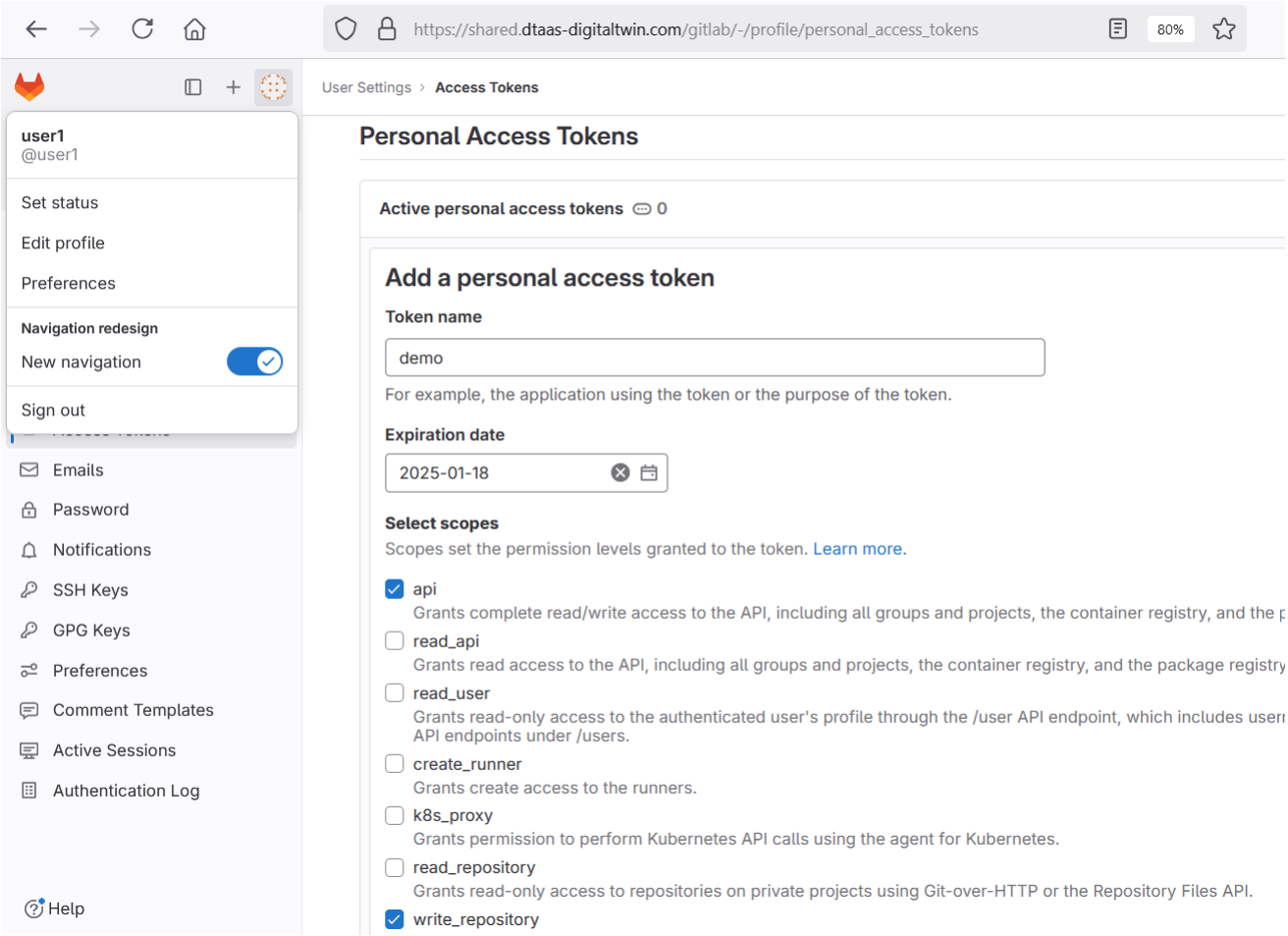
Use the standard git commands and workflows. A GitLab project can be used as a remote git server in two ways:

1. Over SSH using a personal SSH key
2. Over HTTPS using personal access tokens (PAT)

This tutorial uses a PAT to work with the GitLab server.

First, create a PAT.

Copyright © 2022 - 2026 The INTO-CPS Association

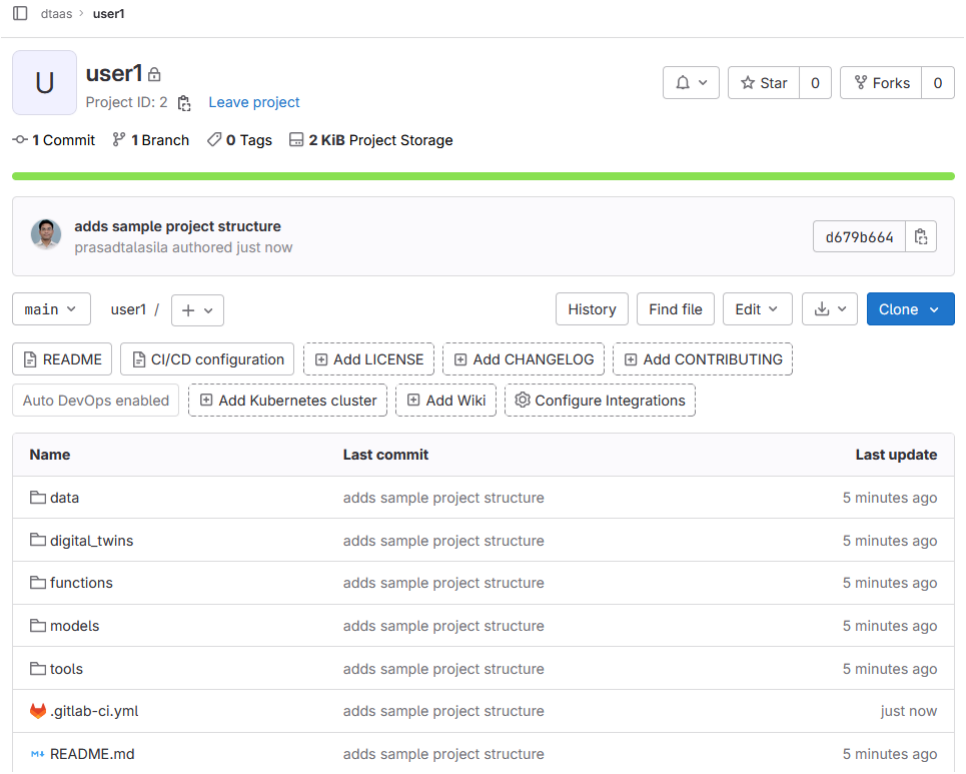


Copy the token and use it to clone the git repository.

3.7.3 Library Assets

GitLab stores the reusable **Library** assets of all users. Library assets, including digital twins, must follow a mandatory structure. A properly initialized GitLab project has the following structure.

Copyright © 2022 - 2026 The INTO-CPS Association



Pay special attention to `.gitlab-ci.yml`. It must be a valid GitLab DevOps configuration. The example repo has a sample structure.

For example, with `PAT1` as the PAT for the `dtaas/user1` repository, the command to clone the repository is:

```
$git clone \
  https://user1:PAT1@dtaas-digitaltwin.com/gitlab/dtaas/user1.git
$cd user1
```

After adding the required Library assets:

```
$git push origin
```

3.7.4 Next Steps

A GitLab runner must be linked to the project repository. Some runners may already be installed with the DTaaS platform; check the runners page of the GitLab project. You can also install custom runners and link them to the repository.

You can then use the Digital Twins Page to access the DevOps features of the DTaaS platform.

Copyright © 2022 - 2026 The INTO-CPS Association

3.8 Runner

A utility service for the safe execution of remote scripts and commands. The user launches it from the command line, and the utility manages the commands to be executed.

The runner runs as a service and provides a REST API for executing remote commands safely. Several runners can be active at the same time on one computer. Commands are sent through the REST API and executed on the computer running the runner.

> **Info**
>
> The runner works on Linux and Windows. On Windows, it runs the PowerShell version of each script.

3.8.1 Install

NPM Registry

The package is available on npmjs.

Install the package with the following command:

```
sudo npm install -g @into-cps-association/runner
```

GitHub Registry

The package is available in GitHub packages registry.

Set the registry and install the package with the following commands:

```
sudo npm config set @into-cps-association:registry \
  https://npm.pkg.github.com
sudo npm install -g @into-cps-association/runner
```

The *npm install* command asks for a username and password. The username is your GitHub username and the password is a GitHub personal access token. For npm to download the package, the personal access token needs the *read:packages* scope.

3.8.2 Configure

The utility needs a configuration file in YAML format. The template configuration file is:

```
port: 5000
location: 'scripts'  # directory of scripts, relative to this config file
commands:  # list of permitted scripts
  - create
  - execute
  - terminate
```

Name the configuration file *runner.yaml* and place it in the directory from which the *runner* microservice is run.

`location` is the path of the scripts directory relative to the *runner.yaml* file.

You can choose any configuration filename and any `location`. The path to the configuration file can be relative or absolute, but the `location` path is always relative to the configuration file.

> **Note**
>
> The commands must be executable. On Linux, make sure the commands have execute permission.

Copyright © 2022 - 2026 The INTO-CPS Association

3.8.3 Create Commands

The runner needs commands or scripts to run. Place them in the `location` directory specified in the *runner.yaml* file. The location is relative to the directory of the *runner.yaml* file.

On Windows, each command needs a PowerShell script with the `.ps1` extension, for example *create.ps1* for the *create* command. The runner executes it with `powershell.exe -NoProfile -ExecutionPolicy Bypass -File`. A directory can hold both versions of a script, so that the same commands work on Linux and Windows.

The runner can also run in a docker container. See the docker instructions.

3.8.4 Use

Display help.

```
$runner -h
Usage: runner [options]

Remote code execution for humans

Options:
  -v --version          package version
  -c --config <string>  runner config file specified in yaml format (default: "runner.yaml")
  -h --help             display help
```

The config option is optional. Without it, **runner** looks for *runner.yaml* in the directory from which it is run. Once launched, the utility listens on the port specified in the *runner.yaml* file.

```
runner  #use runner.yaml of the present working directory
runner -c FILE-PATH       #absolute or relative path to config file
runner --config FILE-PATH #absolute or relative path to config file
```

On the computer where it runs, the service is available at `http://localhost:<port>`.

Other computers on the network can reach it at `http://<ip or hostname>:<port>/`.

Application Programming Interface (API)

Three REST API methods are available. Their routes and responses are:

REST API Route	HTTP Method	Return Value	Comment
localhost:port	POST	Returns the execution status of command	Executes the command provided. Each invocation appends to *array* of commands executed so far.
localhost:port	GET	Returns the execution status of the last command sent via POST request.	
localhost:port/history	GET	Returns the array of POST requests received so far.	

Copyright © 2022 - 2026 The INTO-CPS Association

POST REQUEST TO /

Executes a command. The command must be listed under `commands` in the configuration file and exist in the *location* directory. A command that is not permitted, or that fails, gets an HTTP 400 response.

Valid HTTP Request | **HTTP Response - Valid Command** | **HTTP Response - Invalid Command**

```
POST / HTTP/1.1
Host: intocps.org
Content-Type: application/json
Content-Length: 388

{
  "name": "<command-name>"
}
```

```
Connection: close
Content-Length: 134
Content-Type: application/json; charset=utf-8
Date: Tue, 09 Apr 2024 08:51:11 GMT
Etag: W/"86-ja15r8P5HJu72JcROfBTV4sAn2I"
X-Powered-By: Express

{
  "status": "success"
}
```

```
Connection: close
Content-Length: 28
Content-Type: application/json; charset=utf-8
Date: Tue, 09 Apr 2024 08:51:11 GMT
Etag: W/"86-ja15r8P5HJu72JcROfBTV4sAn2I"
X-Powered-By: Express

{
  "status": "invalid command"
}
```


Copyright © 2022 - 2026 The INTO-CPS Association

GET REQUEST TO /

Shows the status of the command last executed.

Valid HTTP Request

```
GET / HTTP/1.1
Host: intocps.org
Content-Type: application/json
Content-Length: 388

{
  "name": "<command-name>"
}
```

HTTP Response - Valid Command

```
Connection: close
Content-Length: 134
Content-Type: application/json; charset=utf-8
Date: Tue, 09 Apr 2024 08:51:11 GMT
Etag: W/"86-ja15r8P5HJu72JcROfBTV4sAn2I"
X-Powered-By: Express

{
  "name": "<command-name>",
  "status": "valid",
  "logs": {
    "stdout": "<output log of command>",
    "stderr": "<error log of command>"
  }
}
```

HTTP Response - Invalid Command

```
Connection: close
Content-Length: 70
Content-Type: application/json; charset=utf-8
Date: Tue, 09 Apr 2024 08:51:11 GMT
Etag: W/"86-ja15r8P5HJu72JcROfBTV4sAn2I"
X-Powered-By: Express

{
  "name": "<command-name>",
  "status": "invalid",
  "logs": {
    "stdout": "",
    "stderr": ""
  }
}
```

GET REQUEST TO /HISTORY

Returns the array of POST requests received so far. Both valid and invalid commands are recorded in the history.

```
[
  {
    "name": "valid command"
  },
  {
    "name": "valid command"
  },
  {
    "name": "invalid command"
  }
]
```


Copyright © 2022 - 2026 The INTO-CPS Association

3.9 Examples

3.9.1 DTaaS Examples

The DTaaS platform comes with several example digital twins. Work through the steps in this **Examples** section to try the features of the platform and to learn good practice for managing digital twins on it. The following slides and video give an overview of these examples.

The following demos show the DTaaS in use in two projects:

Project	Slides and Videos
CP-SENS project	Project Introduction: slides and video
	Wind turbine testing with the demo inside user workspace
	Python package and demo video
	Videos of a digital twin for structural health monitoring applications: Data Acquisition System: Part-1, Data Acquisition System: Part-2, Operational Modal Analysis, and Model Updating
Incubator	video

Workspace Examples

These examples run interactively inside the DTaaS user workspace. The workspace examples overview lists them and explains how to download and run them.

Automation Examples

These examples use GitLab DevOps pipelines to run the lifecycle scripts of digital twins on the DTaaS platform. See the automation examples overview for details.

DTaaS examples

Copyright © 2022 - 2026 The INTO-CPS Association

3.9.2 Workspace

Workspace Examples

These digital twins run interactively inside the DTaaS user workspace. You can download each example and run it with its lifecycle scripts.

COPY EXAMPLES

Copy all the example code into the user workspace with this script:

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

This copies all examples into the `/workspace/examples` directory.

The script also works when run as root. The examples are then given to the user who ran `sudo`, or to the owner of the directory containing the examples, so that the workspace user can run them. Set `REPOSITORY` and `BRANCH` to download the examples from another repository or branch, for example `REPOSITORY=<user>/DTaaS-examples BRANCH=<branch> bash getExamples.sh`.

The lifecycle scripts locate the models, tools, data and common assets relative to their own location. The examples work from `/workspace/examples` and from any other directory that contains a copy of the DTaaS-examples repository.

EXAMPLE LIST

The examples vary in complexity. Try them in the following order. Some examples need external services, such as RabbitMQ, MQTT or InfluxDB, to be running and reachable from the workspace. Configure the access details of these services as described in the README of each example. The demo video of each example shows it running in a terminal of the workspace desktop of

Copyright © 2022 - 2026 The INTO-CPS Association

DTaaS v1.0 (`workspace-dex-localhost` installation). The videos are sped up: results play at normal speed, installations and long runs at 16x, and everything else at 4x.

#	Example	External services	Video
1	Mass Spring Damper	None	demo
2	Water Tank Fault Injection	None	demo
3	Water Tank Model Swap	None	demo
4	Heart Digital Twin	None	demo
5	Desktop Robotti and RabbitMQ	RabbitMQ	demo
6	Water Treatment Plant and OPC-UA	None (uses a mock OPC-UA server)	demo
7	Water Treatment Plant and OPC-UA (complete version, without lifecycle scripts)	OPC-UA server (mock servers included)	–
8	Three Water Tanks with TwinManager Framework	None	demo
9	Flex Cell with Two Industrial Robots	RabbitMQ, MQTT	demo
10	Incubator	RabbitMQ, InfluxDB	demo
11	Firefighters in Emergency Environments	MQTT, InfluxDB	demo
12	Mass Spring Damper with NuRV Runtime Monitor FMU	None	demo
13	Water Tank Fault Injection with NuRV Runtime Monitor FMU	None	demo
14	Incubator Co-Simulation with NuRV Runtime Monitor FMU	None	demo
15	Incubator with NuRV Runtime Monitor as Service	RabbitMQ, InfluxDB	demo
16	Incubator with NuRV Runtime Monitor FMU as Service	RabbitMQ, InfluxDB	demo
17	Incubator with TeSSLa Runtime Monitor as Service	RabbitMQ, InfluxDB	demo
18	Incubator with TeSSLa Runtime Monitor for Active Control	RabbitMQ, InfluxDB	demo
19	BIM - Building Information Modeling	MQTT	demo

DTaaS examples

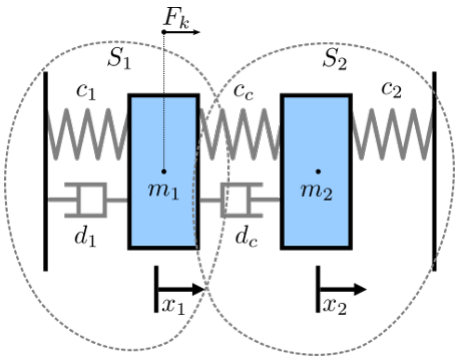
Copyright © 2022 - 2026 The INTO-CPS Association

Mass Spring Damper

OVERVIEW

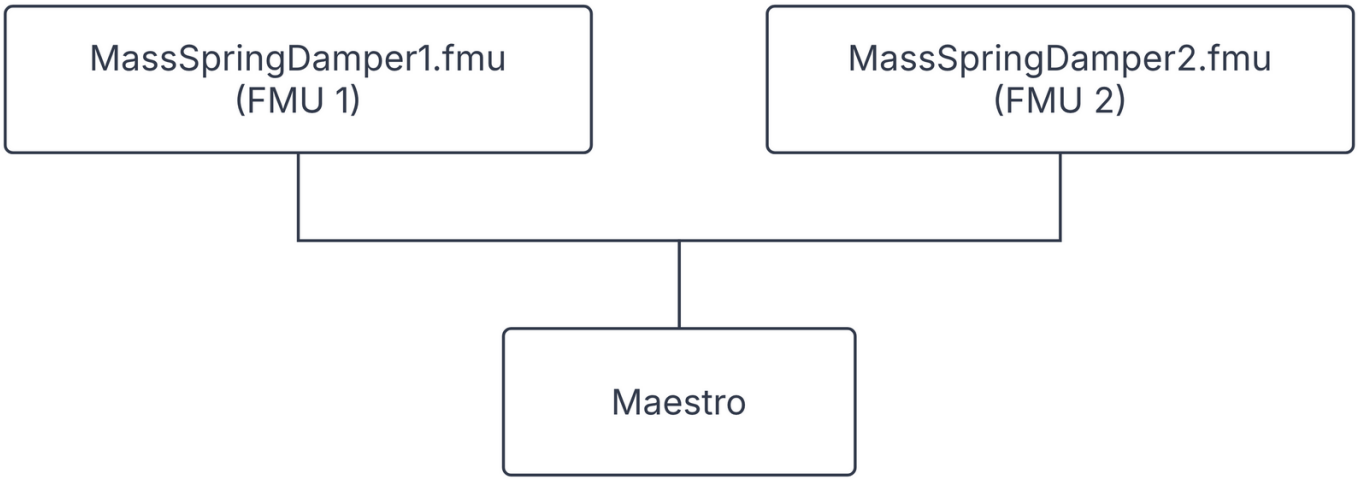
The mass spring damper digital twin (DT) has two mass spring dampers. It shows how to use a co-simulation based DT within the DTaaS.

EXAMPLE DIAGRAM



EXAMPLE STRUCTURE

The study includes two simulators, each representing a mass spring damper system. The first simulator calculates the displacement and speed of mass \(m_1\) for a given force \(F_k\) acting on \(m_1\). The second simulator calculates the force \(F_k\) from a given displacement and speed of mass \(m_1\). Coupling the two simulators computes how the positions of the two masses evolve.



DIGITAL TWIN CONFIGURATION

This example uses two models and one tool:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	MassSpringDamper1.fmu	Private	Yes
	MassSpringDamper2.fmu	Private	Yes
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes

The DT is executed with two configuration files, `cosim.json` and `time.json`. Edit them to customise the DT.

Copyright © 2022 - 2026 The INTO-CPS Association

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 to the example directory.

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

If required, make the lifecycle scripts executable, for example:

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

Then run the following scripts.

Create

Installs Open Java Development Kit 21 in the workspace.

```
lifecycle/create
```

Execute

Runs the digital twin. Because this is a co-simulation based digital twin, the Maestro co-simulation tool runs a co-simulation of the two FMU models.

```
lifecycle/execute
```

Examine the Results

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

The run logs are in */workspace/examples/digital_twins/mass-spring-damper*.

Terminate phase

Removes the debug files and co-simulation output files.

```
lifecycle/terminate
```

REFERENCES

More information about co-simulation techniques and the mass spring damper case study is 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 is in the mass spring damper GitHub repository.

Copyright © 2022 - 2026 The INTO-CPS Association

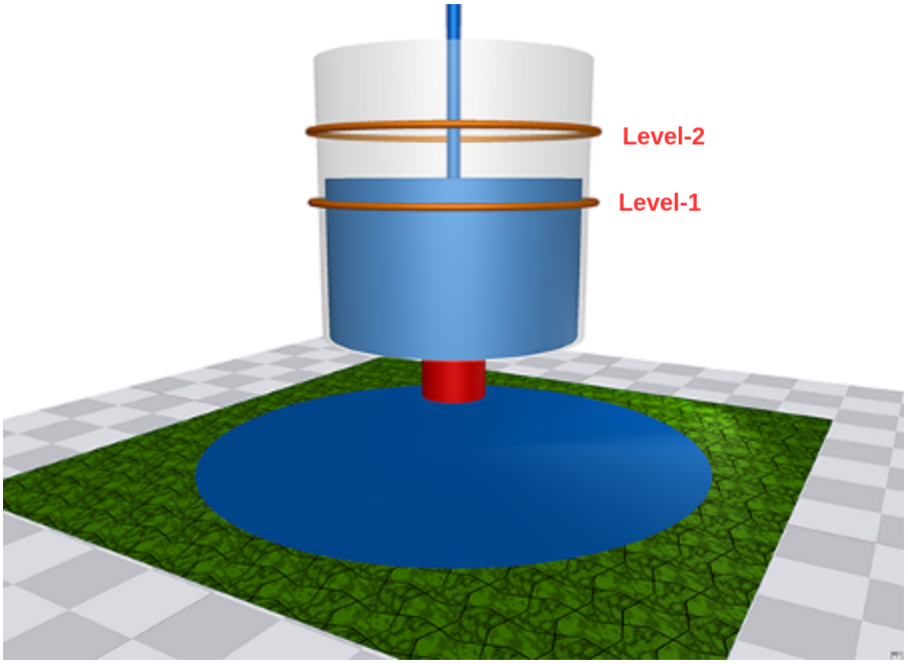
Water Tank Fault Injection

OVERVIEW

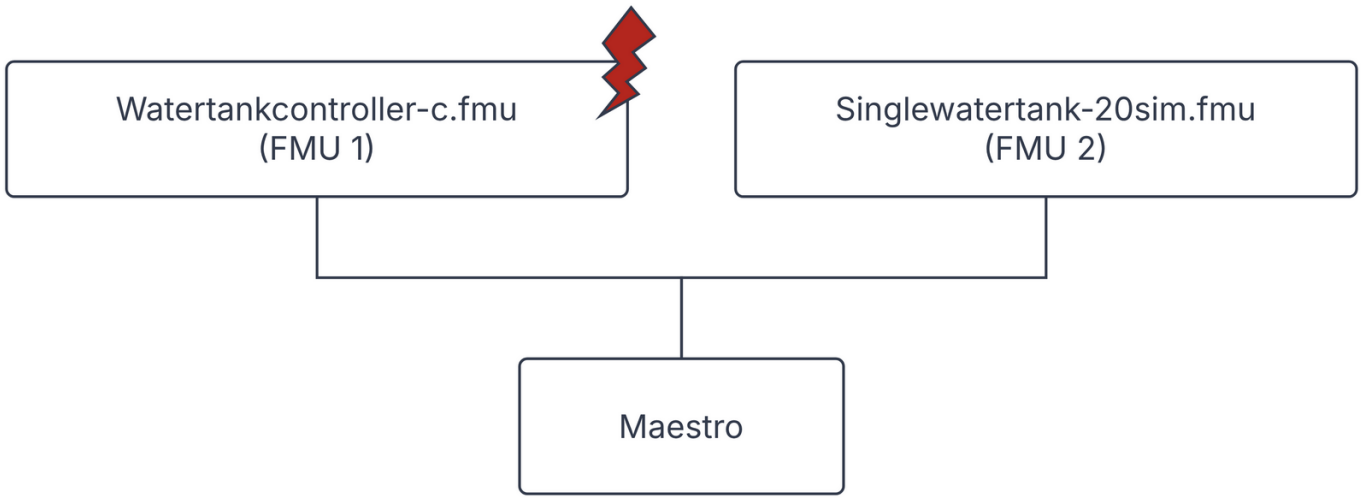
This example shows a digital twin (DT) with fault injection (FI). A live DT receives simulated faults from the environment. The faults are part of the DT configuration and can be changed for new instances of the DT.

This co-simulation based DT uses a water tank case study. The co-simulation consists of a tank and a controller, and its goal is to keep the water level in the tank between `Level-1` and `Level-2`. Faults are injected into the output of the water tank controller (**watertankcontroller-c.fmu**) from 12 to 20 time units. The faults close the tank outlet for a period of time, and the water level rises above the desired level (`Level-2`).

EXAMPLE DIAGRAM



EXAMPLE STRUCTURE



Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses two models and one tool:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	watertankcontroller-c.fmu	Private	Yes
	singlewatertank-20sim.fmu	Private	Yes
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes

The DT is executed with two configuration files, `multimodelFI.json` and `simulation-config.json`. Edit them to customise the DT.

Note

The faults are defined in `/workspace/examples/data/water_tank_FI/input/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 intermediate co-simulation files

RUN THE EXAMPLE

To run the example, change to the example directory.

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

If required, make the lifecycle scripts executable, for example:

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

Then run the following scripts.

Create

Installs Open Java Development Kit 21 and the pandas and matplotlib Python packages from the Ubuntu repositories (`python3-pandas` and `python3-matplotlib`).

```
lifecycle/create
```

Execute

Runs the co-simulation and writes the `outputs.csv` file to `/workspace/examples/data/water_tank_FI/output`.

```
lifecycle/execute
```

Analyze phase

Processes the co-simulation output and saves a plot in `/workspace/examples/data/water_tank_FI/output/plots/`.

```
lifecycle/analyze
```

Examine the results

The results are in the */workspace/examples/data/water_tank_FI/output directory*.

Copyright © 2022 - 2026 The INTO-CPS Association

The run logs are in */workspace/examples/digital_twins/water_tank_FI*.

Terminate phase

Removes the temporary files.

```
lifecycle/terminate
```

REFERENCES

More details on this case study are 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 extends the Maestro co-orchestration engine. It can replace the inputs and outputs of FMUs in an FMI-based co-simulation with tampered values. More details on the plugin are in the fault injection Git repository. The same repository holds the source code for this example in its example directory.

Copyright © 2022 - 2026 The INTO-CPS Association

Water Tank Model Swap

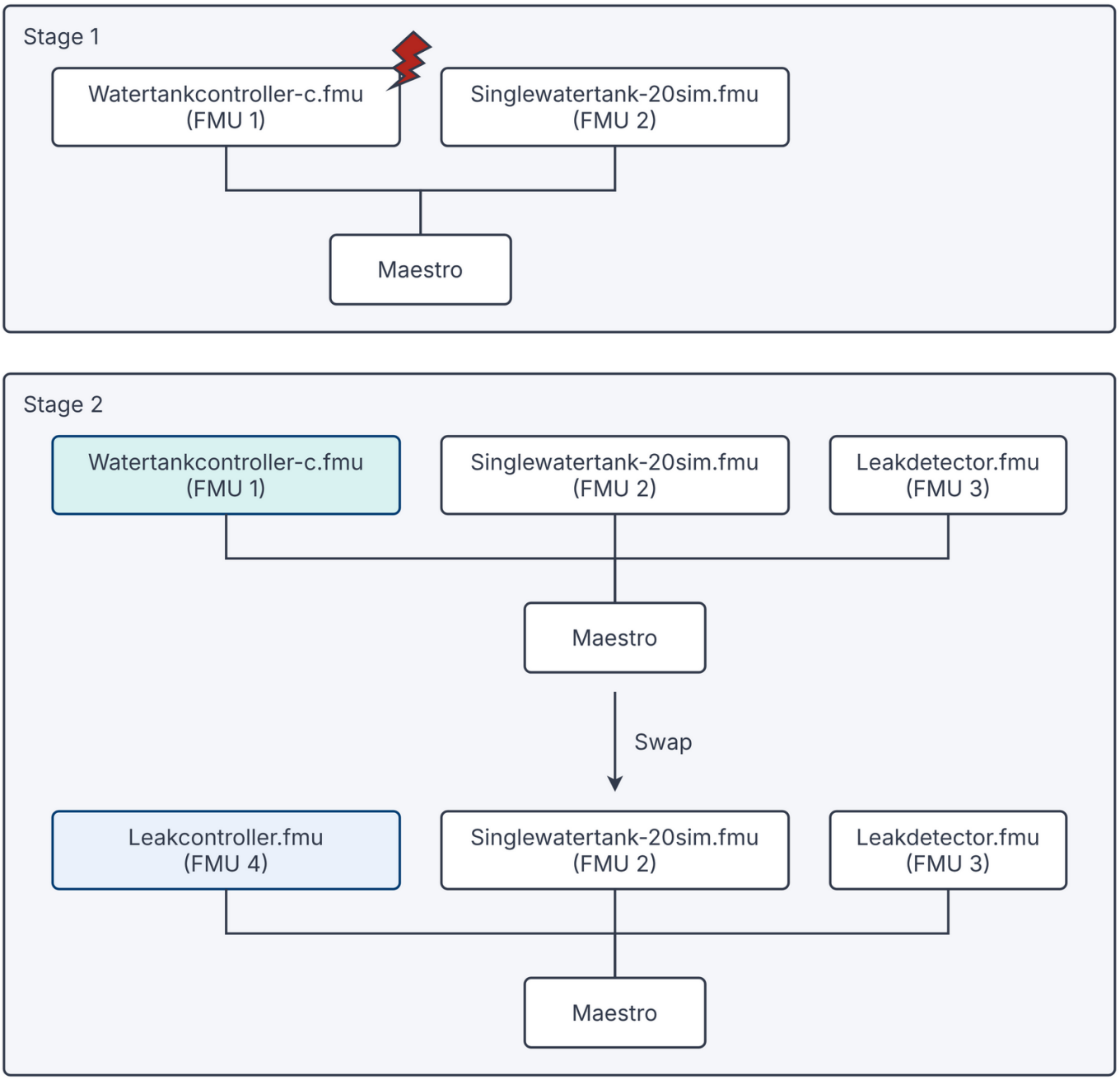
OVERVIEW

This example shows multi-stage execution and dynamic reconfiguration of a digital twin (DT). It demonstrates two features of DTs:

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

The DT is built with the co-simulation methodology.

EXAMPLE STRUCTURE



Copyright © 2022 - 2026 The INTO-CPS Association

CONFIGURATION OF ASSETS

This example uses four models and one tool:

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-4.1.0-jar-with-dependencies.jar	Common	Yes

The DT is executed in two stages, and each stage has its own configuration files. The following table lists 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	Processes the co-simulation output and produces plots
Clean	Clears run logs and intermediate co-simulation files

RUN THE EXAMPLE

To run the example, change to the example directory.

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

If required, make the lifecycle scripts executable, for example:

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

Then run the following scripts.

Create

Installs Open Java Development Kit 21 and, from the Ubuntu repositories, the Python packages used by the plotting script and the Python FMUs (`python3-pandas` , `python3-matplotlib` , `python3-protobuf` and `python3-zmq`).

```
lifecycle/create
```

Execute

This DT runs in two stages. The first stage runs a co-simulation of the watertankcontroller-c.fmu and singlewatertank-20sim.fmu models. During this stage, faults are injected into one of the models (watertankcontroller-c.fmu) and the system performance is checked.

Copyright © 2022 - 2026 The INTO-CPS Association

The second stage runs another co-simulation with three FMUs: watertankcontroller-c, singlewatertank-20sim and leak_detector. The Maestro tool has a built-in monitor, which is enabled during this stage, and a swap condition is set at the start of the stage. When the swap condition is satisfied, Maestro swaps out the watertankcontroller-c.fmu model and swaps in the leak_controller.fmu model. This model swap demonstrates the dynamic reconfiguration of a DT.

At the end of the execution phase, the `outputs.csv` file of each stage is copied to the `stage1` and `stage2` subdirectories of `/workspace/examples/data/water_tank_swap/output`.

```
lifecycle/execute
```

Analyze phase

Processes the co-simulation output and saves a plot in `/workspace/examples/data/water_tank_swap/output/plots/`.

```
lifecycle/analyze
```

Examine the results

The results are in the */workspace/examples/data/water_tank_swap/output directory*.

The run logs are in */workspace/examples/digital_twins/water_tank_swap*.

Terminate phase

Removes the temporary files.

```
lifecycle/terminate
```

REFERENCES

The complete source of this example is in the model swap GitHub repository.

The run-time model (FMU) swap mechanism used in this example is described 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).
```

Runtime reconfiguration of a co-simulation by changing the Functional Mockup Units (FMUs) it uses is described further 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.
```


Copyright © 2022 - 2026 The INTO-CPS Association

Heart Digital Twin

OVERVIEW

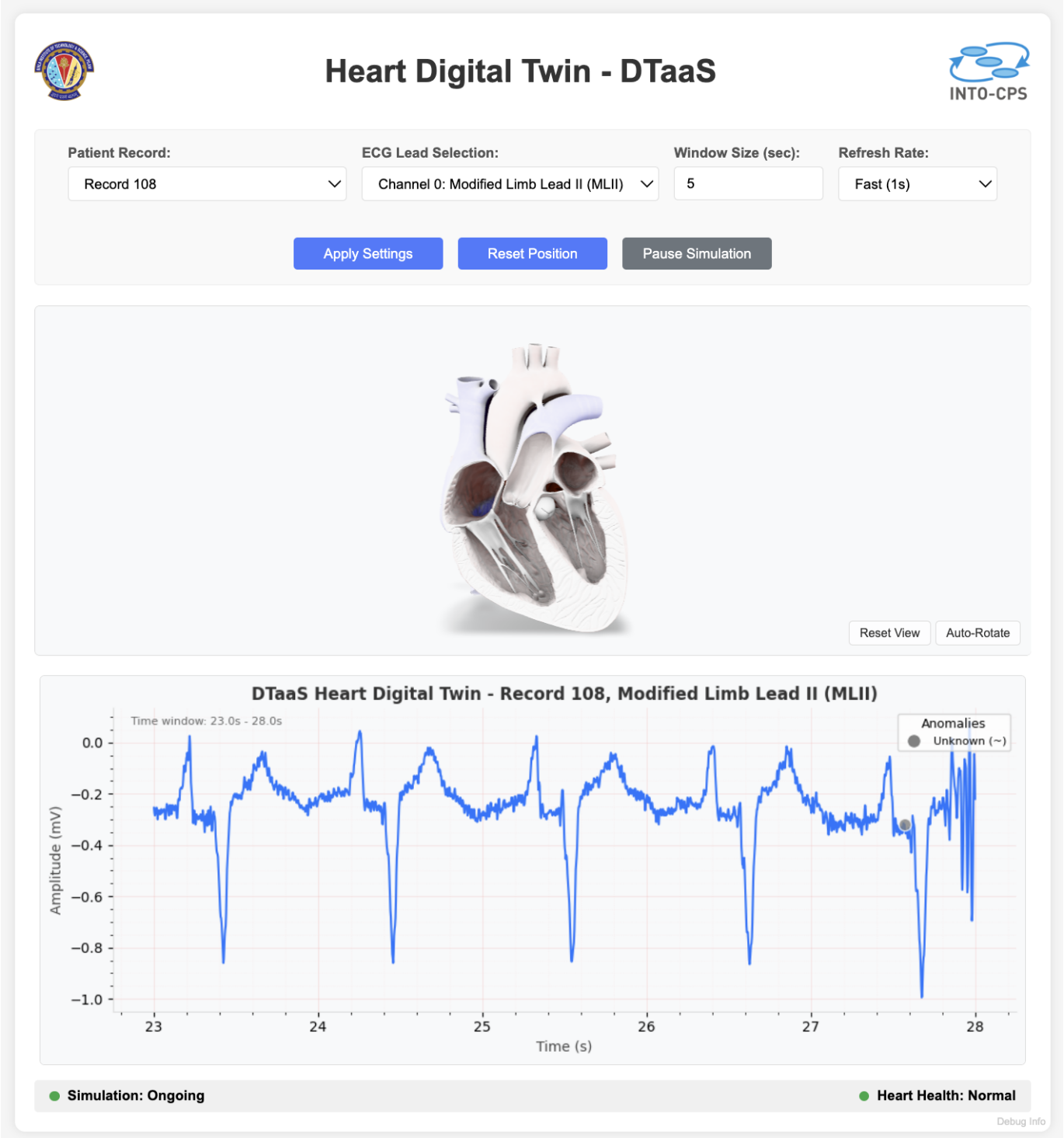
The Heart Digital Twin is a real-time cardiovascular monitoring system that uses ECG data from the MIT-BIH Arrhythmia Database. It shows live ECG plots with anomaly detection, alongside an interactive 3D heart model that beats in time with the cardiac rhythm.

EXAMPLE STRUCTURE

The Heart Digital Twin has these components:

- **ECG Data Processing**: Real-time processing and visualisation of electrocardiogram signals from the MIT-BIH database
- **3D Heart visualisation**: Interactive 3D heart model with synchronised pumping animation
- **Anomaly Detection**: Real-time identification and highlighting of cardiac anomalies
- **Web Interface**: Flask-based web application for real-time monitoring and control

Copyright © 2022 - 2026 The INTO-CPS Association



MIT-BIH ARRHYTHMIA DATABASE

This digital twin uses the **MIT-BIH Arrhythmia Database**, which contains 48 half-hour excerpts of two-channel ambulatory ECG recordings. The database includes:

- Records from 47 subjects studied by the BIH Arrhythmia Laboratory
- 360 Hz sampling frequency with 11-bit resolution
- Expert annotations for arrhythmia detection and classification
- Multiple ECG channels (MLII, V1, V2, V4, V5) depending on the record

Records 100-109 are available for simulation. Each contains different types of cardiac rhythms and anomalies.

Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Data	MIT-BIH Arrhythmia Database Records (100-109)	Public	Yes
Models	3D Heart Model (Beating heart.glb)	Public	Yes
Functions	ECG Processing & Anomaly Detection	Public	Yes
Tools	Flask Web Server & UI Components	Public	Yes

You can customise the heart digital twin with these parameters:

- **Record Selection**: Choose from records 100-109
- **Channel Selection**: Switch between available ECG channels
- **Window Size**: Adjust the time window for real-time display
- **Update Frequency**: Control the refresh rate of the simulation

3D HEART ANIMATION

The digital twin includes a **synchronised 3D heart pumping animation**:

- **Real-time Synchronisation**: Heart model animation matches the ECG rhythm
- **Interactive Controls**:
- Camera controls for 360° viewing
- Auto-rotate functionality
- Reset view option
- **Animation Features**:
- Realistic beating heart model
- Smooth cardiac cycle animation
- Pause/resume functionality synchronised with ECG simulation
- **Model Format**: GLB (GL Transmission Format Binary) 3D model
- **Rendering**: WebGL-based rendering using model-viewer component

The 3D heart moves in step with the electrical activity shown in the ECG, which helps to relate the electrical signals to the mechanical function of the heart.

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Sets up Python virtual environment and installs required dependencies (NumPy, Pandas, Matplotlib, WFDB, Flask)
Execute	Starts the Flask web server and launches the heart monitoring interface accessible via web browser
Clean	Terminates the running application and removes the virtual environment

RUN THE EXAMPLE

To run the example, go to the following directory:

```
cd /workspace/examples/digital_twins/heart_dt
```


Copyright © 2022 - 2026 The INTO-CPS Association

If required, make the lifecycle scripts executable:

```
chmod +x lifecycle/create
chmod +x lifecycle/execute
chmod +x lifecycle/clean
```

Now, run the following scripts:

Create

Sets up the Python environment and installs the required dependencies, including WFDB for MIT-BIH data access, Flask for the web interface, and scientific computing libraries.

```
lifecycle/create
```

Execute

Starts the Heart Digital Twin web application in the background. Open it in a web browser for real-time ECG monitoring and 3D heart visualisation.

```
lifecycle/execute
```

The application is available at `http://localhost:5001`. On start-up it:

- Loads the required MIT-BIH records
- Initialises the ECG data processing
- Starts real-time visualisation
- Enables 3D heart model synchronisation

Examine the results

Use the web interface to:

- **Monitor Real-time ECG**: View live electrocardiogram signals
- **Control Simulation**: Select different records, channels, and update frequencies
- **Interact with 3D Heart**: Rotate, zoom, and control the heart animation
- **Detect Anomalies**: Observe highlighted cardiac irregularities
- **Pause/Resume**: Control the simulation flow

The application writes its logs to the `logs/heart_dt.log` file.

Clean

Stops the running application and cleans up the environment.

```
lifecycle/clean
```

This script:

- Stops the Flask web server
- Removes the Python virtual environment

REFERENCES

More information about the MIT-BIH Arrhythmia Database and cardiac monitoring:


```
Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database.
IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001).
```

```
Goldberger, A., et al. PhysioBank, PhysioToolkit, and PhysioNet:
Components of a new research resource for complex physiologic signals.
Circulation 101(23):e215-e220, 2000.
```


Copyright © 2022 - 2026 The INTO-CPS Association

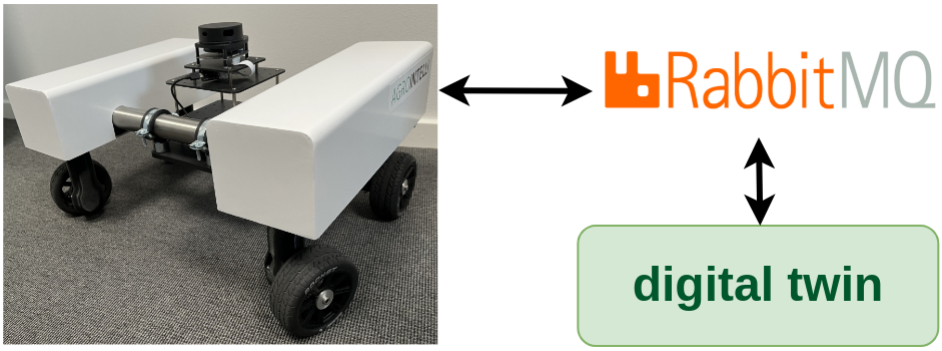
The MIT-BIH Arrhythmia Database is available at: https://physionet.org/content/mitdb/

Copyright © 2022 - 2026 The INTO-CPS Association

Desktop Robotti with RabbitMQ

OVERVIEW

This example shows bidirectional communication between a mock physical twin and a digital twin of a mobile robot (Desktop Robotti). A RabbitMQ broker carries the messages between them.



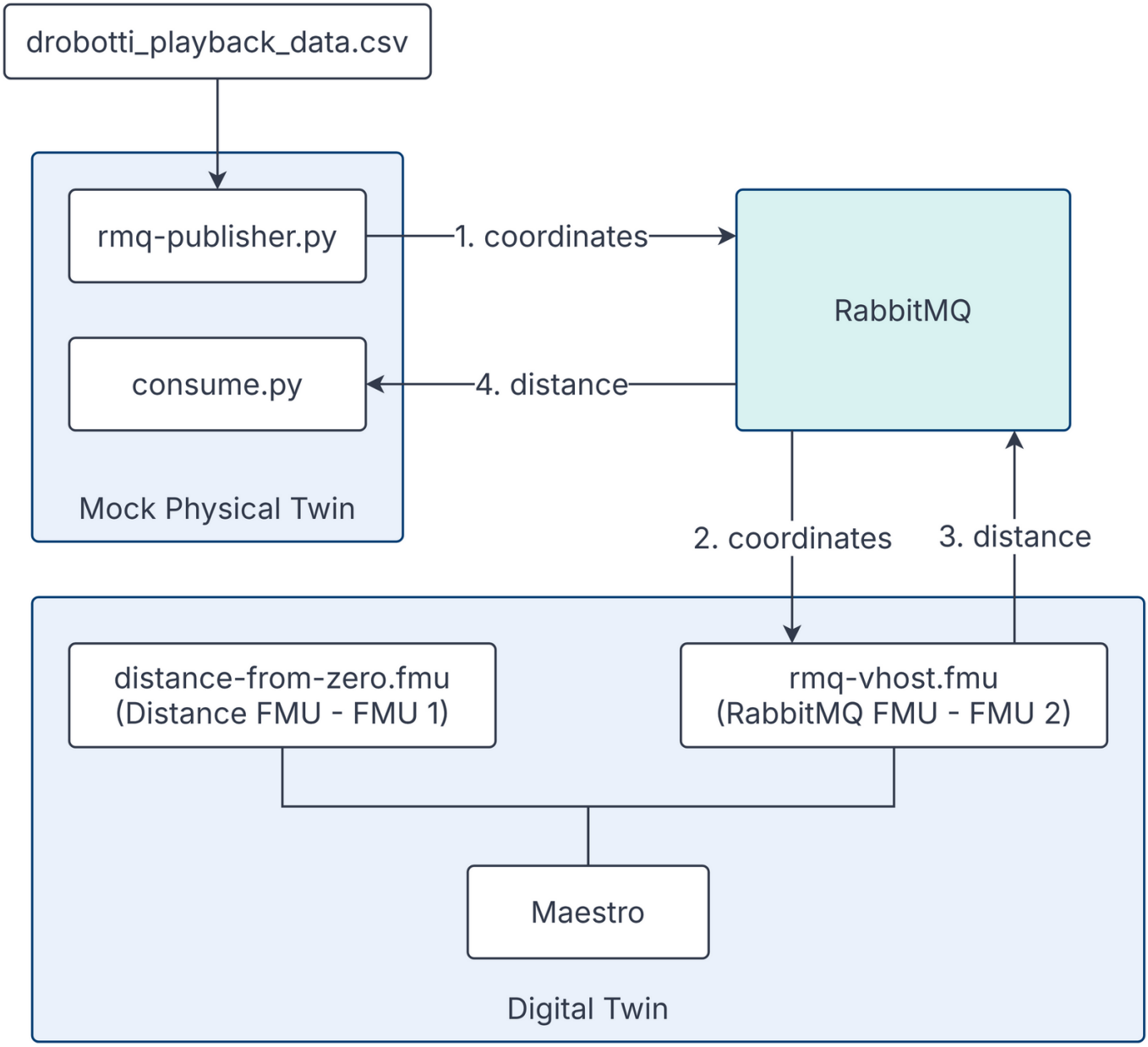
EXAMPLE STRUCTURE

Two Python scripts make up the mock physical twin of the mobile robot:

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

The mock physical twin sends its location as *(x,y)* coordinates and expects the digital twin to return the Cartesian distance \(\sqrt(x^2 + y^2)\).

Copyright © 2022 - 2026 The INTO-CPS Association



The *rmq-publisher.py* script reads the recorded *(x,y)* coordinates of the mobile robot from a data file and publishes them to the RabbitMQ broker, where the digital twin consumes them.

The *consume.py* script subscribes to the RabbitMQ broker and waits for the distance calculated by the digital twin.

The digital twin is an FMI-based co-simulation, with Maestro as the co-orchestration engine. The co-simulation uses two FMUs: the RMQ FMU (rmqfmu-vhost.fmu) and the distance FMU (distance_from_zero.fmu). The RMQ FMU receives the *(x,y)* coordinates from *rmq-publisher.py* and sends the calculated distance to *consume.py*. It uses the RabbitMQ broker to communicate with the mock mobile robot, that is, *rmq-publisher.py* and *consume.py*. The distance FMU calculates the distance between *(0,0)* and *(x,y)*. The two FMUs exchange values during the co-simulation.

Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses two models, one tool, one data file and two scripts that form the 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
	rmqfmu-vhost.fmu	Private	Yes
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes
Data	drobotti_playback_data.csv	Private	No
Mock PT	rmq-publisher.py	Private	No
	consume.py	Private	No

The DT has several configuration files. `coe.json` and `multimodel.json` are used to execute the digital twin and can be modified to customise the DT.

The RabbitMQ access credentials go in `multimodel.json` for the RMQ FMU and in `rabbitMQ-credentials.json` for the mock PT Python scripts. Add the appropriate credentials to both files.

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool and Python 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 to the example directory.

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

If required, give the lifecycle scripts execute permission, for example:

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

Now run the following scripts.

Create

Installs Open Java Development Kit 21 in the workspace, along with the Python packages needed by the *rmq-publisher.py* and *consume.py* scripts and by the *distance_from_zero.fmu* model (`python3-pika` , `python3-protobuf` and `python3-zmq`).

```
lifecycle/create
```

Execute

Starts the Python scripts of the mock physical twin and then the digital twin. Since this is a co-simulation based digital twin, the Maestro co-simulation tool runs the co-simulation with the two FMU models.

```
lifecycle/execute
```

Examine the results

The results are in the */workspace/examples/digital_twins/drobotti_rmqfmu* directory.

Copyright © 2022 - 2026 The INTO-CPS Association

Executing the DT generates and launches a co-simulation (RMQFMU and distance FMU) and two Python scripts: one publishes data read from a file, and the other consumes what the distance FMU sends.

In this example the DT runs for 10 seconds with a step size of 100 ms. Afterwards, the logs are in `/workspace/examples/digital_twins/drobotti_rmqfmu/target`. The `outputs.csv` file records the outputs of each FMU: xpos and ypos for the RMQFMU, and the distance for the distance FMU. Each FMU and the publisher scripts also write their own logs. Note that the RMQFMU sends data only when the current input differs from 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 the documentation and source code of the RabbitMQ 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.
```


Copyright © 2022 - 2026 The INTO-CPS Association

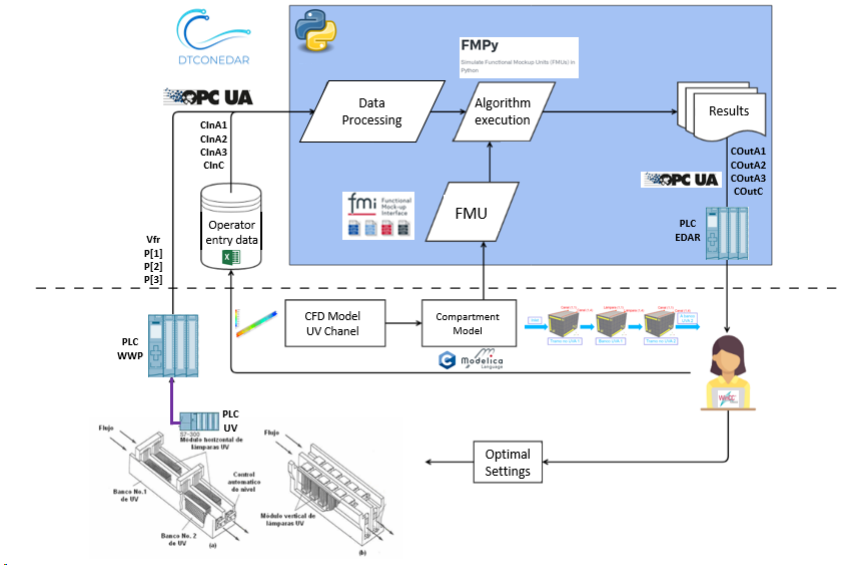
Waste Water Plant with OPC-UA

INTRODUCTION

Waste water treatment (WWT) plants must keep substance and species concentrations within regulatory limits to ensure water quality. This is usually checked by taking periodic samples and analysing them in the laboratory, so plant operators lack continuous information for their decisions and set operating setpoints higher than needed to guarantee water quality. Some WWT processes consume a lot of power, so adjusting the setpoints could reduce energy consumption significantly.

PHYSICAL TWIN OVERVIEW

This example shows 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 digital twin provides "virtual sensors" that give the plant operator continuous information for decision making.



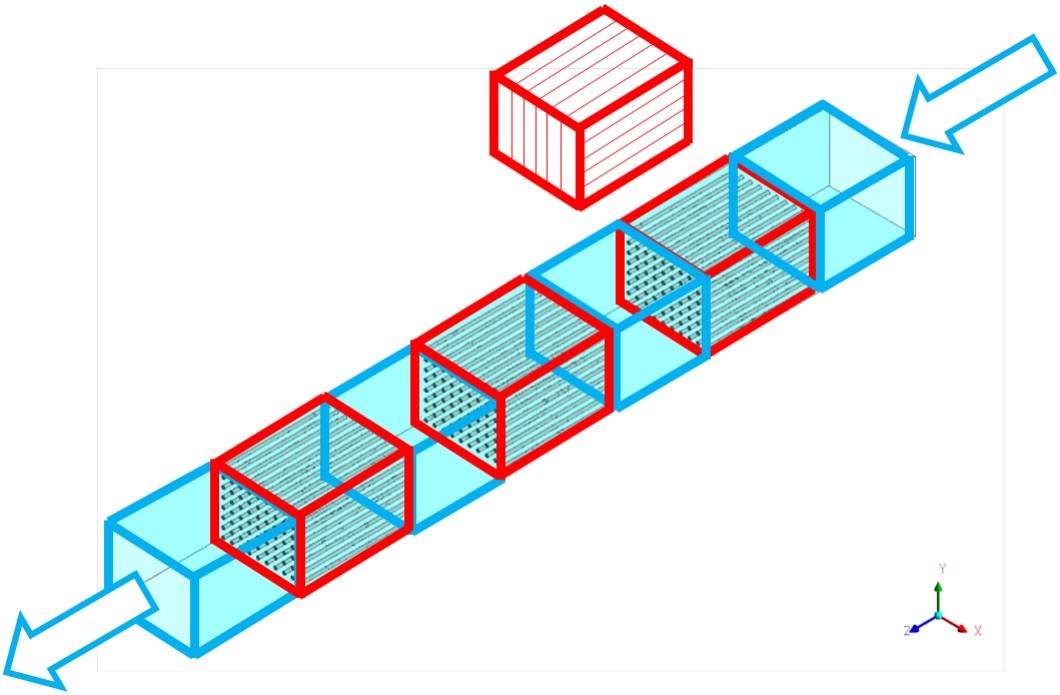
The physical twin is an ultraviolet channel. A PLC controls the power of the UV lamps needed to kill all the pathogens in the flow. The channel has 3 groups of UV lamps, so the real channel (and its mathematical model) is divided into 7 zones: 4 without UV lamps (the entrance and exit of the channel, plus 2 zones between lamp groups) and 3 for the UV lamps.

The dose to apply (related to the power) changes with the residence time, computed from the measured volume flow, and with the UV intensity measured by the intensity sensor.

The volumetric flow and the power in the three parts of the channel are transmitted to the PLC of the plant. The PLC also works as an OPC UA server that sends data to and receives data from an OPC UA client. Some sizing parameters and initial values, such as the initial concentrations of disinfectant and pathogens, are read from a spreadsheet that the plant operator fills in. The spreadsheet is an Open Office file (.ods) because of the software installed on the SCADA PC. Some of the values the operator defines are input signals that are not measured yet but are expected to be measured in the future.

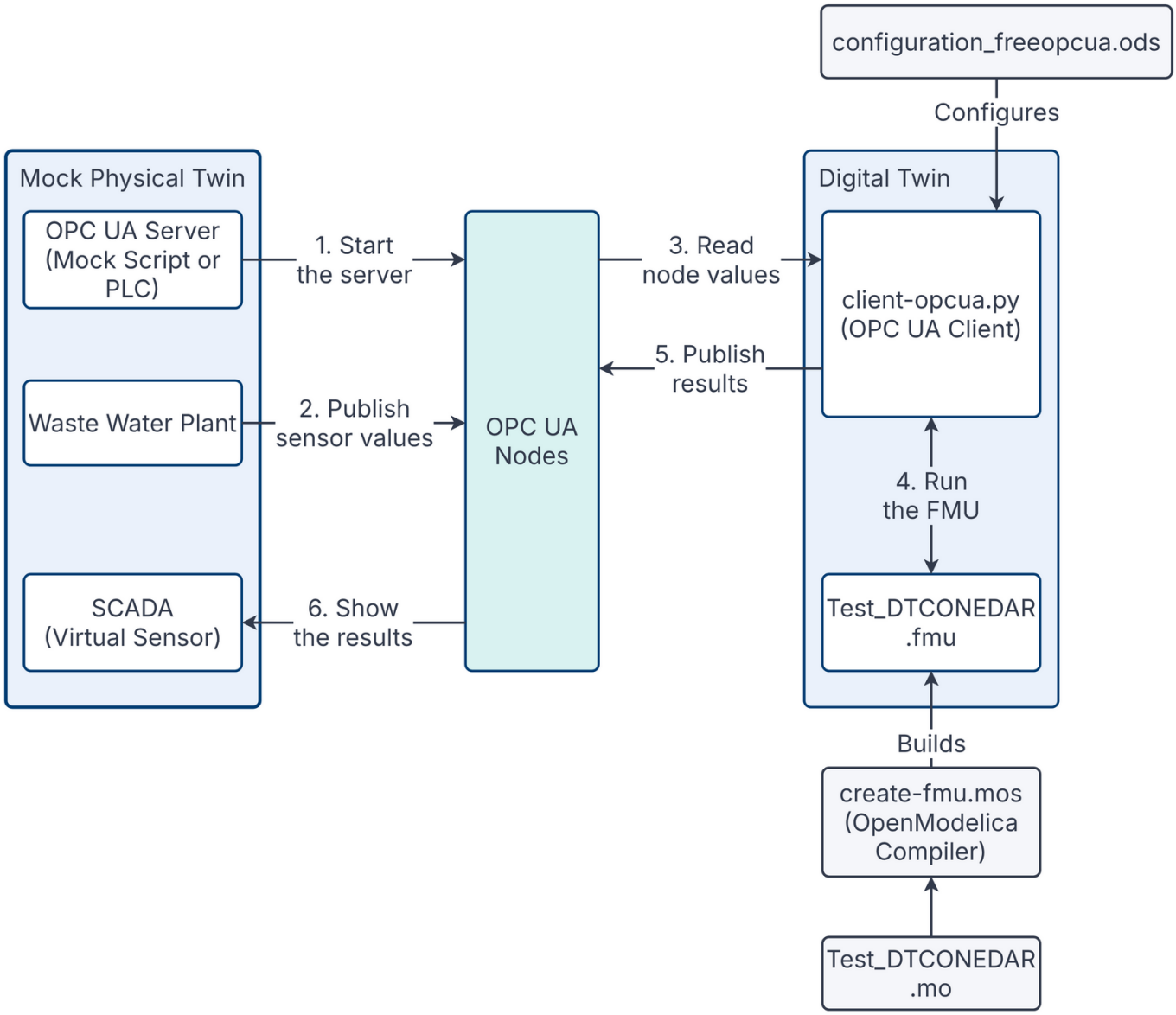
Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN OVERVIEW



The digital twin is a reduced model, written in C, that solves the physical conservation laws (mass, energy and momentum) but simplifies details, mainly the geometry, to calculate in real time while keeping the results accurate. Its results are compared with those of the CFD model. An OpenModelica model uses the C solver, and OpenModelica exports it as an FMU so that the OPC UA client (*client-opcua.py*) can integrate it.

Copyright © 2022 - 2026 The INTO-CPS Association



DIGITAL TWIN CONFIGURATION

Asset Type	Name of Asset	Visibility	Reuse in Other Examples
Model	Test_DTCONEDAR.mo	private	No
Data	configuration_freeopcua.ods	private	No
	model_description.csv (generated by client-opcua.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

This example uses a dummy model of the plant instead of the real model. The simplified model, which does not contain the real equations, is written in OpenModelica (`Test_DTCONEDAR.mo`). The FMU is generated with OpenModelica to obtain the binaries needed to run it. A previously generated FMU can be used, but to make sure the right binaries are used, install the OpenModelica compiler and run `create-fmu.mos` to build the FMU from the Modelica file `Test_DTCONEDAR.mo`.

Copyright © 2022 - 2026 The INTO-CPS Association

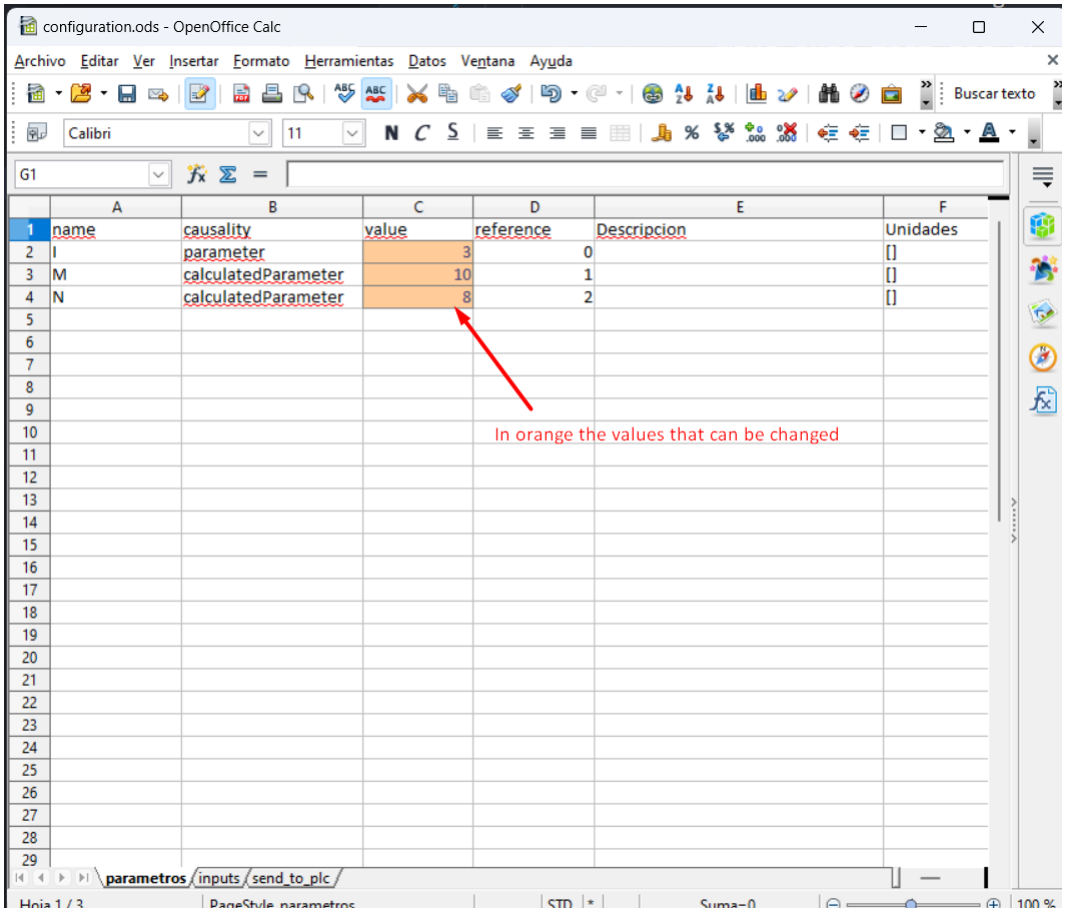
The FMU model description file (`modelDescription.xml` inside `Test_DTCONEDAR.fmu`) holds the value references, and `configuration_freeopcua.ods` holds the OPC-UA node IDs. The variable name links the two.

The client Python script (`client-opcua.py`) performs these actions:

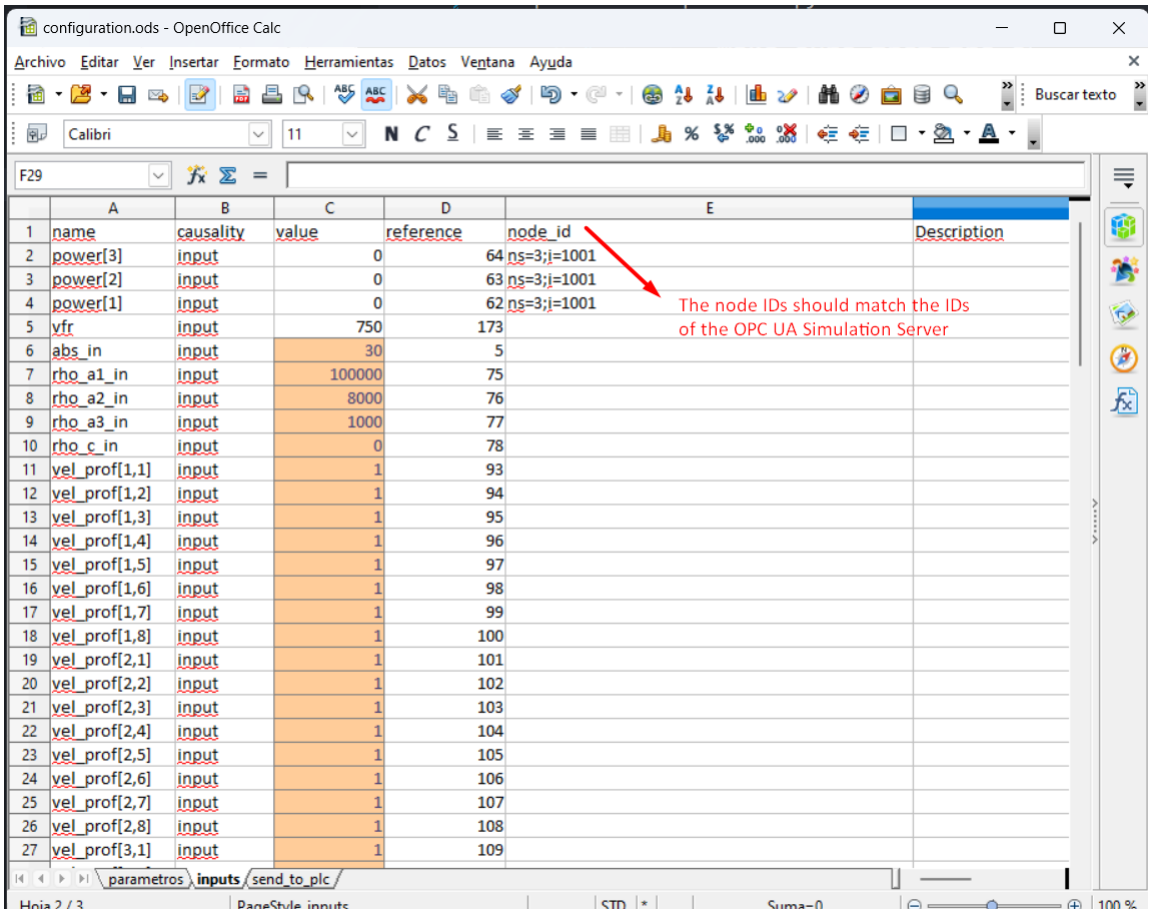
- Reads the variable names and value references from the model description file of `Test_DTCONEDAR.fmu` .
- Reads *configuration_freeopcua.ods* to obtain the OPC UA node IDs and assigns them to the variables read from the FMU.
- Reads *configuration_freeopcua.ods* to set initial values, parameters and some inputs (a reasonable value is assumed for inputs that are not measured).
- Reads values from the PLC using an OPC UA client.
- Runs the algorithm with the FMPy library, using the FMU created from the compartment model (based on CFD):
- Obtains the results.
- Sends the results to the PLC over OPC UA, so that they appear in the SCADA and help the plant operator make decisions.

Input Data Variables

The *configuration_freeopcua.ods* data file sets the initial input data values used by the server.



Copyright © 2022 - 2026 The INTO-CPS Association



	A	B	C	D	E	
1	name	causality	value	reference	node_id	Description
2	power[3]	input	0	64	ns=3;i=1001	
3	power[2]	input	0	63	ns=3;i=1001	
4	power[1]	input	0	62	ns=3;i=1001	
5	vfr	input	750	173		
6	abs_in	input	30	5		
7	rho_a1_in	input	100000	75		
8	rho_a2_in	input	8000	76		
9	rho_a3_in	input	1000	77		
10	rho_c_in	input	0	78		
11	vel_prof[1,1]	input	1	93		
12	vel_prof[1,2]	input	1	94		
13	vel_prof[1,3]	input	1	95		
14	vel_prof[1,4]	input	1	96		
15	vel_prof[1,5]	input	1	97		
16	vel_prof[1,6]	input	1	98		
17	vel_prof[1,7]	input	1	99		
18	vel_prof[1,8]	input	1	100		
19	vel_prof[2,1]	input	1	101		
20	vel_prof[2,2]	input	1	102		
21	vel_prof[2,3]	input	1	103		
22	vel_prof[2,4]	input	1	104		
23	vel_prof[2,5]	input	1	105		
24	vel_prof[2,6]	input	1	106		
25	vel_prof[2,7]	input	1	107		
26	vel_prof[2,8]	input	1	108		
27	vel_prof[3,1]	input	1	109		

DT Config

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

config.json M

```
{
    "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
}
```

These optional parameters can be modified:

- stop_time
- step_size
- record = True, to save the simulation results
- record_interval. When the simulation step_size is small, the results file can become too big. For instance, if the step_size is 0.01 seconds, increase the record_interval to reduce the file size.
- record_variables: the list of variables to record.
- enable_send = True, to send the results to the OPC UA server.

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE PHASES

The example covers these lifecycle phases:

Lifecycle Phase	Completed Tasks
Install	Installs Open Modelica and a Python virtual environment with 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 to the example directory.

```
cd /workspace/examples/digital_twins/opc-ua-waterplant
```

If required, give the lifecycle scripts execute permission.

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

Now run the following scripts.

Install

Installs OpenModelica and a Python virtual environment (`.venv`) with the required pip dependencies.

```
lifecycle/install
```

Create

Creates the `Test_DTCONEDAR.fmu` co-simulation model from the `Test_DTCONEDAR.mo` OpenModelica file.

```
lifecycle/create
```

Execute

Starts the mock OPC UA server in the background, then runs the OPC UA client.

```
lifecycle/execute
```

Clean

Removes the temporary files created by OpenModelica and the output files generated by the 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.

Copyright © 2022 - 2026 The INTO-CPS Association

Waste Water Plant with OPC UA

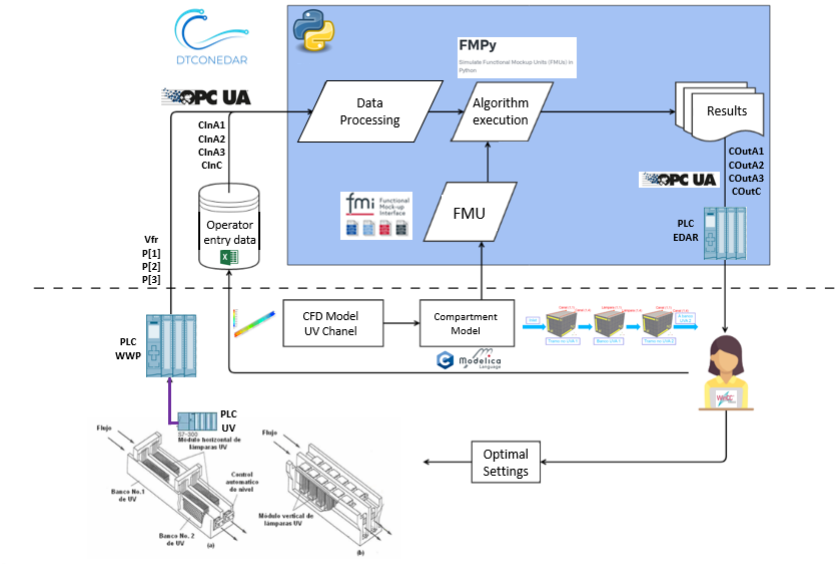
NOTE: This directory contains complete version of the `digital_twins/opc-ua-waterplant` digital twin. This directory contains code for multiple versions of mock physical twin and the example given here works on both Windows and Linux.

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.

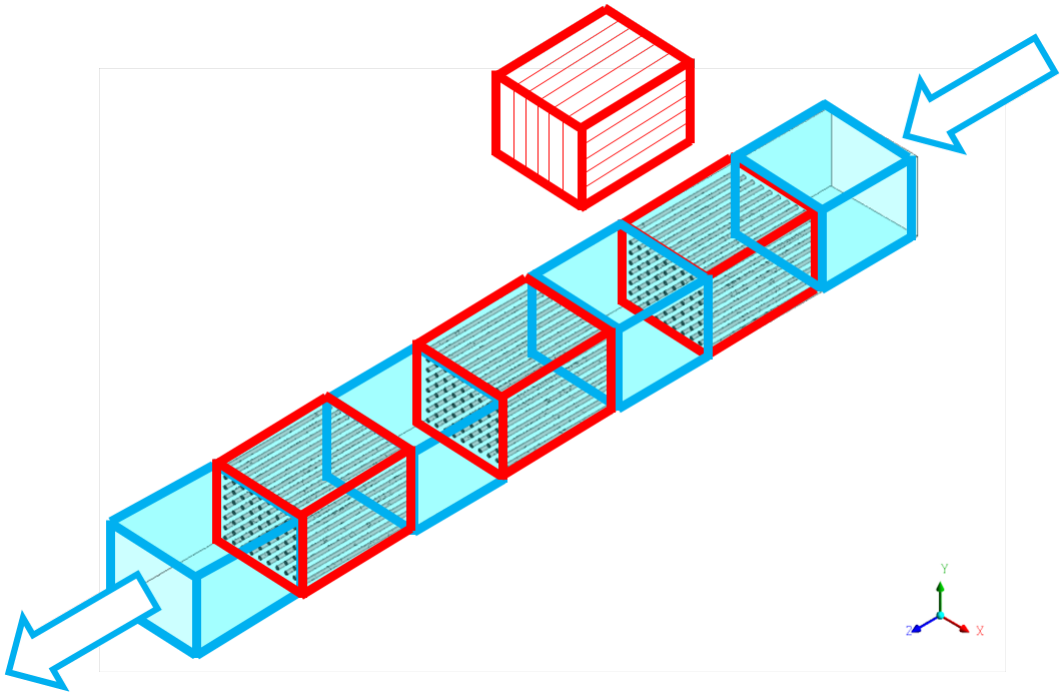


The physical twin of the waste water plant is composed of an ultraviolet channel controlled by a PLC that controls the power of the UV lamps needed to kill all the pathogens of the flow. The channel has 3 groups of UV lamps, therefore the real channel (and is mathematical model) is subdivided in 7 zones: 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.

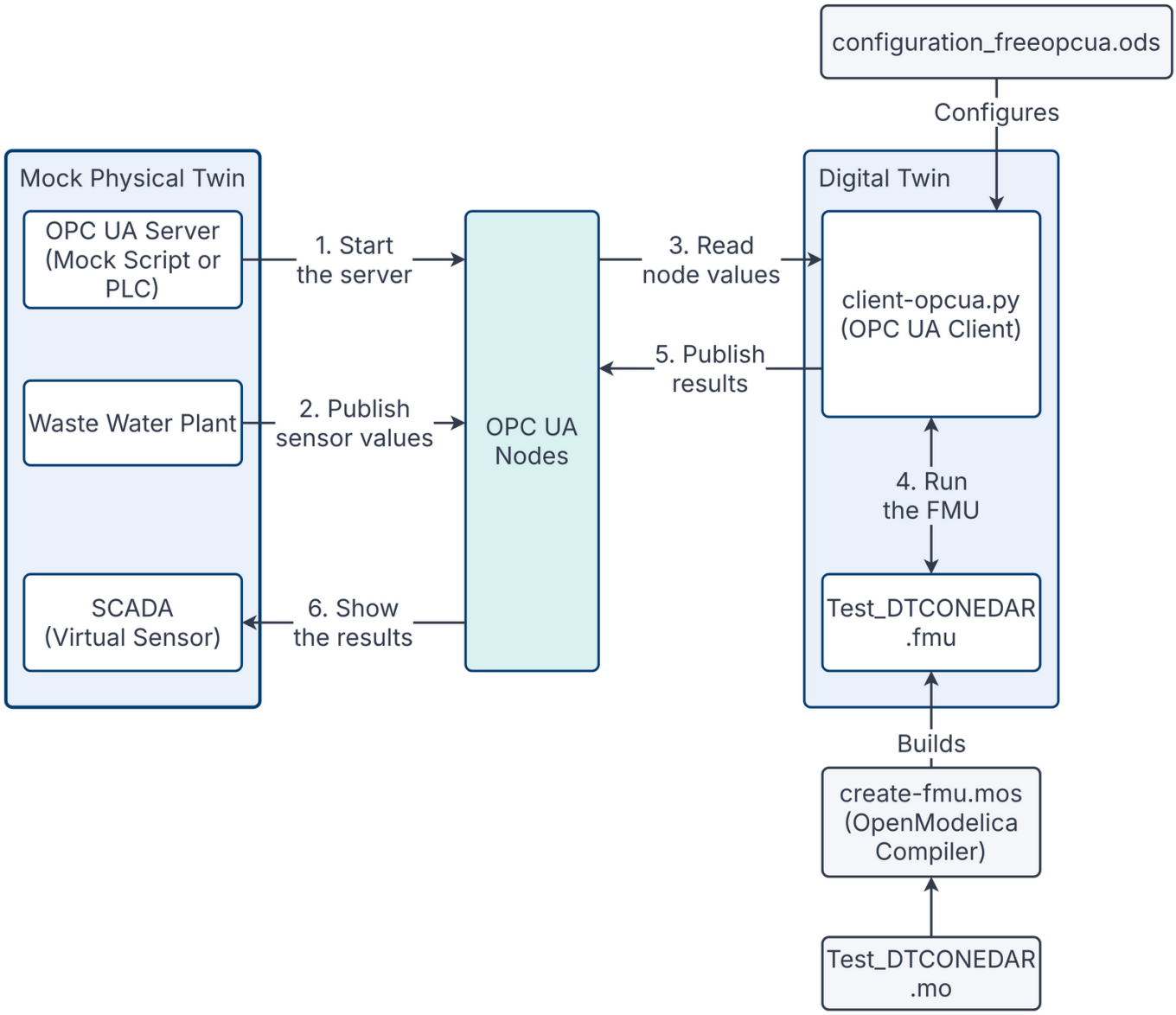
Copyright © 2022 - 2026 The INTO-CPS Association

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 FMI standard, to be integrated in the main script execution (*client-opcua.py*).

Copyright © 2022 - 2026 The INTO-CPS Association



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

- Read values from PLC using a client OPC.
- Read the spreadsheet to fix initial values, parameters and some inputs (those inputs that are not being measured, a reasonable value is assumed). And read the node IDs assigned to each input and output variable of the OPC UA server.
- 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.

Copyright © 2022 - 2026 The INTO-CPS Association

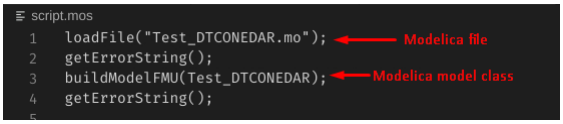
DIGITAL TWIN CONFIGURATION

Asset Type	Name of Asset	Visibility	Reuse in Other Examples
Models	Test_DTCONEDAR_linux.fmu	private	No
	Test_DTCONEDAR_windows.fmu	private	No
	Test_DTCONEDAR.mo	private	No
Data	model_description.csv (generated by client-XXX.py)	private	No
	configuration_freopcua.ods	private	No
	configuration_prosys.ods	private	No
	nodes-export.xml	private	No
Tool	**Server**: server-opcua.py, server-asyncua.py, server-opcua-xml.py, server-asyncua-xml.py	private	No
	Client: client-opcua.py, client-asyncua.py	private	No
	FMU builder: create-fmu.mos	private	No

The *configurationXXX.ods* file is used for customizing the initial values to your needs. 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 **Modelica (Test_DTCONEDAR.mo)**. The FMU is generated from the Open Modelica interface in Linux and Windows to obtain the needed binaries to run the FMU in both operating systems. 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 `create-fmu.mos` to build the FMU from de Modelica file `Test_DTCONEDAR.mo`

INSTALL AND DEMONSTRATE

1. Install GLIBC 2.33 or newer versions. The Windows has Visual Studio Redistributable which comes with GLIBC equivalent libraries.
2. **Linux**: You can install with `sudo apt-get install -y glibc-source` on Ubuntu 22.04. Please see install GLIBC-2.35 for Ubuntu 20.04.
3. **Windows**: Install Visual Studio Redistributable to install the newest version of GLIBC.
4. Install Open Modelica Compiler for Linux or Windows. The minimal installation is enough since the graphical clients (OMEdit, OMShell and OMNotebook) are not necessary. **This allows to compile an FMU from the terminal, therefore if the FMU is already built with the correct binaries, this step is optional**
5. Run the `create-fmu.mos` file to compile the FMU from the Modelica file.



Linux

```
sudo apt-get install cmake
omc create-fmu.mos
```

Windows: omc usually is located in a path like this

```
"C:\Program Files\OpenModelica1.21.0-64bit\bin\omc.exe" create-fmu.mos
```


Copyright © 2022 - 2026 The INTO-CPS Association

Two FMUs, namely `Test_DTCONEDAR_linux.fmu` and `Test_DTCONEDAR_windows.fmu` have been provided as samples. The `Test_DTCONEDAR_linux.fmu` has been generated on Ubuntu Linux 22.04 and the `Test_DTCONEDAR_windows.fmu` has been generated on Windows 10.

1. Install Python 3.8 or above. It has been tested in 3.8, 3.10 and 3.11 versions. However **version 3.10 is recommended**:
2. **Linux**: Install Python 3.10.
3. **Windows**: Install Python 3.10
4. Install pip for Python
5. **Linux**: Install pip3 for python3.10. `curl -sS https://bootstrap.pypa.io/get-pip.py | python3.10`
6. **Windows**: pip is installed by default when installing Python.
7. Upgrade pip and install pip dependencies.

Copyright © 2022 - 2026 The INTO-CPS Association

Linux

```
``` bash
sudo apt-get install python-is-python3 python3-pip
python3 -m pip install --upgrade pip
python3 -m pip install -r requirements_linux.txt
```

Windows

```cmd
py -m pip install --upgrade pip
py -m pip install -r requirements_windows.txt
```
```

Copyright © 2022 - 2026 The INTO-CPS Association

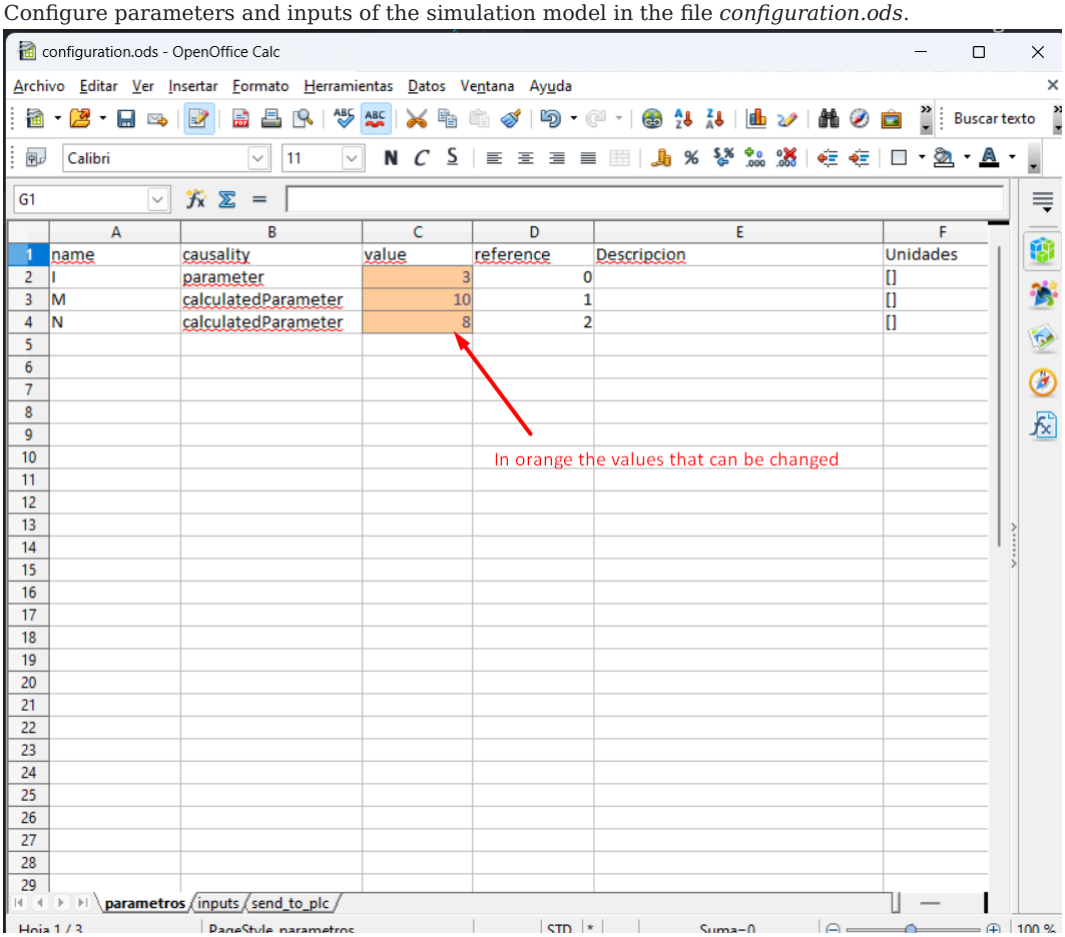
1. Configure an OPC UA simulation server. Two options are available.
2. Use Free OPC-UA Library. This is the recommended approach, because its configuration is more straightforward. Two libraries are provided within Free OPC-UA Library.

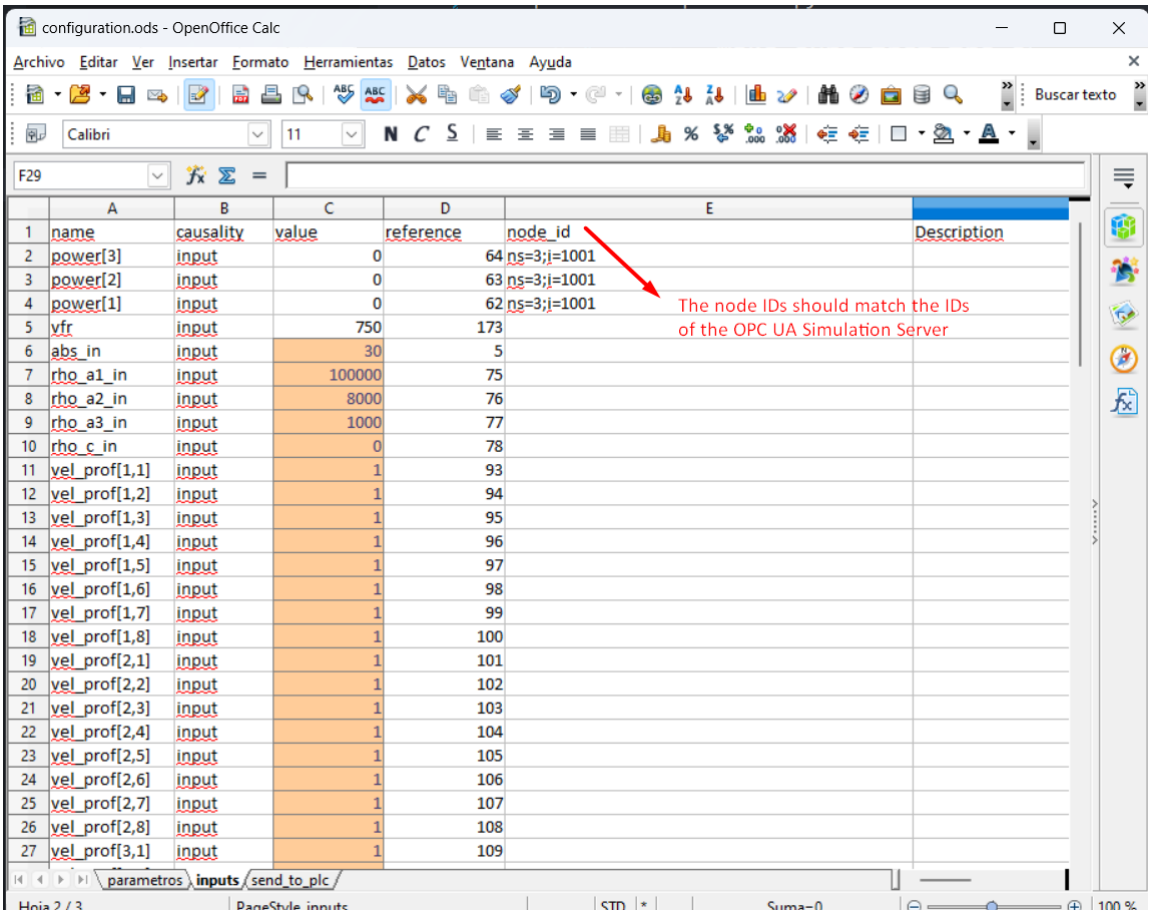
- opcua is deprecated, but still works fine to create OPC-UA servers and clients.
- asyncua is very similar, but allows asynchronous programming through the asyncio Python native library.

3. Download and install Prosys OPC UA Simulation Server. **This server is not running properly in DTaaS user workspace but works well on Ubuntu 22.04. It seems there is a bug in the OPC UA server. This needs debugging.** To configure Prosys OPC UA Simulation Server see How to configure Prosys.
4. Start the OPC-UA Server (two options available):
5. **Free OPC-UA Library**: run any of these scripts (`server-opcua.py` , `server-opcua-xml.py` , `server-asyncua.py` , `server-asyncua-xml.py`). All of them create the same server, the difference between them is that ones use opcua library and others asyncua and ones define each node by hand and others import a list of nodes from a .xml file.
6. **Prosys OPC UA Simulation Server**: just run the application.

Copyright © 2022 - 2026 The INTO-CPS Association

7. Configure parameters and inputs of the simulation model in the file *configuration.ods*.





Copyright © 2022 - 2026 The INTO-CPS Association

8. Open the configuration file *config.json* and fill in the values:



9. Replace `"url" : "opc.tcp://0.0.0.0:4840"` with new **Connection Address**. The default localhost is:
   - `opc.tcp://0.0.0.0:4840` for Linux
   - `opc.tcp://localhost:4840` for Windows
10. Select the .ods configuration file.
    - If using Free OPC-UA select `configuration_freeopcua.ods`
    - If using Prosys select `configuration_prosys.ods`
11. Make sure that your are running the FMU that suits your Operating System. The easiest way is to provide the same *fmu_filename* that has been created after running **step 3**.
12. Besides the mentioned parameters, 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.
13. Run any of the following client scripts: `client-opcua.py` or `client-asyncua.py`

FUTURE STEPS

1. FreeOPCUA server in DTaaS user worksapce is working properly, but Prosys server does not run in DTaaS user workspace. **There is an error when running the FMU: 'version GLIBC 2.33 not found'**.
2. Develop an FMU for OPC UA communication and test the cosimulation in DTaaS of the two FMUs:

a. The FMU that models the physic behaviour of the plant and

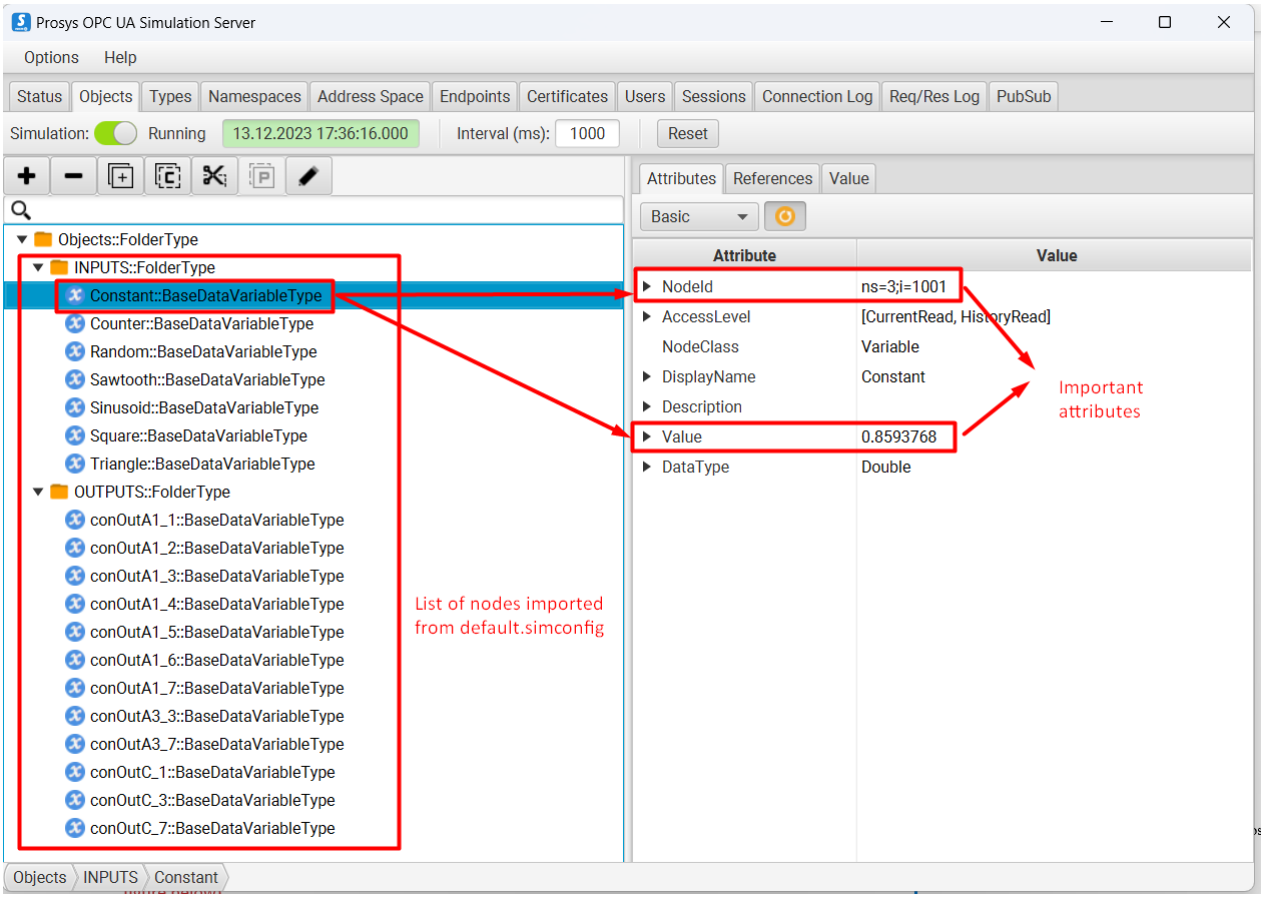
b. The FMU responsible for the OPC UA communication.

HOW TO CONFIGURE PROSYS

1. Start OPC UA server once and then close. This action will create `~/.prosysopc/prosys-opc-ua-simulation-server` directory (within your operating system).

Copyright © 2022 - 2026 The INTO-CPS Association

2. Go to `prosys` folder (within this repository) and copy `default.simconf` file into `~/.prosysopc/prosys-opc-ua-simulation-server` directory. After that, when running again Prosys OPC UA the following nodes should appear.



3. Start the OPC UA server again. Make sure that the server is running at a port. Copy **Connection Address (UA TCP)** from the **Status** tab.

REFERENCES

1. 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

ACKNOWLEDGEMENTS

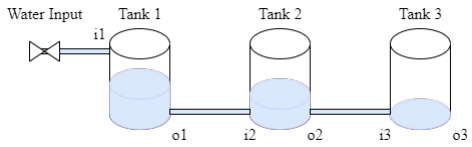
This project is subsidised thanks to the support grants for Agrupación Empresarial Innovadora (AEI) of the Ministry of Industry, Trade and Tourism (MINCOTUR) with the aim of improving the competitiveness of small and medium-sized enterprises within the Recovery, Transformation and Resilience Plan (PRTR)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

Copyright © 2022 - 2026 The INTO-CPS Association

Three-Tank System Digital Twin

OVERVIEW

The three-tank system is a simple case study of three components coupled in a cascade: 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 simulated components, so there is no configuration for a connection to the physical system.

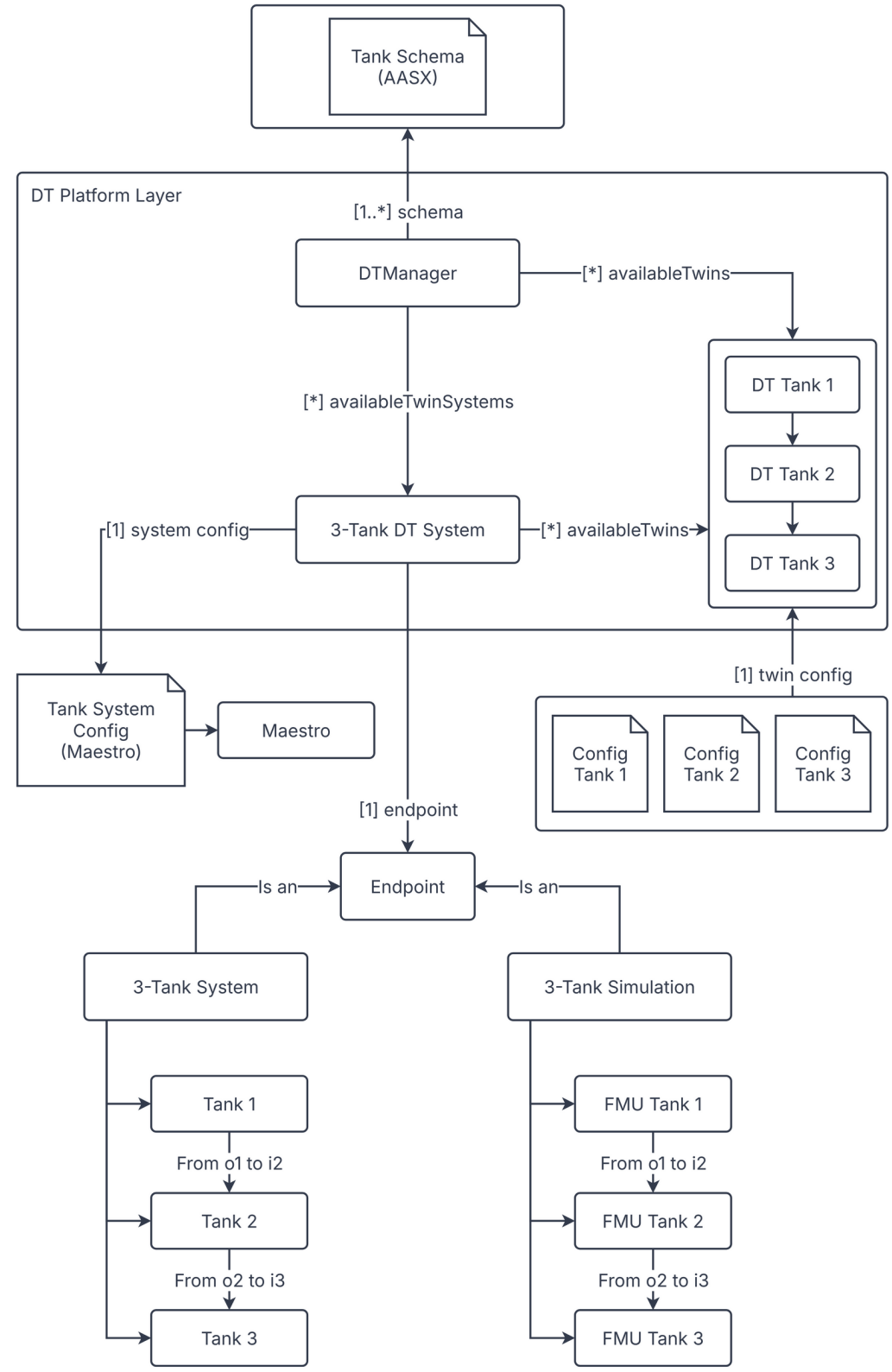
The `TwinManager` manages the three-tank system. It is packaged as a jar library in the tools and run from a Java main file. The `TwinManager` uses Maestro as a slave for co-simulation, so it generates the output of the co-simulation.

The main file can be changed to suit the application, that is, `/workspace/examples/tools/three-tank/TankMainDTaaS.java` can be modified to get a different result.

The `/workspace/examples/models/three-tank/` folder contains `Linear.fmu`, a non-realistic model of a tank with an input and an output, and `TankSystem.aasx`, the schema representation as an Asset Administration Shell. The three tank instances use the same `.fmu` file and the same schema because they belong to the same object class. The `TwinManager` reads the values from the co-simulation output.

Copyright © 2022 - 2026 The INTO-CPS Association

EXAMPLE STRUCTURE



Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses two models, three tools and one data file. 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	TwinManagerFramework-0.0.2.jar (wraps Maestro)	Common	Yes
	maestro-4.1.0-jar-with-dependencies.jar (used by TwinManager)	Common	Yes
	TankMainDTaaS.java (main script)	Private	No
Data	outputs.csv	Private	No

The DT has several configuration files. The Maestro tool uses *coe.json* and *multimodel.json*. The files *tank1.conf*, *tank2.conf* and *tank3.conf* configure three instances of one model (Linear.fmu).

LIFECYCLE PHASES

The example covers these lifecycle phases:

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit 21 and Maven, and builds the TwinManager
Execute	The TwinManager executes the three-tank digital twin and produces output in `data/three-tank/output` directory
Terminate	Stops the background processes
Clean	Removes the output

RUN THE EXAMPLE

To run the example, change to the example directory.

```
cd /workspace/examples/digital_twins/three-tank
```

If required, give the lifecycle scripts execute permission, for example:

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

Maestro 4.1.0 needs Java 21 or newer. The **create** script installs Java 21. If other Java versions are already installed, the default *java* may point to another version. Check and change the default version with these commands.

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

Then run the following scripts.

Create

Installs Open Java Development Kit 21 and Maven, then builds the `TwinManager` tool (TwinManagerFramework-0.0.2.jar) from source code.

```
lifecycle/create
```


Copyright © 2022 - 2026 The INTO-CPS Association

Execute

Executes the three-tank digital twin using the TwinManager, which in turn runs the co-simulation using Maestro. The co-simulation output file `outputs.csv` is written to `/workspace/examples/data/three-tank/output`.

```
lifecycle/execute
```

Terminate

Stops Maestro running in the background, along with any other Java process started during the **execute** phase.

```
lifecycle/terminate
```

Clean

Removes the output generated during the execute phase.

```
lifecycle/clean
```

EXAMINING THE RESULTS

Executing this digital twin generates a co-simulation output. The results can also be monitored by adding a set of `getAttributeValue` calls to `/workspace/examples/tools/three-tank/TankMainDTaaS.java`, as shown in the code.

The main file runs the digital twin and its internal components online.

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

In the default example, the co-simulation runs for 10 seconds in steps of 0.5 seconds. Both the duration and the step size can be changed. The output stored in `outputs.csv` contains the level, in/out flow and leak values.

The example neither generates nor uses data from the physical twin.

REFERENCES

More information about the TwinManager is available at:

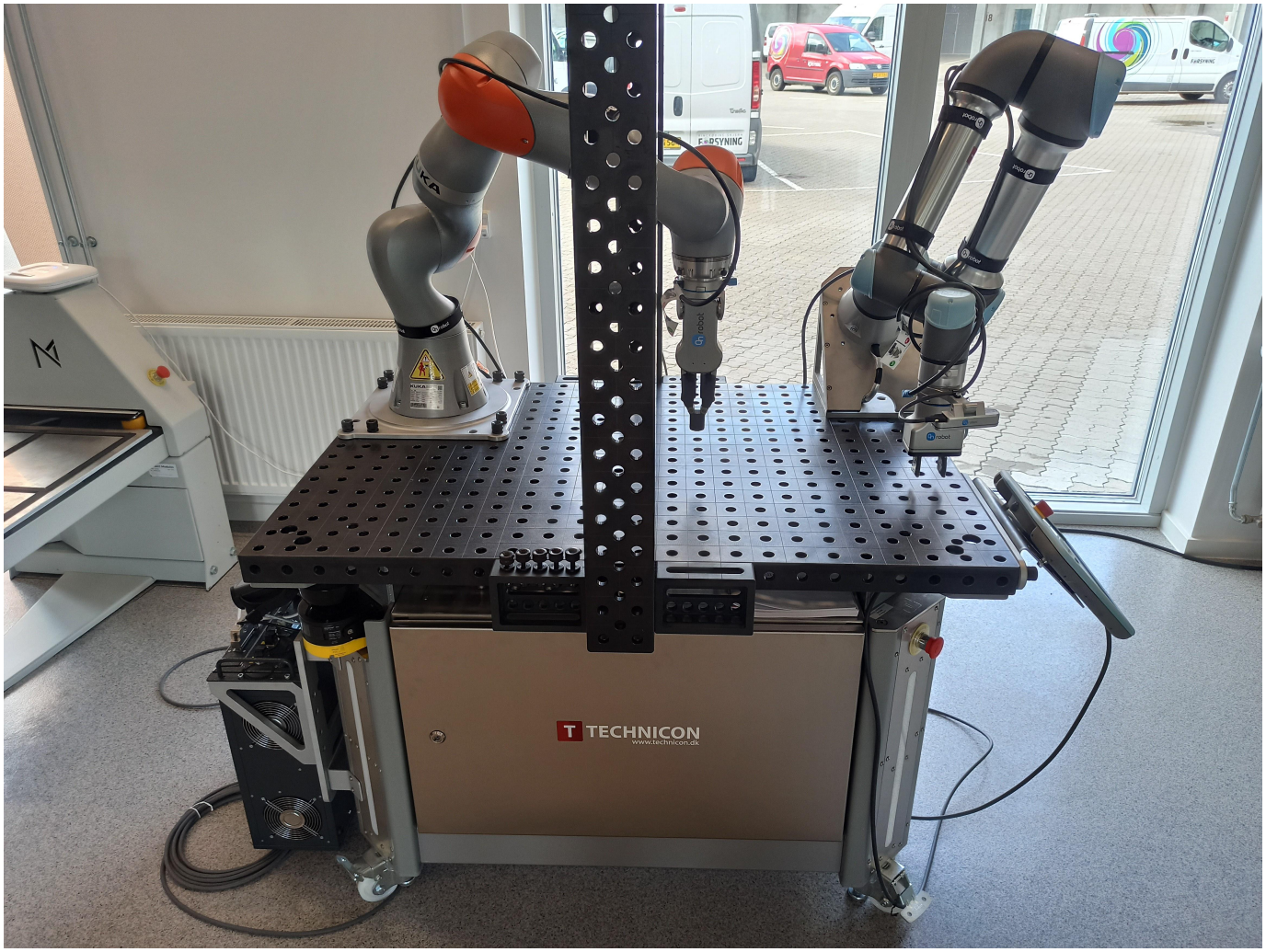

```
D. Lehner, S. Gil, P. H. Mikkelsen, P. G. Larsen and M. Wimmer,
"An Architectural Extension for Digital Twin Platforms to Leverage
Behavioral Models," 2023 IEEE 19th International Conference on
Automation Science and Engineering (CASE), Auckland, New Zealand,
2023, pp. 1-8, doi: 10.1109/CASE56687.2023.10260417.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


Copyright © 2022 - 2026 The INTO-CPS Association

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 together in a manufacturing cell.



The case study focuses on positioning the robots in the discrete Cartesian space of the flex-cell. Both robots accept (X,Y,Z) commands that give the target hole and height to move to.

The `TwinManager` (formerly `DT Manager`) manages the flex-cell case study. It is packaged 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 co-simulation output and, with the proper configuration and setup, can interact with the real robots at the same time. The main file can be changed to suit the application, that is, `/workspace/examples/tools/flex-cell/FlexCellDTaaS.java` can be modified 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 visualisation (including the grippers), and the `.aasx` files for the schema representation as an Asset Administration Shell.

The case study also uses the RabbitMQ FMU to inject values into the co-simulation, so the models folder contains that FMU as well. For now, the RabbitMQ FMU only injects values into the co-simulation and does not send values out of it. The `TwinManager` reads the values from the co-simulation output and the current state of the physical twins.

EXAMPLE STRUCTURE

The example structure shows the components of the flex-cell DT implementation using the `TwinManager` architecture.

Copyright © 2022 - 2026 The INTO-CPS Association

The TwinManager orchestrates the flex-cell DT through the *Flex-cell DT System*, which is composed of two smaller digital twins, the *DT UR5e* and the *DT Kuka lbr iiwa 7*. The TwinManager also provides the interface to the physical twins, *PT UR5e* and *PT Kuka lbr iiwa 7*. Each physical twin and digital twin system has its own endpoint (with a different specialization), which is initialized from configuration files and a data model (twin schema).

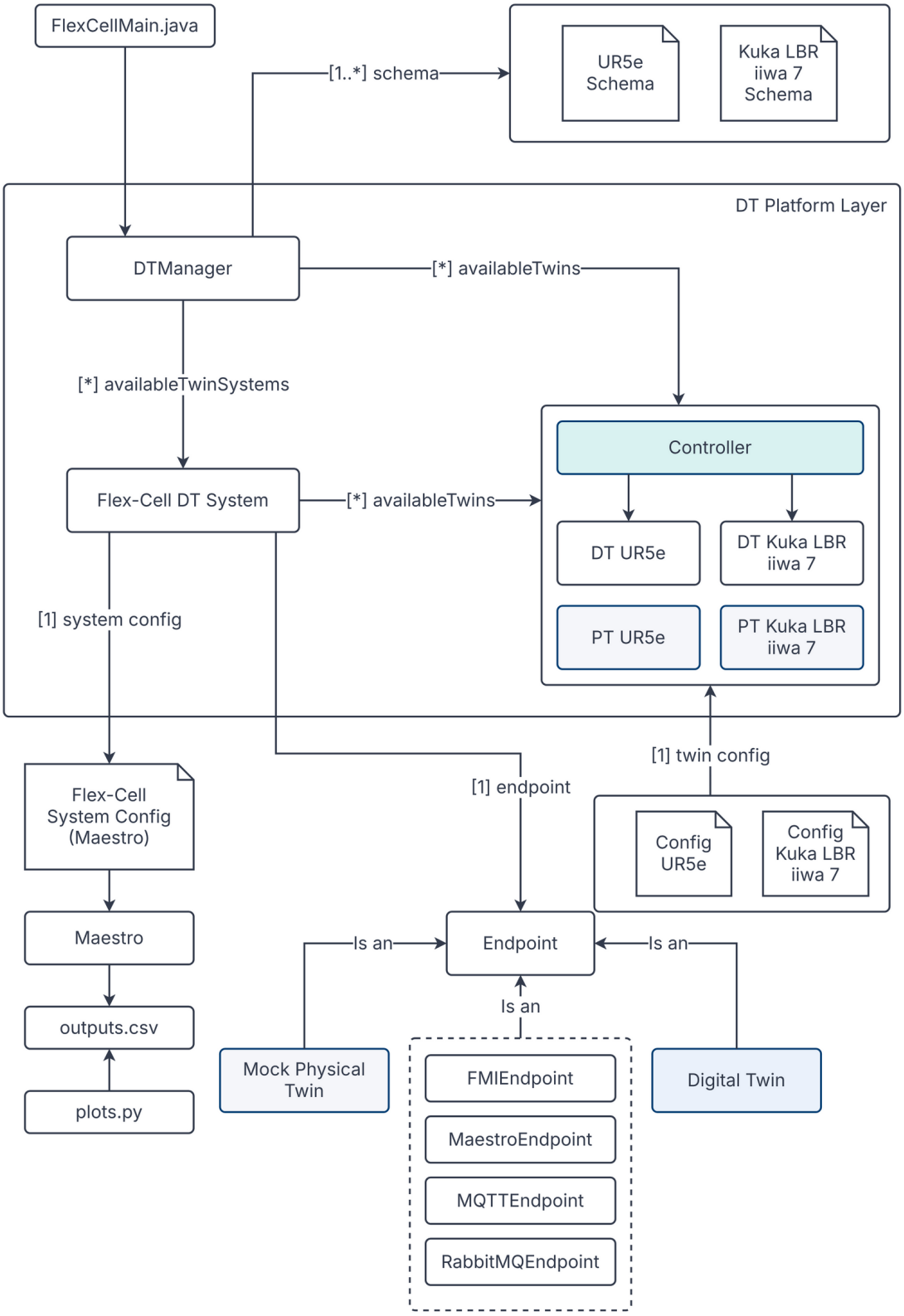
The current endpoints used in this implementation are:

Digital or Physical Twin	Endpoint
Flex-cell DT System	MaestroEndpoint
DT UR5e	FMIEndpoint
DT Kuka lbr iiwa 7	FMIEndpoint
PT UR5e	MQTTEndpoint and RabbitMQEndpoint
PT Kuka lbr iiwa 7	MQTTEndpoint and RabbitMQEndpoint

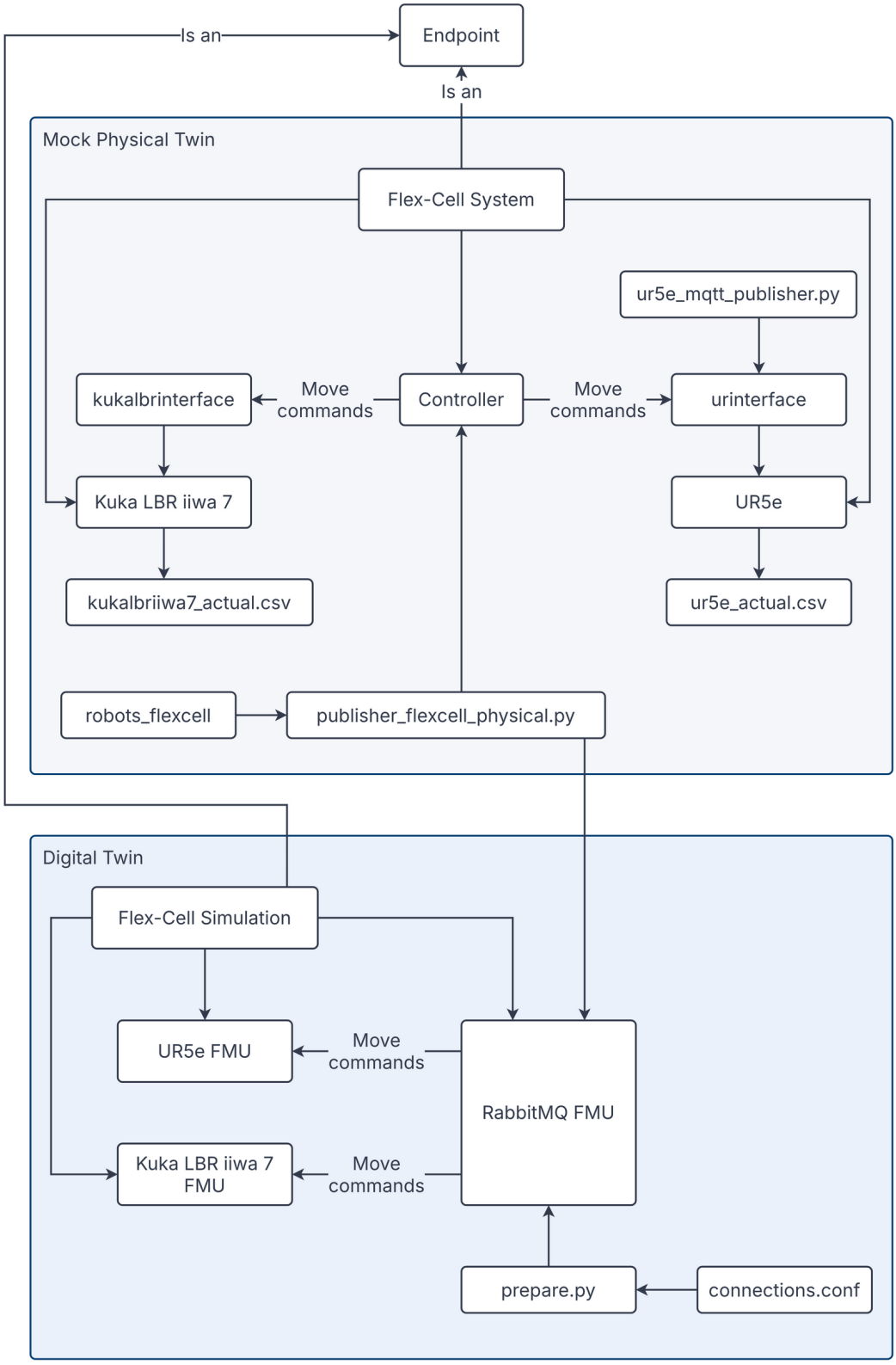
The Flex-cell DT System has a separate configuration for its integration with the Maestro co-simulation engine.

In the lower part of the diagram, the Flex-cell System is the composed physical twin, including the two robotic arms and the controller. The Flex-cell Simulation is the mock-up of the real system, implemented by FMU blocks and their connections.

Copyright © 2022 - 2026 The INTO-CPS Association



Copyright © 2022 - 2026 The INTO-CPS Association



Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses seven models, five tools, six data files, 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
	rmqfmu-vhost.fmu	Private	Yes
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes
	TwinManagerFramework-0.0.2.jar	Private	Yes
	urinterface (installed with pip)	Private	No
	kukalbrinterface	Private	No
	robots_flexcell	Private	No
	FlexCellDTaaS.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

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE PHASES

The example covers these lifecycle phases:

1. Installing dependencies in the create phase.
2. Preparing the connection credentials in the prepare phase.
3. Running the experiment in the execute phase.
4. Saving experiments in the save phase.
5. Plotting the co-simulation results and the real data from the robots in the analyze phase.
6. Stopping the background processes in the terminate phase and cleaning up the outputs in the clean phase.

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit 21 and Maven, compiles the source code of TwinManager 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 TwinManager 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 to the example directory.

```
cd /workspace/examples/digital_twins/flex-cell
```

If required, give the lifecycle scripts execute permission, for example:

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

This example requires Java 21 because Maestro 4.1.0 needs it. The **create** script installs Java 21. If other Java versions are already installed, the default *java* may point to another version. Check and change the default version with these commands.

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

Now run the following scripts.

Create

Installs Open Java Development Kit 21, Maven, and a Python virtual environment with pip dependencies. It also builds the `TwinManager` tool (TwinManagerFramework-0.0.2.jar) from source code.

```
lifecycle/create
```

Prepare

This step configures the DT assets with connection credentials, using the `functions/flex-cell/prepare.py` script. To set up the connection, update `/workspace/examples/data/flex-cell/input/connections.conf` with the connection parameters for MQTT and RabbitMQ, then run the `prepare` script.

```
lifecycle/prepare
```


Copyright © 2022 - 2026 The INTO-CPS Association

The script writes the configuration into these files, which need credentials to connect to the RabbitMQ and MQTT brokers:

1. `/workspace/examples/digital_twins/flex-cell/kuka_actual.conf`
2. `/workspace/examples/digital_twins/flex-cell/ur5e_actual.conf`
3. `/workspace/examples/data/flex-cell/input/publisher-flexcell-physical.py`
4. `modelDescription.xml` of the RabbitMQ FMU

Execute

Executes the flex-cell digital twin using the TwinManager, which in turn runs the co-simulation using Maestro. The co-simulation output file `outputs.csv` is written to `/workspace/examples/data/flex-cell/output`. Stop the execution with `control + c`, because the `TwinManager` runs the application in an endless loop.

```
lifecycle/execute
```

Save

Each execution of the DT is treated as a single run. The save script stores the results of one execution as a time-stamped co-simulation output file in the `data/flex-cell/output/saved_experiments` directory.

```
lifecycle/save
```

The `execute` and `save` scripts can be run in that order any number of times. Each iteration adds a new file to the `data/flex-cell/output/saved_experiments` directory.

Analyze

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

```
lifecycle/analyze
```

Terminate

Stops Maestro running in the background, along with any other Java process started during the **execute** phase.

```
lifecycle/terminate
```

Clean

Removes the output generated during the execute phase.

```
lifecycle/clean
```

EXAMINING THE RESULTS

Executing this digital twin generates a co-simulation output. The results can also be monitored by adding a set of `getAttributeValue` calls to `/workspace/examples/tools/flex-cell/FlexCellDTaaS.java`, as shown in the code. The main file runs the digital twin and the physical twin online and compares them at the same time and at the same abstraction level.

The output is written to the `/workspace/examples/data/flex-cell/output` folder. To keep a specific experiment, the **save** lifecycle script stores the co-simulation results in the `/workspace/examples/data/flex-cell/output/saved_experiments` folder.

In the default example, the co-simulation runs for 10.5 seconds in steps of 0.2 seconds. Both the duration and the step size can be changed. The output stored in `outputs.csv` contains the joint positions of both robotic arms and the current discrete (X,Y,Z) position of the robot TCP. More 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 the documentation and source code of the RabbitMQ FMU.

Copyright © 2022 - 2026 The INTO-CPS Association

More information about the TwinManager (formerly DT Manager) and the case study is available in:

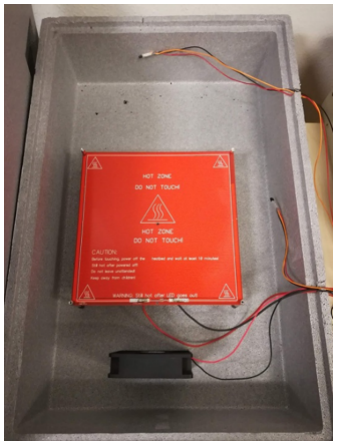
1. D. Lehner, S. Gil, P. H. Mikkelsen, P. G. Larsen and M. Wimmer, "An Architectural Extension for Digital Twin Platforms to Leverage Behavioral Models," 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), Auckland, New Zealand, 2023, pp. 1-8, doi: 10.1109/CASE56687.2023.10260417.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

Copyright © 2022 - 2026 The INTO-CPS Association

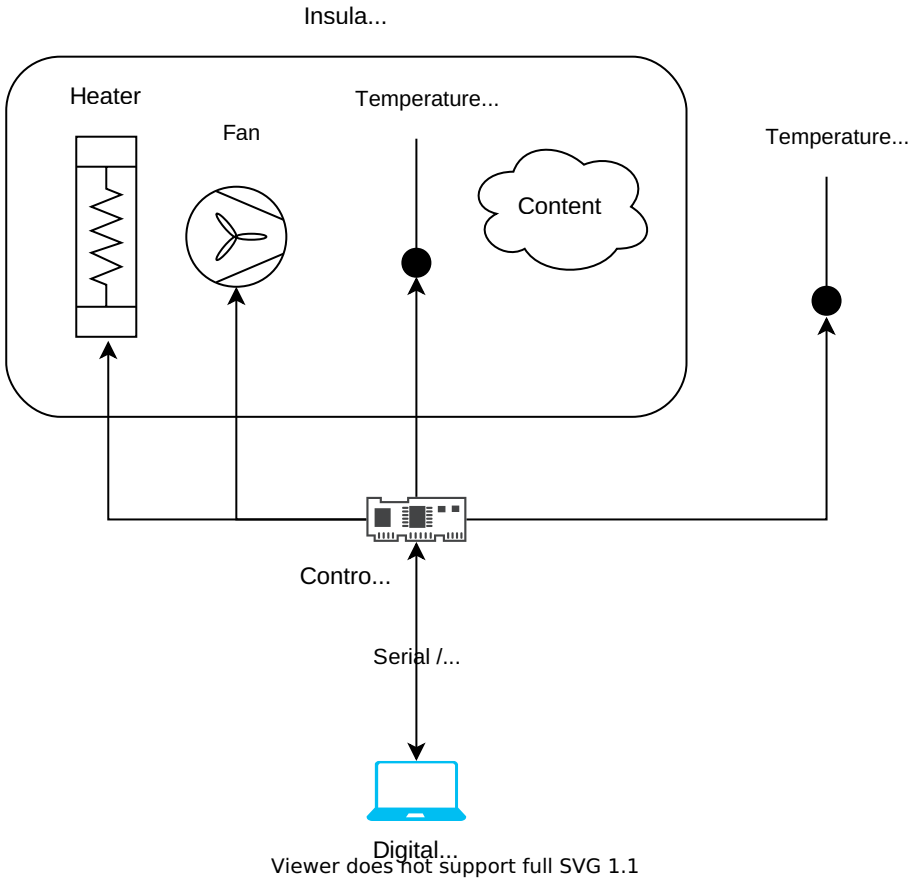
Incubator Digital Twin

OVERVIEW

This case study of an incubator shows the steps involved in developing a digital twin system. The incubator is an insulated container that can maintain a temperature by heating, but it cannot cool. A picture of the incubator is shown below.



The system reaches a set temperature inside a box and holds that temperature regardless of the content. An overview of the system is shown below:



Copyright © 2022 - 2026 The INTO-CPS Association

The system consists of:

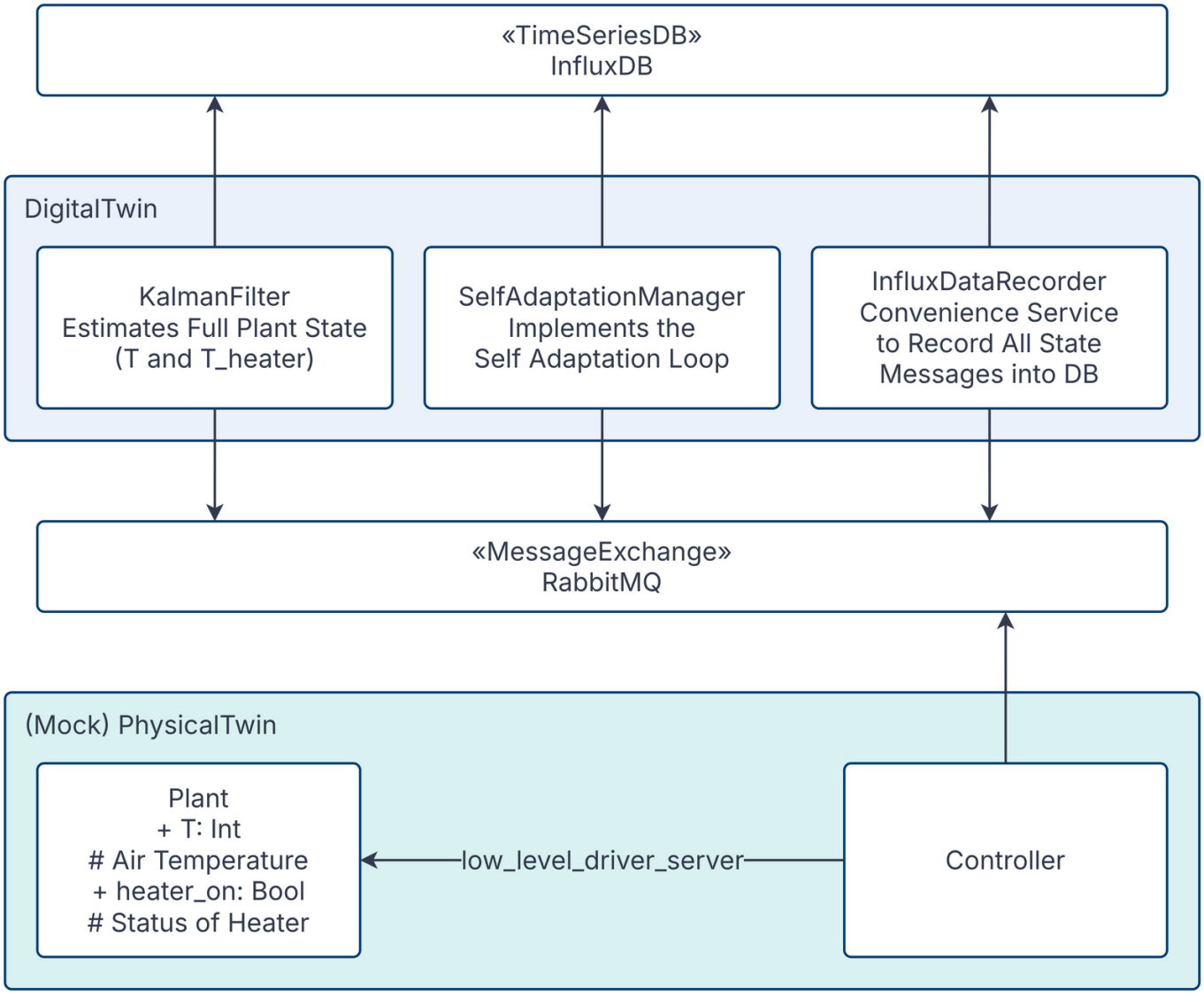
- 1x Styrofoam box as the insulated container
- 1x heat source to heat the content of the Styrofoam box
- 1x fan to distribute the heat within the box
- 2x temperature sensors to monitor the temperature inside the box
- 1x temperature sensor to monitor the temperature outside the box
- 1x controller that actuates the heat source and the fan, reads the temperature sensors, and communicates with the digital twin

The example comes from the original repository. This trimmed version of the codebase leaves out:

- docker support
- tests
- datasets

The original repository has the complete documentation of the example, including the full system architecture and instructions for running with a physical twin or a 3D visualisation of the incubator.

DIGITAL TWIN STRUCTURE



Copyright © 2022 - 2026 The INTO-CPS Association

This diagram shows the main components and the interfaces they use to communicate. All components communicate through the RabbitMQ message exchange, and the data is stored in a time series database. RabbitMQ and InfluxDB are platform services hosted by the DTaaS.

The Incubator digital twin is pre-packaged. It can be used as is or integrated with other digital twins.

The mock physical twin runs from the `incubator/mock_plant/real_time_model_solver.py` script.

DIGITAL TWIN CONFIGURATION

This example uses many Python scripts to run the digital twin. By default, it runs with a mock physical twin. It also needs RabbitMQ and InfluxDB instances.

There is one configuration file: `simulation.conf`. The RabbitMQ and InfluxDB configuration parameters must be updated.

LIFECYCLE PHASES

The example covers these lifecycle phases:

Lifecycle Phase	Completed Tasks
Create	Installs missing system packages and the Python dependencies in a virtual environment
Execute	Executes the Incubator digital twin and produces output in the terminal and in *incubator/log.log*.
Clean	Removes the log file.

RUN THE EXAMPLE

To run the example, change to its directory.

```
cd /workspace/examples/digital_twins/incubator
```

If required, make the lifecycle scripts executable, for example:

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

Now, run the following scripts:

Create

Installs any missing system packages (Python 3 and venv) and the Python dependencies in a virtual environment in *common/digital_twins/incubator/.venv*.

```
lifecycle/create
```

Execute

Executes the Incubator digital twin with a mock physical twin. Writes the results to the terminal, *incubator/log.log* and InfluxDB.

```
lifecycle/execute
```

Clean

Removes the output log file.

```
lifecycle/clean
```

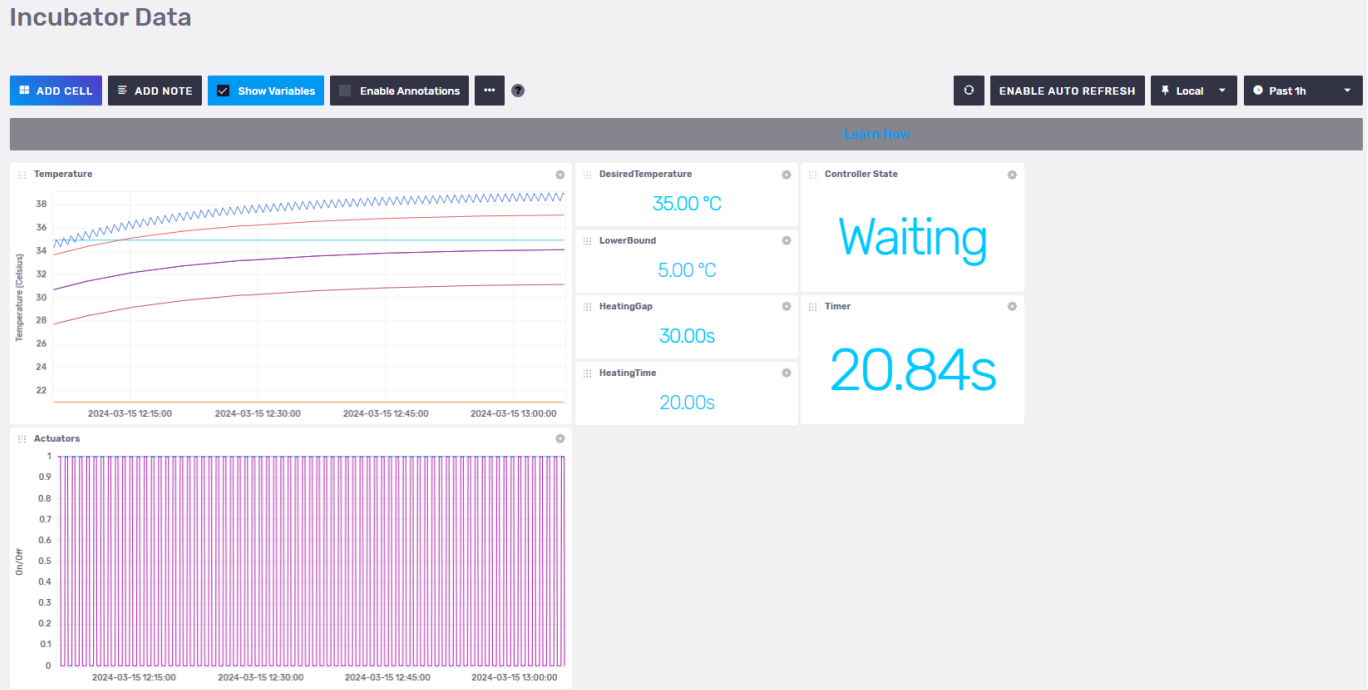

Copyright © 2022 - 2026 The INTO-CPS Association

EXAMINING THE RESULTS

Once all services have started, the controller service produces output like the following:

```
time           execution_interval  elapsed  heater_on  fan_on   room   box_air_temperature  state
19/11 16:17:59  3.00                0.01     True       False    10.70  19.68                Heating
19/11 16:18:02  3.00                0.03     True       True     10.70  19.57                Heating
19/11 16:18:05  3.00                0.01     True       True     10.70  19.57                Heating
19/11 16:18:08  3.00                0.01     True       True     10.69  19.47                Heating
19/11 16:18:11  3.00                0.01     True       True     10.69  19.41                Heating
```

An InfluxDB dashboard can be set up from `incubator/digital_twin/data_access/influxdbserver/dashboards/incubator_data.json`. Once the dashboard is set up in InfluxDB, it shows the following visualisation:



REFERENCES

Forked from: Incubator repository with commit ID: 989ccf5909a684ad26a9c3ec16be2390667643aa

The following resources explain what a digital twin is:

1. Feng, Hao, Cláudio Gomes, Casper Thule, Kenneth Lausdahl, Alexandros Iosifidis, and Peter Gorm Larsen. “Introduction to Digital Twin Engineering.” In 2021 Annual Modeling and Simulation Conference (ANNSIM), 1–12. Fairfax, VA, USA: IEEE, 2021. https://doi.org/10.23919/ANNSIM52504.2021.9552135.
2. Video recording of presentation by Claudio Gomes

Copyright © 2022 - 2026 The INTO-CPS Association

Firefighter Mission in a Burning Building

During a firefighter mission, the oxygen level of each firefighter's Self Contained Breathing Apparatus (SCBA) must be monitored in the context of the mission.

PHYSICAL TWIN OVERVIEW



Image: Schematic overview of a firefighter mission. Note the mission commander on the lower left documenting the air supply pressure levels provided by radio communication from the firefighters inside and around the burning building. This image was created with the assistance of DALL·E.

The following scenario is assumed:

- a set of firefighters work to extinguish a burning building
- they each use an SCBA with pressurised oxygen to breathe
- a mission commander outside coordinates the efforts and monitors the oxygen levels

DIGITAL TWIN OVERVIEW

This example implements a monitor that calculates how much time the firefighters have left before they need to leave the building. It uses these inputs:

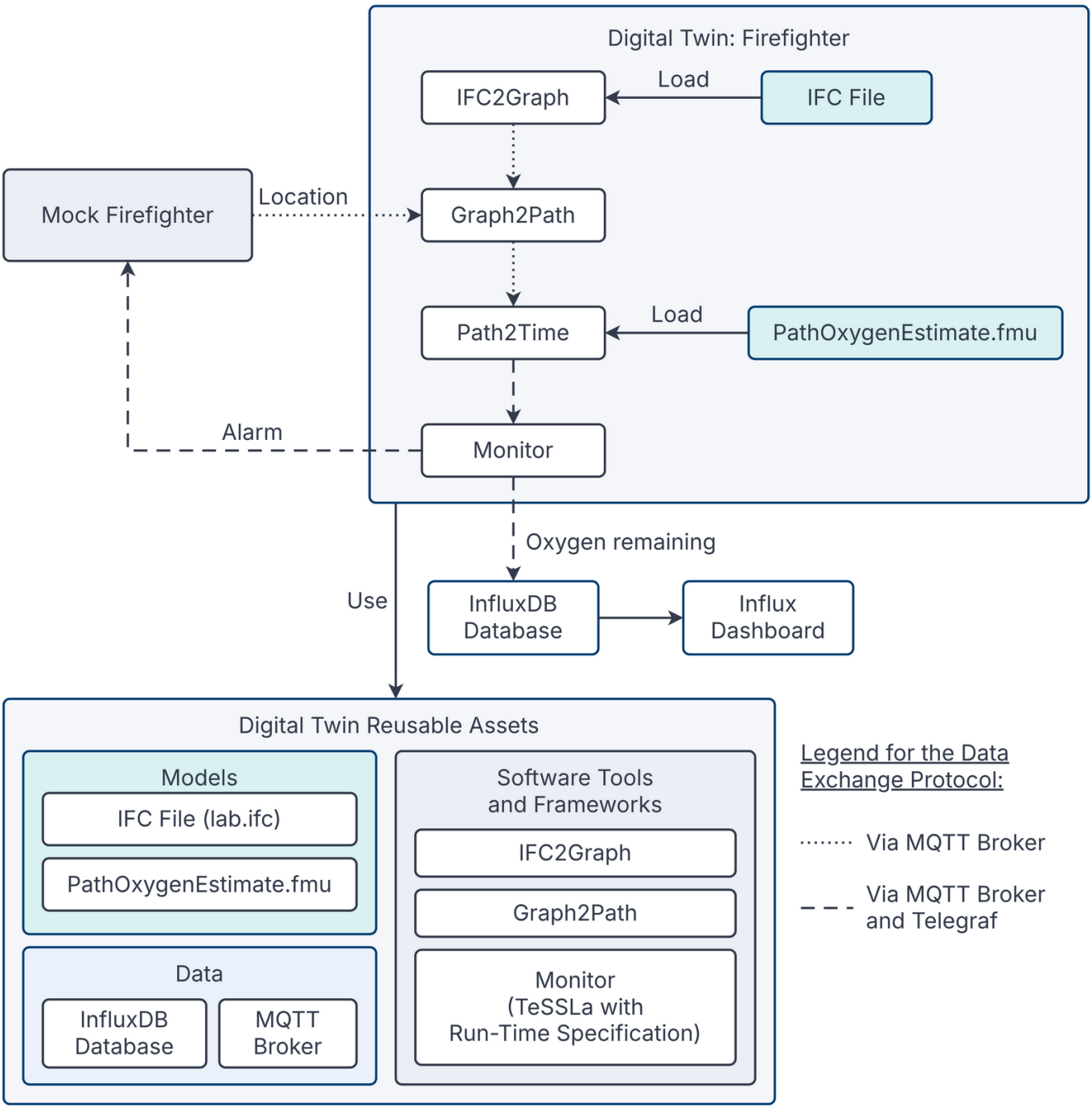
- a 3D model of the building in which the mission takes place,
- pressure data from a firefighter's SCBA, and
- the firefighter's location inside the building

to estimate:

- the shortest way out,
- how long it takes, and
- how much time is left until the SCBA runs out of oxygen.

The remaining mission time is monitored, and the firefighters receive a warning if it drops below a threshold.

Copyright © 2022 - 2026 The INTO-CPS Association



This example implements the paper *Digital Twin for Rescue Missions--a Case Study* by Leucker et al.

Copyright © 2022 - 2026 The INTO-CPS Association

Quick Check

Before running this example, check that the following files are in the correct locations:

```
/workspace/examples/
   data/o5g/input/
      runTessla.sh
      sensorSimulation.py
      telegraf.conf

   models/
      lab.ifc
      makefmu.mos
      PathOxygenEstimate.mo

   tools/
      graph2Path.py
      ifc2graph
      path2Time.py
      tessla-telegraf-connector/
         tessla.jar
         specification.tessla (run-time specification)

   digital_twins/o5g/
      main.py
      config
      lifecycle/ (scripts)
```

Digital Twin Configuration

All configuration for this example is in `digital_twins/o5g/config`.

The MQTT server needs account information. The topics have default values that let the DT read the sensor metrics of the mock physical twin and send back alerts.

```
export O5G_MQTT_SERVER=
export O5G_MQTT_PORT=
export O5G_MQTT_USER=
export O5G_MQTT_PASS=

export O5G_MQTT_TOPIC_SENSOR='vgiot/ue/metric'
export O5G_MQTT_TOPIC_AIR_PREDICTION='vgiot/dt/prediction'
export O5g_MQTT_TOPIC_ALERT='vgiot/dt/alerts'
```

This example stores data in InfluxDB, which needs the matching access details. Configure it as follows:

- Log into the InfluxDB Web UI.
- Note the **org** name (below the *username* in the sidebar).
- Create a data bucket in `Load Data -> Buckets` if one does not already exist.
- Create an API access token in `Load Data -> API Tokens`. Copy and save this token immediately, because it cannot be viewed again later.

```
export O5G_INFLUX_SERVER=
export O5G_INFLUX_PORT=
export O5G_INFLUX_TOKEN=
export O5G_INFLUX_ORG=
export O5G_INFLUX_BUCKET=
```

LIFECYCLE PHASES

The example covers these lifecycle phases:

Lifecycle Phase	Completed Tasks
Install	Installs Open Modelica, Rust, Telegraf and the required pip dependencies
Create	Create FMU from Open Modelica file
Execute	Execute the example in the background tmux terminal session
Terminate	Terminate the tmux terminal session running in the background
Clean	Delete the temporary files

Copyright © 2022 - 2026 The INTO-CPS Association

RUN THE EXAMPLE

The *install* script creates the Python virtual environment *o5g-venv* in the directory of the example, and the *execute* script activates it from there. The lifecycle scripts can be run from any directory, for example from the directory of the example.

```
cd /workspace/examples/digital_twins/o5g
```

Install

Run the install script by executing

```
lifecycle/install
```

This installs the required apt packages, including Rust, as well as Open Modelica and Telegraf from their own repositories and the required pip packages.

Create

Run the create script by executing

```
lifecycle/create
```

This compiles the Modelica model to a Functional Mockup Unit (FMU) for the correct platform.

Execute

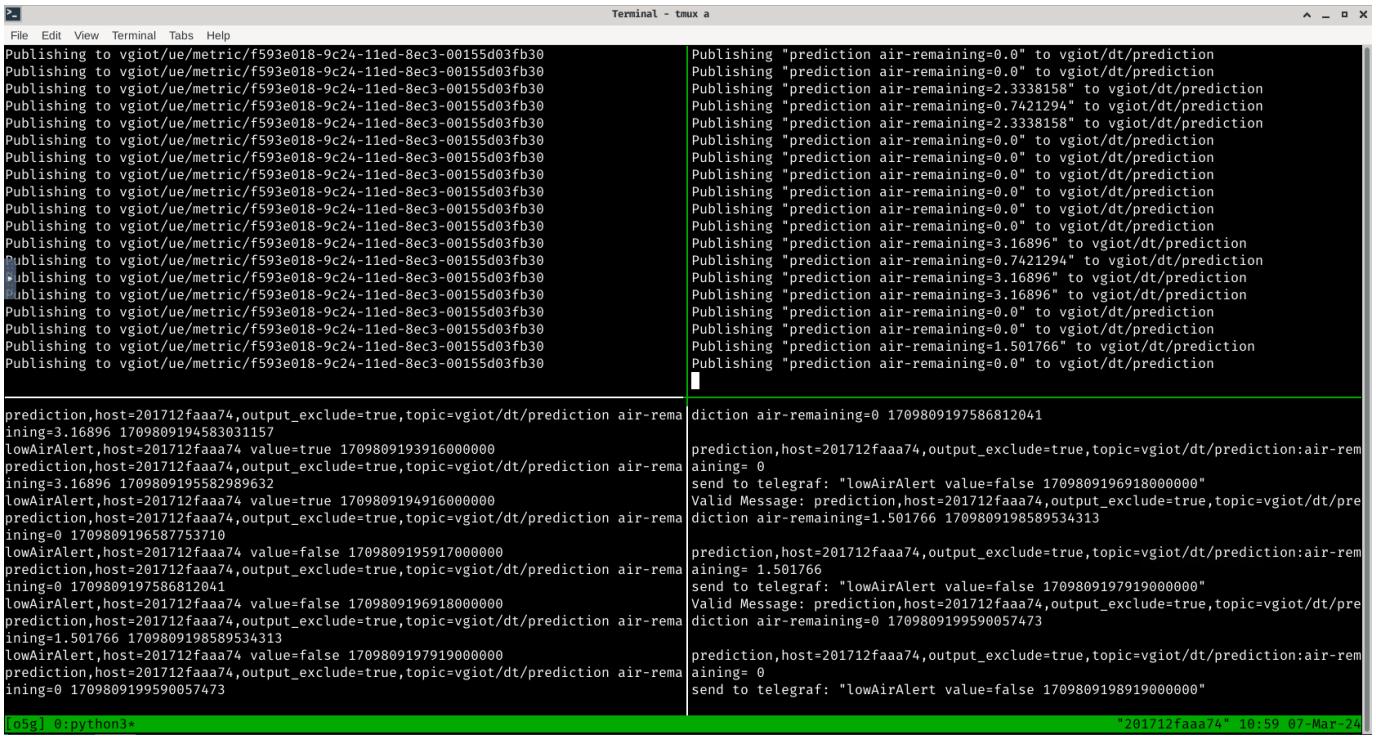
To run the digital twin, execute

```
lifecycle/execute
```

This starts all the required components in a single tmux session called `o5g` in the background. To view the running digital twin, attach to this tmux session by executing

```
tmux a -t o5g
```

To detach, press `Ctrl-b` followed by `d`.



Copyright © 2022 - 2026 The INTO-CPS Association

The *tmux* session contains four components of the digital twin:

Panel location	Purpose
Top Left	Sensor simulator generating random location and O2-level data
Top Right	Main Digital Twin receives the sensor data and calculates an estimate of how many minutes of air remain
Bottom Left	Telegraf to convert between different message formats, also displays all messages between components
Bottom Right	TeSSLa monitor raises an alarm if the remaining time is too low.

Examine the Results

For more mission awareness, use the InfluxDB data visualisation. The file *influx-dashboard.json* has a dashboard configuration. Log in to the InfluxDB server (usually on port 8086) to import it. A screenshot of the dashboard is given here.



The data is stored in the `o5g->prediction->air-remaining->37ae3e4fb3ea->true->vgiot/dt/prediction` variable of InfluxDB. Besides importing the dashboard configuration above, you can create custom dashboards from the stored data.

Terminate

To stop all components and close the *tmux* session, execute

```
lifecycle/terminate
```

Clean

To remove the temporary files created during execution, execute

```
lifecycle/clean
```


Copyright © 2022 - 2026 The INTO-CPS Association

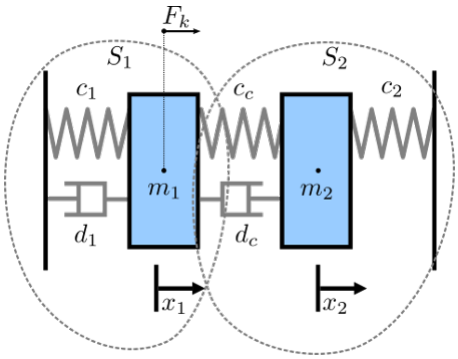
Mass Spring Damper with NuRV Runtime Monitor

OVERVIEW

This digital twin is derived from the Mass Spring Damper digital twin.

The mass spring damper digital twin (DT) has two mass spring dampers. It shows how to use a co-simulation based DT within the DTaaS. This version of the example adds a monitor generated by NuRV. More information about NuRV is available on the NuRV tool page.

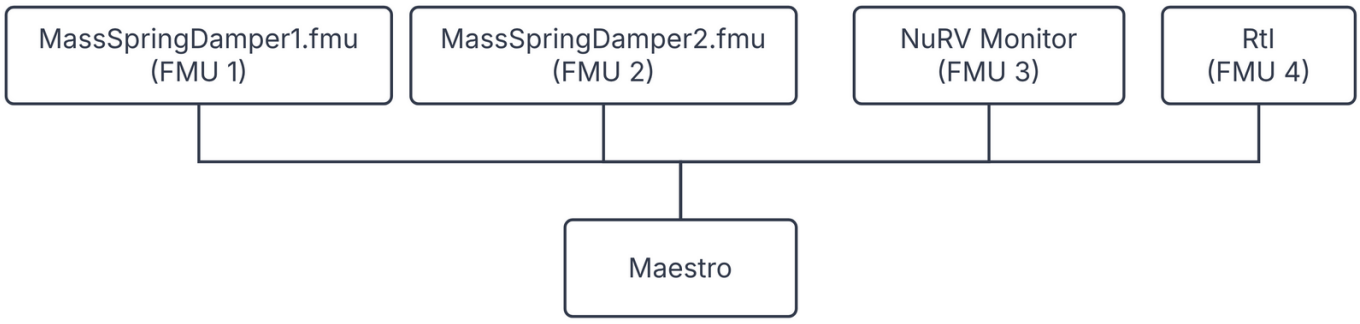
EXAMPLE DIAGRAM



EXAMPLE STRUCTURE

The study includes two simulators, each representing a mass spring damper system. The first simulator calculates the displacement and speed of mass \(m_1\) for a given force \(F_k\) acting on \(m_1\). The second simulator calculates the force \(F_k\) from a given displacement and speed of mass \(m_1\). Coupling the two simulators computes how the positions of the two masses evolve.

A monitor added to the simulation checks at runtime whether the displacement of the two masses stays below a maximum threshold.



DIGITAL TWIN CONFIGURATION

This example uses four models, one specification and one tool:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	MassSpringDamper1.fmu	Private	Yes
	MassSpringDamper2.fmu	Private	Yes
	m2.fmu	Private	No
	RtI.fmu	Private	Yes
Specification	m2.smv	Private	No
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes

Copyright © 2022 - 2026 The INTO-CPS Association

The DT is executed with two configuration files, `cosim.json` and `time.json`. Edit them to customise the DT.

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool Generates and compiles the monitor FMU
Execute	Produces and stores output in data/mass-spring-damper-monitor/output directory
Analyze	Processes the co-simulation output and produces a plot
Clean	Clears run logs and outputs

RUN THE EXAMPLE

To run the example, change to the example directory.

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

If required, make the lifecycle scripts executable, for example:

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

Then run the following scripts.

Create

- Installs Open Java Development Kit 21 and the pandas and matplotlib Python packages in the workspace.
- Generates and compiles the monitor FMU from the NuRV specification.

```
lifecycle/create
```

Execute

Runs the digital twin. Because this is a co-simulation based digital twin, the Maestro co-simulation tool runs a co-simulation of the FMU models.

```
lifecycle/execute
```

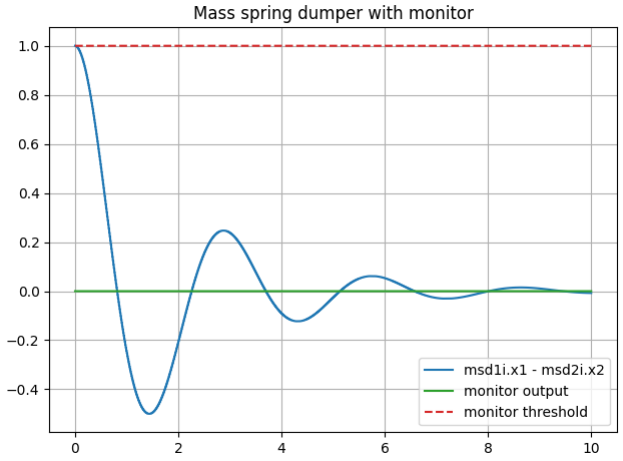
Analyze phase

Processes the co-simulation output and saves a plot in `/workspace/examples/data/mass-spring-damper-monitor/output/plots`.

```
lifecycle/analyze
```

Here is a sample plot.

Copyright © 2022 - 2026 The INTO-CPS Association



The plot uses three colours. The blue line is the distance between the two masses, and the green line is the monitor's verdict. The red dashed line marks the distance threshold that the monitor checks. The distance between the masses always stays below the threshold, so the monitor output stays at `unknown` (`0`).

Examine the results

The results are in the */workspace/examples/data/mass-spring-damper-monitor/output directory*.

The run logs are in */workspace/examples/digital_twins/mass-spring-damper-monitor*.

Terminate phase

Removes the debug files and co-simulation output files.

```
lifecycle/terminate
```

REFERENCES

More information about co-simulation techniques and the mass spring damper case study is 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 is in the mass spring damper GitHub repository.

Copyright © 2022 - 2026 The INTO-CPS Association

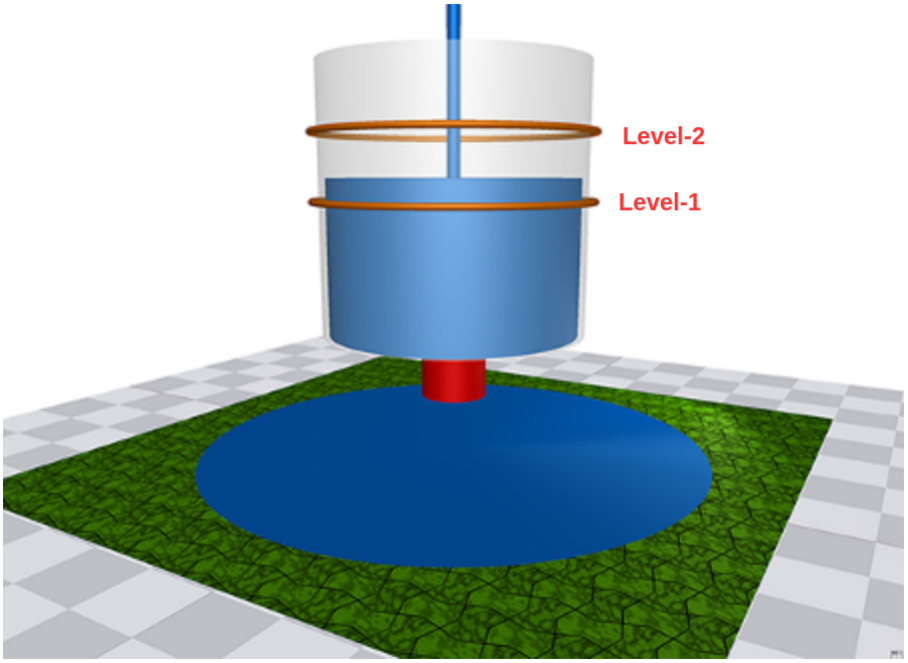
Water Tank Fault Injection with NuRV Runtime Monitor

OVERVIEW

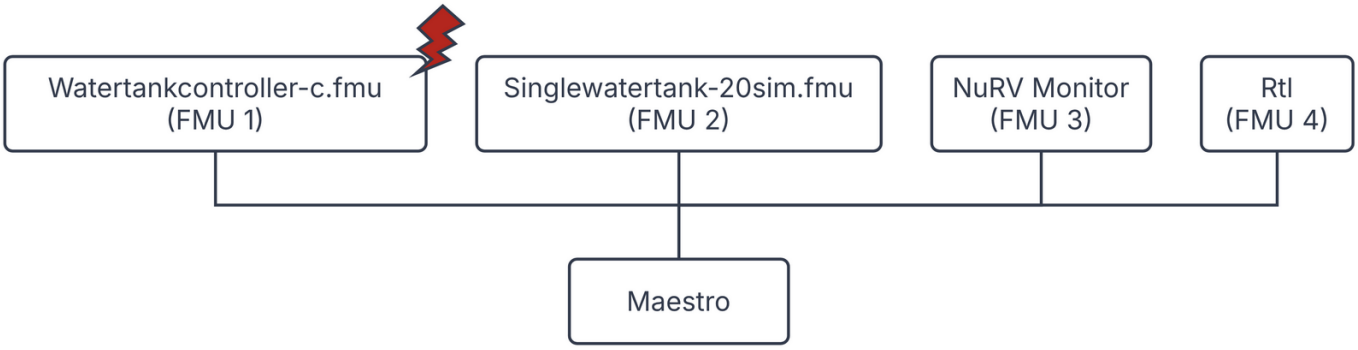
This example shows a digital twin (DT) with fault injection (FI). A live DT receives simulated faults from the environment. The faults are part of the DT configuration and can be changed for new instances of the DT. This version of the example adds a monitor generated by NuRV. More information about NuRV is available on the NuRV tool page.

This co-simulation based DT uses a water tank case study. The co-simulation consists of a tank and a controller, and its goal is to keep the water level in the tank between `Level-1` and `Level-2`. Faults are injected into the output of the water tank controller (**watertankcontroller-c.fmu**) from 12 to 20 time units. The faults close the tank outlet for a period of time, and the water level rises above the desired level (`Level-2`). A monitor added to the simulation checks at runtime whether the water level stays below a maximum threshold.

EXAMPLE DIAGRAM



EXAMPLE STRUCTURE



Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses four models, one specification and one tool:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Models	watertankcontroller-c.fmu	Private	Yes
	singlewatertank-20sim.fmu	Private	Yes
	m1.fmu	Private	No
	RtI.fmu	Private	Yes
Specification	m1.smv	Private	No
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes

The DT is executed with two configuration files, `multimodelFI.json` and `simulation-config.json`. Edit them to customise the DT.

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

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool Generates and compiles the monitor FMU
Execute	Produces and stores output in data/water_tank_FI_monitor/output directory
Analyze	Processes the co-simulation output and produces a plot
Clean	Clears run logs and intermediate co-simulation files

RUN THE EXAMPLE

To run the example, change to the example directory.

```
cd /workspace/examples/digital_twins/water_tank_FI_monitor
```

If required, make the lifecycle scripts executable, for example:

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

Then run the following scripts.

Create

Installs Open Java Development Kit 21 and the pandas and matplotlib Python packages from the Ubuntu repositories (`python3-pandas` and `python3-matplotlib`). It then generates and compiles the monitor FMU from the NuRV specification.

```
lifecycle/create
```

Execute

Runs the co-simulation and writes the `outputs.csv` file to `/workspace/examples/data/water_tank_FI_monitor/output`.

```
lifecycle/execute
```

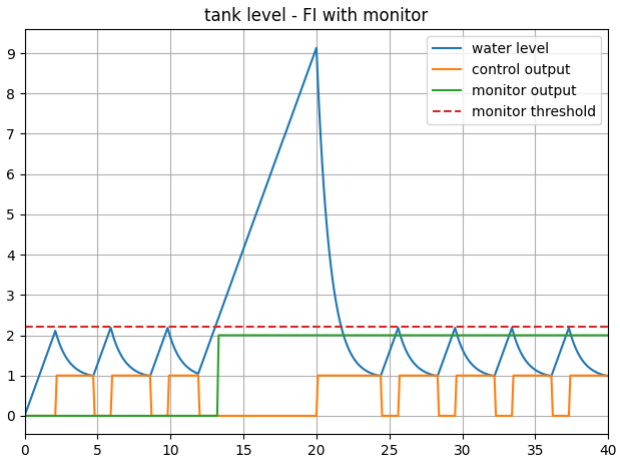

Copyright © 2022 - 2026 The INTO-CPS Association

Analyze phase

Processes the co-simulation output and saves a plot in `/workspace/examples/data/water_tank_FI_monitor/output/plots/`.

```
lifecycle/analyze
```

Here is a sample plot.



The plot uses four colours. The blue line is the water level in the tank, the orange line is the control output and the green line is the monitor's verdict. The red dashed line marks the water level threshold that the monitor checks. When the water level rises above this threshold, the monitor's verdict changes from `unknown` (`0`) to `false` (`2`).

Examine the results

The results are in the */workspace/examples/data/water_tank_FI_monitor/output directory*.

The run logs are in */workspace/examples/digital_twins/water_tank_FI_monitor*.

Terminate phase

Removes the temporary files.

```
lifecycle/terminate
```

REFERENCES

More details on this case study are 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 extends the Maestro co-orchestration engine. It can replace the inputs and outputs of FMUs in an FMI-based co-simulation with tampered values. More details on the plugin are in the fault injection Git repository. The same repository holds the source code for this example in its example directory.

Copyright © 2022 - 2026 The INTO-CPS Association

Incubator Co-Simulation Digital Twin validation with NuRV Monitor

OVERVIEW

This example shows how to validate digital twin components using FMU monitors. Here the monitors are generated with NuRV[1].

SIMULATED SCENARIO

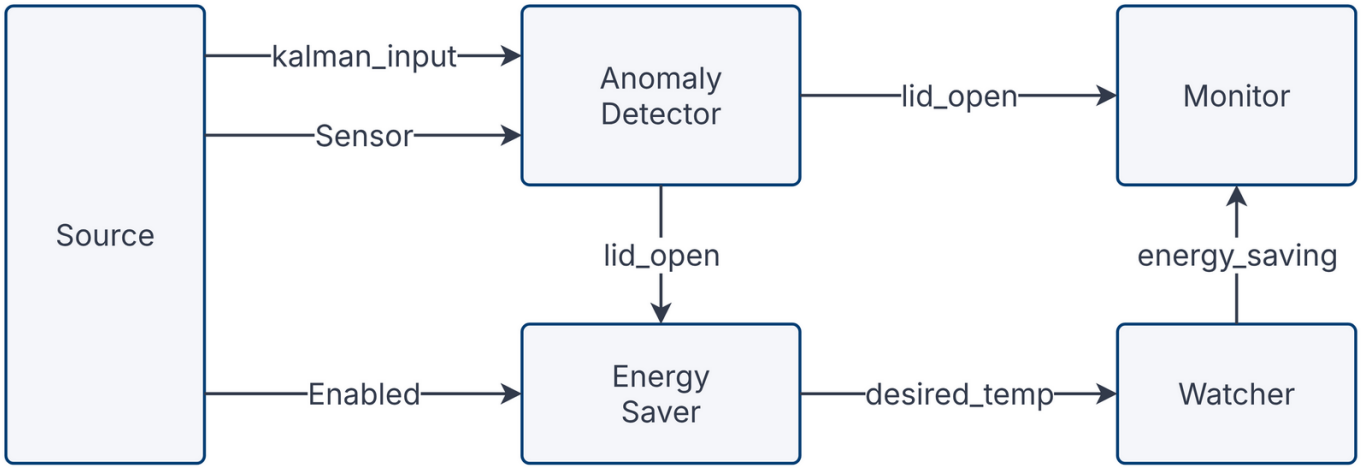
This example validates some components of the Incubator digital twin through a simulation in which the components are wrapped inside FMUs and inspected at runtime by an FMU monitor generated by NuRV. The link to the Incubator digital twin is given only for the details of the incubator physical twin. The digital twin (DT) presented here is a co-simulation DT of the Incubator.

The input data for the simulation comes from a purpose-built FMU component named *source*, which supplies test data to the *anomaly detector* and simulates an anomaly at time t=60s. A second component, the *watcher*, checks whether the *energy saver* activates in response to an anomaly reported by the *anomaly detector*.

The output of the watcher is then passed to the monitor, which checks that the *energy saver* activates within three simulation cycles of a detected anomaly.

EXAMPLE STRUCTURE

The diagram below shows the logical software structure of the example.



DIGITAL TWIN CONFIGURATION

The example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in other Examples
Models	anomaly_detection.fmu	Private	No
	energy_saver.fmu	Private	No
	Source.fmu	Private	No
	Watcher.fmu	Private	No
Specification	safe-operation.smv	Private	No
Tool	maestro-4.1.0-jar-with-dependencies.jar	Common	Yes

The *safe-operation.smv* file contains the default monitored specification described in the Simulated scenario section.

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE PHASES

The lifecycle phases for this example are:

Lifecycle Phase	Completed Tasks
Create	Installs Java Development Kit for Maestro tool Generates and compiles the monitor FMU
Execute	Produces and stores output in data/incubator-NuRV-monitor-validation/output directory
Clean	Clears run logs and outputs

If required, make the lifecycle scripts executable with the following command

```
chmod +x lifecycle/{script}
```

where {script} is the name of the script, for example *create* or *execute*.

RUN THE EXAMPLE

To run the example, change to its directory.

```
cd /workspace/examples/digital_twins/incubator-NuRV-monitor-validation
```

If required, make the lifecycle scripts executable, for example:

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

Now, run the following scripts:

Create

- Installs Open Java Development Kit 21 and, from the Ubuntu repositories, the Python packages used by the plotting script and the Python FMUs (`python3-pandas` , `python3-matplotlib` , `python3-protobuf` and `python3-zmq`).
- Generates and compiles the monitor FMU from the NuRV specification.

```
lifecycle/create
```

Execute

Runs the digital twin. Since this is a co-simulation based digital twin, the Maestro co-simulation tool runs the co-simulation with the FMU models.

```
lifecycle/execute
```

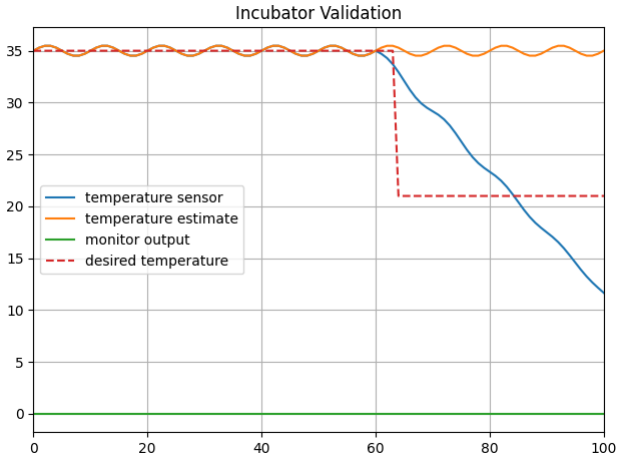
Analyze phase

Processes the co-simulation output to produce a plot in `/workspace/examples/data/incubator-NuRV-monitor-validation/output/plots` .

```
lifecycle/analyze
```

A sample plot is given here.

Copyright © 2022 - 2026 The INTO-CPS Association



The plot has four colour-coded lines. The blue line is the simulated temperature and the orange line is the temperature estimate. The red dashed line is the target temperature set by the energy saver component, and the green line is the monitor's verdict. When the estimated and actual temperatures diverge, the energy saver lowers the target temperature, which shows that the component works as intended. The monitor output therefore stays at `unknown` (`0`), meaning the monitored property is not violated.

Examine the results

The results, including the logs, are in the */workspace/examples/data/incubator-NuRV-monitor-validation/output* directory.

The plots of the results are in the */workspace/examples/data/incubator-NuRV-monitor-validation/output/plots* directory.

Terminate phase

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

```
lifecycle/terminate
```

REFERENCES

1. More information about NuRV is available on the NuRV tool page.

Copyright © 2022 - 2026 The INTO-CPS Association

Incubator Digital Twin with NuRV monitoring service

OVERVIEW

This example shows how a runtime monitoring service (here NuRV[1]) can be connected to the Incubator digital twin to verify the runtime behaviour of the Incubator.

SIMULATED SCENARIO

This example simulates a scenario where the lid of the Incubator is removed and later put back on. The Incubator has anomaly detection, which can detect anomalous behaviour such as the removal of the lid. When it detects an anomaly, the Incubator switches to an energy saving mode in which the heater is turned off.

The monitor checks that the energy saving mode is turned on within 3 simulation steps of an anomaly detection. The property for this is: \(G(anomaly \rightarrow (F_{[0,3]}\space energy_saving))\). The monitor outputs the *Unknown* state as long as the property is satisfied and moves to the *False* state once it detects a violation.

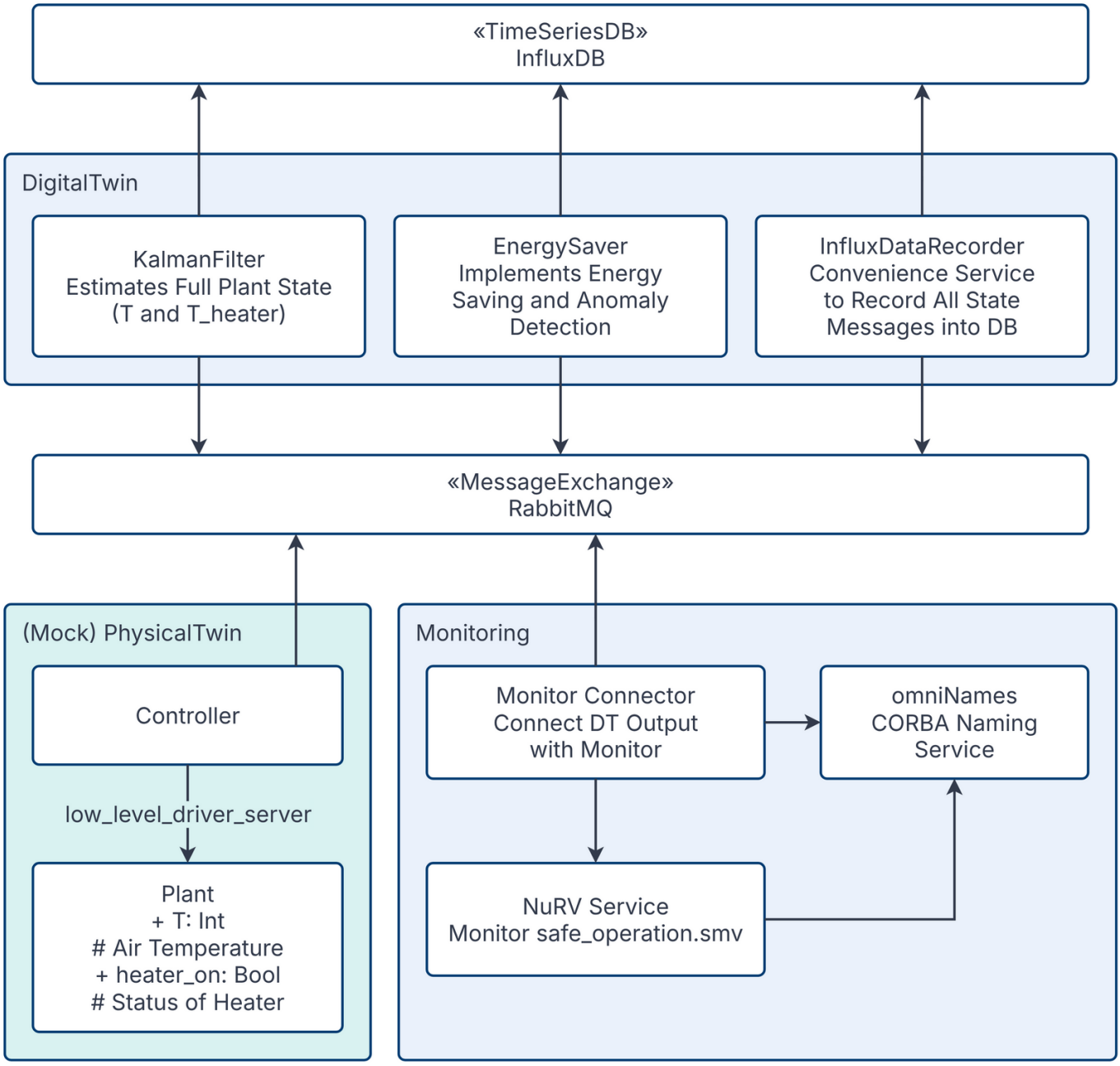
The simulated scenario progresses as follows:

- *Initialization*: The services are initialized and the Kalman filter in the Incubator is given 2 minutes to stabilize. The anomaly detection algorithm sometimes detects an anomaly at startup even though the lid is still on. This disappears after about 15 seconds.
- *After 2 minutes*: The lid is lifted and an anomaly is detected. The energy saver is turned on shortly after.
- *After another 30 seconds*: The energy saver is manually disabled, producing a False verdict.
- *After another 30 seconds*: The lid is put back on and the anomaly detection is given time to detect that the lid is back on. The monitor is then reset, producing an Unknown verdict again. The simulation then ends.

EXAMPLE STRUCTURE

The diagram below shows the logical software structure of the example.

Copyright © 2022 - 2026 The INTO-CPS Association



The *execute.py* script orchestrates and starts all the services in this example: the Incubator DT, the CORBA naming service (omniNames) and the NuRV monitor server. It also implements the *Monitor connector* component, which connects the DT output to the NuRV monitor server.

The NuRV monitor server registers under a specific name with a CORBA naming service. A client can query the naming service for that name to obtain a reference to the monitor server. For more information on how the NuRV monitor server works, refer to [1].

After the connection with the NuRV monitor server is established, the Incubator DT starts and a RabbitMQ client subscribes to changes in the *anomaly* and *energy_saving* states of the DT. Each time either state changes, the full state (the updated state and the last value of the other state) is pushed to the NuRV monitor server, and the verdict is printed to the console.

DIGITAL TWIN CONFIGURATION

Before running the example, configure the *simulation.conf* file with the appropriate RabbitMQ credentials.

Copyright © 2022 - 2026 The INTO-CPS Association

The example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in other Examples
Service	common/services/NuRV/NuRV_orbit	Common	Yes
DT	common/digital_twins/incubator	Common	Yes
Specification	safe-operation.smv	Private	No
Script	execute.py	Private	No

The *safe-operation.smv* file contains the default monitored specification described in the Simulated scenario section. It can be changed as needed.

LIFECYCLE PHASES

The lifecycle phases for this example are:

Lifecycle phase	Completed tasks
create	Downloads the necessary tools and creates a virtual python environment with the necessary dependencies. The omniORBpy CORBA library is compiled from source because there are no packages of it for Python 3.12
execute	Runs a python script that starts up the necessary services as well as the Incubator simulation. Various status messages are printed to the console, including the monitored system states and monitor verdict.
clean	Removes created *data* directory and incubator log files.

If required, make the lifecycle scripts executable with the following command

```
chmod +x lifecycle/{script}
```

where {script} is the name of the script, for example *create* or *execute*.

RUNNING THE EXAMPLE

To run the example, first run the following command in a terminal:

```
cd /workspace/examples/digital_twins/incubator-NuRV-monitor-service/
```

Then run the *create* script, which can take a few minutes depending on the network connection, followed by the *execute* script, using the following command:

```
lifecycle/{script}
```

The *execute* script then prints the system states and the monitor verdict about every 3 seconds, in the format "**State: {anomaly state} & {energy_saving state}, verdict: {Verdict}**", where "*anomaly*" means that an anomaly is detected and "!anomaly" means that no anomaly is currently detected. The energy_saving state uses the same format.

The monitor verdict can be False or Unknown. Unknown means that the monitor does not yet have enough information to decide whether the property is satisfied. Because of the G operator, the monitor never produces a True verdict: satisfaction can only be confirmed over the entire trace. The Unknown state can therefore be read as a tentative True verdict.

Copyright © 2022 - 2026 The INTO-CPS Association

An example output trace is provided below:

```
....
Running scenario with initial state: lid closed and energy saver on
Setting energy saver mode: enable
Setting G_box to: 0.5763498
State: !anomaly & !energy_saving, verdict: Unknown
State: !anomaly & !energy_saving, verdict: Unknown
....
State: anomaly & !energy_saving, verdict: Unknown
State: anomaly & energy_saving, verdict: Unknown
State: anomaly & energy_saving, verdict: Unknown
....
State: anomaly & energy_saving, verdict: Unknown
State: anomaly & !energy_saving, verdict: Unknown
State: anomaly & !energy_saving, verdict: Unknown
State: anomaly & !energy_saving, verdict: Unknown
State: anomaly & !energy_saving, verdict: Unknown
State: anomaly & !energy_saving, verdict: False
```

The connection to the NuRV server often fails at startup and takes a few attempts to establish. The script retries automatically.

REFERENCES

1. Information on the NuRV monitor can be found on FBK website.

Copyright © 2022 - 2026 The INTO-CPS Association

Incubator Digital Twin with NuRV FMU Monitoring Service

OVERVIEW

This example shows how an FMU can serve as a runtime monitoring service (here generated with NuRV[1]) and be connected to the Incubator digital twin to verify the runtime behaviour of the Incubator.

SIMULATED SCENARIO

This example simulates a scenario where the lid of the Incubator is removed and later put back on. The Incubator has anomaly detection, which can detect anomalous behaviour such as the removal of the lid. When it detects an anomaly, the Incubator switches to an energy saving mode in which the heater is turned off.

The monitor checks that the energy saving mode is turned on within 3 messages of an anomaly detection. The property for this is:

$G(anomaly \rightarrow (F_{[0,3]}\space energy_saving))$.

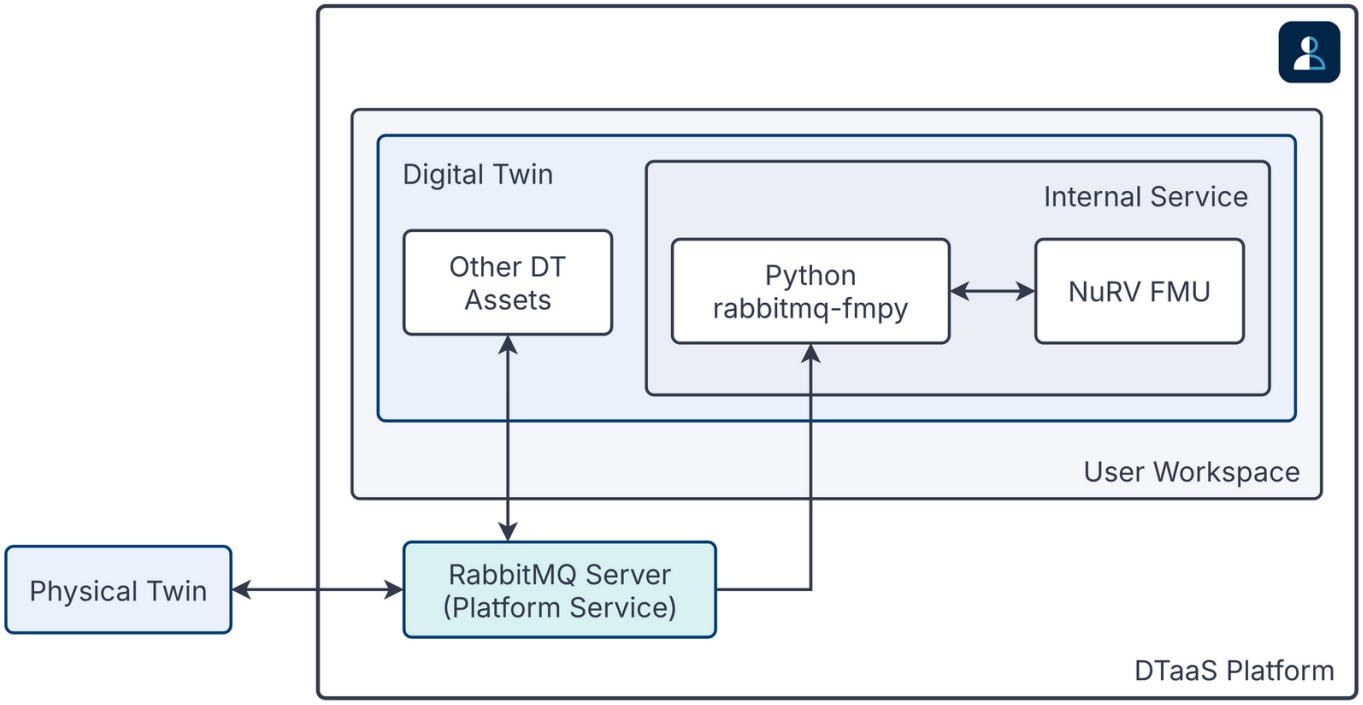
The monitor outputs the *unknown* state as long as the property is satisfied and moves to the *false* state once it detects a violation.

The simulated scenario progresses as follows:

- *Initialization*: The services are initialized and the Kalman filter in the Incubator is given 2 minutes to stabilize. The anomaly detection algorithm sometimes detects an anomaly at startup even though the lid is still on. This disappears after about 15 seconds.
- *After 2 minutes*: The lid is lifted and an anomaly is detected. The energy saver is turned on shortly after.
- *After another 30 seconds*: The energy saver is manually disabled, producing a *false* verdict.
- *After another 30 seconds*: The lid is put back on and the anomaly detection is given time to detect that the lid is back on. The monitor is then reset, producing an *unknown* verdict again. The simulation then ends.

EXAMPLE STRUCTURE

The diagram below shows the logical software structure of the example.



The *execute* script starts the NuRV service and then runs the Python script that controls the scenario (*execute.py*).

Copyright © 2022 - 2026 The INTO-CPS Association

The *execute.py* script starts the Incubator services and runs the example scenario. Once the Incubator DT is started, a RabbitMQ client is created that subscribes to changes in the *anomaly* and *energy_saving* states of the DT, as well as the verdicts produced by the NuRV service. Each time an update arrives, the full state or the verdict is printed to the console.

DIGITAL TWIN CONFIGURATION

Before running the example, configure the *simulation.conf* file with the appropriate RabbitMQ credentials.

The example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in other Examples
Tools	common/tools/NuRV/NuRV	Common	Yes
Other	common/fmi2_headers	Common	Yes
DT	common/digital_twins/incubator	Common	Yes
Specification	safe-operation.smv	Private	No
Script	execute.py	Private	No

The *safe-operation.smv* file contains the default monitored specification described in the Simulated scenario section. It can be changed as needed.

LIFECYCLE PHASES

The lifecycle phases for this example are:

Lifecycle phase	Completed tasks
create	Creates a virtual python environment with the necessary dependencies and generates the monitor FMU from the NuRV specification
execute	Runs a python script that starts up the necessary services as well as the Incubator simulation. Various status messages are printed to the console, including the monitored system states and monitor verdict.

If required, make the lifecycle scripts executable with the following command.

```
chmod +x lifecycle/{script}
```

where {script} is the name of the script, for example *create* or *execute*.

RUNNING THE EXAMPLE

To run the example, first run the following command in a terminal:

```
cd /workspace/examples/digital_twins/incubator-NuRV-fmu-monitor-service/
```

Then run the *create* script followed by the *execute* script using the following command:

```
lifecycle/{script}
```

The *execute* script then prints the system states and the monitor verdict about every 3 seconds. System states are printed in this format:

"**State: {anomaly state} & {energy_saving state}**"

where "*anomaly*" means that an anomaly is detected and "!anomaly" means that no anomaly is currently detected. The energy_saving state uses the same format. NuRV verdicts are printed in this format:

"**Verdict from NuRV: {verdict}**".

Copyright © 2022 - 2026 The INTO-CPS Association

The monitor verdict can be false or unknown. Unknown means that the monitor does not yet have enough information to decide whether the property is satisfied. Because of the G operator, the monitor never produces a true verdict: satisfaction can only be confirmed over the entire trace. The unknown state can therefore be read as a tentative true verdict.

An example output trace is provided below:

```
....
Using LIFECYCLE_PATH: /workspace/examples/digital_twins/incubator-NuRV-fmu-monitor-service/lifecycle
Using INCUBATOR_PATH: /workspace/examples/digital_twins/incubator-NuRV-fmu-monitor-service/lifecycle/../../../common/digital_twins/incubator
Starting NuRV FMU Monitor Service, see output at /tmp/nurv-fmu-service.log
NuRVService.py PID: 13496
Starting incubator
Connected to rabbitmq server.
Running scenario with initial state: lid closed and energy saver on
Setting energy saver mode: enable
Setting G_box to: 0.5763498
State: !anomaly & !energy_saving
State: !anomaly & !energy_saving
Verdict from NuRV: unknown
State: !anomaly & !energy_saving
State: !anomaly & !energy_saving
Verdict from NuRV: unknown
State: !anomaly & !energy_saving
State: !anomaly & !energy_saving
Verdict from NuRV: unknown
State: !anomaly & !energy_saving
State: !anomaly & !energy_saving
Verdict from NuRV: unknown
State: !anomaly & !energy_saving
State: !anomaly & !energy_saving
Verdict from NuRV: unknown
```

REFERENCES

1. Information on the NuRV monitor can be found on FBK website.

Copyright © 2022 - 2026 The INTO-CPS Association

Incubator Digital Twin with TeSSLa Monitoring Service

OVERVIEW

This example demonstrates how a runtime monitoring service (in this example TeSSLa[1]) can be connected to the Incubator Digital Twin to verify the runtime behaviour of the incubator.

SIMULATED SCENARIO

This example simulates a scenario where the lid of the incubator is removed and later replaced. The incubator is equipped with anomaly detection capabilities that can detect anomalous behaviour (i.e. removal of the lid). If an anomaly is detected, the incubator will enter an energy saving mode where the heating is switched off.

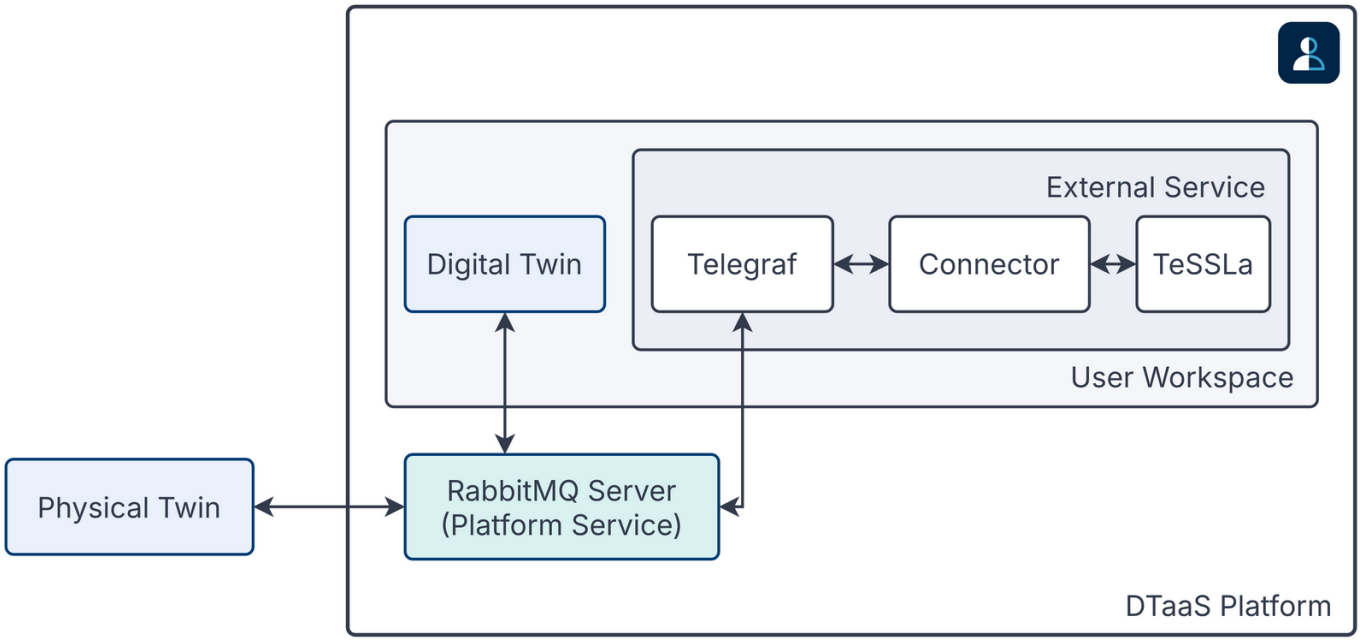
From a monitoring perspective, we want to verify that within approximately 3 simulation steps of an anomaly being detected, the energy saving mode is turned on. To verify this behaviour we construct the property: \(G(anomaly \rightarrow (F_{[0,3]}\space energy_saving))\). The monitor will output the *normal* state as long as the property is satisfied, and will switch to the *alert* state as soon as a violation is detected.

The simulated scenario is as follows

- *Initialisation*: The services are initialised and the Kalman filter in the incubator is given 2 minutes to stabilise.
- After 2 minutes*: The lid is lifted and an anomaly is detected. The power saver is activated shortly after.
- After a further 30 seconds: The power saver is manually deactivated, resulting in a False verdict.
- After another 30 seconds: The lid is replaced and anomaly detection is given time to detect that the lid is back on. The monitor is then reset, producing a *normal* verdict again. The simulation ends.

EXAMPLE STRUCTURE

A diagram showing the logical software structure of the example is shown below.



The *execute.py* script is responsible for orchestrating and starting all relevant services in this example. This includes the incubator DT, the TeSSLa monitor and the telegraf and TeSSLa telegraf connector for communication between the DT and the TeSSLa monitor.

The telegraf client subscribes to the RabbitMQ server used by the DT for communication and forwards the messages on the topics *incubator.diagnosis.plant.lidopen* and *incubator.energysaver.status* to the TeSSLa telegraf connector, which feeds them as inputs to the TeSSLa monitor. TeSSLa then makes a judgement based on this new state together with the previously received states. The judgement is then sent back to Telegraf via the connector and published to the *incubator.energysaver.alert* topic and printed on the console.

Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

Before running the example, please configure the *simulation.conf* file with your RabbitMQ credentials.

The example uses the following assets

Asset type	Asset names	Visibility	Reuse in other examples
DT	common/digital_twins/incubator	Common	Yes
Specification	safe-operation.tessla	Private	No
Configuration	telegraf.conf	Private	No
Script	execute.py	Private	No

The *safe-operation.tessla* file contains the default monitoring specifications as described in the Simulated Scenario section. These can be configured as required.

LIFECYCLE PHASES

The lifecycle phases for this example include:

Lifecycle phase	Completed tasks
create	Downloads the necessary tools and creates a virtual python environment with the necessary dependencies
execute	Runs a python script that starts up the necessary services as well as the Incubator simulation. Various status messages are printed to the console, including the monitored system states and monitor verdict.
clean	Removes created *data* directory and incubator log files.

If required, change the execute permissions of lifecycle scripts you need to execute. This can be done using the following command

```
chmod +x lifecycle/{script}
```

where {script} is the name of the script, e.g. *create*, *execute* etc.

RUNNING THE EXAMPLE

To run the example, first run the following command in a terminal:

```
cd /workspace/examples/digital_twins/incubator-tessla-monitor-service/
```

Then, first execute the *create* script (this can take a few mins depending on your network connection) followed by the *execute* script using the following command:

```
lifecycle/{script}
```

The *execute* script will then start outputting system states and the monitor verdict approx every 3 seconds. The output is printed as follows "**State: {anomaly state} & {energy_saving state}, verdict: {Verdict}**" where "*anomaly*" indicates that an anomaly is detected and "!anomaly" indicates that an anomaly is not currently detected. The same format is used for the energy_saving state.

The monitor verdict can be *normal* or *alert*, where the latte indicates that the monitor does not yet have sufficient informationto determine the satisfaction of the property. The monitor will never produce a True verdict as the entire trace must be verified to ensure satisfaction due to the G operator. Thus the *normal* state can be viewed as a tentative True verdict.

Copyright © 2022 - 2026 The INTO-CPS Association

An example output trace is provided below:

```
....
State: anomaly & energy_saving, verdict: normal
State: anomaly & energy_saving, verdict: normal
State: anomaly & !energy_saving, verdict: normal
State: anomaly & !energy_saving, verdict: normal
State: anomaly & !energy_saving, verdict: normal
State: anomaly & !energy_saving, verdict: normal
State: anomaly & !energy_saving, verdict: alert
State: anomaly & !energy_saving, verdict: alert
```

REFERENCES

1. tessla.io

Copyright © 2022 - 2026 The INTO-CPS Association

Incubator Digital Twin with TeSSLa monitoring service

OVERVIEW

This example demonstrates how a runtime enforcement service (in this example implemented with the specification language TeSSLa[1]) can be connected with the Incubator Digital Twin to steer the behavior of the Incubator.

SIMULATED SCENARIO

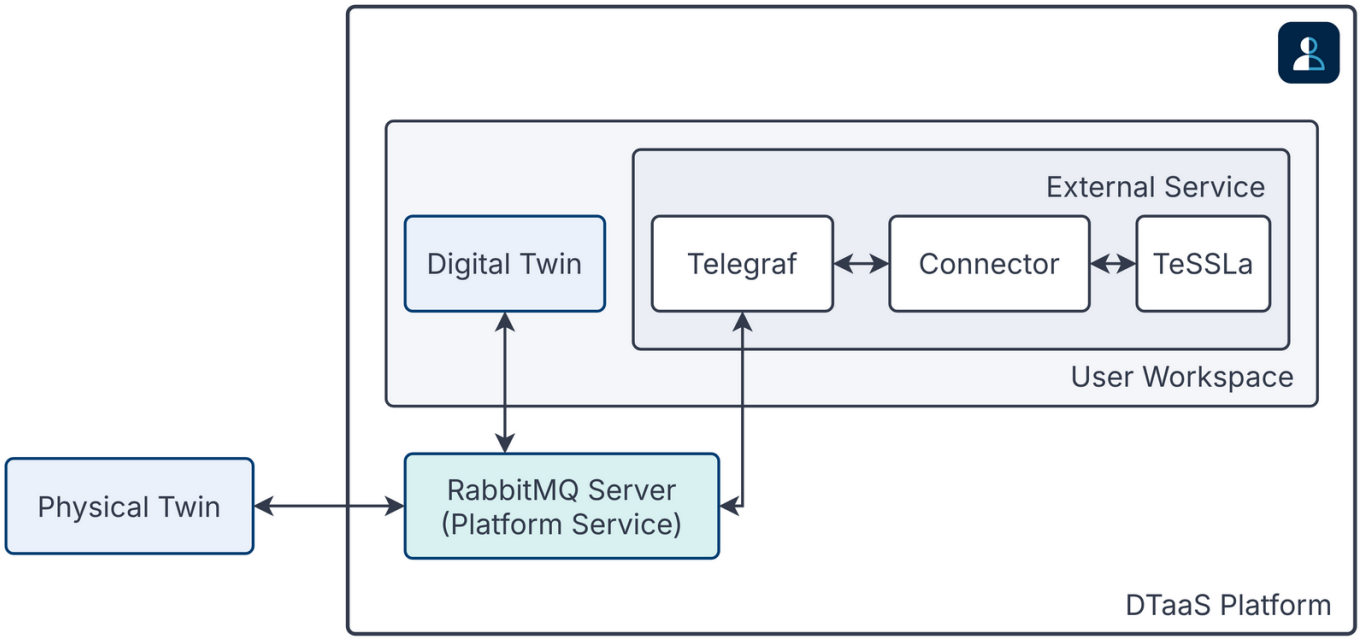
The incubator is a device used to maintain the temperature inside a box within a pre-defined range. In this specific scenario, the lid of the incubator is removed and later replaced. The incubator is equipped with anomaly detection capabilities that can detect anomalous behaviour (i.e. the removal of the lid). If an anomaly is detected, the incubator will, after a period of time, enter a power saving mode where the heating is turned off. The runtime enforcer provides the control signal for the power saving mode.

The simulated scenario is as follows:

- *Initialisation*: The services are initialised and the Kalman filter in the incubator is given 2 minutes to stabilise.
- After 2 minutes*: The lid is lifted and an anomaly is detected. The power saver is activated shortly after.
- After a further 30 seconds: The lid is replaced and the anomaly detector is given time to detect that the lid is back on. The simulation ends.

EXAMPLE STRUCTURE

A diagram showing the logical software structure of the example is shown below.



The *execute.py* script is responsible for orchestrating and starting all relevant services in this example. This includes the incubator DT, the TeSSLa runtime enforce as well as telegraf and the TeSSLa telegraf connector for communication between the DT and TeSSLa.

The telegraf client subscribes to the RabbitMQ server used by the DT for communication and forwards the messages on the topics *incubator.diagnosis.plant.lidopen* and *incubator.energysaver.status* to the TeSSLa telegraf connector, which feeds them as inputs to TeSSLa. TeSSLa then generates a control signal to activate or deactivate the energy saver mode. The signal is then sent back to Telegraf via the connector and published to RabbitMQ and printed on the console.

DIGITAL TWIN CONFIGURATION

Before running the example, please configure the *simulation.conf* file with your RabbitMQ credentials.

Copyright © 2022 - 2026 The INTO-CPS Association

The example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in other Examples
DT	common/digital_twins/incubator	Common	Yes
Specification	safe-operation.tessla	Private	No
Configuration	telegraf.conf	Private	No
Script	execute.py	Private	No

The *safe-operation.tessla* file contains the default monitored specification as described in the Simulated scenario section. These can be configured as desired.

LIFECYCLE PHASES

The lifecycle phases for this example include

lifecycle phase	completed tasks
create	Downloads the required tools and creates a virtual Python environment with the required dependencies
Execute	Runs a Python script that starts the required services and the incubator simulation. Various status messages are printed to the console, including the monitored system states and the monitor verdict.
clean	Removes the created *data* directory and the incubator log files.

If necessary, change the execution privileges of any lifecycle scripts you need to run. This can be done with the following command

```
chmod +x lifecycle/{script}
```

where {script} is the name of the script, e.g. *create*, *execute*, etc.

RUNNING THE EXAMPLE

To run the example, first run the following command in a terminal

```
cd /workspace/examples/digital_twins/incubator-tessla-active-monitor/
```

Then run the *create* script (this may take a few minutes depending on your network connection) followed by the *execute* script using the following command

```
lifecycle/{script}
```

The *execute* script will then start outputting system status and monitor verdict approximately every 3 seconds.

REFERENCES

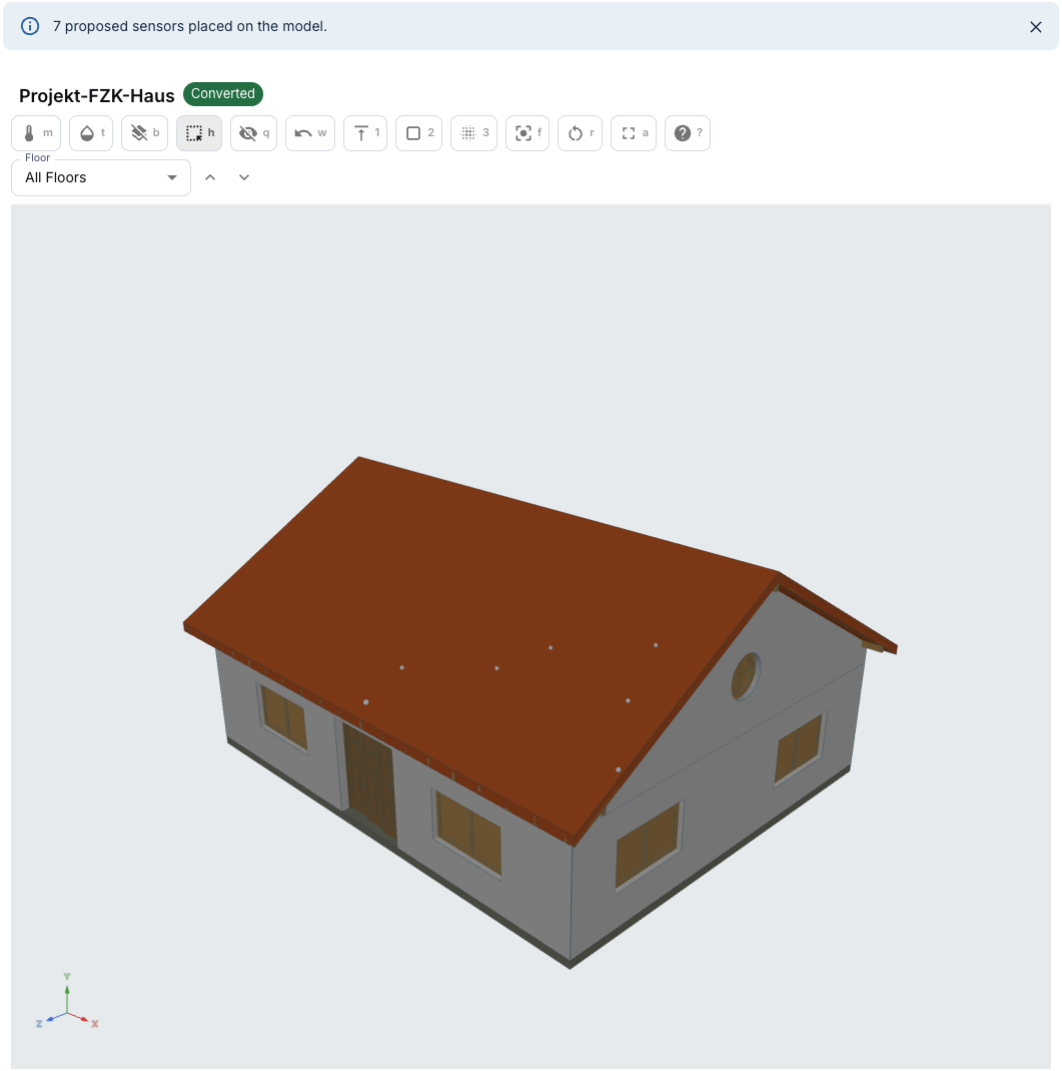
1. tessla.io

Copyright © 2022 - 2026 The INTO-CPS Association

BIM - Building Information Modeling

OVERVIEW

This example puts a building model in the DTaaS library and gives each of its rooms a simulated temperature sensor. The building is the FZK-Haus, a two-storey house modelled by the Karlsruhe Institute of Technology (KIT) as an IFC example. Its model has 7 rooms (`IfcSpace`) and no sensors, so the example proposes one temperature sensor per room. Rooms are not drawn in the 3D view, so each sensor is bound by its IFC `GlobalId` to a wall or column inside the room, the one nearest to the room's centre that no other room has taken. When a room has none left, the sensor goes on another drawn object in it, such as a door.



The Buildings page of the DTaaS client draws the model from the library. The binding manifest written by this example tells the page which reading belongs to which room. A publisher sends a simulated temperature for every room over MQTT, on the topics the manifest lists.

The model is `AC20-FZK-Haus.ifc` from the KIT IFC examples, which KIT provides for unrestricted use. Its content is unchanged, and only its line endings are stored as LF. As KIT asks, publications that use it credit the Institute for Automation and Applied Informatics (IAI) / Karlsruhe Institute of Technology (KIT).

Copyright © 2022 - 2026 The INTO-CPS Association

EXAMPLE STRUCTURE

File	What it does
`models/fzk_haus.ifc`	The building model, IFC4
`lifecycle/make_manifest.py`	Proposes one sensor per room, places it on a wall of the room and writes the binding manifest
`lifecycle/publish_readings.py`	Publishes a simulated temperature per room over MQTT
`lifecycle/show_readings.py`	Prints the first readings that reach the broker
`lifecycle/requirements.txt`	The IFC converters from ifc-utils and the MQTT client
`lifecycle/constraints.txt`	The versions of the packages those tools pull in, each at least ten days old when pinned

The create phase writes four files into `/workspace/common/models`, the folder the Buildings page reads:

File	Contents
`fzk_haus.ifc`	A copy of the building model
`fzk_haus.glb`	The geometry, so the page does not convert the model in the browser
`fzk_haus.json`	The property tree shown when you click an object
`fzk_haus.manifest.json`	One binding per room, from room to MQTT topic

DIGITAL TWIN CONFIGURATION

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Model	fzk_haus.ifc	Common	Yes
Tools	ifc-explorer, ifc-converter	Installed by create	Yes

The publisher reads the broker from these environment variables:

Variable	Default	Meaning
`MQTT_HOST`	`localhost`	Host name of the MQTT broker
`MQTT_PORT`	`1883`	MQTT port of the broker
`MQTT_USERNAME`	not set	User name, when the broker asks for one
`MQTT_PASSWORD`	not set	Password, when the broker asks for one
`MQTT_TLS`	not set	Set to `1` to connect over TLS, for example on port 8883

Without `MQTT_TLS`, the user name and password travel unencrypted, so set it whenever the broker is not on the same machine.

The lifecycle scripts work from any directory, as the other examples do. Two more variables change where they read and write:

Variable	Default	Meaning
`EXAMPLES_PATH`	the repository root	Where `models/fzk_haus.ifc` is read from
`WORKSPACE`	the folder above `EXAMPLES_PATH`, `/workspace` inside the workspace	The library whose `common/models` receives the model

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Installs the tools, places the model and writes the derived files
Execute	Starts the publisher in the background
Terminate	Stops the publisher and clears its log
Clean	Clears the publisher log and PID file

RUN THE EXAMPLE

Change to the example directory:

```
cd /workspace/examples/digital_twins/fzk_haus
```

If required, give the lifecycle scripts execute permission:

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

Create

Installs ifc-explorer, ifc-converter and paho-mqtt into a virtual environment in `.venv`. Then copies the model into `/workspace/common/models` and writes the geometry, the property tree and the binding manifest beside it. This takes about a minute, most of it for the installation.

```
lifecycle/create
```

Open **Buildings** in the DTaaS menu and choose **Projekt-FZK-Haus**, the name the model gives its project, to see it.

Execute

Starts publishing a temperature for each room every two seconds. Point it at the MQTT broker of your installation first, for example the one installed with the DTaaS platform services:

```
export MQTT_HOST=<broker host>
lifecycle/execute
```

Each reading goes to `dtaas/examples/fzk-haus/room-NN/temperature` as one line of comma-separated fields:

```
timestamp,measurement,type,unit,value
20261013T10:00:00,temperature,simulation,°C,21.40
```

The `type` field says `simulation`, so a subscriber can tell these values apart from measured ones. The log is in `publisher.log`.

To see the readings arrive through the broker, print the first round of them:

```
.venv/bin/python lifecycle/show_readings.py /workspace/common/models/fzk_haus.manifest.json
```

It prints one line per room, such as `1, Flur temperature: 22.10 °C at 12:02:14`, and stops with an error when no reading arrives within 10 seconds.

Terminate

Stops the publisher and removes its log:

```
lifecycle/terminate
```

CURRENT LIMITS

The Buildings page draws the model and resolves the manifest, but the DTaaS client does not subscribe to MQTT yet. Until it does, the readings reach any other MQTT subscriber, such as a dashboard, and not the page. The demo video shows this: the

Copyright © 2022 - 2026 The INTO-CPS Association

readings reach the terminal through the broker, and the page draws the model and its sensors, whose cards wait for a first reading.

REFERENCES

The converters come from the ifc-utils repository. The manifest format and the Buildings page are described in DTaaS issue 1762.

Copyright © 2022 - 2026 The INTO-CPS Association

3.9.3 Automation

Automation Examples

These examples use the DevOps features of the DTaaS platform. GitLab CI/CD pipelines run the lifecycle scripts of each digital twin, such as create, execute, and clean, without manual steps.

AVAILABLE EXAMPLES

Example	Description
Mass Spring Damper	Co-simulation DT with automated lifecycle via GitLab CI/CD
Structural Health Monitoring	SHM DT with automated system identification and model update
Heart Digital Twin	Real-time cardiovascular monitoring with web interface
Heart Digital Twin (PaaS)	Cloud deployment of Heart DT using PaaS strategies

CODEBASE

The source code for these examples is available in the DTaaS-examples repository.

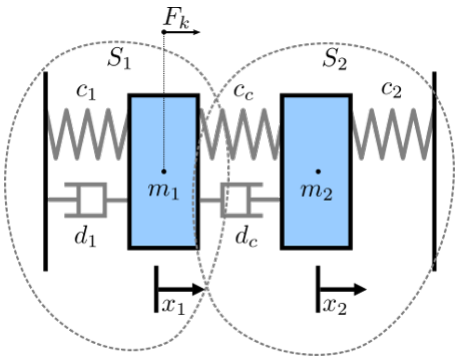
Copyright © 2022 - 2026 The INTO-CPS Association

Mass Spring Damper

OVERVIEW

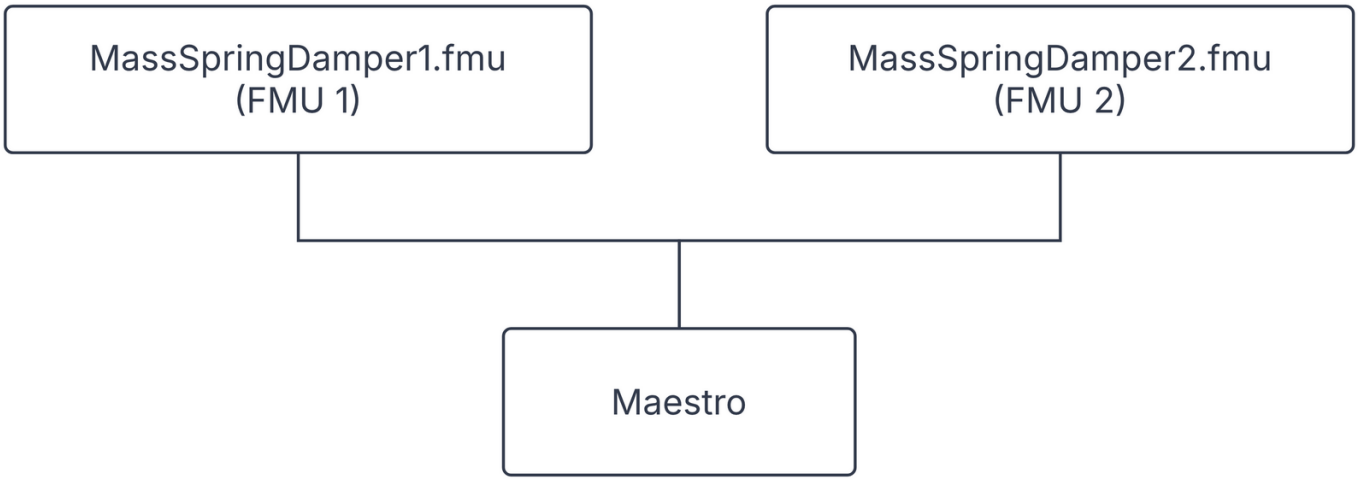
The mass spring damper digital twin (DT) has two mass spring dampers and shows how a co-simulation based DT can be used within DTaaS.

EXAMPLE DIAGRAM



EXAMPLE STRUCTURE

The study includes two simulators, each representing a mass spring damper system. The first simulator calculates the mass displacement and speed of \(m_1\) for a given force \(F_k\) acting on mass \(m_1\). The second simulator calculates force \(F_k\) given a displacement and speed of mass \(m_1\). 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:

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 `cosim.json` and `time.json` files configure the execution of the digital twin. Edit them to customise the DT.

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE PHASES

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

DEVOPS AUTOMATION

This example uses the DevOps features of the DTaaS platform, which the DTaaS provides through GitLab DevOps pipelines.

The `.gitlab-ci.yml` file sets the order in which the lifecycle scripts of the example run. Its configuration format lets you split a program into stages. This first example runs the create and execute scripts in a single stage named *build_and_run*.

GitLab saves the co-simulation outputs as job artefacts. You can find them at `https://<gitlab-host>/<group>/<username>/-/artifacts`. Replace `<gitlab-host>`, `<group>`, and `<username>` with the values for your GitLab instance and project.

RUN THE EXAMPLE

To run the example, go to the following directory.

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

If required, make the lifecycle scripts executable, for example:

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

Now, run the following scripts:

Create

Installs the OpenJDK 17 Java runtime in the workspace.

```
lifecycle/create
```

Execute

Runs the digital twin. Because this is a co-simulation based digital twin, the Maestro co-simulation tool runs a co-simulation of the two FMU models.

```
lifecycle/execute
```

Examine the results

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

The run logs are in the */workspace/examples/digital_twins/mass-spring-damper* directory.

Terminate phase

Removes the debug files and co-simulation output files.

```
lifecycle/terminate
```

REFERENCES

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

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


Copyright © 2022 - 2026 The INTO-CPS Association

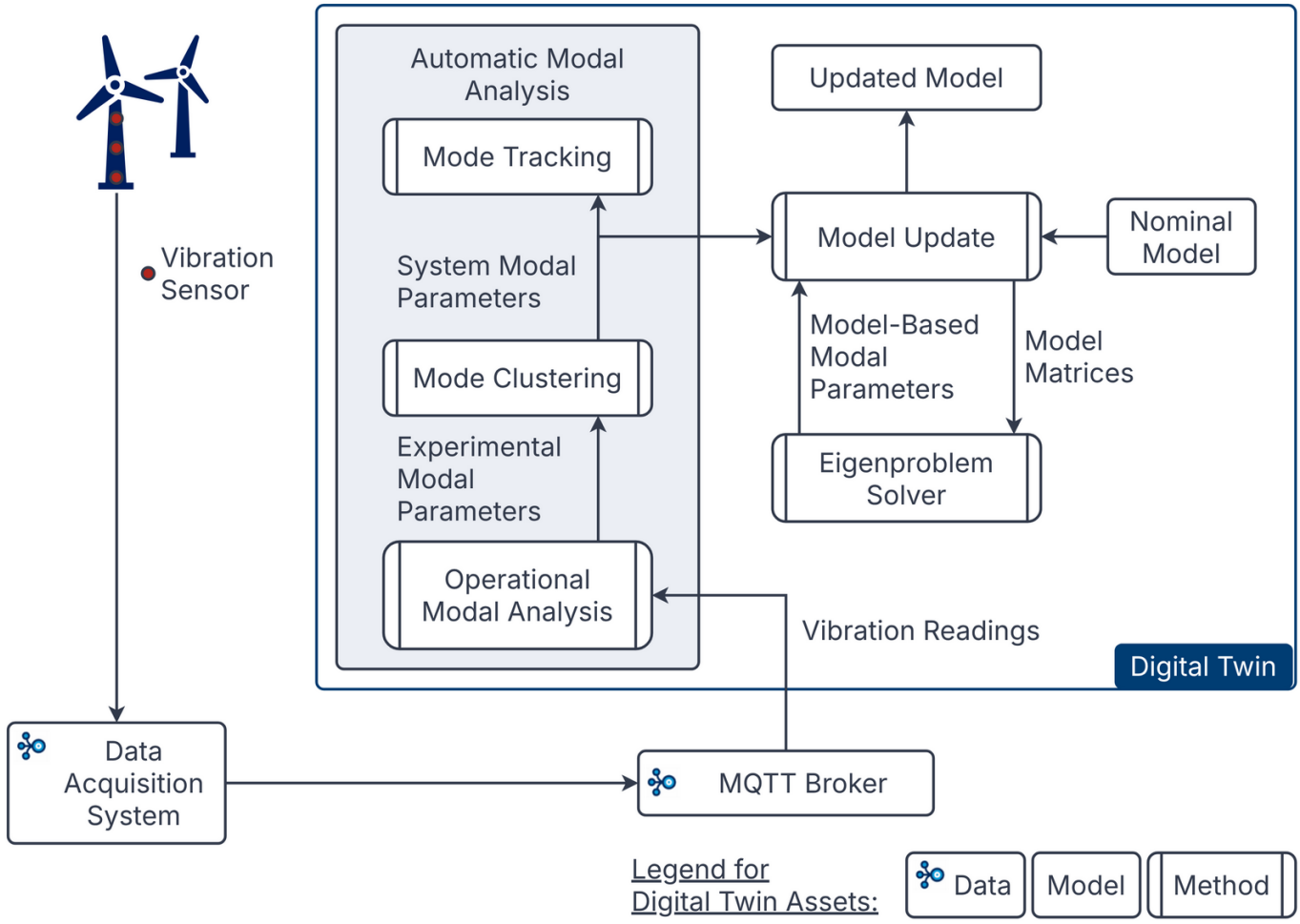
The source code for the models used in this DT is available in the mass spring damper GitHub repository.

Copyright © 2022 - 2026 The INTO-CPS Association

Structural Health Monitoring

OVERVIEW

This example applies digital twin methodology to structural health monitoring (SHM). The digital twin workflow shown below was developed for the SHM use cases.



The complete source code for this example is available online.

EXAMPLE STRUCTURE

This digital twin consists of:

- **SHM package**: implements the digital twin workflow. The package also contains data recorded from the physical twin and replay functions that stream this data into the digital twin workflow.
- **replay.json**: MQTT configuration file for replaying the recorded data

Copyright © 2022 - 2026 The INTO-CPS Association

The package provides these experimental scenarios:

- **replay** replays the recorded readings from the cantilever beam setup.
- **acceleration_readings** shows how the `Accelerometer` class extracts accelerometer measurements from an MQTT data stream.
- **aligning_readings** shows how the `Aligner` class collects and aligns accelerometer measurements from multiple MQTT data streams.
- **sysid** demonstrates `sysid` with four cases:

a. **sysid-and-plot**: plots natural frequencies.

b. **sysid-and-print**: prints sysid output to the console.

c. **sysid-and-publish**: publishes one set of sysid output via MQTT to the broker given in the `sysid` section of the config.

d. **live-sysid-publish**: publishes sysid output continuously via MQTT to the broker given in the `sysid` section of the config.

- **Clustering** demonstrates `clustering` with four cases:

a. **clustering-with-local-sysid**: gets the sysid output by running sysid locally, then runs mode clustering.

b. **clustering-with-remote-sysid**: gets the sysid output by subscribing, then runs mode clustering once.

c. **live-clustering-with-remote-sysid**: gets the sysid output by subscribing, then runs mode clustering in a loop.

d. **live-clustering-with-remote-sysid-and-publish**: gets the sysid output by subscribing, then runs mode clustering in a loop and publishes the cluster results.

- **mode-tracking** demonstrates `mode_tracking` with three cases:

a. **mode-tracking-with-local-sysid**: gets the sysid output by running sysid locally, then runs mode clustering and mode tracking.

b. **mode-tracking-with-remote-sysid**: gets the sysid output by subscribing, then runs mode clustering and mode tracking once.

c. **live-mode-tracking-with-remote-sysid**: gets the sysid output by subscribing, then runs mode clustering and mode tracking in a loop.

- **model-update** demonstrates `model_update` with three cases:

a. **model-update-with-local-sysid**: gets the sysid output, then uses it to update the model and get updated system parameters.

b. **live-model-update-with-remote-sysid**: gets the sysid output by subscribing to an MQTT topic, then runs mode clustering and uses the result to update the model and get updated system parameters.

c. **live-model-update-with-remote-clustering**: gets the mode clustering output by subscribing to an MQTT topic, then uses it to update the model and get updated system parameters.

The following scenarios can run in the automated execution mode:

- **acceleration_readings** shows how the `Accelerometer` class extracts accelerometer measurements from an MQTT data stream.
- **aligning_readings** shows how the `Aligner` class collects and aligns accelerometer measurements from multiple MQTT data streams.
- **sysid** prints the sysid output to the console.
- **Clustering** demonstrates `sysid` and `clustering`.
- **mode-tracking** demonstrates `sysid` and `mode_tracking`.
- **model-update** runs the digital twin workflow up to `model_update`.

CONFIGURATION

This example uses an MQTT broker to replay the data recorded from the physical twin. The configuration file is `replay.json`. Update the MQTT credentials in this file. If no local MQTT broker is available, you can use the Mosquitto test server.

Copyright © 2022 - 2026 The INTO-CPS Association

USE

Run the program without arguments to list the available experimental scenarios:

```
$example-shm
Usage: example-shm [OPTIONS] COMMAND [ARGS]...

Options:
  --config TEXT  Path to config file
  --help         Show this message and exit.

Commands:
  accelerometers
  align-readings
  clustering-with-local-sysid
  clustering-with-remote-sysid
  live-clustering-with-remote-sysid
  live-clustering-with-remote-sysid-and-publish
  live-mode-tracking-with-remote-sysid
  live-model-update-with-remote-clustering
  live-model-update-with-remote-sysid
  live-sysid-publish
  mode-tracking-with-local-sysid
  mode-tracking-with-remote-sysid
  model-update-with-local-sysid
  replay
  sysid-and-plot
  sysid-and-print
  sysid-and-publish
```

Run these examples with the `replay.json` config:

```
$example-shm --config replay.json <experiment-name>
Usage: example-shm --config replay.json accelerometers
```

Automated Execution

The pipeline of this example runs two lifecycle scripts: `lifecycle/sysid_print` for system identification and `lifecycle/model_update_local_sysid` for model update. A truncated sample output follows, with the execution timestamps removed.

```
Using docker image sha256:16f554efe9f9e182e2ec6b86ad37360138407a456a69982ed1b8261716ce5163 for python:3.12 with digest
$ cd digital_twins/shm
$ bash lifecycle/sysid_print
...

pyoma2.setup.base - INFO - Running SSIcovmm_mt... (base:123)
pyoma2.functions.ssi - INFO - Assembling Hankel matrix method: cov_mm... (ssi:82)
pyoma2.functions.ssi - INFO - ... uncertainty calculations... (ssi:94)
pyoma2.functions.ssi - INFO - SSI for increasing model order... (ssi:359)
JSON configuration loaded successfully.
on_connect: Connected with response code Success
Subscribing to topic: cpsens/recorded/1/metadata
on_subscribe: Subscription ID 1 with QoS levels [ReasonCode(Suback, 'Granted QoS 0')]
Extracted Fs from metadata: 256.0
on_connect: Connected with response code Success
Subscribing to topic: cpsens/recorded/1/data
on_subscribe: Subscription ID 1 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 2 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 3 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 4 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 5 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 6 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 7 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
on_subscribe: Subscription ID 8 with QoS levels [ReasonCode(Suback, 'Granted QoS 1')]
Waiting for data for 0.1 seconds
Waiting for data for 0.2 seconds
Waiting for data for 0.3 seconds
...

sysid parameters: {'freq_variance_treshold': 0.1, 'damp_variance_treshold': 1000000, 'Fs': 256, 'model_order_min': 2, 'model_order': 15, 'block_shift': 30,
'mstab': 6, 'tMAC': 0.95, 'bound_multiplier': 2, 'allignment_factor': [0.05, 0.01], 'phi_cri': 0.8, 'freq_cri': 0.2, 'obj_cri': 0.1, 'tMAC_MU': 0.7,
'modes_search_paring': 6, 'pars_to_update': ['k_rot', 'm'], 'MU_start_values': array([10.   ,  0.015]), 'MU_bounds': [(0.01, 1000), (0, 1000)], 'MU_modes':
[1, 2, 3]}

  0%|          | 0/16 [00:00<?, ?it/s]
100%|██████████| 16/16 [00:00<00:00, 23563.51it/s]
Cluster 0.4465797245502472 too short: 1 Must be: > 6
Cluster saved: 27.93849468231201
Updated parameters are:
k_rot: 12.04591179878987
m: 0.0
Model frequencies: [  4.13819299  26.68572155  80.22633861 164.67016633 180.37962416
  326.70816499] [Hz]
[DONE].
```


Copyright © 2022 - 2026 The INTO-CPS Association

REFERENCES

1. This example corresponds to commit ce47244 of the example-shm repository.
2. Talasila, P., Tcherniak, D., Jensen, A. M. D., Mahato, S., Schörghofer-Queiroz, A., Ulriksen, M. D., ... & Damkilde, L. (2025, July). Structural Health Monitoring of Engineering Structures Using Digital Twins: A Digital Twin Platform Approach. In International Conference on Experimental Vibration Analysis for Civil Engineering Structures (pp. 986-996). Cham: Springer Nature Switzerland.

Copyright © 2022 - 2026 The INTO-CPS Association

Heart Digital Twin

OVERVIEW

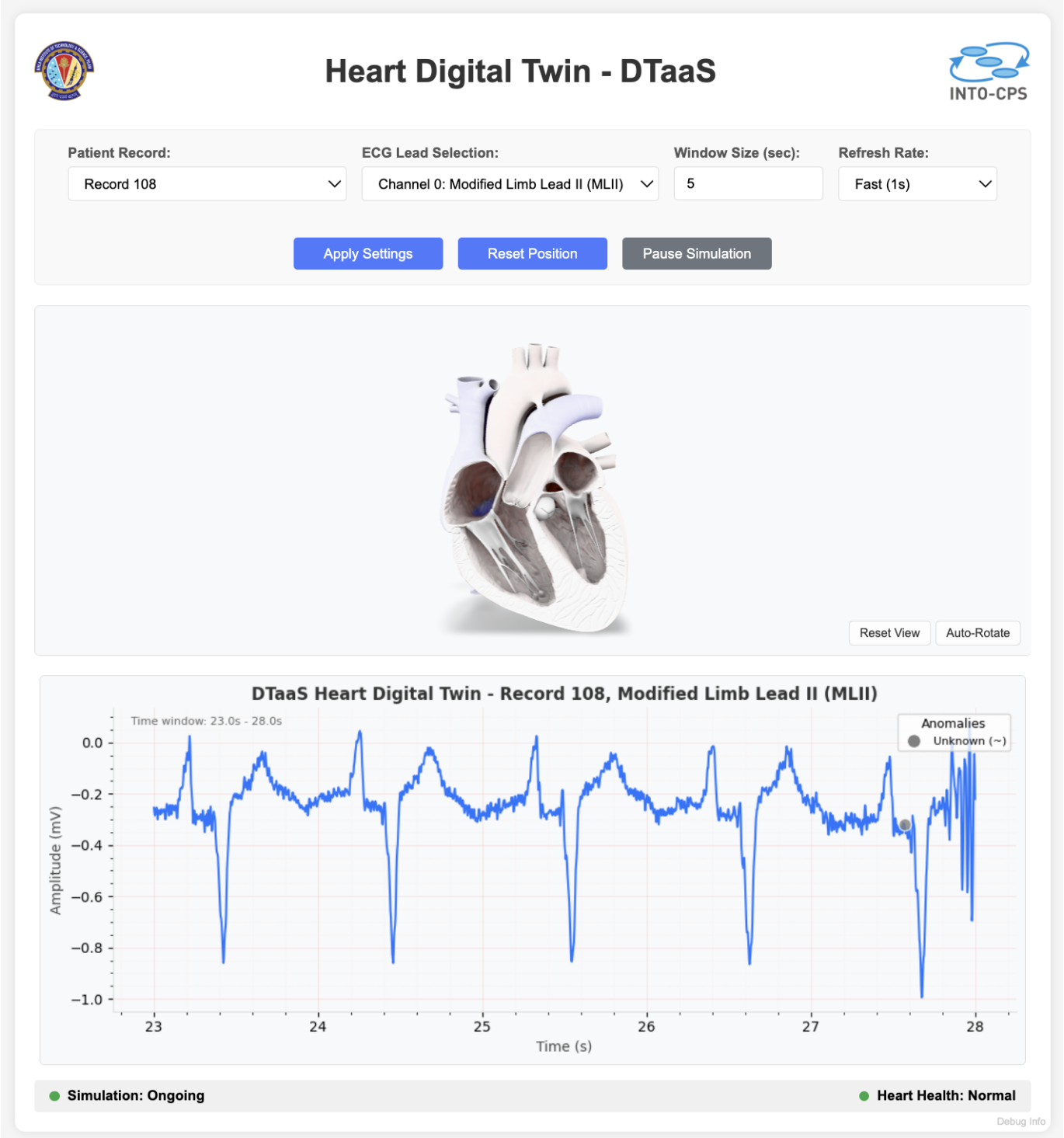
The Heart Digital Twin is a real-time cardiovascular monitoring system that uses ECG data from the MIT-BIH Arrhythmia Database. It shows live ECG plots with anomaly detection, alongside an interactive 3D heart model that beats in time with the cardiac rhythm.

EXAMPLE STRUCTURE

The Heart Digital Twin has these components:

- **ECG Data Processing**: Real-time processing and visualisation of electrocardiogram signals from the MIT-BIH database
- **3D Heart visualisation**: Interactive 3D heart model with synchronised pumping animation
- **Anomaly Detection**: Real-time identification and highlighting of cardiac anomalies
- **Web Interface**: Flask-based web application for real-time monitoring and control

Copyright © 2022 - 2026 The INTO-CPS Association



MIT-BIH ARRHYTHMIA DATABASE

This digital twin uses the **MIT-BIH Arrhythmia Database**, which contains 48 half-hour excerpts of two-channel ambulatory ECG recordings. The database includes:

- Records from 47 subjects studied by the BIH Arrhythmia Laboratory
- 360 Hz sampling frequency with 11-bit resolution
- Expert annotations for arrhythmia detection and classification
- Multiple ECG channels (MLII, V1, V2, V4, V5) depending on the record

Records 100-109 are available for simulation. Each contains different types of cardiac rhythms and anomalies.

Copyright © 2022 - 2026 The INTO-CPS Association

DIGITAL TWIN CONFIGURATION

This example uses the following assets:

Asset Type	Names of Assets	Visibility	Reuse in Other Examples
Data	MIT-BIH Arrhythmia Database Records (100-109)	Public	Yes
Models	3D Heart Model (Beating heart.glb)	Public	Yes
Functions	ECG Processing & Anomaly Detection	Public	Yes
Tools	Flask Web Server & UI Components	Public	Yes

You can customise the heart digital twin with these parameters:

- **Record Selection**: Choose from records 100-109
- **Channel Selection**: Switch between available ECG channels
- **Window Size**: Adjust the time window for real-time display
- **Update Frequency**: Control the refresh rate of the simulation

3D HEART ANIMATION

The digital twin includes a **synchronised 3D heart pumping animation**:

- **Real-time Synchronisation**: Heart model animation matches the ECG rhythm
- **Interactive Controls**:
- Camera controls for 360° viewing
- Auto-rotate functionality
- Reset view option
- **Animation Features**:
- Realistic beating heart model
- Smooth cardiac cycle animation
- Pause/resume functionality synchronised with ECG simulation
- **Model Format**: GLB (GL Transmission Format Binary) 3D model
- **Rendering**: WebGL-based rendering using model-viewer component

The 3D heart moves in step with the electrical activity shown in the ECG, which helps to relate the electrical signals to the mechanical function of the heart.

LIFECYCLE PHASES

Lifecycle Phase	Completed Tasks
Create	Sets up Python virtual environment and installs required dependencies (NumPy, Pandas, Matplotlib, WFDB, Flask)
Execute	Starts the Flask web server and launches the heart monitoring interface accessible via web browser
Clean	Terminates the running application and removes the virtual environment

DEVOPS AUTOMATION

This example uses the DevOps features of the DTaaS platform. GitLab DevOps pipelines run its lifecycle scripts.

Copyright © 2022 - 2026 The INTO-CPS Association

Pipeline Configuration

The `.gitlab-ci.yml` file sets the order in which the lifecycle scripts of the heart digital twin run. It defines two stages:

- **create**: Sets up the Python virtual environment and installs dependencies
- **execute**: Runs the heart digital twin application

Runner Requirements

This digital twin requires a **shell-based GitLab runner**. The application needs:

- Access to the host system's Python environment
- Ability to create and manage Python virtual environments
- File system access for model loading and data processing
- Network access for the web application

Docker-based runners may restrict the network or file system access that this digital twin needs.

Alternative Execution Mode: Docker

The heart digital twin can also run in a **Docker container**, still on a **shell-based GitLab runner**.

The GitLab CI file for this approach is `.gitlab-ci-docker.yml`.

To use it, change the root `.gitlab-ci.yml` to reference this Docker-based pipeline (`.gitlab-ci-docker.yml`) instead of the default one.

Artifacts and Outputs

The `create` stage saves the Python virtual environment as a GitLab pipeline artefact, which the `execute` stage reuses. The artefact expires after one hour.

RUN THE EXAMPLE

To run the example, go to the following directory:

```
cd /workspace/examples/digital_twins/heart_dt
```

If required, make the lifecycle scripts executable:

```
chmod +x lifecycle/create
chmod +x lifecycle/execute
chmod +x lifecycle/clean
```

Now, run the following scripts:

Create

Sets up the Python environment and installs the required dependencies, including WFDB for MIT-BIH data access, Flask for the web interface, and scientific computing libraries.

```
lifecycle/create
```

Execute

Starts the Heart Digital Twin web application in the background. Open it in a web browser for real-time ECG monitoring and 3D heart visualisation.

```
lifecycle/execute
```


Copyright © 2022 - 2026 The INTO-CPS Association

The application is available at `http://localhost:5001`. On start-up it:

- Loads the required MIT-BIH records
- Initialises the ECG data processing
- Starts real-time visualisation
- Enables 3D heart model synchronisation

Examine the results

Use the web interface to:

- **Monitor Real-time ECG**: View live electrocardiogram signals
- **Control Simulation**: Select different records, channels, and update frequencies
- **Interact with 3D Heart**: Rotate, zoom, and control the heart animation
- **Detect Anomalies**: Observe highlighted cardiac irregularities
- **Pause/Resume**: Control the simulation flow

The application writes its logs to the `logs/heart_dt.log` file.

Clean

Stops the running application and cleans up the environment.

```
lifecycle/clean
```

This script:

- Stops the Flask web server
- Removes the Python virtual environment

REFERENCES

More information about the MIT-BIH Arrhythmia Database and cardiac monitoring:


```
Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database.
IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001).
```

```
Goldberger, A., et al. PhysioBank, PhysioToolkit, and PhysioNet:
Components of a new research resource for complex physiologic signals.
Circulation 101(23):e215-e220, 2000.
```


The MIT-BIH Arrhythmia Database is available at: https://physionet.org/content/mitdb/

Copyright © 2022 - 2026 The INTO-CPS Association

Heart Digital Twin - PaaS Edition

OVERVIEW

This example runs the Heart Digital Twin in a **Platform-as-a-Service (PaaS)** deployment mode. This version is meant for cloud deployment and supports more than one deployment strategy, including Render.com and GitLab CI/CD with GitHub mirroring.

DEVOPS AUTOMATION

Setup Instructions

1. **Connect GitHub Repository to Render.com**:
2. Sign up for a Render.com account
3. Connect the GitHub repository
4. **Automatic Deployment**:
5. Render.com deploys automatically when it detects changes in the GitHub repository

The `.gitlab-ci.yml` file provides a mirror-to-GitHub pipeline that:

- Syncs changes from GitLab to a GitHub repository
- Keeps GitHub-based deployments, such as Render.com, up to date
- Requires the GitLab CI/CD variables `GITHUB_USERNAME` and `GITHUB_TOKEN`

REFERENCES

See the Heart Digital Twin documentation for details of the underlying cardiovascular monitoring functionality.

Copyright © 2022 - 2026 The INTO-CPS Association

4. Admin

4.1 Installation Steps

There are two ways to install the DTaaS platform:

1. **DTaaS CLI (recommended).** The `dtaas` command generates, validates, installs and operates a deployment for every supported scenario. The CLI installation guide covers the full workflow.
2. **Manual scenario packages (advanced).** A docker compose package for each scenario, edited by hand. Choose this path only when a scenario needs customisation beyond what `dtaas.toml` offers, or when you want to study the deployment internals. Start at the manual installation scenarios.

4.1.1 Platform Services (Optional)

To install additional data and infrastructure services (MQTT, InfluxDB, Grafana and others), use the Platform Services CLI.

4.1.2 Manual Installation Scenarios (Advanced)

> **Note**
>
> These packages are the manual counterpart of what `dtaas deployment generate` produces. The CLI performs the steps below automatically. Prefer it unless you need to modify the deployment beyond its configuration options.

Start by selecting a scenario package.

DTaaS

DTaaS scenarios are complete platform deployments that bundle the web client, gateway, user workspaces and core services. Choose a scenario by target environment, security needs and GitLab setup. Each scenario has its own Install and Config pages.

Scenario	Purpose
localhost	Single-user DTaaS package over HTTP
localhost on portainer	GUI-based localhost deployment with Portainer for single-users
secure server	Compatibility package for secure server installs
secure server and GitLab	Multi-user DTaaS package with integrated GitLab

Workspace

Workspace scenarios cover user workspace access, identity provider setup and route-level protection. Choose localhost for a quick start with Dex, or secure server for a production-style deployment with Keycloak and OIDC. Each scenario has Install, Configuration and identity provider pages.

Scenario	Purpose
localhost	Single-user workspace deployment with Dex
secure server	Multi-user workspace deployment with Keycloak

Other

- Vagrant

Copyright © 2022 - 2026 The INTO-CPS Association

After selecting a scenario package, follow these steps:

1. Configure OAuth and Client Settings

- DTaaS packages:
- Frontend OAuth setup: client auth
- Backend OAuth setup (forward-auth): servers auth
- Client runtime settings: client config
- Workspace Dex localhost:
- Configure `.env` and `config/dex-config.yaml`
- Workspace Keycloak secure server:
- Follow package guides in workspace config and Keycloak config

2. Configure Filesystem and Certificates

For DTaaS or workspace server deployments:

- Create user workspaces under `files/`
- Provide TLS certificates under `certs/` (secure server packages)
- Create `config/.env` from its example and update it
- Create `config/client.js` from its example and update it
- For the workspace Dex localhost package, create `config/dex-config.yaml` from its example (no changes needed)

3. Start Services

Run the following command from the package directory:

```
docker compose --env-file config/.env up -d
```

4. Validate and Iterate

- Confirm the login flow in a browser
- Check the workspace routes (`/user1` , `/user2`)
- Check service logs and health

4.1.3 Independent Packages

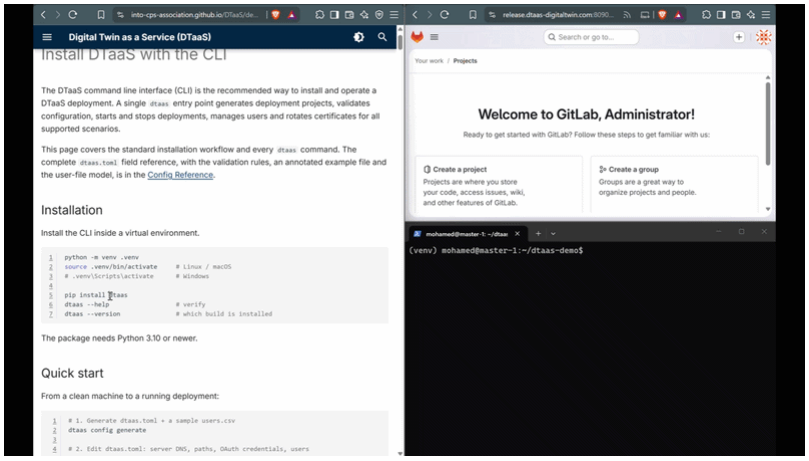
Reusable package details are listed on packages.

4.1.4 Walkthrough Videos

These walkthroughs help when selecting or validating an installation scenario. Where a `4x` recording exists, the preview and MP4 link below use it.

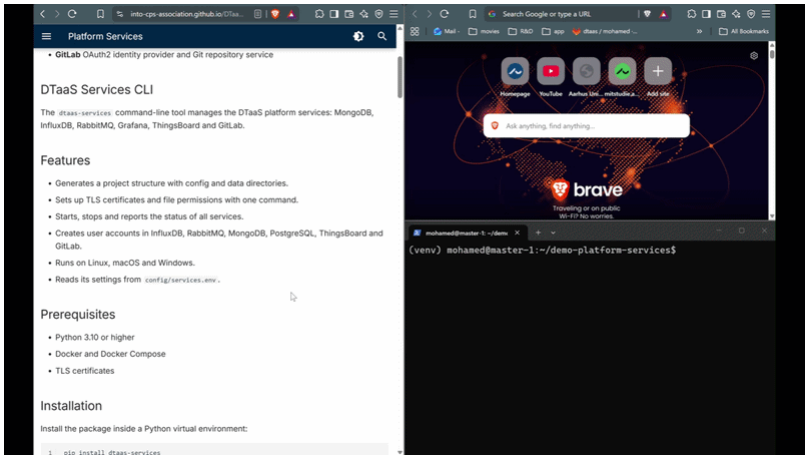
Copyright © 2022 - 2026 The INTO-CPS Association

DTaaS CLI



Open MP4

DTaaS Platform Services CLI



Open MP4

4.1.5 Administration

- DTaaS CLI
- GitLab server guidance
- GitLab runner guidance

Copyright © 2022 - 2026 The INTO-CPS Association

4.2 Auto Installation

4.2.1 Install DTaaS with the CLI

The DTaaS command line interface (CLI) is the recommended way to install and operate a DTaaS deployment. A single `dtaas` entry point generates deployment projects, validates configuration, starts and stops deployments, manages users and rotates certificates for all supported scenarios.

This page covers the standard installation workflow and every `dtaas` command. The complete `dtaas.toml` field reference, with the validation rules, an annotated example file and the user-file model, is in the Config Reference.

Installation

Install the CLI inside a virtual environment.

```
python -m venv .venv
source .venv/bin/activate     # Linux / macOS
# .venv\Scripts\activate      # Windows

pip install dtaas
dtaas --help                  # verify
dtaas --version               # which build is installed
```

The package needs Python 3.10 or newer.

Quick start

From a clean machine to a running deployment:

```
# 1. Generate dtaas.toml + a sample users.csv
dtaas config generate

# 2. Edit dtaas.toml: server DNS, paths, OAuth credentials, users

# 3. Validate; fix any reported errors before continuing
dtaas config validate

# 4. Generate deployment files for your chosen scenario
dtaas deployment generate --type secure-server

# 5. Bring the deployment up
dtaas platform install
```

Tear it down again with `dtaas platform uninstall`. Add `--remove-user-files` to also delete the per-user workspace directories. That flag asks for confirmation; pass `--yes` to skip the prompt.

DEPLOYMENT TYPES

`--type`	When to use	Support level
`localhost`	Local development or demo, single user	dev/demo only
`insecure-server`	Multi-user HTTP demo	insecure/demo only
`secure-server`	Multi-user HTTPS	**production-supported**
`secure-server-gitlab`	HTTPS with bundled GitLab	**production-supported**
`workspace-localhost`	Workspace with Dex on localhost	dev/demo only
`workspace-secure-server`	Workspace with Keycloak	**production-supported**

Copyright © 2022 - 2026 The INTO-CPS Association

> **Warning**
>
> Types marked *dev/demo only* or *insecure/demo only* run over plain HTTP with default or static credentials. **Do not expose them to the internet or shared networks.** Production-supported types still require the manual hardening steps documented in the `README.md` and configuration notes (`CONFIG.md`, or `CONFIGURATION.md` for `workspace-secure-server`) shipped with each generated project.

The generated packages match the manual installation scenarios. The CLI fills in the configuration and file structure that the manual path asks you to prepare by hand.

Configuration

The CLI reads all its settings from `dtaas.toml`. It looks for the file in `--output-dir` first and then in the current working directory, so one top-level `dtaas.toml` can serve a deployment generated into a subdirectory.

The main sections are:

- `[common]`: the server DNS name and the absolute path of the DTaaS installation directory.
- `[common.security]`: the `tls` flag and `certs-src`, the source directory for TLS certificates.
- `[common.resources]`: default resource limits for user workspace containers (`cpus`, `mem_limit`, `pids_limit`, `shm_size`), with a `set_limits` flag that turns enforcement off. Adjust them to the host capacity and your tenancy policy.
- `[[users]]`: the starting users, one record each.
- `[gitlab]`: optional GitLab account provisioning for `user add` (see GitLab provisioning).
- `[frontend]`: the OAuth application of the DTaaS web client, for the server types.
- One section per deployment type, such as `[secure-server]`, which holds the OAuth credentials for that type.

Run `dtaas config validate` after every edit. It reports all problems at once; see Validation Rules for the full list.

Copyright © 2022 - 2026 The INTO-CPS Association

ABRIDGED CONFIGURATION

An abridged `dtaas.toml`:

```
name="Digital Twin as a Service (DTaaS)"
version="2.2.1"
owner="The INTO-CPS-Association"
git-repo="https://github.com/into-cps-association/DTaaS.git"

[common]
server-dns="localhost"
# Absolute path to the DTaaS installation. Separate it with / or \\, never a
# single \, which fails with "Invalid unicode value".
path="/home/XXX/DTaaS"
# Windows example
#path="C:\\Users\\XXX\\DTaaS"

[common.security]
tls=true
certs-src='/etc/letsencrypt/archive/intocps.org'

[common.resources]
# Enforce the per-user limits below; false creates users with no CPU, memory
# or process caps and ignores the fields.
set_limits=true
cpus=4 # Virtual CPU
mem_limit="4G"
pids_limit=4960 # Number of processes
shm_size="512m" # Shared memory size

[[users]]
username="username1"
email="username1@intocps.org"
groups=["default","dtaas"]
load_balance=true

[gitlab]
provision=false
api_url="https://gitlab.com"
ssl_verify=true
#pat="glpat-xxxxxxxxxxxxxxxxxxx"

[frontend]
react-app-client-id="your_client_id_here"
react-app-oauth-url="https://gitlab.com"
...
```

Notes

- Edits to `dtaas.toml` affect only user containers created after the change.
- To apply new limits to existing containers, recreate or restart the user containers, for example by deleting and re-adding the user with the CLI or by restarting the container with Docker Compose.
- Give memory and shared memory values a unit (`M`, `G`).

Commands

Every command follows a `dtaas <noun> <verb>` grammar with four nouns, listed in workflow order:

Noun	Acts on
`config`	`dtaas.toml` and the user registry
`deployment`	The generated deployment files
`platform`	The installed core services, as one unit
`user`	Additional users, one at a time

The `config`, `deployment` and `platform` commands accept `--output-dir` (default `.`). The `user` commands always act on the current directory, so run them from the deployment directory.

If a command fails because a path is root-owned (for example TLS certificates under `/etc/letsencrypt`), the CLI suggests re-running it as `sudo -E env PATH="$PATH" dtaas <command>`.

Copyright © 2022 - 2026 The INTO-CPS Association

`CONFIG GENERATE` / `CONFIG VALIDATE`

```
dtaas config generate    # writes dtaas.toml + a sample users.csv
dtaas config validate    # checks an existing dtaas.toml
```

`generate` writes `dtaas.toml` and a sample `users.csv` (bulk input for `user add --file`) into `--output-dir`. `--force` overwrites an existing `dtaas.toml`. `validate` reports every problem in the file at once; see Validation Rules.

Options:

Option	Default	Description
`--output-dir PATH`	`.`	For `generate`: target directory (created if missing). For `validate`: search location
`--force`	off	(`generate` only) Overwrite an existing `dtaas.toml`

`CONFIG RECONCILE`

Reports drift between `dtaas.users.registry.json` (the users who **should** be provisioned) and the live `compose.users.yml` services (the users who **are** provisioned):

```
dtaas config reconcile
```

It lists users in these categories:

- **missing**: registered but not provisioned.
- **unexpected**: provisioned but not in the registry. This can come from a manual edit or a partial delete, so investigate it.
- **drifted**: provisioned, but the live config no longer matches what `.dtaas.state.json` recorded at the last provisioning.
- **absent**: provisioned (the user has a compose service), but the container is gone, for example after an interrupted `user add`.
- **desired-status drift**: provisioned, but the live container state does not match the user's registry `desired_status` (for example `desired 'paused' but container is 'running'`).

When everything matches, it prints `In sync: no drift detected.` If Docker is unreachable, the command warns that it could not verify container state and reports only the drift it can read from the compose file. It never reports `In sync` in that case.

Without `--fix`, the command is read-only. With `--fix`, it reprovisions **missing** and **drifted** users, restarts the containers of **absent** users, and then pauses, stops or starts every provisioned user to match its `desired_status`:

```
dtaas config reconcile --fix
```

`--fix` runs the same code as `dtaas user add`, so it acts on the current directory. Combining it with an `--output-dir` other than the current directory is rejected. It also refuses to run when Docker is unreachable.

`--fix` never touches **unexpected** services, because removing something that is running should be a deliberate action. Use `dtaas user delete` for those.

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory to inspect
`--fix`	off	Reprovision missing/drifted registry users, and enforce desired status, after reporting

Copyright © 2022 - 2026 The INTO-CPS Association

`DEPLOYMENT GENERATE`

Copies the project structure for one deployment scenario (`docker-compose.yml` , config examples and supporting files) into a target directory. It also writes the user-workspace templates and the `files/template/` skeleton, and, for the types that run the logger, creates `logs/logger/` (see `platform install`).

```
dtaas deployment generate --type <name>
```

Options:

Option	Default	Description
`--type NAME`	*(required)*	Deployment scenario (see Deployment Types)
`--output-dir PATH`	`.`	Target directory (created if missing)
`--force`	off	Overwrite files that already exist

Examples:

```
# Localhost demo in the current directory
dtaas deployment generate --type localhost

# Production HTTPS server in a subdirectory
dtaas deployment generate --type secure-server --output-dir ./my-server

# Regenerate, overwriting existing files
dtaas deployment generate --type insecure-server --output-dir ./demo --force
```

Generated files besides the scenario's compose tree:

Item	Purpose
`users.server.yml`	Docker Compose template for HTTP deployments
`users.server.secure.yml`	Docker Compose template for HTTPS/TLS deployments
`users.resources.yml`	Per-user resource-limit overlay, merged in when `set_limits` is true
`files/template/`	Skeleton copied into each new user workspace

When `dtaas.toml` is present, its values are substituted into the generated files (dotenv files, the client config and, for TLS types, the `certs/` directory). See Configuration Substitution for details. If `dtaas.toml` is not found, the command prints a note and the generated files keep their placeholder values.

> **Verify the Docker image tag**
>
> `users.server.yml` and `users.server.secure.yml` pin a workspace image tag (e.g. `intocps/workspace:main-56c6f68`). Check Docker Hub and update the tag to a current, stable version before deploying.

`PLATFORM INSTALL`

Starts a generated deployment.

```
dtaas platform install
```

The command runs `docker compose up -d` against the `docker-compose.yml` in the installation directory. Before that, it makes sure a workspace directory exists for every `[[users]]` record, creating any missing one from `files/template/` , and sets its ownership to `1000:100` .

For the `localhost` , `insecure-server` , `secure-server` and `secure-server-gitlab` types, it also creates `logs/logger/` , where the logger microservice writes its events, and gives it to `1000:1000` , the user that runs the logger. This is the logger step of the manual

Copyright © 2022 - 2026 The INTO-CPS Association

installation guides, so an installation done with the CLI needs no extra step. Only `logs/logger/` changes owner: `secure-server-gitlab` also mounts `logs/` into GitLab, so `logs/` itself is left alone.

Both ownership changes use `sudo` on Linux and macOS and are skipped on Windows. Before `sudo` runs, the command says what it is for, since `sudo` may ask for your password. `logs/logger/` is left alone when it already belongs to `1000:1000`, for example when you are that user. When a change fails, for example because `sudo` is not available, the command prints a warning with the commands to run by hand and carries on:

```
Setting the owner of logs/logger/ to 1000:1000 with sudo (it may ask for your password)
Warning: could not set the owner of logs/logger/
  Run: sudo chown -R 1000:1000 -- /home/admin/dtaas/logs/logger
```

When `logs/` or `logs/logger/` is a symbolic link, the command does not change its owner and prints a warning instead.

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory containing the generated deployment

Because the CLI looks for `dtaas.toml` in `--output-dir` first and then in the current working directory, a single top-level `dtaas.toml` can serve a deployment generated into a subdirectory:

```
dtaas platform install --output-dir ./insecure
```

The command fails with an error if `docker-compose.yml` is missing, `dtaas.toml` cannot be found or the Docker daemon is unreachable.

`PLATFORM UNINSTALL`

Tears the whole installation down, stopping and removing containers and networks.

```
dtaas platform uninstall
```

Containers of users added with `user add` run as a separate Compose project. The command removes them first so that they do not hold the shared network open. **Per-user workspace files are kept by default.**

To also delete the per-user workspace directories **and** the CLI-owned `dtaas.users.registry.json` and `.dtaas.state.json`:

```
dtaas platform uninstall --remove-user-files
```

This is destructive, so the command asks for confirmation. Skip the prompt in non-interactive scripts with `--yes`:

```
dtaas platform uninstall --remove-user-files --yes
```

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory
`--remove-user-files`	off	Also delete per-user workspace dirs + registry/state files
`--yes` / `-y`	off	Skip the confirmation prompt for `--remove-user-files`

> **Note**
>
> `--remove-user-files` removes only the per-user directories inside `<output-dir>/files/`. It keeps `files/common/` and `files/template/`, so a later `platform install` can recreate the user directories. It refuses to follow a symlinked `files/`. Double-check `--output-dir` before using this flag.

Copyright © 2022 - 2026 The INTO-CPS Association

LIFECYCLE OPERATIONS

These commands work on an **installed** deployment. `status` reports its state; `stop` / `start` and `pause` / `resume` suspend and resume it. None of them remove containers or networks; `uninstall` does that. They exit `0` on success, including when nothing is installed, and non-zero on failure, so CI and ops scripts can call them safely.

`platform stop`, `start`, `pause` and `resume` act on the **core services only** and never touch per-user containers. Suspend or resume additional users with `dtaas user stop` / `pause` / `resume`. `platform status` still reports the whole installation, core services **and** user containers.

Lifecycle command matrix:

Command	`docker compose` verb	Effect	Containers kept?	Reverse with
`platform install`	`up -d`	Create and start every service	n/a	`platform stop` / `platform uninstall`
`platform status`	`ps` (read-only)	Report per-service state; no change	n/a	n/a
`platform stop`	`stop`	Terminate the processes, keep the containers	yes	`platform start`
`platform start`	`start`	Start previously stopped containers	yes	`platform stop`
`platform pause`	`pause`	Freeze the processes (memory preserved)	yes	`platform resume`
`platform resume`	`unpause`	Thaw previously paused processes	yes	n/a
`platform uninstall`	`down`	Stop **and remove** containers and networks	no	`platform install`

`stop` vs `pause`

`stop` sends `SIGTERM` / `SIGKILL`: the processes end, and a restart runs them from scratch (reverse with `platform start`). `pause` uses the kernel cgroup freezer: the processes are suspended in place with their memory intact and resume instantly (reverse with `platform resume`), but a paused container still holds its resources. Use `stop` to free CPU and `pause` for a short suspension. `pause` expects running containers and fails if the deployment is already stopped.

`platform status`

Reports the state of every service, for both the core services and the user containers.

```
dtaas platform status
```

```
PROJECT     SERVICE           STATE
deployment  traefik           running
deployment  client            running
deployment  gitlab            not created
users       alice             paused
```

The state is one of `running`, `paused`, `stopped` (a terminated container, which Docker calls `exited`), `restarting` or `not created` (a service defined in `docker-compose.yml` that has no container yet).

Copyright © 2022 - 2026 The INTO-CPS Association

For automation, `--json` prints the same records as JSON:

```
dtaas platform status --json
```

```
[
  {"project": "deployment", "service": "traefik", "state": "running"}
]
```

For the user containers alone, use `dtaas user status`.

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory
`--json`	off	Emit machine-readable JSON instead of the table

`platform stop` / `platform start`

`stop` stops the core services with `docker compose stop`; `start` brings the stopped containers back with `docker compose start`. Containers and networks are **kept**, so `stop` is not an uninstall.

```
dtaas platform stop
dtaas platform start
```

After `stop`, the command notes how many per-user containers are still running and suggests `dtaas user stop --all`. Both commands print `There is no existing DTaaS / Workspace installation` and exit `0` when nothing is installed (no containers in any state), so they are safe to call repeatedly.

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory

`platform pause` / `platform resume`

`pause` freezes the running core-service containers in place with `docker compose pause`; `resume` thaws them with `docker compose unpause`. Memory is preserved and resuming is almost instant.

```
dtaas platform pause
dtaas platform resume
```

Like `stop`, `pause` notes any per-user containers still running. Both commands report the absent installation and exit `0` when nothing is installed (no containers in any state).

Options:

Option	Default	Description
`--output-dir PATH`	`.`	Installation directory

`PLATFORM UPDATE --CERTS`

Replaces the TLS certificates of a running deployment in place, without regenerating the project or copying files by hand.

```
dtaas platform update --certs
```


Copyright © 2022 - 2026 The INTO-CPS Association

The command reads `[common.security].certs-src` from `dtaas.toml`, then:

1. **Validates** the new certificate pair (parseable, private key matches the certificate, no expired intermediates).
2. **Stops** the `traefik` service to release open file handles.
3. **Swaps** the validated files into `<output-dir>/certs/`, backing up the live pair and restoring it on any failure.
4. **Restricts** the private key to `0600` (POSIX; prints a warning on Windows).
5. **Restarts** `traefik` and waits for it to come back up.

If validation fails, the live certificates are left untouched. The command is safe to run repeatedly.

Options:

Option	Default	Description
`--certs`	*(required)*	Refresh the deployment's TLS certificates
`--output-dir PATH`	`.`	Installation directory

`PLATFORM UPDATE --CONFIG`

Re-applies the values in `dtaas.toml` to the service config files of an installed deployment without regenerating the project.

```
dtaas platform update --config
```

The command runs the same substitution as `deployment generate`. If any file changed, it recreates all deployment services with `docker compose up -d --force-recreate`. It detects the deployment type from `docker-compose.yml`.

```
# Preview changes without writing or restarting
dtaas platform update --config --dry-run

# Apply changes and restart
dtaas platform update --config

# Update a deployment in a subdirectory
dtaas platform update --config --output-dir ./my-server
```

`--config` validates `dtaas.toml` before changing anything and refuses to apply an invalid file. A second run with no `dtaas.toml` changes reports `No configuration changes` and restarts nothing. If secrets are still template placeholders after the update, the command lists them.

Options:

Option	Default	Description
`--config`	*(required)*	Re-apply `dtaas.toml` to installed service config files
`--dry-run`	off	Report what would change without writing or restarting
`--output-dir PATH`	`.`	Installation directory

`--certs` and `--config` can be combined in one invocation; at least one of them is required.

User management

User management spans three files, each with a single owner; see User Files for the full model.

- **`dtaas.users.registry.json`** is the CLI-owned record of every *additional* user (added after installation with `user add`), including their `desired_status` (`running` / `paused` / `stopped`). Only `user add`, `delete`, `pause`, `stop` and `resume` change it; do not edit it by hand.
- **`.dtaas.state.json`** is a git-ignored, disposable cache of what is running: per-user container id, status and a config hash, refreshed on every add and delete. It is safe to delete; the CLI rebuilds it on the next provisioning run. `config reconcile` compares it against the registry to detect drift.

Copyright © 2022 - 2026 The INTO-CPS Association

ADD USERS

The starting users an instance is installed with are declared in *dtaas.toml* as `[[users]]` records (see Abridged Configuration). You edit them once at installation time, and the CLI never rewrites them.

Users added later are **not** written to *dtaas.toml*. Add them with `dtaas user add`, either one at a time:

```
dtaas user add --email alice@intocps.org --group dtaas --load-balance alice
```

`--group` is repeatable, not comma-separated. Pass it once per group to add a user to several groups:

```
dtaas user add --email alice@intocps.org --group dtaas --group testers alice
```

or in bulk from a CSV file (`dtaas config generate` writes a sample `users.csv` next to `dtaas.toml`):

```
dtaas user add --file users.csv
```

```
username,email,groups,load_balance,password
alice,alice@intocps.org,additional,true,
bob,bob@intocps.org,additional;beta-testers;load-balanced-tier1,false,
```

`groups` is a `;`-separated list and `load_balance` is `true` or `false`. The `password` column is used only with GitLab provisioning (see below). Give either a `USERNAME` or `--file`, not both. A bare `dtaas user add` with neither is rejected, so it cannot reprovision the whole registry by accident.

Both forms merge the users into the CLI-owned `dtaas.users.registry.json` with `desired_status` set to `running`. **Only the newly added users are started**; users already running are left alone, so adding one user never recreates the others. A username already in the registry is reported (`'alice' already exists, skipping`) and never added twice or overwritten. A username declared in `dtaas.toml`'s `[[users]]` is reported as a starting user, whose workspace `dtaas.toml` manages; with GitLab provisioning on, naming one creates its GitLab account (see below). A run that adds nobody ends with `No new users added.` and exits `0`.

To reprovision **every** registry user at once (for example after `compose.users.yml` was lost), use `dtaas config reconcile --fix`.

Options:

Option	Default	Description
`USERNAME`	—	Add one user (requires `--email`, except for a `dtaas.toml` starting user)
`--file PATH` / `-f`	—	Bulk-add users from a CSV
`--email TEXT`	—	Email for `USERNAME` (enables forward-auth routing); read from `[[users]]` for a starting user, and another address given for one is reported and not applied
`--group TEXT`	`additional`	Group tag for `USERNAME`; repeat the flag for multiple groups
`--load-balance` / `--no-load-balance`	on	Mark `USERNAME` for load balancing
`--password TEXT`	—	Initial GitLab password for `USERNAME`; only used when GitLab provisioning is enabled (see below). It wins over `[[users]].password`. Prefer the `users.csv` `password` column or the interactive prompt, because a command-line value is visible in shell history and the process list

For each registry user, the command copies `files/template/` into `files/<username>`, creating the directory when it is missing, and sets its ownership to `1000:100` where it has the permission to; without that permission it leaves the ownership as it is and carries on.

Copyright © 2022 - 2026 The INTO-CPS Association

When a user has an *email*, the CLI adds the matching traefik-forward-auth routing rule to `config/conf.server`, so you do not need to edit `conf.server` by hand. A user added from a `users.csv` row with an empty `email` cell gets no rule. Restart the container for the change to take effect:

```
docker compose --env-file config/.env up -d --force-recreate traefik-forward-auth
```

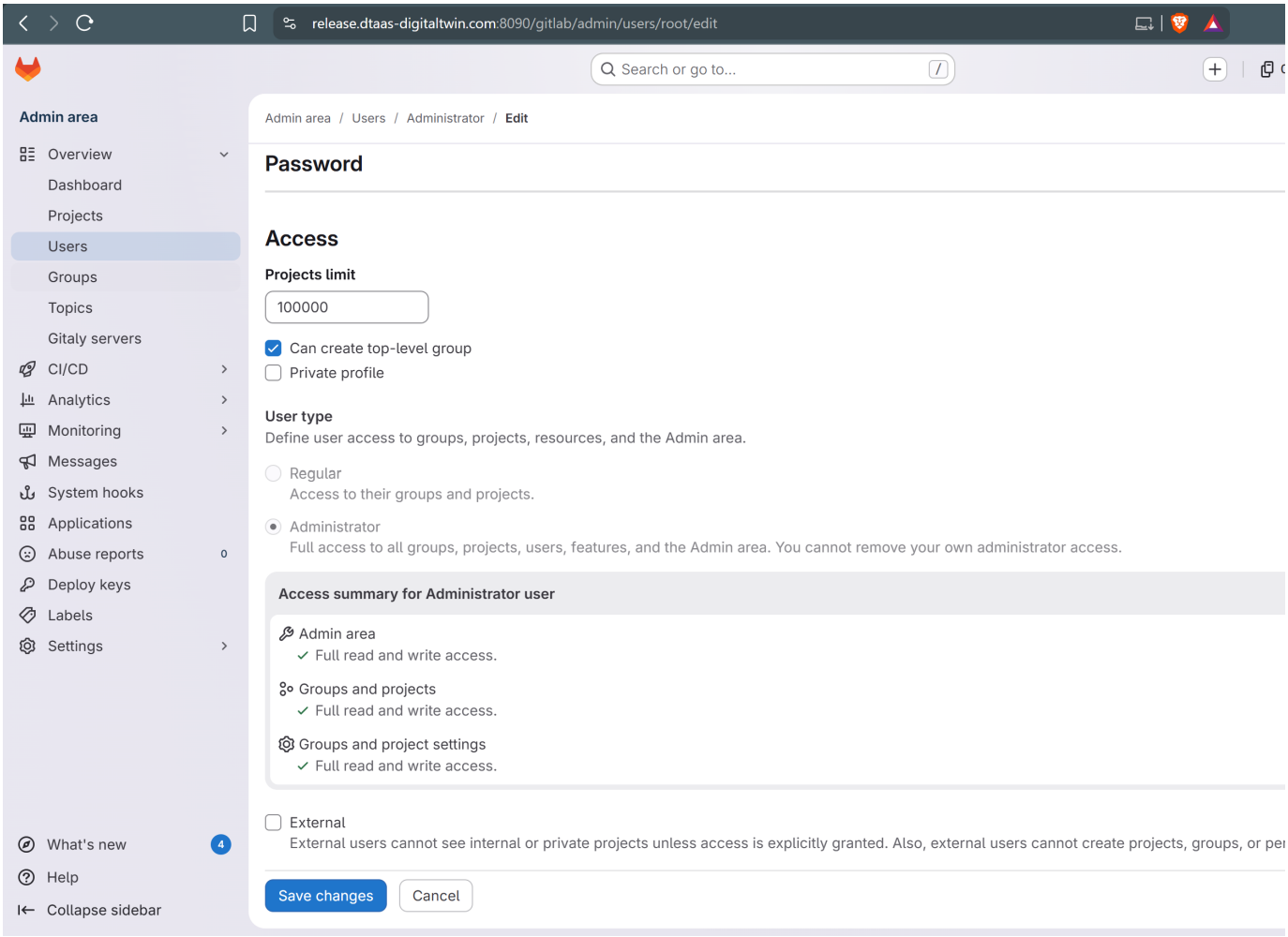
GitLab provisioning (optional)

When `[gitlab].provision = true` in `dtaas.toml` (off by default), `user add` also creates a GitLab account for each user it adds and issues that account a Personal Access Token (PAT). The containers are provisioned first; a GitLab failure never undoes them, but it does make the command exit non-zero. The `[gitlab]` keys, the password sources and the files written are described in GitLab provisioning settings.

Set up the admin token

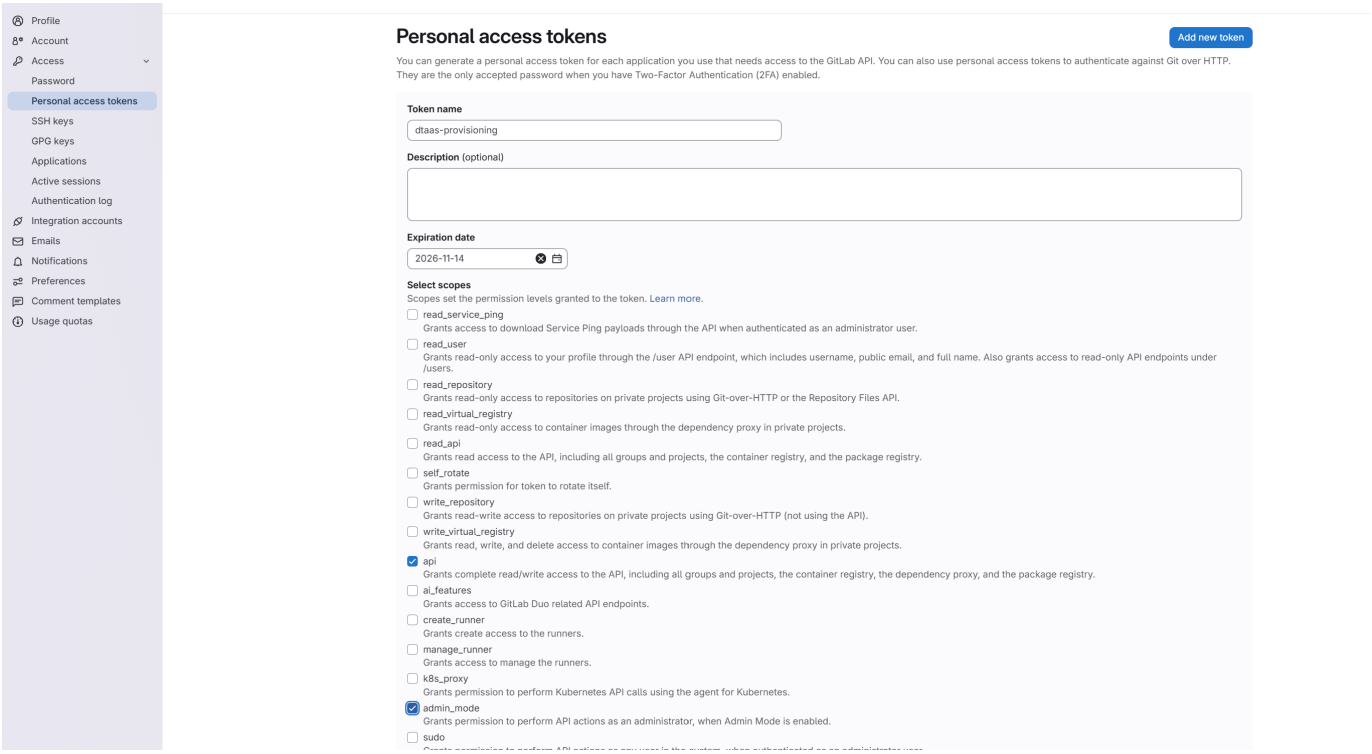
Provisioning needs an admin token, set up once per installation by a GitLab administrator:

1. Use an administrator account. In **Admin area > Users**, click **Edit** on the account and check that **User type** is **Administrator**.



2. Create its token, signed in as that account, under **User settings > Access > Personal access tokens** (**Add new token**). Tick the `api` scope, and `admin_mode` too if the instance has Admin Mode enabled, then export the token where you run `dtaas`:

Copyright © 2022 - 2026 The INTO-CPS Association




```
export DTAAS_GITLAB_PAT="glpat-xxxxxxxxxxxxxxxxxxx"
```

Choose an expiry you will rotate, because provisioning stops working the day the token lapses. When `DTAAS_GITLAB_PAT` and `[gitlab].pat` hold different tokens, the exported one is used and the CLI prints `Note: using the admin token from DTAAS_GITLAB_PAT.`

Scenarios

Each scenario shows what `dtaas user add` prints. The output of `docker compose` while containers start is left out.

Add one user. Without `--password`, the CLI asks for the initial GitLab password with hidden input, twice:

```
$ dtaas user add --email alice@intocps.org alice
GitLab password for 'alice':
Repeat for confirmation:
GitLab account and token created for 'alice'.
  The token is in gitlab_user_tokens.json.
Users added successfully
```

Add users from a CSV. Fill in the `password` column of `users.csv`, then make the file readable by you alone:

```
username,email,groups,load_balance,password
alice,alice@intocps.org,additional,true,Kq7v-Rm3x-Zt9w
bob,bob@intocps.org,additional;beta-testers,false,Wd4p-Hs8n-Qc2y
```

```
$ chmod 600 users.csv
$ dtaas user add --file users.csv
GitLab account and token created for 'alice'.
  The token is in gitlab_user_tokens.json.
GitLab account and token created for 'bob'.
  The token is in gitlab_user_tokens.json.
Users added successfully
```


Copyright © 2022 - 2026 The INTO-CPS Association

A starting user. Naming a `[[users]]` user of `dtaas.toml` creates only their GitLab account and token. Their workspace already runs from `docker-compose.yml`, their email is the one in `dtaas.toml`, and their password is `[[users]].password` or, when that is unset, the prompt:

```
$ dtaas user add username1
'username1' is a dtaas.toml starting user.
  Its workspace is managed there, not by 'user add'.
GitLab account and token created for 'username1'.
  The token is in gitlab_user_tokens.json.
No new users added.
```

An `--email` that differs from the one in `dtaas.toml` is not applied, and the run says so:

```
$ dtaas user add --email other@intocps.org username1
Warning: the email given for 'username1' is not used.
  dtaas.toml declares username1@intocps.org for this starting user, and that is the address GitLab gets.
...
```

A user with no password. When a named user has no password and no account from an earlier run, the whole run is refused before anything is registered or started. Fill in the `password` column and run the same command again:

```
$ dtaas user add --file users.csv
Error: No GitLab password for 'bob'.
  [gitlab].provision is enabled in dtaas.toml, so a new user needs an initial GitLab password: add a 'password' column to their users.csv row (and chmod 600
the file), pass --password for a single user, or set [[users]].password for a dtaas.toml starting user.
```

A password GitLab refuses. GitLab refuses a password that is too short or built from common words, for example one that contains `pass` or the username. The user's container is up, but their account is not created. The line under `GitLab account creation failed.` is GitLab's own reason, for example:

```
$ dtaas user add --file users.csv
GitLab account and token created for 'alice'.
  The token is in gitlab_user_tokens.json.
GitLab provisioning failed for 'bob'.
  GitLab account creation failed.
    Failed to create user 'bob': 400: {'password': ['must not contain commonly used combinations of words and letters']}
Error: Error while adding users
  GitLab provisioning failed for: bob
    Their containers were still provisioned.
```

Retry after a failure. Fix the cause (here, bob's password in `users.csv`) and run the same command again. Both users are already registered, so their containers are left alone; alice already has a token and is skipped, and only bob's account is created:

```
$ dtaas user add --file users.csv
'alice' already exists, skipping
'bob' already exists, skipping
GitLab account step skipped for 'alice'.
  A Personal Access Token was already issued on an earlier run (see gitlab_user_tokens.json). A re-run does not reissue one.
GitLab account and token created for 'bob'.
  The token is in gitlab_user_tokens.json.
No new users added.
```

A retry never leaves a second live token for a user, so repeating the command after a partial failure is always safe. When the account was created but its token was not saved, the retry first revokes any token named `dtaas` that an earlier run issued, then issues a new one to that same account (the registry keeps its id) instead of creating the account again. No password is needed for that, so naming the user again is enough:

```
$ dtaas user add --email alice@intocps.org alice
'alice' already exists, skipping
GitLab token issued for 'alice'.
  The token is in gitlab_user_tokens.json.
No new users added.
```

A token that is lost or revoked is replaced in GitLab directly.

A token that cannot be saved. Before creating any account, `user add` checks `gitlab_user_tokens.json`. A leftover `gitlab_user_tokens.json.tmp` from an interrupted run is removed with a warning;

Copyright © 2022 - 2026 The INTO-CPS Association

a file that is not a JSON object, or cannot be written, stops the GitLab step for every user. Should a save still fail after a token was issued (a full disk, for example), that token is revoked at once, and no further token is issued in that run:

```
GitLab token for 'alice' not saved; revoked.
  [Errno 28] No space left on device
  Once tokens can be saved to gitlab_user_tokens.json again, run 'dtaas user add alice' to issue a new one.
GitLab provisioning stopped for: bob
  Tokens cannot be saved to gitlab_user_tokens.json, so no further token is issued this run.
Error: Error while adding users
  GitLab provisioning failed for: alice, bob
    Their containers were still provisioned.
```

If the token cannot be revoked either, the first line reads `GitLab token for 'alice' not saved, still valid.` and says to revoke it in GitLab.

An account that already exists. When the username, or the email, is already taken on GitLab, the account is left as it is and no token is issued: this run did not create it and does not know its credentials. The container is still provisioned, and the command fails. Delete the account in GitLab, or use another username or email, then run the same command again:

```
$ dtaas user add --email carol@intocps.org carol
GitLab password for 'carol':
Repeat for confirmation:
GitLab provisioning failed for 'carol'.
  GitLab account already exists; no token issued.
    The username is already registered, or its email is already on another account, and this CLI did not create it, so its credentials are unknown. Delete
that account in GitLab, or use another username or email, then run 'user add' again. GitLab said: 409: Username has already been taken
Error: Error while adding users
  GitLab provisioning failed for: carol
    Their containers were still provisioned.
```

No admin token. Without `DTAAS_GITLAB_PAT` or `[gitlab].pat`, the containers are still provisioned, the GitLab step is skipped for everyone, and the command fails. Export the token and run the same command again:

```
$ dtaas user add --email alice@intocps.org alice
GitLab password for 'alice':
Repeat for confirmation:
GitLab provisioning skipped for every user.
  No GitLab PAT configured for provisioning.
    Export DTAAS_GITLAB_PAT, or set [gitlab].pat in dtaas.toml.
Error: Error while adding users
  GitLab provisioning failed for: alice
    Their containers were still provisioned.
```

A token GitLab rejects. When GitLab answers `401` (an invalid or expired token) or `403` (a token whose owner is not an administrator, or that lacks a scope), each user's failure ends with a hint on the token:

```
GitLab provisioning failed for 'alice'.
  GitLab account creation failed.
    Failed to create user 'alice': 403: 403 Forbidden
    Check DTAAS_GITLAB_PAT (or [gitlab].pat): creating users and managing their tokens needs a non-expired personal access token with the 'api' scope, owned
by an administrator account, and the 'admin_mode' scope as well on an instance with Admin Mode enabled.
```

What the user receives

Each issued token is named `dtaas`, has the `read_repository` and `write_repository` scopes only, and expires after 365 days. It is saved to `gitlab_user_tokens.json` the moment it is issued; see Files written for that file and the registry markers. Hand each user their token, which they use to clone and push their repositories, and their initial password.

Copyright © 2022 - 2026 The INTO-CPS Association

Resource limits (optional)

By default each user container gets the CPU, memory, process and shared-memory caps from `[common.resources]`, merged in from the `users.resources.yml` overlay. To add users without any caps, set `set_limits = false` in that section:

```
# Constrained users (default): limits enforced
[common.resources]
set_limits = true
cpus       = 4
mem_limit  = "4G"
pids_limit = 4960
shm_size   = "512m"
```

```
# Unconstrained users: no caps written, limit fields optional
[common.resources]
set_limits = false
```

The CLI reads the flag on every `user add`, so one deployment can host both constrained and unconstrained users if you change `set_limits` between runs.

DELETE USERS

Pass one or more usernames to `dtaas user delete`:

```
dtaas user delete alice bob
```

or delete in bulk from the same CSV format (only the `username` column is used):

```
dtaas user delete --file users.csv
```

Give either `USERNAMES` or `--file`; one of them is required. Add `--dry-run` to see which users would be removed without deleting anything:

```
dtaas user delete alice bob --dry-run
```

Each deleted user is deprovisioned (container stopped, compose service and forward-auth rule removed) and dropped from `dtaas.users.registry.json`. Users who are not provisioned are reported and skipped, but they are still removed from the registry.

The CLI removes the traefik-forward-auth routing rules of deleted users from `config/conf.server`. Restart the container for the change to take effect:

```
docker compose --env-file config/.env up -d --force-recreate traefik-forward-auth
```

> **The GitLab account stays**
>
> `user delete` removes the user from the DTaaS installation and revokes the Personal Access Tokens named `dtaas` that the CLI issued to them, so they lose repository access. Their GitLab account stays, so the user can still sign in there. A revoked token is dropped from `gitlab_user_tokens.json`. Revoking uses the same admin token as `user add`, even when `[gitlab].provision` is now off. To remove an account, a GitLab administrator deletes it in GitLab. Do that before adding the same username again: `user add` cannot issue a token for an account it did not create.
>
> A `dtaas.toml` starting user is handled the same way on GitLab: its token is revoked and its markers are cleared. Its workspace is one of the installation's own, defined in `docker-compose.yml`, so `user delete` keeps its container and forward-auth rule.

```
GitLab token revoked for 'alice'.
  The GitLab account stays. Delete it in GitLab before adding the same username again.
Users deleted successfully
```


Copyright © 2022 - 2026 The INTO-CPS Association

When a token cannot be revoked (GitLab unreachable, admin token missing or refused), the rest of the delete still completes, the token stays in `gitlab_user_tokens.json`, and the command fails naming the user. Revoke that token in GitLab:

```
GitLab token not revoked for 'alice'.
  No GitLab PAT configured for provisioning.
    Export DTAAS_GITLAB_PAT, or set [gitlab].pat in dtaas.toml.
  The token is still valid and stays in gitlab_user_tokens.json; revoke it in GitLab.
Error: Error while deleting users
  GitLab tokens not revoked for: alice
    Their workspaces were still deleted.
```

Options:

Option	Default	Description
`USERNAMES`	—	One or more usernames to remove
`--file PATH` / `-f`	—	Bulk-delete users listed in a CSV (only the `username` column is used)
`--dry-run`	off	Preview the removal without making any changes

`USER PAUSE` / `STOP` / `RESUME`

Suspend or resume **specific additional users** without touching the rest of the installation. Target one or more `USERNAMES`, a `--file` / `-f users.csv` (only the `username` column is read, as with `user delete`), or every additional user with `--all`.

```
dtaas user pause alice bob
dtaas user stop alice
dtaas user resume alice bob
dtaas user pause --file users.csv
dtaas user stop --all
```

Command	`docker compose` verb	Effect	Reverse with
`user pause`	`pause`	Freeze the named users' containers (memory preserved)	`user resume`
`user stop`	`stop`	Terminate the named users' containers, keep them	`user resume`
`user resume`	`unpause` or `start`, as needed	Thaw a paused user, or restart a stopped one	—

Each command also records a `desired_status` (`"paused"` / `"stopped"` / `"running"`) in `dtaas.users.registry.json` for the users it acted on. This makes the suspension durable: a later `dtaas user add` or `dtaas config reconcile --fix` checks `desired_status` and does **not** restart a user you paused or stopped. Only `user resume` (or editing the registry by hand) sets it back to `"running"`.

Only additional users can be targeted. Naming a `dtaas.toml` starting user is an error. To suspend or resume the whole installation, starting users included, use `dtaas platform pause` / `stop` / `resume` together with `--all` here.

A username that is not in the registry, or is registered but not provisioned (for example because `user add` never ran for it), is reported and skipped, and the rest of the batch continues:

```
'carol' is not a registered user, skipping
alice, bob paused successfully
```

Options:

Option	Default	Description
`USERNAMES`	—	One or more usernames to target
`--file PATH` / `-f`	—	Bulk-target users listed in a CSV (only the `username` column is used)
`--all`	off	Target every additional user; cannot be combined with `USERNAMES` or `--file`

Copyright © 2022 - 2026 The INTO-CPS Association

USER STATUS

Reports the state of every additional user's container, or of one named user. It is the per-user part of `platform status`, using the same table and `--json` format.

```
dtaas user status
dtaas user status alice
dtaas user status --json
```

A username that is not in `dtaas.users.registry.json` is rejected, so a typo is not mistaken for a stopped user. A registered user without a container is reported as not provisioned. The command exits `0` whenever it can read the deployment.

Options:

Option	Default	Description
`USERNAME`	—	Report only this user
`--json`	off	Emit machine-readable JSON instead of the table

ADDITIONAL POINTS TO REMEMBER

- `user add` writes every registry user into `compose.users.yml` but starts only the users added in that run. Users whose `desired_status` is not `running` are never started.
- `config reconcile --fix` with an empty registry does nothing.
- Usernames may contain `.`, `_` and `-` and must start with a letter or digit. Whitespace, path separators and shell metacharacters are rejected.
- `user add` does not enable AuthMS authentication.

KNOWN LIMITATIONS

- Validation accepts usernames that contain `.`, but `.` is a special character in Docker Compose labels, so containers for such users can fail to provision. A future release will replace `.` with `-` or `_` in the generated labels.

Copyright © 2022 - 2026 The INTO-CPS Association

4.2.2 CLI Configuration Reference

This page is the reference for `dtaas.toml`, the configuration file that every DTaaS CLI command reads. Generate a template with `dtaas config generate`, fill it in, and check it with `dtaas config validate` before running any other command. The installation workflow is covered in Install with DTaaS CLI.

Which sections does my deployment need?

SERVER DEPLOYMENTS

Section	`localhost`	`insecure-server`	`secure-server`	`secure-server-gitlab`
`[common]`	Required	Required	Required	Required
`[common.security]`	Not used	Optional (`tls = false`)	Required	Required
`[common.resources]`	Optional	Optional	Optional	Optional
`[gitlab]`	Optional	Optional	Optional	Optional
`[[users]]`	Optional	Required	Required	Required
`[frontend]`	Not used	Required	Required	Required
`[localhost]`	Required	Not used	Not used	Not used
`[insecure-server]`	Not used	Required	Not used	Not used
`[secure-server]`	Not used	Not used	Required	Not used
`[secure-server-gitlab]`	Not used	Not used	Not used	Required

`dtaas user add` reads `[common.security].tls` for every deployment type to choose the per-user compose template. For `insecure-server`, leave the section out or set `tls = false`; `tls = true` gives the new users the HTTPS template, which the HTTP-only deployment does not serve.

WORKSPACE DEPLOYMENTS

Section	`workspace-localhost`	`workspace-secure-server`
`[common]`	Required	Required
`[common.security]`	Not used	Required
`[common.resources]`	Optional	Optional
`[gitlab]`	Optional	Optional
`[[users]]`	Optional	Required
`[workspace-localhost]`	Required	Not used
`[workspace-secure-server]`	Not used	Required

Copyright © 2022 - 2026 The INTO-CPS Association

Annotated `dtaas.toml`

The file below shows every key with inline comments. Copy it as a starting point and delete the sections that your deployment type does not use (see the tables above).

Copyright © 2022 - 2026 The INTO-CPS Association

```
# ── Metadata (top level) ──────────────────────────────────────────────
# Describes the instance. git-repo, the URL of the DTaaS repository, is
# required for all deployment types; validate checks no other key here.
name     = "Digital Twin as a Service (DTaaS)"
version  = "2.2.1"
owner    = "The INTO-CPS-Association"
git-repo = "https://github.com/into-cps-association/DTaaS.git"

# ── Common settings (all deployment types) ────────────────────────────
[common]
# Public hostname of the server. Use "localhost" for local deployments,
# a fully-qualified domain name (e.g. "dtaas.example.com") for servers,
# or a bare IP address.
server-dns = "dtaas.example.com"

# Absolute path to the DTaaS installation directory on the deployment host.
# Must exist before running validate.
path = "/opt/dtaas"

# ── TLS settings (required for: secure-server, secure-server-gitlab,
#                               workspace-secure-server) ──────────────
[common.security]
tls = true

# Absolute path to the directory containing fullchain.pem and privkey.pem.
# Used by deployment generate (seeds certs/) and platform update --certs.
certs-src = "/etc/letsencrypt/live/dtaas.example.com"

# ── Per-user container resource limits (optional, all types) ──────────
[common.resources]
# Enforce the limits below. Set to false to add users without any caps.
# The 4 fields are then optional and ignored. Defaults to true when omitted.
set_limits = true
cpus       = 4        # CPU cores; may be fractional, e.g. 0.5
mem_limit  = "4G"     # memory limit unit required: G, m, k …
pids_limit = 4960     # maximum number of processes per container (integer)
shm_size   = "512m"   # shared memory unit required

# ── Starting users (all deployment types) ─────────────────────────────
# One self-contained [[users]] block per user, hand-edited once at install
# time. Presence in this file is the desired state there are no add/delete
# lists. Additional users added later with `dtaas user add` live in the
# CLI-owned dtaas.users.registry.json instead.
# Usernames must match GitLab accounts and be unique across the array.
#
# email enables traefik-forward-auth routing rules automatically;
# groups/load_balance carry per-user tags. password is optional: it is the
# initial GitLab password when `dtaas user add <username>` provisions this
# starting user's account. Avoid committing a real secret here; prefer
# supplying it at runtime instead.
[[users]]
username     = "alice"
email        = "alice@example.com"
groups       = ["default", "dtaas"]
load_balance = true

[[users]]
username     = "bob"
email        = "bob@example.com"
groups       = ["default", "dtaas"]
load_balance = false
# password   = "Kq7v-Rm3x-Zt9w"   # initial GitLab password, see above

# ── React web client OAuth app (insecure-server, secure-server,
#                                secure-server-gitlab) ─────────────────
# This is a SEPARATE OAuth application from the traefik-forward-auth app
# configured in [insecure-server] / [secure-server] below.
# Redirect URI: https://<server-dns>/Library
# Scopes: api read_user read_repository openid profile   Type: not Confidential
[frontend]
react-app-client-id = "dtaas-client"
react-app-oauth-url = "https://gitlab.example.com"

# ── GitLab user provisioning (optional, all types) ────────────────────
# Off by default. When provision = true, `dtaas user add` also creates
# each new user's GitLab account and a Personal Access Token (see
# "Add Users" in cli.md). The provisioning token must belong to an
# administrator and carry the api scope (plus admin_mode on an instance with
# Admin Mode enabled); supply it via the DTAAS_GITLAB_PAT environment variable
# rather than committing it here. A [gitlab].pat key is the fallback when the
# variable is unset or empty. ssl_verify may be a CA-bundle path for a GitLab
# behind an internal CA; false disables verification (a warning is printed)
# and true (default) uses the system trust store.
[gitlab]
provision  = false
api_url    = "https://gitlab.example.com"
# pat      = "glpat-xxxxxxxxxxxxxxxxxxxx"   # prefer DTAAS_GITLAB_PAT
# ssl_verify = "/etc/ssl/certs/corp-ca.pem" # or true (default) / false

# ── localhost deployment (dev / demo only) ────────────────────────────
# GitLab OAuth app for the web client; auth-authority is the GitLab URL.
[localhost]
default-user   = "alice"
client-id      = "dtaas-local"
```


Copyright © 2022 - 2026 The INTO-CPS Association

```
auth-authority = "https://gitlab.example.com"

# ── insecure-server deployment (HTTP, demo only not internet-facing) ──────────
# GitLab OAuth app for traefik-forward-auth.
# Redirect URI: http://<server-dns>/_oauth
# Scopes: openid profile read_user   Type: Confidential
[insecure-server]
oauth-url           = "https://gitlab.example.com"
oauth-client-id     = "abc123"
oauth-client-secret = "s3cr3t"
oauth-secret        = "random-signing-string"   # random; used to sign session cookies

# ── secure-server deployment (HTTPS/TLS production-ready) ─────────────────────
# Same GitLab OAuth app as insecure-server, with Redirect URI using https.
[secure-server]
oauth-url           = "https://gitlab.example.com"
oauth-client-id     = "abc123"
oauth-client-secret = "s3cr3t"
oauth-secret        = "random-signing-string"

# ── secure-server-gitlab deployment (bundled GitLab production-ready) ────────
# oauth-url points at the bundled GitLab, served under /gitlab.
[secure-server-gitlab]
oauth-url           = "https://dtaas.example.com/gitlab"
oauth-client-id     = "abc123"
oauth-client-secret = "s3cr3t"
oauth-secret        = "random-signing-string"

# ── workspace-localhost deployment (Dex on localhost dev / demo only) ────────
[workspace-localhost]
default-user   = "alice"
client-id      = "workspace-local"
auth-authority = "http://localhost:5556/dex"

# ── workspace-secure-server deployment (Keycloak production-ready) ──────────
[workspace-secure-server]
keycloak-admin          = "admin"
keycloak-admin-password = "change-me"
keycloak-realm          = "dtaas"
keycloak-issuer-url     = "https://keycloak.example.com/realms/dtaas"
keycloak-client-id      = "workspace"
keycloak-client-secret  = "s3cr3t"
oauth-secret            = "random-signing-string"
# Not written to config/client.js (auth-authority is only checked to be
# a URL). Set REACT_APP_CLIENT_ID and REACT_APP_AUTH_AUTHORITY there by hand.
client-id               = "dtaas-frontend"
auth-authority          = "https://keycloak.example.com/realms/dtaas"
```


Copyright © 2022 - 2026 The INTO-CPS Association

Validation rules

`dtaas config validate` reads `dtaas.toml` (from `--output-dir` first, then the current directory) and reports all problems at once:

Field	Rule
`git-repo`	Must be an `http(s)` URL
`[common].server-dns`	Must be `localhost`, an IP, or a fully qualified hostname
`[common].path`	Must be an absolute path to an existing directory
`[common.security].certs-src`	When present, must be an absolute path to an existing directory
`[common.resources].set_limits`	`false` turns the limits off; any other value, or no key, keeps them on
`[common.resources].cpus`	Positive number (e.g. `4` or `0.5`)
`[common.resources].pids_limit`	Integer
`[common.resources].mem_limit`, `shm_size`	Byte size with required unit (e.g. `4G`, `512m`)
`[[users]]`	When present, must be an array of tables; usernames must be unique
`[[users]].username`	Required, valid username
`[[users]].email`	Required, valid RFC 5321/5322 address (no DNS lookup)
`[[users]].groups`	When present, must be a list of strings
`[[users]].load_balance`	When present, must be `true` or `false`
`[[users]].password`	When present, must be a string (the starting user's initial GitLab password, used when `user add` names them)
`[gitlab].provision`	When present, `true` or `false` (default `false`)
`[gitlab].api_url`	Must be an `http(s)` URL; **required** when `provision` is `true`, optional otherwise
`[gitlab].pat`	When present, must be non-empty (remove the key to use `DTAAS_GITLAB_PAT`)
`[gitlab].ssl_verify`	When present, `true`, `false`, or a CA-bundle path string
Deployment-section URLs	When present, must be `http(s)` URLs
Deployment-section `default-user`	When present, must be a valid username

Deployment-section URLs are `react-app-oauth-url`, `oauth-url`, `auth-authority` and `keycloak-issuer-url` in `[frontend]`, `[localhost]`, `[insecure-server]`, `[secure-server]`, `[secure-server-gitlab]`, `[workspace-localhost]` and `[workspace-secure-server]`. Each is checked only when its section is present. `config validate` checks every section in the file, while `platform update --config` checks only the section of the installed deployment type, plus `[frontend]`.

For `workspace-secure-server`, no command writes `client-id` or `auth-authority` into `config/client.js`; `auth-authority` is only checked to be a URL. Edit `REACT_APP_CLIENT_ID` and `REACT_APP_AUTH_AUTHORITY` in that file by hand.

`path` and `certs-src` are checked against the local filesystem, so run `validate` on the deployment host. If a check fails with a permission error (for example, `certs-src` under the root-owned `/etc/letsencrypt`), the CLI prints the command to re-run with `sudo`.

The `[common.resources]` limit fields (`cpus`, `pids_limit`, `mem_limit`, `shm_size`) are required only when `set_limits` is `true`, the default. With `set_limits = false` they are optional and ignored, but any value that is present is still validated.

Copyright © 2022 - 2026 The INTO-CPS Association

Configuration substitution

When `dtaas.toml` is present, `deployment generate` substitutes its values into the generated files. `platform update --config` repeats the same substitution on an installed deployment. Each `--type` reads from its matching top-level section in `dtaas.toml`. Depending on the type, values go into the dotenv files (`config/.env`, `config/conf.server`), the React client config (`config/client.js`, or `config/env.local.js` for `workspace-localhost`) and the Dex config (`config/dex-config.yaml` for `workspace-localhost`).

For the server types, the `[frontend]` section supplies `REACT_APP_CLIENT_ID` and `REACT_APP_AUTH_AUTHORITY` for the DTaaS web client. That is a separate OAuth application from the traefik-forward-auth credentials in `[insecure-server]`, `[secure-server]` and `[secure-server-gitlab]`. The redirect URIs in `config/client.js` are built from `[common].server-dns`, and the first two `[[users]]` records fill `USERNAME1` and `USERNAME2`.

If `dtaas.toml` is not found, the command prints a note and the generated files keep their placeholder values.

TLS CERTIFICATE PLACEMENT

For the TLS types (`secure-server`, `secure-server-gitlab`, `workspace-secure-server`), `deployment generate` also fills the `certs/` directory in the output. It reads `[common.security].certs-src` from `dtaas.toml` and copies the latest `fullchain.pem` and `privkey.pem` from there.

User files

User management spans three files, each with a single owner. The split follows the one Terraform makes between `.tf` and `terraform.tfstate` files:

File	Owner	Contents	Git
`dtaas.toml` `[[users]]`	Human, at install time	**Starting** users: one self-contained record per user (`username`, `email`, `groups`, `load_balance`, optional `password`)	Hand-edited; keep out of Git, as it can hold credentials
`dtaas.users.registry.json`	CLI (`user add` / `delete` / `pause` / `stop` / `resume`)	**Additional** users under `users`: the same fields, plus `desired_status` (`running` / `paused` / `stopped`) and any GitLab markers. Its `starting_users` section holds the GitLab markers alone, for starting users that `user add` has provisioned on GitLab	Tracked, CLI-written, never hand-edited
`.dtaas.state.json`	CLI, at provisioning time	Observed runtime facts: container id, status, provisioned-at, config hash	Ignored, runtime cache

- **`dtaas.toml`** is written by a person and never rewritten by the CLI, so its comments and reviewed values stay as they are.
- **`dtaas.users.registry.json`** is a database that the CLI owns and updates atomically, the way `useradd` owns `/etc/passwd`. Change its users with `dtaas user add --file users.csv`, `delete`, `pause`, `stop` and `resume`, not by hand. The `users.csv` written by `dtaas config generate` is the bulk input for `add` and `delete`. `desired_status` is `running` for a user who has never been paused or stopped, and `user add` and `config reconcile --fix` do not start any user whose `desired_status` is something else.
- **`.dtaas.state.json`** is a disposable cache of what is running, refreshed on every add and delete. It is git-ignored and safe to delete.

CHECKING FOR DRIFT

`dtaas config reconcile` compares `dtaas.users.registry.json` (the users who **should** be provisioned) with the live `compose.users.yml` services (the users who **are** provisioned). It reports **missing**, **unexpected**, **drifted**, **absent** and **desired-status** drift; see `config reconcile` for what each means. When everything matches, it prints `In sync: no drift detected.`

Without `--fix`, the command is read-only. `--fix` reprovisions missing and drifted users, restarts absent users' containers, and pauses, stops or starts every provisioned user to match its `desired_status`. It never touches **unexpected** services; remove those with `dtaas user delete`.

Copyright © 2022 - 2026 The INTO-CPS Association

Pausing or stopping a user with `dtaas user pause` or `stop` is recorded as that user's `desired_status`. A later `reconcile` treats the suspension as the desired state instead of reporting it as drift.

GitLab provisioning settings

The `[gitlab]` section turns on GitLab account provisioning for `dtaas user add`. For what the command does and prints in each case, see GitLab provisioning.

```
[gitlab]
provision  = true
api_url    = "https://gitlab.example.com"
# pat      = "glpat-xxxxxxxxxxxxxxxxxxxx"   # prefer DTAAS_GITLAB_PAT
# ssl_verify = "/etc/ssl/certs/corp-ca.pem" # or true (default) / false
```

Key	Default	Meaning
`provision`	`false`	`true` makes `user add` create each user's GitLab account and token
`api_url`	none	The GitLab instance; required when `provision` is `true`
`pat`	none	The admin token, used only when `DTAAS_GITLAB_PAT` is unset or empty
`ssl_verify`	`true`	`true` uses the system trust store, a path uses that CA bundle, `false` turns verification off

ADMIN TOKEN

The CLI uses `DTAAS_GITLAB_PAT` whenever it is exported and not empty, and `[gitlab].pat` otherwise, so a rotated token needs no edit to a `dtaas.toml` the whole installation shares. When both are set to different tokens, a `Note:` line says the exported one is used. The token must belong to a GitLab administrator and carry the `api` scope, plus `admin_mode` on an instance with Admin Mode enabled.

For a self-hosted GitLab behind an internal CA, point `ssl_verify` at the CA bundle rather than turning it off. With `ssl_verify = false`, the admin token and every user's initial password cross the network without certificate checks, and the CLI prints a warning on each run.

INITIAL PASSWORDS

Each new account needs an initial password. The CLI takes the first one it finds:

1. `--password` on the command line, or the `password` column of the `users.csv` given with `--file`.
2. `[[users]].password` in `dtaas.toml`, for a starting user.
3. A hidden prompt, asked twice, when a single user is added.

A password is used once, to create the account. The CLI never writes it to `dtaas.users.registry.json`, `.dtaas.state.json` or a log. A run in which a named user has no password and no account from an earlier run is refused before anything changes. `--password` is visible in shell history and in the process list, so prefer the prompt, or the CSV column with `chmod 600 users.csv` (the CLI sets that mode only on the sample `users.csv` that `config generate` writes).

`[[users]].password` and the CSV column hold passwords in plaintext. When a run takes a password from `dtaas.toml` or `users.csv` and that file is readable by other users, the CLI prints a warning with the `chmod 600` command (on Linux and macOS; Windows file modes do not say who can read a file). Once a starting user's account exists, the CLI also reminds you to remove their `[[users]].password`: it stays a valid GitLab login until the user changes it.

FILES WRITTEN

`gitlab_user_tokens.json` holds the token issued to each user, keyed by username. It is written to the current directory, next to `dtaas.toml` (not to `--output-dir`), with mode `0600`, and each user's token is added the moment it is issued:

```
{
  "alice": "glpat-xxxxxxxxxxxxxxxxxxxx",
  "bob": "glpat-yyyyyyyyyyyyyyyyyyyy"
}
```


Copyright © 2022 - 2026 The INTO-CPS Association

A run adds to the file and never rewrites the other entries. Should a user's entry ever be replaced, the old token is kept under a `"<username> (superseded <timestamp>)"` key and a warning says to revoke it in GitLab if it is still active. `user delete` revokes the deleted users' tokens on GitLab, starting users' included, and then drops their entries; a token that could not be revoked keeps its entry. Their GitLab accounts stay.

Before creating any account, `user add` checks that this file can be read and written, and removes a leftover `gitlab_user_tokens.json.tmp` from an interrupted run. A token that still cannot be saved is revoked on GitLab at once, and the run issues no further tokens.

`dtaas.users.registry.json` records, per user, the GitLab account id (`gitlab_user_id`) and whether a token was issued (`gitlab_pat_issued`). That is how a re-run knows what is already done: a user with an account id but no token gets a new token for that account, with no password needed, after any token an earlier run issued but never saved is revoked, and a user with a token gets nothing more. Starting users carry the same two markers in the separate `starting_users` section, since they have no entry under `users`:

```
{
  "users": {
    "alice": {
      "email": "alice@intocps.org",
      "groups": [
        "additional"
      ],
      "load_balance": true,
      "desired_status": "running",
      "gitlab_user_id": 42,
      "gitlab_pat_issued": true
    }
  },
  "starting_users": {
    "username1": {
      "gitlab_user_id": 7,
      "gitlab_pat_issued": true
    }
  }
}
```

`dtaas.toml`, `users.csv` and `gitlab_user_tokens.json` are gitignored and written with mode `0600`, since they can hold the admin token, user passwords and user tokens respectively.

The registry and `gitlab_user_tokens.json` are read and rewritten without a lock, so run one `dtaas` command at a time per installation: two `user add` runs at once can lose each other's changes.

Copyright © 2022 - 2026 The INTO-CPS Association

4.3 Manual Installation (Advanced)

4.3.1 DTaaS

Localhost

INSTALL

Digital Twin as a Service

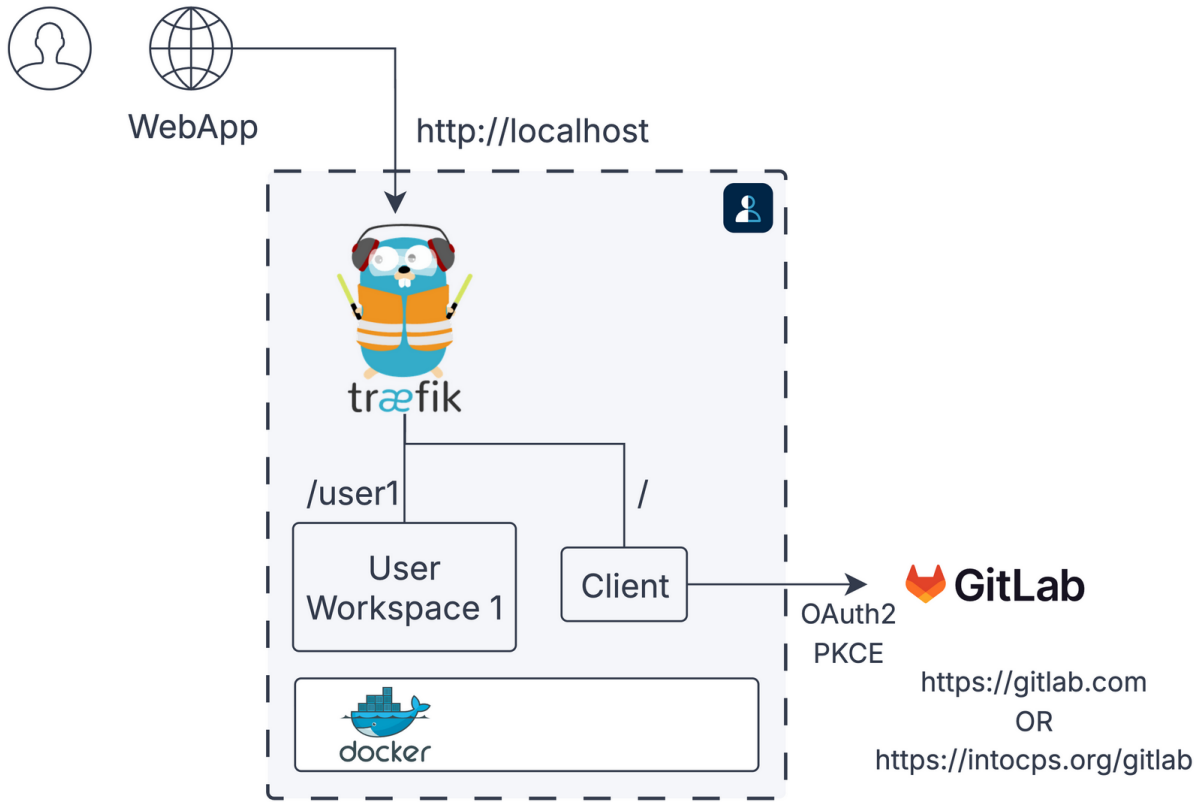
This guide installs DTaaS on localhost over **HTTP** with a single user workspace.

Download **dtaas-localhost-xx.zip** from the release / pre-release page.

For the configuration reference, see config.md.

Design

The diagram shows the Docker containers in this package.



Requirements

- Docker Engine with the Compose plugin
- A GitLab account for OAuth sign-in

Copyright © 2022 - 2026 The INTO-CPS Association

Quick Start

1. Create Configuration Files

```
cp config/.env.example config/.env
cp config/client.js.example config/client.js
```

2. Create User Workspace Directory

Set `DEFAULT_USER` in `config/.env`, then create a matching workspace:

```
cp -R files/template files/<DEFAULT_USER>
sudo chown -R 1000:100 files/*
```

The logger microservice writes its events to `logs/logger`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

3. Start Services

```
docker compose --env-file config/.env up -d
```

4. Open DTaaS

- http://localhost

Sign in with the OAuth provider configured in `config/client.js` (the default authority is `https://gitlab.com/`).

Run

Start and stop the application with:

```
docker compose --env-file config/.env up -d
docker compose --env-file config/.env down
```

Notes

- This package does not include `libms` or backend forward-auth.
- For secure production deployments, see `deploy/dtaas/docker/secure-server_with_integrated-gitlab`.

Documentation

See https://into-cps-association.github.io/DTaaS/development/index.html for the complete documentation.

References

Image sources: Traefik logo, gitlab

Copyright © 2022 - 2026 The INTO-CPS Association

CONFIGURATION REFERENCE

This page describes each configuration file in this package. For the quick-start installation guide, see install.md.

config/.env

Source: `config/.env.example`

```
cp config/.env.example config/.env
```

Variable	Example	Description
`DEFAULT_USER`	`user1`	Workspace path prefix and workspace folder name
`COMPOSE_PROJECT_NAME`	`dtaas`	Docker Compose project name

`docker-compose.yml` uses `DEFAULT_USER` in:

- the router path prefix (`/${DEFAULT_USER}`)
- the workspace volume mount (`./files/${DEFAULT_USER}`)
- the `MAIN_USER` environment variable of the workspace container

config/client.js

Source: `config/client.js.example`

```
cp config/client.js.example config/client.js
```

Variable	Example	Description
`REACT_APP_URL`	`http://localhost`	Base URL of DTaaS
`REACT_APP_CLIENT_ID`	*(OAuth client id)*	Client OAuth application ID
`REACT_APP_AUTH_AUTHORITY`	`https://gitlab.com/`	OAuth authority URL
`REACT_APP_REDIRECT_URI`	`http://localhost/Library`	Redirect URI after sign-in
`REACT_APP_LOGOUT_REDIRECT_URI`	`http://localhost/`	Redirect URI after sign-out
`LOGGER_URL`	`http://localhost/logger`	URL of the logger service

files/

`files/common` is mounted read-write at `/workspace/common`.

Create a per-user workspace from `files/template`:

```
cp -R files/template files/<DEFAULT_USER>
sudo chown -R 1000:100 files/*
```

logs/logger/

The logger microservice writes the captured events to `logs/logger/workflow-logs.jsonl`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

Reload client config

After editing `config/client.js`, reload only the client service:

```
docker compose --env-file config/.env up -d --force-recreate client
```


Copyright © 2022 - 2026 The INTO-CPS Association

Secure Server

INSTALL

Digital Twin as a Service

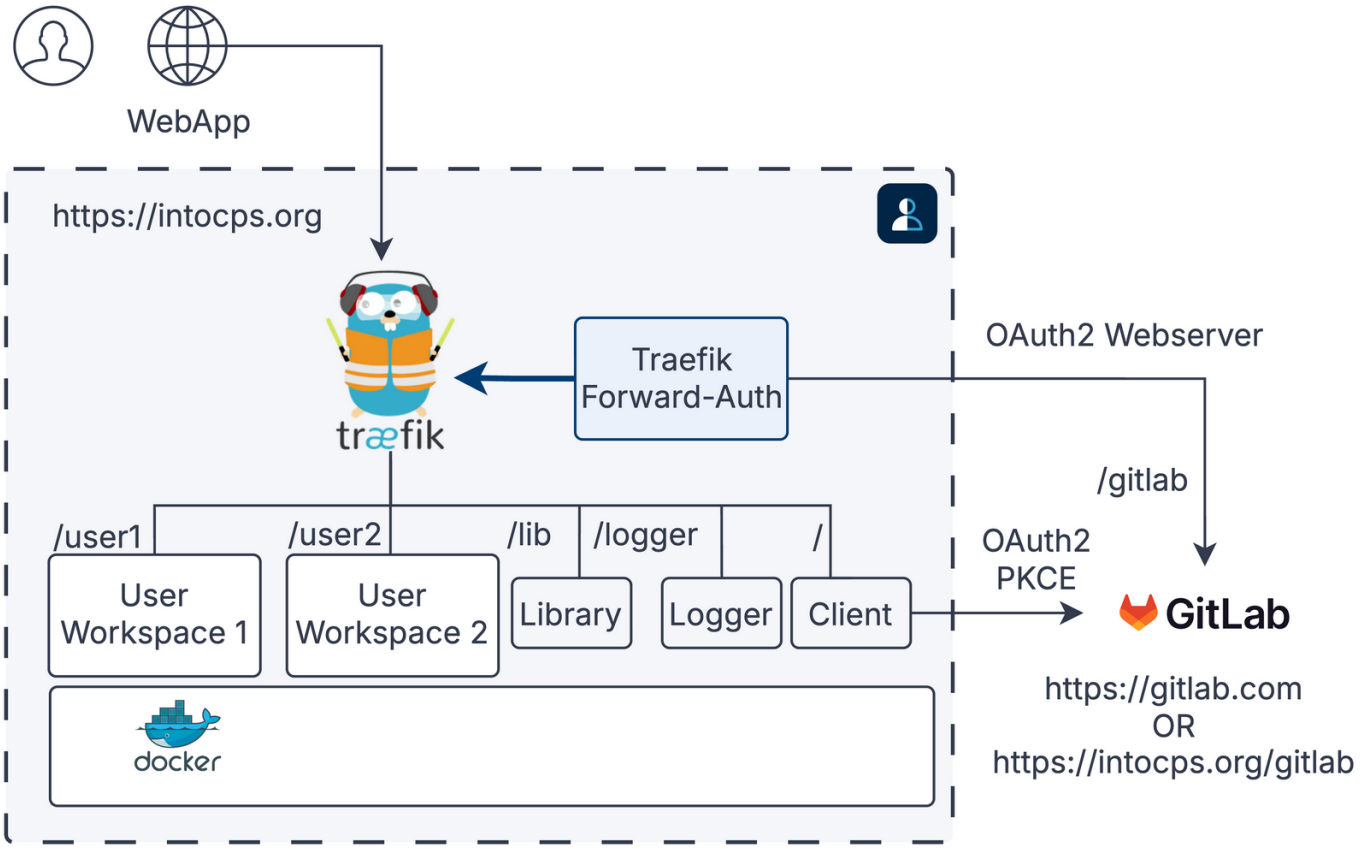
This guide installs a **secure, multi-user DTaaS deployment** that uses an **external GitLab instance** for OAuth 2.0 authorisation.

Download **dtaas-secure-server-xx.zip** from the latest release / pre-release page.

For the full configuration reference, see config.md.

Overview

This package deploys DTaaS with Traefik, OAuth forward-auth and two user workspaces.



A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

Copyright © 2022 - 2026 The INTO-CPS Association

The `docker-compose.yml` starts the following services:

Service	Purpose
traefik	Reverse proxy with TLS termination
client	DTaaS React frontend
logger-ms	Logger microservice
user1 / user2	JupyterLab user workspaces
libms	Library management microservice
traefik-forward-auth	OAuth 2.0 authorisation middleware

Prerequisites

Requirement	Details
Docker Engine	v28 or later with Compose plugin
Domain name	Public DNS name or server IP
TLS certificates	`fullchain.pem` and `privkey.pem` for the target domain
OAuth provider	External GitLab (`gitlab.com` or self-hosted GitLab)

Quick Start

1. Create Configuration Files

```
cp config/.env.example config/.env
cp config/conf.server.example config/conf.server
cp config/client.js.example config/client.js
```

Edit `config/.env` :

- `SERVER_DNS`
- `USERNAME1`
- `USERNAME2`
- `OAUTH_URL`
- `OAUTH_CLIENT_ID`
- `OAUTH_CLIENT_SECRET`
- `OAUTH_SECRET`

Edit `config/client.js` :

- `REACT_APP_CLIENT_ID`
- `REACT_APP_AUTH_AUTHORITY`
- `REACT_APP_REDIRECT_URI`
- `REACT_APP_LOGOUT_REDIRECT_URI`

See config.md for details.

2. Create User Workspace Directories

```
cp -R files/template files/<USERNAME1>
cp -R files/template files/<USERNAME2>
sudo chown -R 1000:100 files/*
```


Copyright © 2022 - 2026 The INTO-CPS Association

The logger microservice writes its events to `logs/logger`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

3. Add TLS Certificates

```
cp /path/to/fullchain.pem certs/fullchain.pem
cp /path/to/privkey.pem certs/privkey.pem
```

If the certificates are missing or invalid, Traefik falls back to self-signed certificates.

4. Configure OAuth 2.0 Applications in GitLab

1. Create user accounts (see GitLab docs). The usernames **must** match `USERNAME1` / `USERNAME2` in `config/.env`.
2. Register two OAuth 2.0 applications in GitLab (Admin Area → Applications):
3. **DTaaS Client Authorization** for the React SPA frontend. See the client auth docs.
4. **DTaaS Server Authorization** for the Traefik forward-auth backend. See the server auth docs.
5. Put the generated OAuth 2.0 credentials into the configuration files:
6. Set `REACT_APP_CLIENT_ID` and `REACT_APP_AUTH_AUTHORITY` in `config/client.js`.
7. Set `OAUTH_URL`, `OAUTH_CLIENT_ID` and `OAUTH_CLIENT_SECRET` in `config/.env`.

5. Start Services

```
docker compose --env-file config/.env up -d
```

6. Verify

URL	Expected result
`https://<SERVER_DNS>`	DTaaS web interface
`https://<SERVER_DNS>/user1`	User 1 workspace after sign-in
`https://<SERVER_DNS>/user2`	User 2 workspace after sign-in
`https://<SERVER_DNS>/lib`	Library service

Stop

```
docker compose --env-file config/.env down
```

Directory Layout

```
.
| - certs/                # TLS certificates (fullchain.pem, privkey.pem) |
| - config/                                                               |
|                                                                         |
|                                                                         |
|                                                                         |
|                                                                         |
|                                                                         |
|                                                                         |
| \- tls.yml            # Traefik TLS provider configuration              |
| - files/                                                                |
|                                                                         |
| \- template/           # sample user workspace files                    |
| - logs/                                                                 |
| \- logger/             # Logger microservice events                     |
| - docker-compose.yml     # Service definitions                          |
| - CONFIG.md              # Detailed configuration reference             |
\- README.md             # This file
```


Copyright © 2022 - 2026 The INTO-CPS Association

CONFIGURATION REFERENCE

This page describes each configuration file in this package. For the quick-start installation guide, see install.md.

config/.env - Docker Compose Environment

Source: `config/.env.example`

This file holds the environment variables that `docker-compose.yml` reads.

```
cp config/.env.example config/.env
```

Server Settings

Variable	Example	Description
`SERVER_DNS`	`intocps.org`	Domain name or IP address of the server. Do not include `https://`.
`USERNAME1`	`user1`	Path prefix and workspace name for the first user
`USERNAME2`	`user2`	Path prefix and workspace name for the second user
`COMPOSE_PROJECT_NAME`	`dtaas`	Docker Compose project name

OAuth 2.0 Settings

These values come from the OAuth 2.0 applications on the external GitLab.

Variable	Example	Description
`OAUTH_URL`	`https://gitlab.com`	GitLab base URL used by traefik-forward-auth
`OAUTH_CLIENT_ID`	*(from GitLab)*	Application ID from the **DTaaS Server Authorization** OAuth app
`OAUTH_CLIENT_SECRET`	*(from GitLab)*	Secret from the **DTaaS Server Authorization** OAuth app
`OAUTH_SECRET`	*(random string)*	Encryption key for OAuth session cookies. Example generation: `openssl rand -base64 32`

How Variables Map to Services

Variable	Used by
`SERVER_DNS`	client, logger-ms, user1, user2, libms, traefik-forward-auth
`USERNAME1` / `USERNAME2`	user1, user2 (routing and workspace volumes)
`OAUTH_URL`	traefik-forward-auth (authorize, token, userinfo endpoints)
`OAUTH_CLIENT_ID` / `OAUTH_CLIENT_SECRET`	traefik-forward-auth
`OAUTH_SECRET`	traefik-forward-auth

config/client.js - DTaaS Web Client

Source: `config/client.js.example`

This JavaScript file is mounted into the React client container and configures the DTaaS web application at runtime.

```
cp config/client.js.example config/client.js
```


Copyright © 2022 - 2026 The INTO-CPS Association

Variable Reference

Variable	Example	Description
`REACT_APP_ENVIRONMENT`	`prod`	Environment name
`REACT_APP_URL`	`https://intocps.org`	Base URL of the DTaaS web application
`REACT_APP_URL_BASENAME`	`''`	Optional URL base path
`REACT_APP_URL_DTLINK`	`/lab`	URL path for the Digital Twin workbench
`REACT_APP_URL_LIBLINK`	`''`	URL path for the Library
`REACT_APP_CLIENT_ID`	*(from GitLab)*	Application ID from **DTaaS Client Authorization** OAuth app
`REACT_APP_AUTH_AUTHORITY`	`https://gitlab.com`	OAuth issuer URL
`REACT_APP_REDIRECT_URI`	`https://intocps.org/Library`	Redirect URI after sign-in
`REACT_APP_LOGOUT_REDIRECT_URI`	`https://intocps.org/`	Redirect URI after sign-out
`REACT_APP_GITLAB_SCOPES`	`openid profile read_user read_repository api`	Requested OAuth scopes
`LOGGER_URL`	`https://intocps.org/logger`	URL of the logger service

config/conf.server - Traefik Forward-Auth Rules

Source: `config/conf.server.example`

This file defines per-path authorisation rules for traefik-forward-auth. Each rule restricts a URL path to specific email addresses.

```
cp config/conf.server.example config/conf.server
```

Format

```
rule.<NAME>.action=auth
rule.<NAME>.rule=PathPrefix(`/<path>`)
rule.<NAME>.whitelist=<email>
```

Default Rules

```
rule.libms.action=auth
rule.libms.rule=PathPrefix(`/lib`)

rule.onlyu1.action=auth
rule.onlyu1.rule=PathPrefix(`/user1`)
rule.onlyu1.whitelist=user1@emailservice.com

rule.onlyu2.action=auth
rule.onlyu2.rule=PathPrefix(`/user2`)
rule.onlyu2.whitelist=user2@emailservice.com
```

Replace the usernames and email addresses with those of the actual users.

Important Rules

- Usernames in `config/.env` (`USERNAME1`, `USERNAME2`) must match `PathPrefix` values in `config/conf.server`.
- If a route exists in `docker-compose.yml` but has no rule in `config/conf.server`, any signed-in user can open it.
- If a rule exists in `config/conf.server` but no router serves that path, the URL returns 404.

Copyright © 2022 - 2026 The INTO-CPS Association

certs - TLS Certificates

Place these TLS certificate files in `certs/`:

```
certs/
| - fullchain.pem   # Public certificate or certificate chain |
\- privkey.pem     # Private key
```

The certificates must be valid for `SERVER_DNS`.

files - User Workspace Directories

Each user workspace container mounts a directory from `files/` as its `/workspace` volume. All workspaces share `files/common/`.

```
files/
| - common/    # Shared files (mounted to /workspace/common) |
| - user1/     # User 1 workspace files                      |
\- user2/     # User 2 workspace files
```

Create the user directories:

```
cp -R files/template files/<USERNAME>
sudo chown -R 1000:100 files/*
```

logs - Logger Microservice Events

The logger microservice writes the captured events to `logs/logger/workflow-logs.jsonl`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

OAuth 2.0 Application Setup

The external GitLab instance needs two OAuth 2.0 applications.

DTaaS Client Authorization (React Frontend)

1. In GitLab, open **Applications**.
2. Create an application:
3. **Name**: DTaaS Client Authorization
4. **Redirect URI**: `https://<SERVER_DNS>/Library`
5. **Confidential**: unticked (public SPA client)
6. **Scopes**: `openid`, `profile`, `read_user`, `read_repository`, `api`
7. Save the **Application ID** and set `REACT_APP_CLIENT_ID` in `config/client.js`.
8. Set `REACT_APP_AUTH_AUTHORITY` in `config/client.js` to the GitLab URL.

DTaaS Server Authorization (Traefik Forward-Auth)

1. In GitLab, create another application:
2. **Name**: DTaaS Server Authorization
3. **Redirect URI**: `https://<SERVER_DNS>/_oauth`
4. **Confidential**: ticked
5. **Scopes**: `read_user`
6. Save the **Application ID** and **Secret**.
7. Set `OAUTH_CLIENT_ID`, `OAUTH_CLIENT_SECRET` and `OAUTH_URL` in `config/.env`.
8. Generate `OAUTH_SECRET` and set it in `config/.env`.

Copyright © 2022 - 2026 The INTO-CPS Association

Reload After Configuration

```
docker compose --env-file config/.env up -d --force-recreate client traefik-forward-auth
```

Adding More Users

To add a third user:

1. Add a `user3` service to `docker-compose.yml`, modelled on `user1` and `user2`.
2. Add `USERNAME3=<name>` to `config/.env`.
3. Create the `files/<name>` directory from `files/template`.
4. Add a matching authorisation rule to `config/conf.server`.
5. Restart:

```
docker compose --env-file config/.env up -d
```

Troubleshooting

Authentication redirect loop

- Check `OAUTH_URL` in `config/.env`.
- Check `REACT_APP_AUTH_AUTHORITY` in `config/client.js`.
- Clear the browser cookies for the domain.
- Check the logs:

```
docker compose --env-file config/.env logs traefik-forward-auth
```

404 on user workspace

- Check that the usernames are the same in:
- `config/.env`
- `config/conf.server`
- `docker-compose.yml`

TLS warning in browser

- Replace `certs/fullchain.pem` and `certs/privkey.pem` with valid certificates.

Copyright © 2022 - 2026 The INTO-CPS Association

Secure Server and GitLab

INSTALL

Digital Twin as a Service

This guide installs a **secure, multi-user DTaaS deployment** that uses an **integrated GitLab instance** for OAuth 2.0 authorisation.

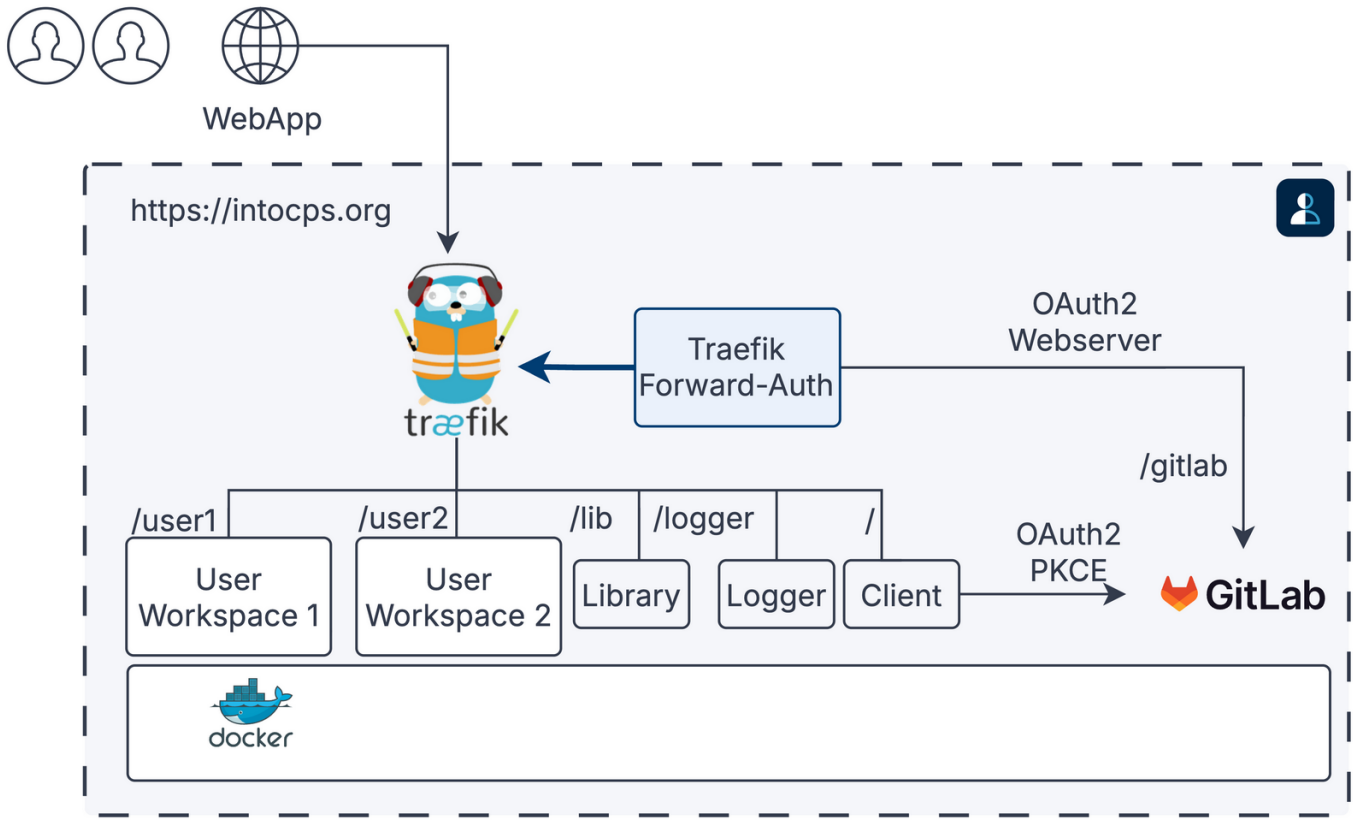
Download **dtaas-secure-server-integrated-gitlab-xx.zip** from the latest release / pre-release page.

For the configuration reference, see config.md.

> **Important**
>
> This guide uses the hostname `intocps.org` as an example. Replace it with the hostname of your server.

Overview

This package installs DTaaS as a multi-user web application with an integrated GitLab CE instance. GitLab is both the OAuth 2.0 authorisation provider and the DevOps backend for digital twin projects.



A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

Copyright © 2022 - 2026 The INTO-CPS Association

The `docker-compose.yml` starts the following services:

Service	Purpose
traefik	Reverse proxy with TLS termination
client	DTaaS React frontend
logger-ms	Logger microservice
user1 / user2	JupyterLab user workspaces
libms	Library management microservice
traefik-forward-auth	OAuth 2.0 authorisation middleware
gitlab	Integrated GitLab CE (OAuth 2.0 provider)

Prerequisites

Requirement	Details
Docker Engine	v28 or later with Compose plugin
Domain name	A domain name (e.g. `intocps.org`) or IP address
TLS certificate	`fullchain.pem` and `privkey.pem` for the domain. Obtain via certbot or a certificate provider

Quick Start

1. Create Configuration Files

```
cp config/.env.example       config/.env
cp config/conf.server.example config/conf.server
cp config/client.js.example     config/client.js
```

Edit `config/.env` and set `SERVER_DNS`, `USERNAME1` and `USERNAME2`. Leave the `OAUTH_*` variables as placeholders for now; you fill them in once the GitLab instance is running (see step 5).

See config.md for a reference of every variable.

2. Create User Workspace Directories

```
cp -R files/template files/<USERNAME1>
cp -R files/template files/<USERNAME2>
sudo chown -R 1000:100 files/*
```

The logger microservice writes its events to `logs/logger`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

`logs/` itself holds the GitLab logs, so change the owner of `logs/logger` only.

3. Add TLS Certificates

```
cp /path/to/fullchain.pem certs/fullchain.pem
cp /path/to/privkey.pem   certs/privkey.pem
```

Traefik falls back to self-signed certificates if these files are absent or invalid.

4. Start Services

```
docker compose --env-file config/.env up -d
```


Copyright © 2022 - 2026 The INTO-CPS Association

GitLab takes a few minutes to become healthy. Watch its status with:

```
watch docker ps
```

5. Configure GitLab

   Once the GitLab container shows as `healthy`:

1. Log in to `https://intocps.org/gitlab` as `root`. The initial password is in `config/gitlab/initial_root_password`.

> **Warning**
>
> GitLab **deletes this file 24 hours** after the first start. Save the password immediately.

2. Create user accounts (see GitLab docs). The usernames **must** match `USERNAME1` / `USERNAME2` in `config/.env`.
3. Register two OAuth 2.0 applications in GitLab (Admin Area → Applications):
4. **DTaaS Client Authorization** for the React SPA frontend. See the client auth docs.
5. **DTaaS Server Authorization** for the Traefik forward-auth backend. See the server auth docs.
6. Put the generated OAuth 2.0 credentials into the configuration files:
7. Set `REACT_APP_CLIENT_ID` and `REACT_APP_AUTH_AUTHORITY` in `config/client.js`.
8. Set `OAUTH_URL`, `OAUTH_CLIENT_ID` and `OAUTH_CLIENT_SECRET` in `config/.env`.
9. Update the user permissions in `config/conf.server`.
10. Reload the services:

```
docker compose --env-file config/.env up -d --force-recreate client traefik-forward-auth
```

6. Verify

URL	Expected result
`https://intocps.org`	DTaaS web interface (redirects to GitLab sign-in)
`https://intocps.org/gitlab`	Integrated GitLab instance
`https://intocps.org/user1`	User 1 workspace (after sign-in)
`https://intocps.org/user2`	User 2 workspace (after sign-in)

Stop

```
docker compose --env-file config/.env down
```

Directory Layout

```
.
├── certs/                # TLS certificates (fullchain.pem, privkey.pem)
├── config/
│   ├── .env              # Docker compose environment variables
│   ├── conf.server       # Traefik forward-auth authorisation rules
│   ├── client.js         # DTaaS React client configuration
│   ├── gitlab/           # GitLab config (mounted as /etc/gitlab)
│   └── tls.yml           # Traefik TLS provider configuration
├── data/                 # GitLab persistent data (/var/opt/gitlab)
├── files/
│   ├── common/           # Shared files across all workspaces
│   └── template/         # sample user workspace files
├── logs/                 # GitLab logs (/var/log/gitlab)
│   └── logger/           # Logger microservice events
├── docker-compose.yml    # Service definitions
├── CONFIG.md             # Detailed configuration reference
└── README.md             # This file — quick-start guide
```


Copyright © 2022 - 2026 The INTO-CPS Association

References

Image sources: Traefik logo, GitLab

Copyright © 2022 - 2026 The INTO-CPS Association

CONFIGURATION REFERENCE

This page describes every configuration file in this package. For the quick-start installation guide, see install.md.

> **Important**
>
> This guide uses the hostname `intocps.org` as an example. Replace it with the hostname of your server.

config/.env - Docker Compose Environment

Source: `config/.env.example`

This file holds the environment variables that `docker-compose.yml` reads.

```
cp config/.env.example config/.env
```

Server Settings

Variable	Example	Description
`SERVER_DNS`	`intocps.org`	Domain name or IP address of the server. Do **not** include `https://`.
`USERNAME1`	`user1`	Path prefix and workspace name for the first user
`USERNAME2`	`user2`	Path prefix and workspace name for the second user
`COMPOSE_PROJECT_NAME`	`dtaas`	Docker Compose project name (rarely needs changing)

OAuth 2.0 Settings

Fill these in after the GitLab instance is running and the OAuth 2.0 applications exist (see OAuth 2.0 Application Setup).

Variable	Example	Description
`OAUTH_URL`	`https://intocps.org/gitlab`	GitLab instance URL used for browser-side authorisation redirects. No trailing slash.
`OAUTH_CLIENT_ID`	*(from GitLab)*	Application ID from the **DTaaS Server Authorization** OAuth 2.0 application
`OAUTH_CLIENT_SECRET`	*(from GitLab)*	Secret from the **DTaaS Server Authorization** OAuth 2.0 application
`OAUTH_SECRET`	*(random string)*	Encryption key for OAuth session cookies. Generate with: `openssl rand -base64 32`

How Variables Map to Services

Variable	Used by
`SERVER_DNS`	client, logger-ms, user1, user2, libms, traefik-forward-auth, gitlab
`USERNAME1` / `USERNAME2`	user1, user2 (routing and workspace volumes)
`OAUTH_URL`	traefik-forward-auth (browser redirect URL)
`OAUTH_CLIENT_ID` / `OAUTH_CLIENT_SECRET`	traefik-forward-auth
`OAUTH_SECRET`	traefik-forward-auth

traefik-forward-auth reaches the token and user endpoints over the internal Docker network (`http://gitlab/gitlab/...`), so only the authorisation redirect uses `OAUTH_URL` .

config/client.js - DTaaS Web Client

Source: `config/client.js.example`

Copyright © 2022 - 2026 The INTO-CPS Association

This JavaScript file is mounted into the React client container and configures the DTaaS web application at runtime.

```
cp config/client.js.example config/client.js
```

Variable Reference

Variable	Example	Description
`REACT_APP_ENVIRONMENT`	`prod`	Environment name. Use `prod` for production.
`REACT_APP_URL`	`https://intocps.org`	Base URL of the DTaaS web application
`REACT_APP_URL_BASENAME`	`''`	Optional URL base path (leave empty for root hosting)
`REACT_APP_URL_DTLINK`	`/lab`	URL path for the Digital Twin workbench
`REACT_APP_URL_LIBLINK`	`''`	URL path for the Library
`REACT_APP_CLIENT_ID`	*(from GitLab)*	Application ID from the **DTaaS Client Authorization** OAuth 2.0 application
`REACT_APP_AUTH_AUTHORITY`	`https://intocps.org/gitlab`	URL of the GitLab instance (OAuth 2.0 issuer)
`REACT_APP_REDIRECT_URI`	`https://intocps.org/Library`	Where GitLab sends users after sign-in
`REACT_APP_LOGOUT_REDIRECT_URI`	`https://intocps.org/`	Where users land after sign-out
`REACT_APP_GITLAB_SCOPES`	`openid profile read_user read_repository api`	OAuth 2.0 scopes requested during sign-in
`LOGGER_URL`	`https://intocps.org/logger`	URL of the logger service

Example

```
if (typeof window !== 'undefined') {
  window.env = {
    REACT_APP_ENVIRONMENT: 'prod',
    REACT_APP_URL: 'https://intocps.org',
    REACT_APP_URL_BASENAME: '',
    REACT_APP_URL_DTLINK: '/lab',
    REACT_APP_URL_LIBLINK: '',
    REACT_APP_CLIENT_ID: '<APPLICATION_ID>',
    REACT_APP_AUTH_AUTHORITY: 'https://intocps.org/gitlab',
    REACT_APP_REDIRECT_URI: 'https://intocps.org/Library',
    REACT_APP_LOGOUT_REDIRECT_URI: 'https://intocps.org/',
    REACT_APP_GITLAB_SCOPES: 'openid profile read_user read_repository api',
    LOGGER_URL: 'https://intocps.org/logger',
  };
};
```

config/conf.server - Traefik Forward-Auth Rules

Source: `config/conf.server.example`

This file defines per-path authorisation rules for traefik-forward-auth. Each rule restricts a URL path to specific GitLab email addresses.

```
cp config/conf.server.example config/conf.server
```

Format

```
rule.<NAME>.action=auth
rule.<NAME>.rule=PathPrefix(`/<path>`)
rule.<NAME>.whitelist=<email>
```


Copyright © 2022 - 2026 The INTO-CPS Association

Default Rules

```
rule.libms.action=auth
rule.libms.rule=PathPrefix(`/lib`)

rule.onlyu1.action=auth
rule.onlyu1.rule=PathPrefix(`/user1`)
rule.onlyu1.whitelist=user1@emailservice.com

rule.onlyu2.action=auth
rule.onlyu2.rule=PathPrefix(`/user2`)
rule.onlyu2.whitelist=user2@emailservice.com
```

Replace `user1`, `user2` and the email addresses with those of the actual GitLab accounts.

Important Rules

> **Warning**
>
> The usernames in `config/.env` (`USERNAME1`, `USERNAME2`) must match the `PathPrefix` values in `config/conf.server`. A mismatch causes routing or authorisation failures.

Scenario	Behaviour
Route in `config/.env` but **missing** from `config/conf.server`	Any signed-in user can access the route (default forward-auth behaviour)
Route in `config/conf.server` but **missing** from `config/.env`	Traefik returns **404** (route not served)
The `/lib` rule has **no whitelist**	Any signed-in user can access the library service

certs/ - TLS Certificates

Place the TLS certificate files here:

```
certs/
├── fullchain.pem   # Public certificate (or certificate chain)
└── privkey.pem     # Private key
```

The certificates must be valid for `SERVER_DNS` (e.g. `intocps.org` or `*.intocps.org`).

To obtain certificates with certbot:

```
# Using certbot (Let's Encrypt)
sudo certbot certonly --standalone -d intocps.org
sudo cp /etc/letsencrypt/live/intocps.org/fullchain.pem certs/
sudo cp /etc/letsencrypt/live/intocps.org/privkey.pem   certs/
```

If the certificate files are absent or invalid, Traefik uses self-signed certificates and browsers show a security warning.

files/ - User Workspace Directories

Each user workspace container mounts a directory from `files/` as its `/workspace` volume. All workspaces share the `files/common/` directory, which is mounted at `/workspace/common` in each container.

```
files/
├── common/    # Shared files (mounted to /workspace/common in each container)
└── template/  # template workspace files
```

Create a directory for each user:

```
cp -R files/template files/<USERNAME>
sudo chown -R 1000:100 files/*
```

UID `1000` and GID `100` match the default user inside the workspace container.

Copyright © 2022 - 2026 The INTO-CPS Association

logs/logger/ - Logger Microservice Events

The logger microservice writes the captured events to `logs/logger/workflow-logs.jsonl`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```

`logs/` itself holds the GitLab logs, so change the owner of `logs/logger` only.

OAuth 2.0 Application Setup

Once the GitLab instance is running, register two OAuth 2.0 applications to connect DTaaS and Traefik forward-auth to the integrated GitLab.

DTaaS Client Authorization (React Frontend)

1. In GitLab, go to **Admin Area → Applications** (or the user's **Edit Profile → Applications**).
2. Create a new application:
3. **Name**: DTaaS Client Authorization
4. **Redirect URI**: `https://intocps.org/Library`
5. **Confidential**: unticked (public SPA client)
6. **Scopes**: `openid`, `profile`, `read_user`, `read_repository`, `api`
7. Save the **Application ID**.
8. Set `REACT_APP_CLIENT_ID` in `config/client.js` to this Application ID.
9. Set `REACT_APP_AUTH_AUTHORITY` in `config/client.js` to `https://intocps.org/gitlab`.

For details, see the client auth documentation.

DTaaS Server Authorization (Traefik Forward-Auth)

1. In GitLab, go to **Admin Area → Applications**.
2. Create a new application:
3. **Name**: DTaaS Server Authorization
4. **Redirect URI**: `https://intocps.org/_oauth`
5. **Confidential**: ticked
6. **Scopes**: `read_user`
7. Save the **Application ID** and **Secret**.
8. Set `OAUTH_CLIENT_ID` and `OAUTH_CLIENT_SECRET` in `config/.env`.
9. Set `OAUTH_URL` in `config/.env` to `https://intocps.org/gitlab`.
10. Generate `OAUTH_SECRET` with `openssl rand -base64 32` and set it in `config/.env`.

For details, see the server auth documentation.

Reload After Configuration

After putting the OAuth 2.0 credentials into the configuration files, reload the affected services:

```
docker compose --env-file config/.env up -d --force-recreate client traefik-forward-auth
```


Copyright © 2022 - 2026 The INTO-CPS Association

Adding More Users

To add a third user:

1. **Add a service to `docker-compose.yml`:**

```
user3:
  image: intocps/workspace:main-56c6f68
  restart: unless-stopped
  environment:
    - MAIN_USER=${USERNAME3:-user3}
  volumes:
    - "./files/common:/workspace/common"
    - "./files/${USERNAME3:-user3}:/workspace"
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.u3.rule=Host(`${SERVER_DNS:-localhost}`) && PathPrefix(`/${USERNAME3:-user3}`)"
    - "traefik.http.routers.u3.tls=true"
    - "traefik.http.routers.u3.middlewares=traefik-forward-auth"
  networks:
    - users
```

1. **Add to `config/.env`:**

```
USERNAME3=alice
```

1. **Create the workspace directory:**

```
cp -R files/template files/alice
sudo chown -R 1000:100 files/alice
```

1. **Add an authorisation rule to `config/conf.server`:**

```
rule.onlyu3.action=auth
rule.onlyu3.rule=PathPrefix(`/alice`)
rule.onlyu3.whitelist=alice@emailservice.com
```

1. **Create a GitLab account** for `alice` in the integrated GitLab instance.
2. **Restart:**

```
docker compose --env-file config/.env up -d
```

Troubleshooting

GitLab Takes Too Long to Start

GitLab CE needs substantial resources, and the first startup can take 5 to 10 minutes. Follow progress with `docker compose --env-file config/.env logs -f gitlab`. Make sure the host has at least 4 GB of RAM available for GitLab.

Authentication Redirect Loop

1. Check that `OAUTH_URL` in `config/.env` matches the URL that the user's browser can reach (e.g. `https://intocps.org/gitlab`).
2. Check that `REACT_APP_AUTH_AUTHORITY` in `config/client.js` matches the same URL.
3. Clear the browser cookies for the domain.
4. Check the traefik-forward-auth logs: `docker compose --env-file config/.env logs traefik-forward-auth`

404 on User Workspace

- Make sure `USERNAME1` / `USERNAME2` in `config/.env` match the `PathPrefix` values in `config/conf.server`.
- Make sure a matching service exists in `docker-compose.yml`.

GitLab "502 Bad Gateway"

GitLab is still initialising. Wait until `docker ps` shows the container as `healthy`.

Self-Signed Certificate Warning in Browser

Copyright © 2022 - 2026 The INTO-CPS Association

The TLS certificate files in `certs/` are missing or invalid. Replace them with valid certificates for the domain.

Copyright © 2022 - 2026 The INTO-CPS Association

4.3.2 Workspace

Localhost

INSTALL



Digital Twin as a Service

Install

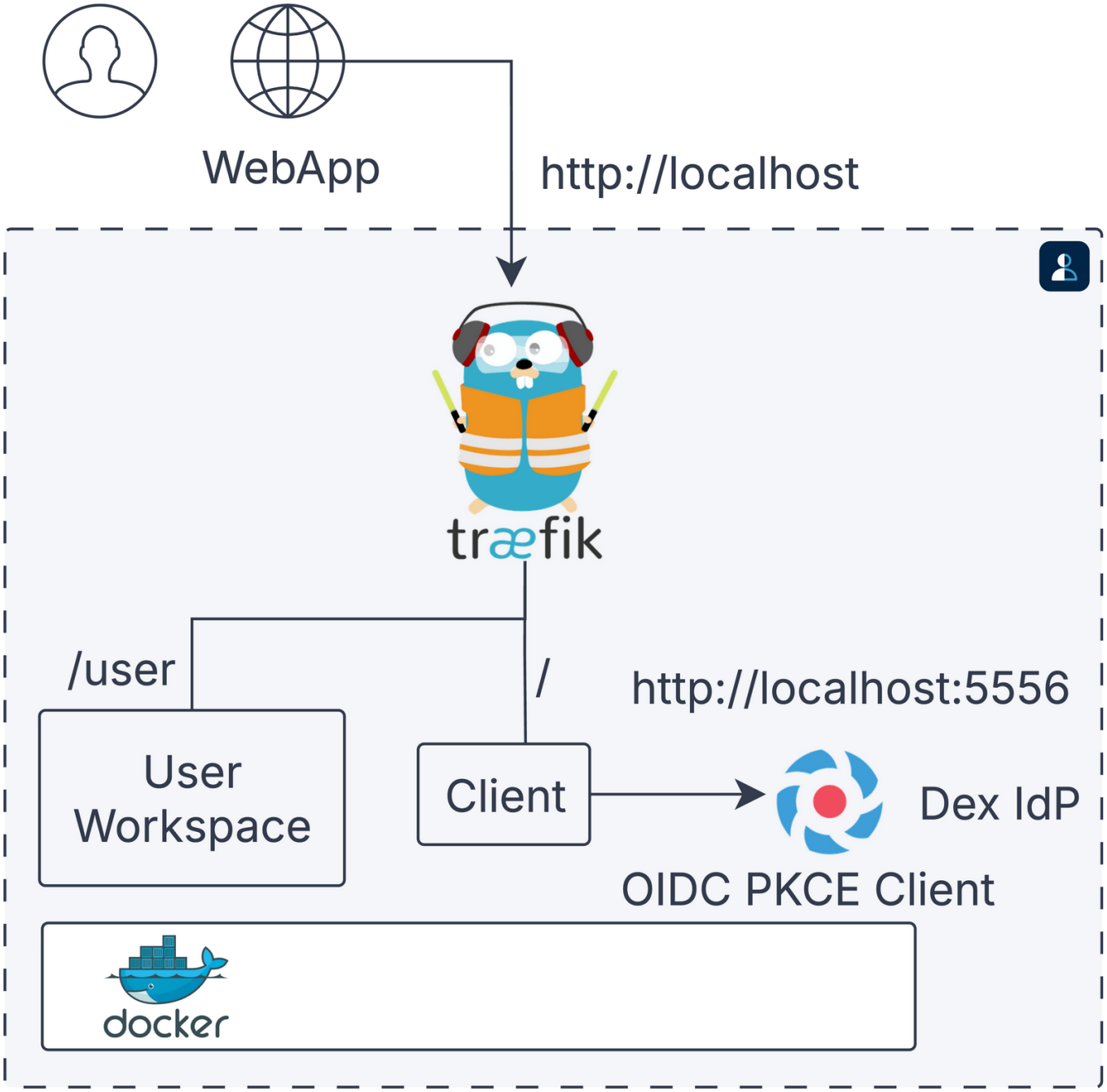
These instructions run **DTaaS on localhost, served over HTTP**. This installation suits a single user running DTaaS on their own computer.

Download **workspace-dex-localhost-xx.zip** from the latest release / pre-release page.

Design

The installation setup is shown here.

Copyright © 2022 - 2026 The INTO-CPS Association



A path such as `/user` is the URL route of that service, for example `http://localhost/user`.

Prerequisites

- Docker Engine v27 or later
- Ports 80 and 5556 available on the host
- At least 2GB RAM available

Quick Demo

Run all the commands below from the extracted package directory.

Copy the example configuration files first:

```
cp config/.env.example config/.env
cp config/dex-config.yaml.example config/dex-config.yaml
```


Copyright © 2022 - 2026 The INTO-CPS Association

The workspace container runs as uid 1000 and gid 100. Give it ownership of the user files it mounts:

```
sudo chown -R 1000:100 files
```

Start the demo:

```
docker compose --env-file config/.env up -d
```

Open http://localhost in a web browser and log in with the default credentials:

- **User email:** `user@intocps.org`
- **Password:** `user`

Stop the demo:

```
docker compose --env-file config/.env down
```

Limitations

1. All DTaaS functionality except the DevOps features is available through the single page client.
2. The installation has default user credentials. See the instructions for changing them.

Documentation

See https://into-cps-association.github.io/DTaaS/development/index.html for the complete documentation.

References

Image sources:

Traefik logo: https://www.laub-home.de/wiki/

Dex IdP: https://dexidp.io/

Copyright © 2022 - 2026 The INTO-CPS Association

DEX CONFIGURATION GUIDE

The `dex-config.yaml.example` file is the Dex configuration template. Copy `config/dex-config.yaml.example` to `config/dex-config.yaml`.

Files

- `dex-config.yaml.example`: template for local password database mode

Common options

`issuer`

Set to `http://localhost:5556/dex`.

This must match the URL used by the web client (`REACT_APP_AUTH_AUTHORITY`).

`storage.type: memory`

Keeps state in memory for local development. Tokens, keys and sessions reset when the Dex container is recreated.

`web.http: 0.0.0.0:5556`

Dex listens on container port `5556`.

`web.allowedOrigins: ['*']`

Permissive CORS setting for localhost development.

`staticClients`

One client is configured:

- `id: mock`
- `redirectURIs: ['http://localhost/Library']`
- `public: true` (no client secret, which suits a browser SPA)

Local password database file (`dex-config.yaml`)

`expiry.idTokens: "2h"`

ID tokens are valid for two hours.

`oauth2.skipApprovalScreen: true`

Skips the consent page during login.

`enablePasswordDB: true` and `staticPasswords`

Enables Dex local user authentication with static users.

Configured user fields:

- `email`: login identifier used on the Dex login screen
- `hash`: bcrypt hash of the password
- `username` / `name` / `preferredUsername`: identity claims returned by Dex
- `groups`: included when the `groups` scope is requested
- `userID`: stable Dex subject identifier

In `dex-config.yaml.example`, the sample user's password is `user` (bcrypt-hashed).

Generate a bcrypt password hash

Use the generated value for the `hash` field in `staticPasswords`.

Copyright © 2022 - 2026 The INTO-CPS Association

Install the Python dependency if needed:

```
python3 -m pip install bcrypt
```

Generate a bcrypt hash interactively:

```
python3 - <<'PY'
import bcrypt
import getpass

password = getpass.getpass('Password: ').encode('utf-8')
print(bcrypt.hashpw(password, bcrypt.gensalt()).decode('utf-8'))
PY
```

Copy the printed hash into `config/dex-config.yaml` as the value of `hash`.

Username alignment with `config/.env`

DTaaS user-scoped routes use the `DEFAULT_USER` value from `config/.env`. `DEFAULT_USER` sets only the workspace URL route (`/<DEFAULT_USER>`) and the `MAIN_USER` of the workspace container. The workspace files always come from `files/user`: `docker-compose.yml` mounts `./files/user` at `/workspace` whatever the value of `DEFAULT_USER`.

In local password database mode, keep these aligned:

- `config/.env`: `DEFAULT_USER=<user>`
- `config/dex-config.yaml`: set the static user's `username` and `preferredUsername` to the same `<user>`

This prevents path mismatches in user-scoped URLs.

Copyright © 2022 - 2026 The INTO-CPS Association

CUSTOM USER



Digital Twin as a Service

This document explains how to change the username and password of the localhost **DTaaS** installation.

Update Configuration

Copy `config/dex-config.yaml.example` to `config/dex-config.yaml`. Update these fields in `config/dex-config.yaml`:

- `email`
- `hash` (bcrypt hash of the chosen password)
- `username`, `name`, `preferredUsername`

The Dex configuration is described in `dex.md`.

Edit `config/.env`.

Variable	Example Value	Explanation
DEFAULT_USER	'user'	Dex username set in `dex-config.yaml`

The username must match in both files when Dex uses its local password database:

- In `config/.env`, choose `DEFAULT_USER=<user>`.
- In `config/dex-config.yaml`, set the static user's `username` and `preferredUsername` to the same value.

This keeps the user-scoped DTaaS routes in line with the OIDC identity claims.

Changing `DEFAULT_USER` changes only the workspace URL route (`/<DEFAULT_USER>`) and the `MAIN_USER` of the workspace container. The workspace files still come from `files/user`, because `docker-compose.yml` mounts `./files/user` at `/workspace`; do not rename that directory.

Run

Start the application:

```
docker compose --env-file config/.env up -d
```

Stop the application:

```
docker compose --env-file config/.env down
```

Use

Open http://localhost in a web browser. Sign in with the new user credentials from the Dex configuration file (`config/dex-config.yaml`).

All DTaaS functionality except the DevOps features is available through the single page client.

Documentation

See https://into-cps-association.github.io/DTaaS/development/index.html for the complete documentation.

Copyright © 2022 - 2026 The INTO-CPS Association

OAuth Scope Notes

Scopes in local password database mode:

- Supported: `openid profile email groups offline_access`
- Not supported by the Dex local password database: `read_user read_repository api`

References

Image sources:

Traefik logo: https://www.laub-home.de/wiki/

Dex IdP: https://dexidp.io/

Copyright © 2022 - 2026 The INTO-CPS Association

Secure Server

INSTALL

Digital Twin as a Service

This guide explains how to deploy the workspace application for **secure multi-user deployments with Keycloak** for authentication.

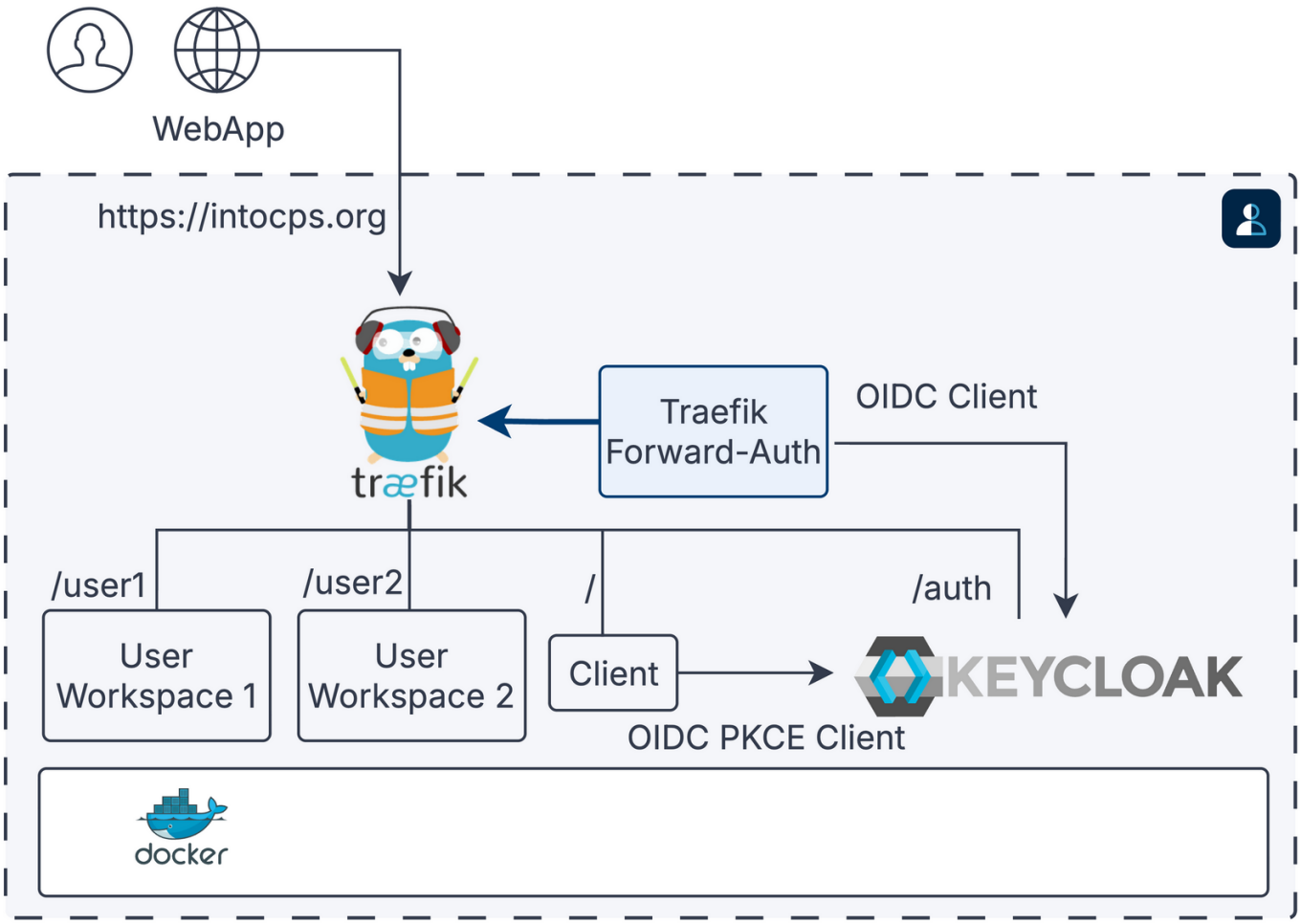
Prerequisites

- Docker Engine v27 or later
- Sufficient system resources (at least 2GB RAM per workspace instance)
- Ports 80 and 443 available on the host
- Valid TLS certificates
- A domain name pointing to the server

Download **workspace-keycloak-production-xx.zip** from the latest release / pre-release page.

Design

The installation setup is shown here.



Copyright © 2022 - 2026 The INTO-CPS Association

A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

User Directories

Every deployment option needs user directories for the workspace files. Create them for the `USERNAME1` and `USERNAME2` values set in `config/.env`.

```
# create required files
cp -R files/template files/<USERNAME1>
cp -R files/template files/<USERNAME2>
# set file permissions for use inside the container
sudo chown -R 1000:100 files
```

Configuration

Follow the pre-installation steps in `configuration.md` to create a valid configuration.

Start the application:

```
docker compose --env-file config/.env up -d
```

Temporary Issues

The following startup issues are expected at this stage and go away after the post-installation steps:

- The `traefik-forward-auth` service keeps restarting.
- Visiting `https://intocps.org` shows `HTTP ERROR 500`.

Now complete the post-installation steps in `configuration.md`, then restart `traefik-forward-auth`:

```
docker compose --env-file config/.env up -d --force-recreate traefik-forward-auth
```

The application is now available at https://intocps.org. Log in with the user credentials set in **Keycloak**.

Stop the application:

```
docker compose --env-file config/.env down
```

To stop and remove volumes:

```
docker compose --env-file config/.env down -v
```

Customisation

Adding More Users

Create a user account for USERNAME3 in **Keycloak**.

Create the user's workspace directory:

```
cp -r files/template files/<USERNAME3>
sudo chown -R 1000:100 files/<USERNAME3>
```


Copyright © 2022 - 2026 The INTO-CPS Association

Add a new service in `docker-compose.yml`:

```
  user3:
    image: intocps/workspace:main-56c6f68
    restart: unless-stopped
    environment:
      - MAIN_USER=${USERNAME3:-user3}
    volumes:
      - "./files/common:/workspace/common"
      - "./files/${USERNAME3:-user3}:/workspace"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.u3.entryPoints=web-secure"
      - "traefik.http.routers.u3.rule=Host(`${SERVER_DNS:-localhost}`) && PathPrefix(`/${USERNAME3:-user3}`)"
      - "traefik.http.routers.u3.tls=true"
      - "traefik.http.routers.u3.middlewares=traefik-forward-auth"
    networks:
      - users
```

Add the `USERNAME3` variable to `config/.env`:

```
# Username Configuration
# These usernames will be used as path prefixes for user workspaces
# Example: https://intocps.org/user1, https://intocps.org/user2
USERNAME1=user1
USERNAME2=user2
USERNAME3=user3 # <--- replace "user3" with your desired username
```

Add a Forward Auth rule for user3 to `config/forward-auth-conf`:

```
rule.user3_access.action=auth
rule.user3_access.rule=PathPrefix(`/user3`)
rule.user3_access.whitelist = user3@localhost
```

Troubleshooting

Certificate Issues

Problem: "NET::ERR_CERT_INVALID" in the browser

Solutions:

- Verify that the certificate files exist in the `./certs/` directory
- Check the certificate file permissions
- For self-signed certificates, add a security exception in the browser

OAuth2 Issues

Problem: Redirect loop after OAuth2 login

Solutions:

- Verify that the OAuth2 callback URL matches `https://intocps.org/_oauth`
- Check that the `SERVER_DNS` environment variable is set correctly
- Ensure that `SERVER_DNS` is correct (`COOKIE_DOMAIN` is derived from it)
- Verify that the OAuth2 application is approved and active

Service Access Issues

Problem: Cannot access the workspace after authentication

Solutions:

- Check service health: `docker compose --env-file config/.env ps`
- View logs: `docker logs`
- Verify Traefik routes: `docker compose --env-file config/.env logs traefik`
- Test the OAuth2 service: `docker compose --env-file config/.env logs traefik-forward-auth`

Copyright © 2022 - 2026 The INTO-CPS Association

Port Conflicts

Problem: Ports 80 or 443 already in use

Solutions:

- Check for other services: `sudo netstat -tlnp | grep -E ':(80|443)'`
- Stop conflicting services
- Or change the port mappings in the compose file (not recommended for production)

Additional Resources

- Traefik Documentation
- Traefik Forward Auth
- Let's Encrypt Documentation
- Docker Compose Documentation
- OAuth 2.0 Specification

Copyright © 2022 - 2026 The INTO-CPS Association

DTAAS CONFIGURATION

This document describes the configuration needed for the docker compose file. It has pre-install and post-install parts. Complete the pre-install tasks before bringing up the docker compose services, and the post-install tasks after.

Pre-install Configuration Tasks:

- Environment
- Domain
- TLS Certificates
- Usernames
- Forward Auth

Post-install Configuration Tasks:

- Keycloak Integration
- Web Client

Environment

The compose commands in the setup guides read their environment from an environment file. An example of this file is at `config/.env.example`.

Create a copy of this example file without the example suffix:

```
cp config/.env.example config/.env
```

Domain

Decide whether the deployment is local or remote.

Wherever `<DOMAIN_NAME>` or `intocps.org` appears in this guide, replace it with the domain name of the remote machine. The remote machine must have a domain name and be reachable from the internet.

Open `config/.env` and replace the current value of the `SERVER_DNS` variable with the domain name:

```
# Server Configuration
# Replace with the actual domain name
SERVER_DNS=<DOMAIN_NAME>
```

TLS Certificates

Ensure that valid TLS certificates are present on the machine. The `fullchain.pem` and `privkey.pem` files must be in the `certs/` directory. Traefik reads them through `config/tls.yml`.

There are several ways to obtain TLS certificates. On a web server you can use Certbot from Let's Encrypt:

```
# Install certbot
sudo apt-get update
sudo apt-get install certbot

# Generate certificates
sudo certbot certonly --standalone -d <DOMAIN_NAME>

# Copy certificates to the project
sudo cp /etc/letsencrypt/live/<DOMAIN_NAME>/fullchain.pem ./certs/
sudo cp /etc/letsencrypt/live/<DOMAIN_NAME>/privkey.pem ./certs/
sudo chown $USER:$USER ./certs/*.pem
chmod 644 ./certs/fullchain.pem
chmod 600 ./certs/privkey.pem
```


Copyright © 2022 - 2026 The INTO-CPS Association

Usernames

The usernames of the main workspace users are set in the environment variable file `config/.env`. Change the default values (`user1` and `user2`) to the desired usernames:

```
# Username Configuration
# These usernames will be used as path prefixes for user workspaces
# Example: https://intocps.org/user1, https://intocps.org/user2
USERNAME1=user1
USERNAME2=user2
```

NOTE: These usernames must match the names of the Keycloak users used in the forward auth configuration.

Traefik Forward Auth Configuration

The `config/forward-auth-conf.example` file contains an example configuration for the forward-auth service.

Create a copy of this example file without the example suffix:

```
cp config/forward-auth-conf.example config/forward-auth-conf
```

Then update the Traefik forward auth configuration file with the usernames and emails of the Keycloak users that correspond to `user1` and `user2`.

```
rule.user1_access.action=auth
rule.user1_access.rule=PathPrefix(`/<USERNAME_USER1>`)
rule.user1_access.whitelist = <EMAIL_USER1>

rule.user2_access.action=auth
rule.user2_access.rule=PathPrefix(`/<USERNAME_USER2>`)
rule.user2_access.whitelist = <EMAIL_USER2>
```

NOTE: The usernames set in the Usernames configuration step must be the same as those in the Traefik forward auth configuration file.

Keycloak Integration

The default `docker-compose.yml` uses **Keycloak** for authentication through OIDC (OpenID Connect). The `traefik-forward-auth` and DTaaS `client` docker services use **Keycloak** for authentication and authorisation. Each one needs an OAuth2 application in the integrated **Keycloak** service.

For detailed Keycloak setup instructions, see keycloak-setup.md

Configure Environment Variables

1. Edit `config/.env` and fill in the Keycloak credentials:

```
# Keycloak Admin Credentials
KEYCLOAK_ADMIN=admin
KEYCLOAK_ADMIN_PASSWORD=changeme

# Keycloak Realm
KEYCLOAK_REALM=dtaas

# Keycloak Client Credentials (obtain from Keycloak after creating client)
KEYCLOAK_CLIENT_ID=dtaas-workspace
KEYCLOAK_CLIENT_SECRET=your_client_secret_here

# Keycloak Issuer URL
KEYCLOAK_ISSUER_URL=https://<DOMAIN_NAME>/auth/realms/<REALM>

# Secret key for encrypting OAuth session data
# Generate a random string (at least 16 characters)
# for example, using the following command
# openssl rand -base64 32
OAUTH_SECRET=your-oauth-secret
```

DTaaS Web Client Config

The DTaaS web client is configured with a small JavaScript file. An example is at `config/client.js.example`.

Copyright © 2022 - 2026 The INTO-CPS Association

Create a copy of this example file without the example suffix:

```
cp config/client.js.example config/client.js
```

Then edit the new web client config file and update the following values.

Client OAuth2 Setup

The DTaaS web client is a React SPA that authenticates with Keycloak using the **Authorization Code flow with PKCE**. Follow the Create OAuth2 Client for DTaaS Client Service instructions to create the public PKCE client in Keycloak, then update `config/client.js`:

```
REACT_APP_CLIENT_ID: 'dtaas-client',          // Client ID set in Keycloak
REACT_APP_AUTH_AUTHORITY: 'https://<DOMAIN_NAME>/auth/realms/dtaas',
REACT_APP_REDIRECT_URI: 'https://<DOMAIN_NAME>/Library',
REACT_APP_LOGOUT_REDIRECT_URI: 'https://<DOMAIN_NAME>/',
REACT_APP_GITLAB_SCOPES: 'openid profile',    // OIDC scopes requested from Keycloak
```

`openid` and `profile` are standard OIDC scopes that Keycloak provides by default. `openid` is required for OIDC authentication and issues the ID token. `profile` adds the profile claims, including `preferred_username` and the `profile` claim mapped from the `profile` user attribute that Create Users sets.

The web client takes the username from the first of these claims that is present: `preferred_username`, `username`, `nickname`, `login`, the part of `email` before `@`, the part of `upn` before `@`, the last path segment of the `profile` URL, and `sub`. The `profile` URL is also shown as the user's profile link. The variable is named `REACT_APP_GITLAB_SCOPES` for historical reasons; it carries the Keycloak OIDC scopes.

Replace `<DOMAIN_NAME>` with the value set in the Domain section, and replace `dtaas` with the Keycloak realm name if you chose a different one.

Additional Resources

- Let's Encrypt Documentation
- Docker Compose Documentation
- OAuth 2.0 Specification

Copyright © 2022 - 2026 The INTO-CPS Association

KEYCLOAK SETUP GUIDE FOR DTAAS

This guide explains how to configure Keycloak for authentication in the DTaaS workspace deployment.

Overview

Keycloak uses the OIDC/OAuth2 standards and supports single sign-on, multi-factor authentication and user federation. The deployment is configured through environment variables.

The configuration uses:

- **Keycloak** as the identity provider (IdP) with OIDC support
- **Traefik Forward Auth** to protect routes using OIDC
- **Traefik** as the reverse proxy

Architecture

```
User Request → Traefik → Forward Auth → Keycloak (OIDC)
                  ↓
          Protected Service
```

Quick Start

This guide uses either `<DOMAIN_NAME>` or `intocps.org` as a placeholder for the server hostname of the installation.

1. Configure Environment Variables

The Keycloak-specific environment variables are:

Variable	Purpose	Example
`KEYCLOAK_ADMIN`	Admin username	`admin`
`KEYCLOAK_ADMIN_PASSWORD`	Admin password	`changeme`
`KEYCLOAK_REALM`	Realm name	`dtaas`
`KEYCLOAK_CLIENT_ID`	OIDC client ID	`dtaas-workspace`
`KEYCLOAK_CLIENT_SECRET`	OIDC client secret	`<from-Keycloak>`
`KEYCLOAK_ISSUER_URL`	OIDC issuer URL	`https://intocps.org/auth/realms/dtaas`

`KEYCLOAK_REALM` is informational: `docker-compose.yml` does not use it. The realm that DTaaS uses is the one named at the end of `KEYCLOAK_ISSUER_URL` (`/realms/<realm>`).

Edit the Keycloak configuration in `config/.env` :

```
# Keycloak Admin Credentials (for initial setup)
KEYCLOAK_ADMIN=admin
KEYCLOAK_ADMIN_PASSWORD=changeme

# Keycloak Realm (informational; the realm in KEYCLOAK_ISSUER_URL is used)
KEYCLOAK_REALM=dtaas

# Keycloak Client Configuration (will be created in post-install step)
KEYCLOAK_CLIENT_ID=dtaas-workspace
KEYCLOAK_CLIENT_SECRET=<generated-secret>
```

2. Configure Keycloak

The following instructions are part of the post-install step.

Copyright © 2022 - 2026 The INTO-CPS Association

Access Keycloak Admin Console

1. Go to `https://intocps.org/auth`
2. Click **Administration Console**
3. Log in with the credentials from the `config/.env` file (default: `admin` / `changeme`)

Create a Realm

1. In the top-left dropdown (currently showing "Master"), click **Create Realm**
2. **Realm name**: `dtaas` (or match the realm named in `KEYCLOAK_ISSUER_URL` in `config/.env`)
3. Click **Create**
4. Click **Realm Settings** -> **User Profile**
5. Click **Create attribute** with
6. name: profile
7. display name: profile
8. Permission: **who can edit** selected on admin
9. Click **Create**

Create Users

1. In the left sidebar, click **Users**
2. Click **Create new user**
3. Fill in the user details:
4. **Username**: `user1` (or the desired username)
5. **Email**: the user's email
6. **First name** / **Last name**: required
7. **Email verified**: OFF
8. **Profile**: `https://intocps.org/auth/realms/dtaas/<USERNAME>` The profile URL is the `<REACT_APP_AUTH_AUTHORITY>` URL from `client.js` with `/<USERNAME>` appended
9. Click **Create**
10. Set a password:
11. Go to the **Credentials** tab
12. Click **Set password**
13. Enter a password
14. **Temporary**: OFF (so users do not have to change it on first login)
15. Click **Save**
16. Repeat for additional users (e.g., `user2`)

Copyright © 2022 - 2026 The INTO-CPS Association

Create OAuth2 Client for Traefik Forward-Auth Service

1. In the left sidebar, click **Clients**
2. Click **Create client**
3. Configure the client:
4. **Client type**: OpenID Connect
5. **Client ID**: `dtaas-workspace` (match `KEYCLOAK_CLIENT_ID` in `config/.env`)
6. Click **Next**
7. Capability config:
8. Client authentication: ON
9. Authorization: OFF
10. Authentication flow: enable **Standard flow**
11. Click **Next**
12. Login settings:
13. **Root URL**: `https://intocps.org`
14. **Valid redirect URIs**:
    - `https://intocps.org/_oauth/*`
    - `https://intocps.org/*`
15. **Valid post logout redirect URIs**: `https://intocps.org/*`
16. **Web origins**: `https://intocps.org`
17. Click **Save**
18. Get the client secret:
19. Go to the **Credentials** tab
20. Copy the **Client secret** value
21. Update `KEYCLOAK_CLIENT_SECRET` in the `config/.env` file

Create OAuth2 Client for DTaaS Client Service

Copyright © 2022 - 2026 The INTO-CPS Association

The DTaaS web client is a React single-page application (SPA) that uses the **Authorization Code flow with PKCE** (Proof Key for Code Exchange). It needs a **public** client (no client secret) with PKCE enforced.

1. In the left sidebar, click **Clients**
2. Click **Create client**
3. Configure the client:
4. **Client type**: OpenID Connect
5. **Client ID**: `dtaas-client`
6. Click **Next**
7. Capability config:
8. **Client authentication**: OFF (public client, no secret needed)
9. **Authorization**: OFF
10. **Authentication flow**: enable **Standard flow** only
11. Click **Next**
12. Login settings:
13. **Root URL**: `https://intocps.org`
14. **Valid redirect URIs**: `https://intocps.org/*`
15. **Valid post logout redirect URIs**: `https://intocps.org/*`
16. **Web origins**: `https://intocps.org`
17. Click **Save**
18. Enforce PKCE:
19. Go to the **Advanced** tab of the client
20. Under **Advanced Settings**, set **Proof Key for Code Exchange Code Challenge Method** to `S256`
21. Click **Save**
22. Verify scopes:
23. Go to the **Client Scopes** tab of the client
24. Confirm that `profile` is listed as an assigned scope

3. Restart Services

After configuring Keycloak, restart the services to apply the new client secret:

```
docker compose --env-file config/.env down
docker compose --env-file config/.env up -d
```

4. Test Authentication

1. Go to `https://intocps.org`
2. The browser should redirect to the Keycloak login page
3. Log in with one of the created users
4. The browser should redirect back to the DTaaS landing page

Production Considerations

To use an external Keycloak instance (recommended for production):

1. Update `KEYCLOAK_ISSUER_URL` in `config/.env`:

```
KEYCLOAK_ISSUER_URL=https://keycloak.intocps.org/auth/realms/dtaas
```

1. Update the client redirect URIs in Keycloak to use the production domain.

Copyright © 2022 - 2026 The INTO-CPS Association

Secure Credentials

- Change the default Keycloak admin password
- Use strong client secrets
- Store secrets securely (Docker secrets or external secret managers)
- Rotate secrets regularly

Database Backend

For production, configure Keycloak with a database such as PostgreSQL or MySQL:

```
keycloak:
  environment:
    - KC_DB=postgres
    - KC_DB_URL=jdbc:postgresql://postgres:5432/keycloak
    - KC_DB_USERNAME=keycloak
    - KC_DB_PASSWORD=secure_password
```

Troubleshooting

Cannot Access Keycloak Admin Console

- Ensure that the Keycloak service is running: `docker compose --env-file config/.env ps`
- Check the Keycloak logs: `docker compose --env-file config/.env logs keycloak`
- Verify that ports 80 and 443 are reachable

Authentication Loop/Redirect Issues

- Verify that `KEYCLOAK_ISSUER_URL` contains the correct realm name
- Ensure that the redirect URIs in the Keycloak client include `/_oauth/*`
- Confirm that `SERVER_DNS` is correct (`COOKIE_DOMAIN` is derived from it)
- Clear the browser cookies and retry

"Invalid Client" Error

- Verify that `KEYCLOAK_CLIENT_ID` matches the client ID in Keycloak
- Ensure that `KEYCLOAK_CLIENT_SECRET` is correct
- Confirm that client authentication is enabled for the client

Forward Auth Not Working

- Check the traefik-forward-auth logs: `docker compose --env-file config/.env logs traefik-forward-auth`
- Verify that the environment variables are set correctly
- Ensure that Keycloak is reachable from the traefik-forward-auth container

Advanced Configuration

Custom Claims and Scopes

To access custom user attributes:

1. In Keycloak, create client scopes with mappers
2. Assign the scopes to the client
3. Configure traefik-forward-auth to request the additional scopes

Role-Based Access Control (RBAC)

Keycloak supports RBAC, but the traefik-forward-auth service does not use it by default.

Single Sign-On (SSO)

Copyright © 2022 - 2026 The INTO-CPS Association

Keycloak supports SSO across multiple applications. Configure additional clients for other services as needed.

References

- Keycloak Documentation
- Traefik Forward Auth
- OIDC Specification

Copyright © 2022 - 2026 The INTO-CPS Association

WORKSPACE WITH TRAEFIK, OAUTH2, AND TLS

This guide explains how to deploy the workspace container with the Traefik reverse proxy, OAuth2 authentication and TLS/HTTPS support for secure multi-user deployments.

Prerequisites

- Docker Engine v27 or later
- Docker Compose v2.x
- Sufficient system resources (at least 1GB RAM per workspace instance)
- Valid TLS certificates (production) or self-signed certificates (testing)
- A domain name pointing to the server (production) or localhost (testing)

Design

```
User Request → Traefik → Forward Auth → Keycloak (OIDC)
                  ↓
          User Workspace
```

The `docker-compose.yml` file sets up:

- **Traefik** reverse proxy with TLS termination (ports 80, 443) and an automatic HTTP to HTTPS redirect
- **Keycloak** identity provider, served under `/auth`
- **OAuth2 authentication** through traefik-forward-auth
- the **DTaaS web client**
- **Workspace containers** for each user (user1, user2) behind authentication, using the workspace image
- **Two Docker networks**, `dtaas-frontend` and `dtaas-users`, which separate the frontend services from the user workspaces

Initial Configuration

Follow the steps in `configuration.md` to create a suitable configuration.

Create Workspace Files

Every deployment option needs user directories for the workspace files. Create them for the `USERNAME1` and `USERNAME2` values set in `config/.env`.

```
# create required files
cp -R files/template files/<USERNAME1>
cp -R files/template files/<USERNAME2>
# set file permissions for use inside the container
sudo chown -R 1000:100 files
```

Start Services

To start all services with TLS:

```
docker compose --env-file config/.env up -d
```

This:

1. starts the Traefik reverse proxy with TLS on ports 80 (HTTP → HTTPS redirect) and 443 (HTTPS)
2. starts Keycloak and the DTaaS web client
3. starts the traefik-forward-auth service for OAuth2 authentication
4. starts the workspace instances for user1 and user2, protected by authentication

Accessing Workspaces

Once all services are running, open the workspaces through Traefik over HTTPS:

Copyright © 2022 - 2026 The INTO-CPS Association

User Workspace (workspace)

- **VNC Desktop**: `https://intocps.org/user1/tools/vnc?path=user1%2Ftools%2Fvnc%2Fwebsockify`
- **VS Code**: `https://intocps.org/user1/tools/vscode`
- **Jupyter Notebook**: `https://intocps.org/user1`
- **Jupyter Lab**: `https://intocps.org/user1/lab`

Service Discovery

The workspace has a `/services` endpoint that returns a JSON list of the available services, so frontend applications can discover them.

Example: Get the service list for user1

```
curl https://intocps.org/user1/services
```

Response:

```
{
  "desktop": {
    "name": "Desktop",
    "description": "Virtual Desktop Environment",
    "endpoint": "tools/vnc?path=user1%2Ftools%2Fvnc%2Fwebsockify"
  },
  "vscode": {
    "name": "VS Code",
    "description": "VS Code IDE",
    "endpoint": "tools/vscode"
  },
  "notebook": {
    "name": "Jupyter Notebook",
    "description": "Jupyter Notebook",
    "endpoint": ""
  },
  "lab": {
    "name": "Jupyter Lab",
    "description": "Jupyter Lab IDE",
    "endpoint": "lab"
  }
}
```

The endpoint values are filled in with the username from the `MAIN_USER` environment variable. For `user1`, this variable is set from `USERNAME1` in `config/.env`.

Authentication Flow

1. The user opens a workspace URL.
2. Traefik forwards the request to traefik-forward-auth.
3. If the user is not authenticated, they are redirected to the OAuth2 provider.
4. The user logs in with the OAuth2 provider.
5. The provider redirects back with an authorisation code.
6. traefik-forward-auth exchanges the code for a token and creates a session.
7. The user is redirected to the original URL and gains access.

Stopping Services

To stop all services:

```
docker compose --env-file config/.env down
```

To stop and remove volumes:

```
docker compose --env-file config/.env down -v
```


Copyright © 2022 - 2026 The INTO-CPS Association

Customisation

Adding More Users

To add more workspace instances, add a new service in `docker-compose.yml`:

```
  user3:
    image: intocps/workspace:main-56c6f68
    restart: unless-stopped
    environment:
      - MAIN_USER=${USERNAME3:-user3}
    volumes:
      - "./files/common:/workspace/common"
      - "./files/${USERNAME3:-user3}:/workspace"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.u3.entryPoints=web-secure"
      - "traefik.http.routers.u3.rule=Host(`${SERVER_DNS:-localhost}`) && PathPrefix(`/${USERNAME3:-user3}`)"
      - "traefik.http.routers.u3.tls=true"
      - "traefik.http.routers.u3.middlewares=traefik-forward-auth"
    networks:
      - users
```

Add the `USERNAME3` variable to `config/.env`:

```
# Username Configuration
# These usernames will be used as path prefixes for user workspaces
# Example: https://intocps.org/user1, https://intocps.org/user2
USERNAME1=user1
USERNAME2=user2
USERNAME3=user3 # <--- replace "user3" with your desired username
```

Add a Forward Auth rule for user3 to `config/forward-auth-conf`:

```
rule.user3_access.action=auth
rule.user3_access.rule=PathPrefix(`/user3`)
rule.user3_access.whitelist = user3@localhost
```

The username and email must match the workspace user.

Create the user's directory:

```
cp -r files/template files/<USERNAME3>
sudo chown -R 1000:100 files/<USERNAME3>
```

Bring up the user workspace:

```
docker compose --env-file config/.env up -d --force-recreate user3
```

Troubleshooting

Certificate Issues

Problem: "NET::ERR_CERT_INVALID" in the browser

Solutions:

- Verify that the certificate files exist in the `./certs/` directory
- Check the certificate file permissions
- Ensure that `config/tls.yml` references the correct certificate paths
- For self-signed certificates, add a security exception in the browser

OAuth2 Issues

Problem: Redirect loop after OAuth2 login

Copyright © 2022 - 2026 The INTO-CPS Association

Solutions:

- Verify that the OAuth2 callback URL matches `https://intocps.org/_oauth`
- Check that the `SERVER_DNS` environment variable is set correctly
- Ensure that `SERVER_DNS` is correct (`COOKIE_DOMAIN` is derived from it)
- Verify that the OAuth2 application is approved and active

Port Conflicts

Problem: Ports 80 or 443 already in use

Solutions:

- Check for other services: `sudo netstat -tlnp | grep -E ':(80|443)'`
- Stop conflicting services
- Or change the port mappings in the compose file (not recommended for production)

Additional Resources

- Traefik Documentation
- Traefik Forward Auth
- Let's Encrypt Documentation
- Docker Compose Documentation
- OAuth 2.0 Specification

Copyright © 2022 - 2026 The INTO-CPS Association

4.4 Platform Services

Digital twins running inside DTaaS often need third-party services, and we recommend installing them alongside the platform. *These services can only be installed in secure (TLS) mode.*

The following services can be installed:

- **Influx** time-series database and dashboard service
- **Grafana** visualisation and dashboard service
- **RabbitMQ** AMQP broker and its management interface. The **MQTT plugin** of this broker is enabled, so it can also be used as an **MQTT** broker.
- **MongoDB** database server
- **ThingsBoard** IoT device management and data visualisation platform (with PostgreSQL backend)
- **GitLab** OAuth2 identity provider and Git repository service

4.4.1 DTaaS Services CLI

The `dtaas-services` command-line tool manages the DTaaS platform services: MongoDB, InfluxDB, RabbitMQ, Grafana, ThingsBoard and GitLab.

4.4.2 Features

- Generates a project structure with config and data directories.
- Sets up TLS certificates and file permissions with one command.
- Starts, stops and reports the status of all services.
- Creates user accounts in InfluxDB, RabbitMQ, MongoDB, PostgreSQL, ThingsBoard and GitLab.
- Runs on Linux, macOS and Windows.
- Reads its settings from `config/services.env`.

4.4.3 Prerequisites

- Python 3.10 or higher
- Docker and Docker Compose
- TLS certificates

4.4.4 Installation

Install the package inside a Python virtual environment:

```
pip install dtaas-services
```

This installs the `dtaas-services` command.

To verify the installation:

```
dtaas-services --help
```

Commands follow the form `dtaas-services <noun> <verb>`. The nouns are `project`, `host`, `service` and `user`, listed in the order you use them.

Copyright © 2022 - 2026 The INTO-CPS Association

4.4.5 Quick Start

1. Go to the desired setup location and generate the project structure:

```
dtaas-services project generate
```

This creates: * `config` for the service configuration * `data`, where the services store their data * `log` for the GitLab and ThingsBoard logs * `certs` for the TLS certificates the services need * `compose.services.yml` for the main services, including PostgreSQL * `compose.thingsboard.yml` for ThingsBoard * `compose.gitlab.yml` for GitLab

1. Update `config/services.env` with the environment values:
2. `HOSTNAME`: public hostname of the DTaaS server (used for certificate paths)
3. `CERTS_SRC`: directory that holds the TLS certificates to copy
4. `GITLAB_PORT`: port for the local GitLab instance (default: `8090`)
5. `GITLAB_ROOT_NEW_PASSWORD`: strong password to set for the GitLab root admin
6. `TB_SYSADMIN_NEW_PASSWORD`: new password for the ThingsBoard system admin
7. `TB_TENANT_ADMIN_PASSWORD`: new password for the tenant admin
8. port numbers for each service
9. `USERID` and `GROUPID`: user and group ID that the Grafana container runs as
10. OAuth configuration (optional, to customise the GitLab OAuth app registrations): edit `config/gitlab_oauth.json` to adjust app names, redirect URIs, scopes or confidentiality settings before running `dtaas-services service install -s gitlab`. `dtaas-services project generate` creates this file. To use a different filename, set `OAUTH_APPS=<filename>` in `config/services.env`; the file is still looked up in `config/`.
11. Update `config/credentials.csv` with user accounts (format: `username,password,email`).

Options:

- `--path` Directory to generate the project structure in (default: current directory)
- `--force` Refresh the packaged files of an existing project (see below)

Example:

```
dtaas-services project generate --path /path/to/project
```

Refreshing an existing project

By default an item that already exists is skipped, so re-running the command is safe but picks up nothing new. After upgrading the package, `--force` refreshes the files that ship with it:

```
dtaas-services project generate --force
```

It overwrites the three `compose.*.yml` files and the packaged files under `config/`. It never touches `data/`, `log/` or `certs/`, and it never overwrites these files, which are reported as kept:

- `services.env`, `credentials.csv`, `gitlab_oauth.json`
- `gitlab_tokens.json`, `gitlab_user_tokens.json`, `password.env.current`
- `mongod.conf.secure`, `rabbitmq.conf`, `rabbitmq.enabled_plugins`

The first six hold your settings and live credentials; the last three are service configuration you may have hardened locally. Overwritten files and existing directories keep the permissions they already had, so a tightened `config/` stays tightened. To adopt the packaged version of a kept file, copy it aside, delete it and run the command again.

Review the refreshed files before starting the services, and re-run `dtaas-services host setup` if certificate or service file permissions were affected.

Copyright © 2022 - 2026 The INTO-CPS Association

4.4.6 Usage

Service Setup

After generating the project and configuring the settings, run:

```
dtaas-services host setup
```

This command:

- copies the TLS certificates to the correct locations
- sets up MongoDB certificates and permissions
- sets up InfluxDB certificates and permissions
- sets up RabbitMQ certificates and permissions
- sets up ThingsBoard certificates and permissions

Run the clean command right after it:

```
dtaas-services service clean
```

Managing Services

The `service` commands start, stop, restart, remove and report the status of the services.

Example:

```
dtaas-services service start
```

To act on one service only, name it:

```
dtaas-services service stop -s influxdb
```

Remove services and their volumes:

```
dtaas-services service remove --volumes
# Specify
dtaas-services service remove --volumes -s <service_name>
```

`service status` prints a table by default. For scripting, `--json` prints a JSON array on stdout, one object per container, so services that share a container label are all listed:

```
dtaas-services service status --json
```

```
[
  { "service": "gitlab", "container": "gitlab", "status": "running" },
  { "service": "grafana", "container": "grafana", "status": "removed" }
]
```

Failures are still reported as text on stderr, so a script can read stdout only.

Note: `-s` accepts a comma-separated list. A selector that names no service, such as `-s ""` from an unset shell variable, is an error rather than a silent "all services", so a mistyped selector cannot remove or clean more than you intended. Omit the flag entirely to act on all services.

User Account Management

1. Edit `config/credentials.csv` with user accounts (format: `username,password,email`).

Copyright © 2022 - 2026 The INTO-CPS Association

2. Add users to all supported services:

```
dtaas-services user add
```

This creates user accounts in InfluxDB, RabbitMQ, MongoDB, PostgreSQL, ThingsBoard and GitLab. A service that is not running is skipped. For GitLab, a Personal Access Token is created for each new user and saved to `config/gitlab_user_tokens.json`.

3. Add users to a specific service:

```
dtaas-services user add -s rabbitmq
```

4.4.7 ThingsBoard

We recommend ThingsBoard for digital twins that need IoT device management. It can only be installed in secure (TLS) mode.

The ThingsBoard setup uses two services:

- **ThingsBoard**, an IoT device management and data visualisation platform
- **PostgreSQL**, the database server for ThingsBoard

ThingsBoard Installation

Warning: Running `dtaas-services service install -s thingsboard` more than once re-runs the ThingsBoard schema migration against an already populated PostgreSQL database, which can corrupt it. To reinstall from scratch, first run `dtaas-services service clean -s "postgres,thingsboard"` to wipe all data, then re-run the install command. **Note:** Name the service explicitly with `-s <service>` when installing. Run the clean command before the installation.

```
# It starts PostgreSQL if it is not running, and it checks its health.
dtaas-services service install -s thingsboard
```

```
dtaas-services service start -s thingsboard
```

```
# After installation, wait some time before adding users.
# This creates the tenant, tenant admin, and users.
dtaas-services user add -s thingsboard
```

Reset the ThingsBoard sysadmin and tenant admin passwords to the values configured in `config/services.env`:

```
dtaas-services user reset-password -s thingsboard
```

This command:

- changes the sysadmin password from the default (`"sysadmin"`) to `TB_SYSADMIN_NEW_PASSWORD`
- changes the tenant admin password from the default (`"tenant"`) to `TB_TENANT_ADMIN_PASSWORD`

GitLab Installation

Prerequisites:

- The GitLab container joins the `dtaas-services` Docker network (`platform-services` in the compose files), which is created when the other platform services start. Start them first with `dtaas-services service start` before installing GitLab.
- Set `REACT_APP_AUTH_AUTHORITY` in the client config file to `https://<hostname>:<GITLAB_PORT>/gitlab`.

Note: The DTaaS client uses `react-oidc-context`, which forces redirects to use HTTPS. GitLab must therefore be served over HTTPS at `https://<hostname>:<GITLAB_PORT>/gitlab`.

Copyright © 2022 - 2026 The INTO-CPS Association

To install and configure the local GitLab instance:

```
dtaas-services service install -s gitlab
```

GitLab takes 5 to 10 minutes to become healthy after the first start. The install command checks GitLab readiness and returns immediately:

- If GitLab is healthy, the command runs the post-install setup (PAT creation, OAuth app registration and root password reset) and saves the tokens to `config/gitlab_tokens.json`.
- If GitLab is still starting, the command prints a status hint and exits. Check progress with `dtaas-services service status -s gitlab` and re-run `dtaas-services service install -s gitlab` once the status shows `running`.

Note: After a successful setup, `config/gitlab_tokens.json` is backed up to `config/backup_gitlab_tokens.json` and the `root_password` entry is removed from the live tokens file. If the GitLab installation becomes corrupted before the root password is changed, the initial password can be found in the backup file `config/backup_gitlab_tokens.json`. If that file is also missing, re-install GitLab to generate a new initial password. **Warning:** The CLI manages `config/password.env.current`, which tracks the current service passwords. Do **not** edit or delete this file manually; doing so may cause the password reset commands to fail.

To complete the OAuth2 integration with DTaaS and set up GitLab Runner, follow the integration guide and the runner setup guide for either Linux or Windows.

Reset the GitLab root admin password to the value configured in `config/services.env` (`GITLAB_ROOT_NEW_PASSWORD`):

```
dtaas-services user reset-password -s gitlab
```

The command reads the new password from `GITLAB_ROOT_NEW_PASSWORD` and applies it through the GitLab API.

4.4.8 GitLab Post-Install Flow

The `dtaas-services service install -s gitlab` command performs these steps:

1. Starts the GitLab Docker container.
2. Checks whether GitLab is healthy.
3. Reads the auto-generated root password from the container (`/etc/gitlab/initial_root_password`).
4. Creates an initial Personal Access Token.
5. Creates the server and client OAuth applications.
6. Saves the root password, token and OAuth application details to `config/gitlab_tokens.json`.
7. Resets the root password to `GITLAB_ROOT_NEW_PASSWORD`.
8. Backs up the tokens file to `config/backup_gitlab_tokens.json` and removes the root password from `config/gitlab_tokens.json`.

4.4.9 Manual Compose Operations

After generating a services project, you can also run the services directly with the compose files:

- `compose.services.yml`
- `compose.thingsboard.yml`
- `compose.gitlab.yml`

Start manually:

```
docker compose -f compose.services.yml --env-file config/services.env up -d
docker compose -f compose.thingsboard.yml --env-file config/services.env up -d
docker compose -f compose.gitlab.yml --env-file config/services.env up -d
```


Copyright © 2022 - 2026 The INTO-CPS Association

Stop manually:

```
docker compose -f compose.services.yml --env-file config/services.env down
docker compose -f compose.thingsboard.yml --env-file config/services.env down
docker compose -f compose.gitlab.yml --env-file config/services.env down
```

4.4.10 Troubleshooting

Permission Issues (Linux/macOS)

```
sudo -E env PATH="$PATH" dtaas-services host setup
```

Docker connectivity

```
docker ps
```

ThingsBoard connection error

ThingsBoard needs time to initialise after it starts. Wait before adding users or changing passwords.

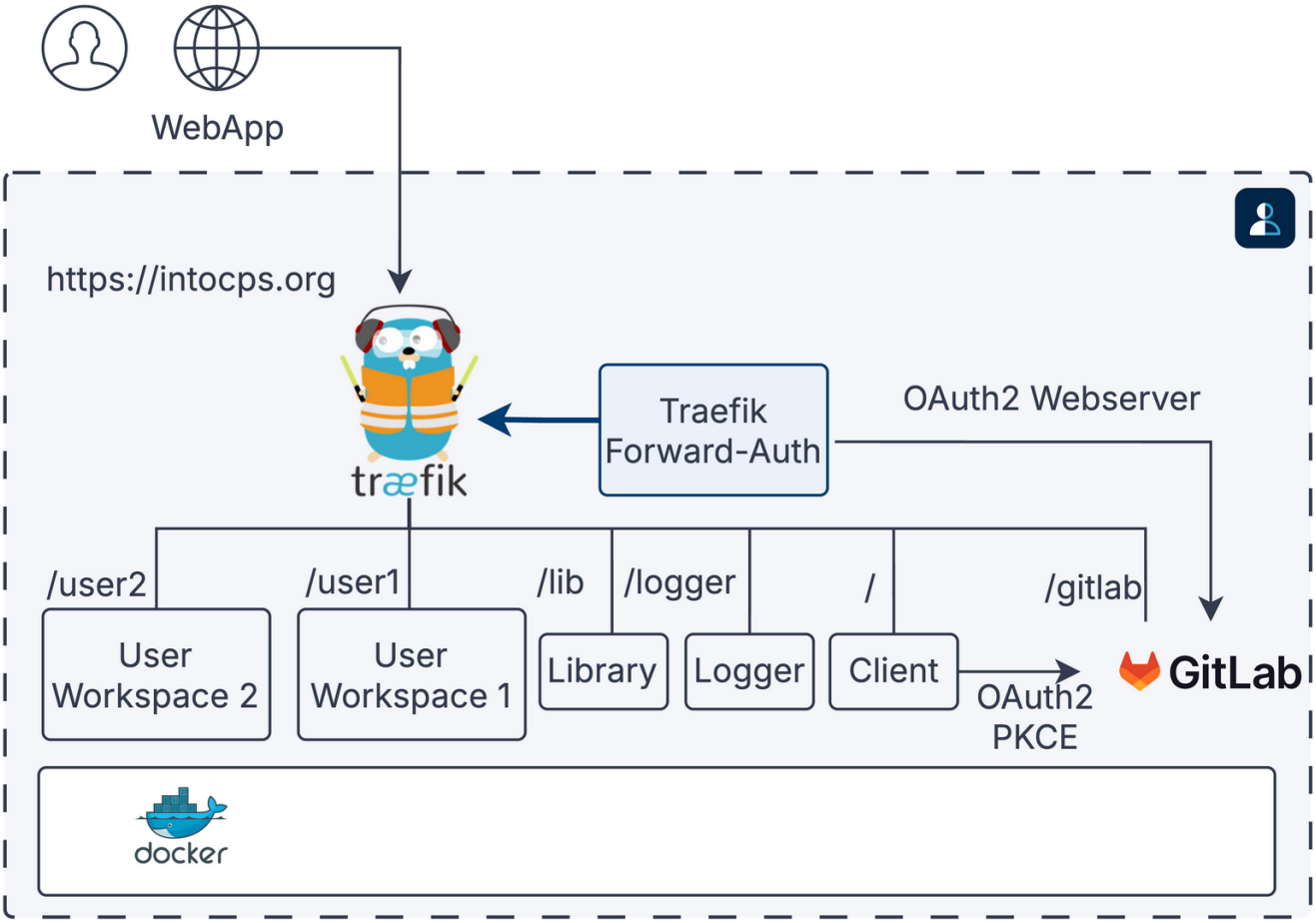
Copyright © 2022 - 2026 The INTO-CPS Association

4.5 GitLab

4.5.1 GitLab Server

DTaaS supports two ways of installing a local GitLab instance.

Option A: Integrated DTaaS Package

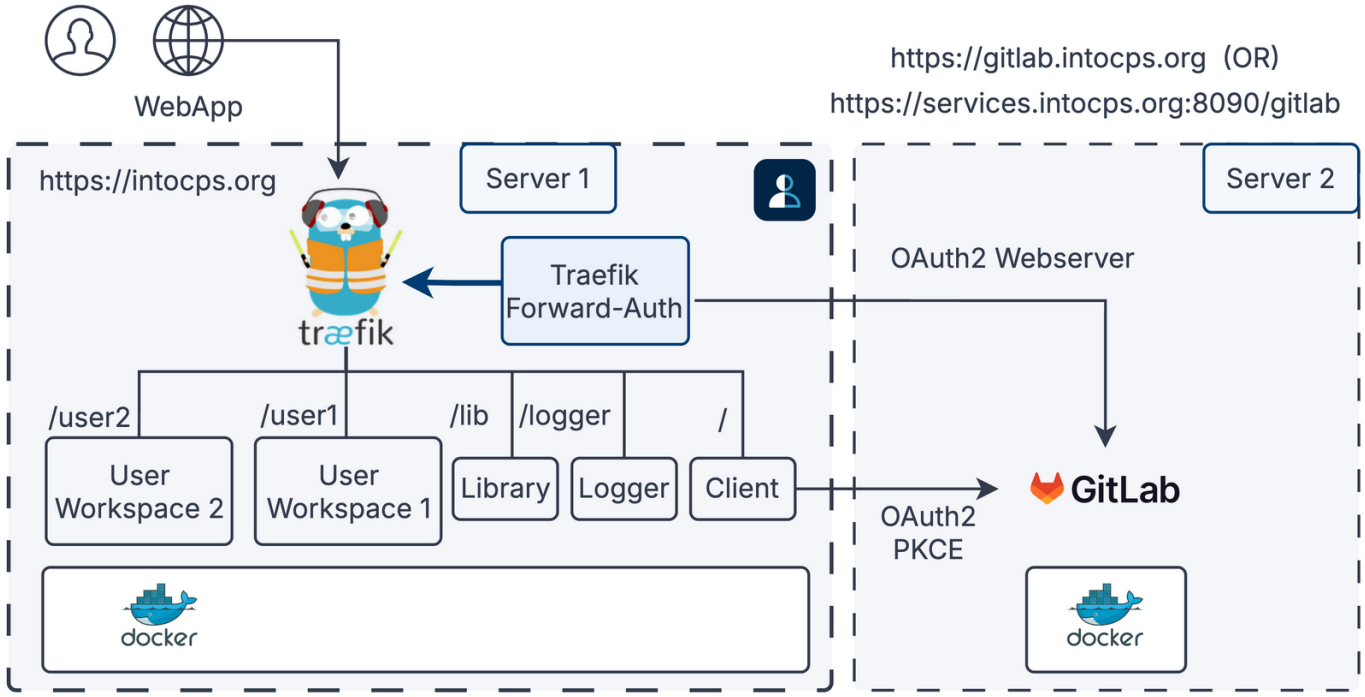


A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

See secure server with GitLab, then continue with the integration guide.

Copyright © 2022 - 2026 The INTO-CPS Association

Option B: Services CLI GitLab Workflow



Server 1 runs DTaaS and server 2 runs GitLab. The routes and the thick arrow mean the same as in Option A.

Use the services CLI if GitLab runs as one of the platform services, then continue with the integration guide.

Notes

- The DTaaS client auth authority must match the actual GitLab endpoint.
- For integrated package deployments, use the package's own `config/` files.
- For services CLI deployments, use the files of the generated services project.

Related

- Integration guide
- Runner setup (Linux)
- Runner setup (Windows)
- DTaaS integrated package config
- Workspace secure server config

Copyright © 2022 - 2026 The INTO-CPS Association

4.5.2 GitLab Integration Guide

This guide connects a local GitLab instance to a DTaaS server installation and sets GitLab up as the OAuth provider for DTaaS.

Afterwards, the GitLab instance is the OAuth provider for both the DTaaS client application and the Traefik forward-auth backend authorisation.

> **Important**
>
> The DTaaS client uses the `react-oidc-context` node package, which incorrectly makes authorisation redirects use the `HTTPS` URL scheme. This is a known issue with the package, and it forces the DTaaS server to use `HTTPS`, either at https://localhost or at https://intocps.org. The rest of this guide uses `intocps.org` to stand for either localhost or a custom domain.

Integration Paths

There are two integration paths:

1. **Integrated package**: see the installation page of the `dtaas-secure-server-integrated-gitlab-xx.zip` package. GitLab is then available over HTTPS at `https://intocps.org/gitlab`.
2. **Platform Services CLI**: see the CLI docs. GitLab is then available over HTTPS on the configured port, at `https://intocps.org:${GITLAB_PORT}/gitlab` (custom domain, default port 8090).

Integration Steps

1. CREATE OAUTH TOKENS IN GITLAB

Follow these guides to create OAuth application tokens for the backend and the client.

This step produces the credentials of the applications named **DTaaS Server Authorization** and **DTaaS Client Authorization**, which the next step uses.

2. USE VALID OAUTH APPLICATION TOKENS

Put the OAuth application details into the configuration files: the **DTaaS Client Authorization** details go into `config/client.js`, and the **DTaaS Server Authorization** details go into `config/.env`.

3. CREATE USERS

Create GitLab users that match `USERNAME1` / `USERNAME2` in `config/.env`.

4. RESTART SERVICES

```
docker compose --env-file config/.env up -d --force-recreate client traefik-forward-auth
```


Copyright © 2022 - 2026 The INTO-CPS Association

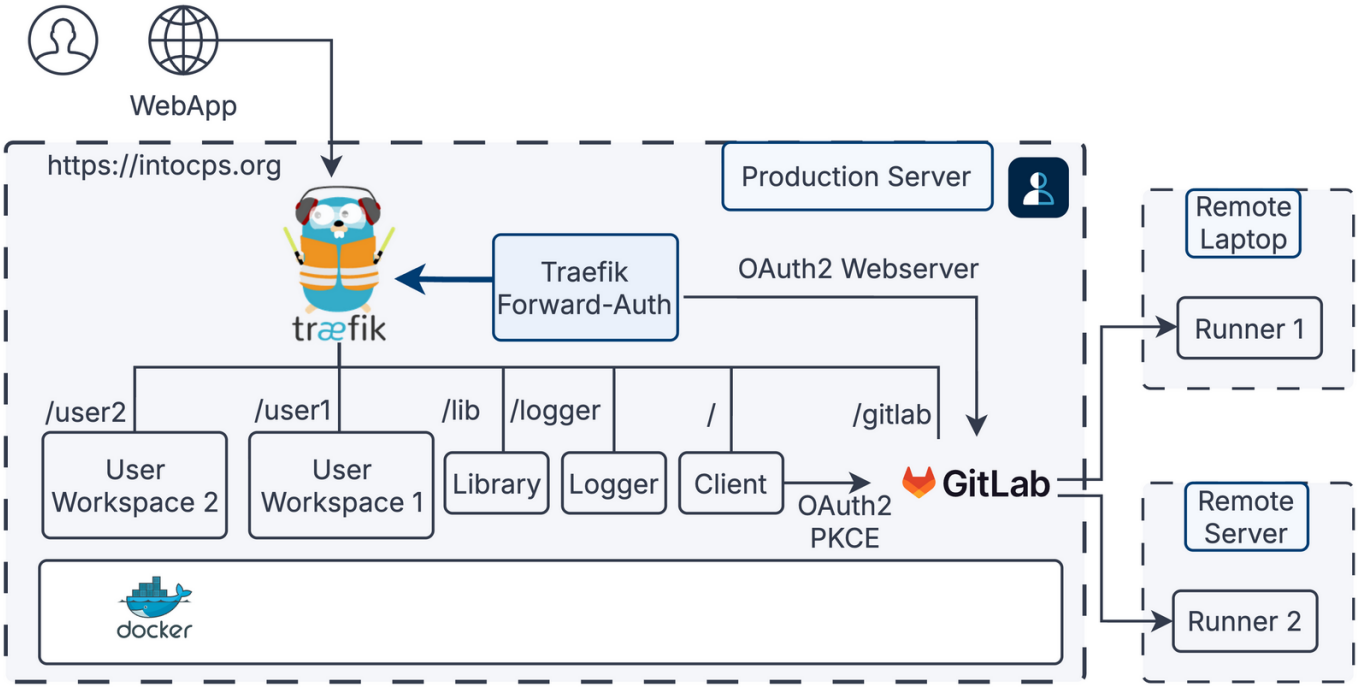
4.5.3 Runner

GitLab Runner Integration

For Windows hosts, see runner-windows.md.

This page creates a `gitlab-runner` that executes digital twins. You can install several such runners and link them to the integrated GitLab.

The diagram shows the intended installation.



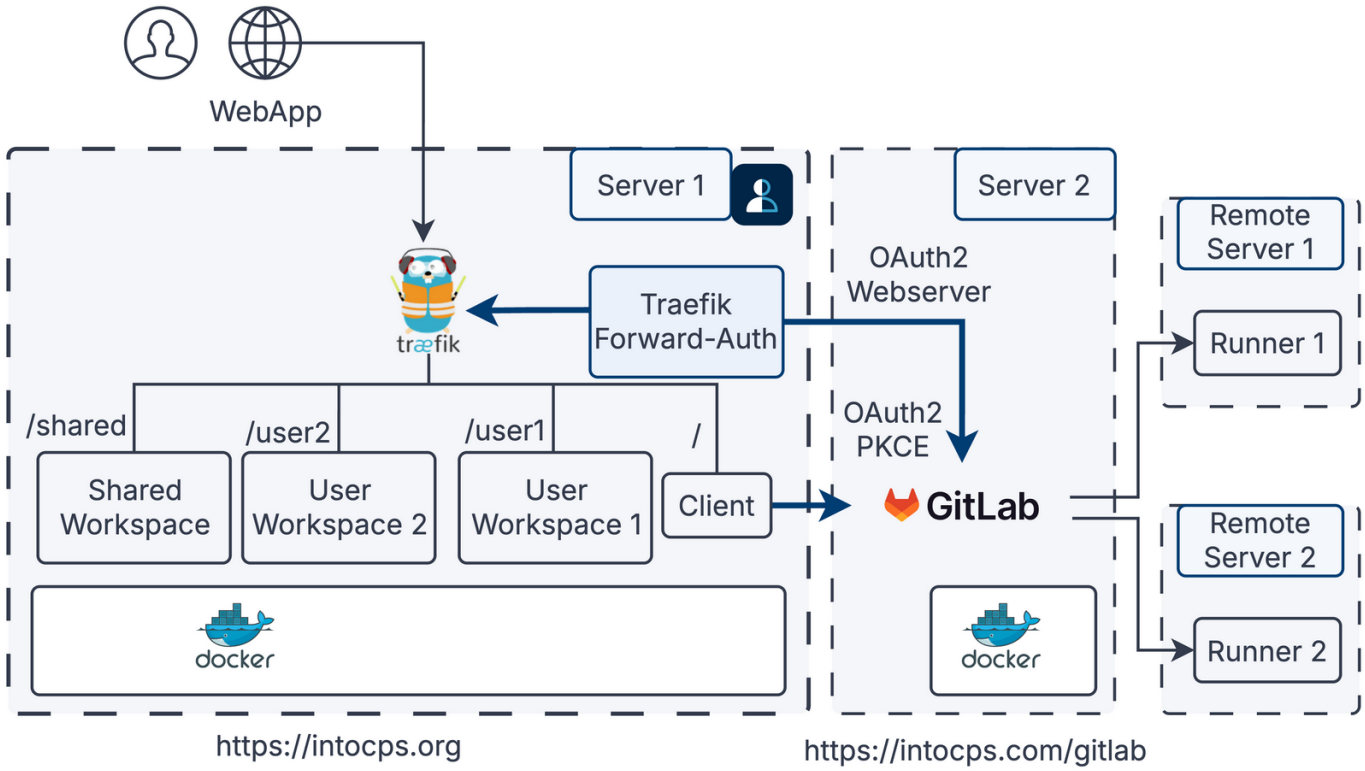
A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

There are two installation scenarios:

1. **Localhost Installation**: the integrated runner runs locally with a GitLab instance hosted at `https://localhost/gitlab`.
2. **Server Installation**: the integrated runner works with a GitLab instance on a production server. That server may be a remote machine rather than the host, and may use TLS with a self-signed certificate.

In a server installation, GitLab can run on a second server, and the runners on further remote servers, as the diagram shows.

Copyright © 2022 - 2026 The INTO-CPS Association



The steps below set up an integrated runner that can execute digital twins from the Digital Twins Preview Page.

PREREQUISITES

A GitLab Runner picks up CI/CD jobs by communicating with a GitLab instance. To set up a GitLab instance that integrates with a DTaaS platform, see the GitLab instance document and the GitLab integration guide.

The rest of this page assumes a running DTaaS platform with a GitLab instance.

RUNNER SCOPES

A GitLab Runner can have one of three scopes:

Runner Scope	Description
Instance Runner	Available to all groups and projects in a GitLab instance.
Group Runner	Available to all projects and subgroups in a group.
Project Runner	Associated with one specific project.

Instance runners are the simplest to set up and are recommended, but any type works. The official GitLab documentation describes the three types.

Copyright © 2022 - 2026 The INTO-CPS Association

OBTAINING A REGISTRATION TOKEN

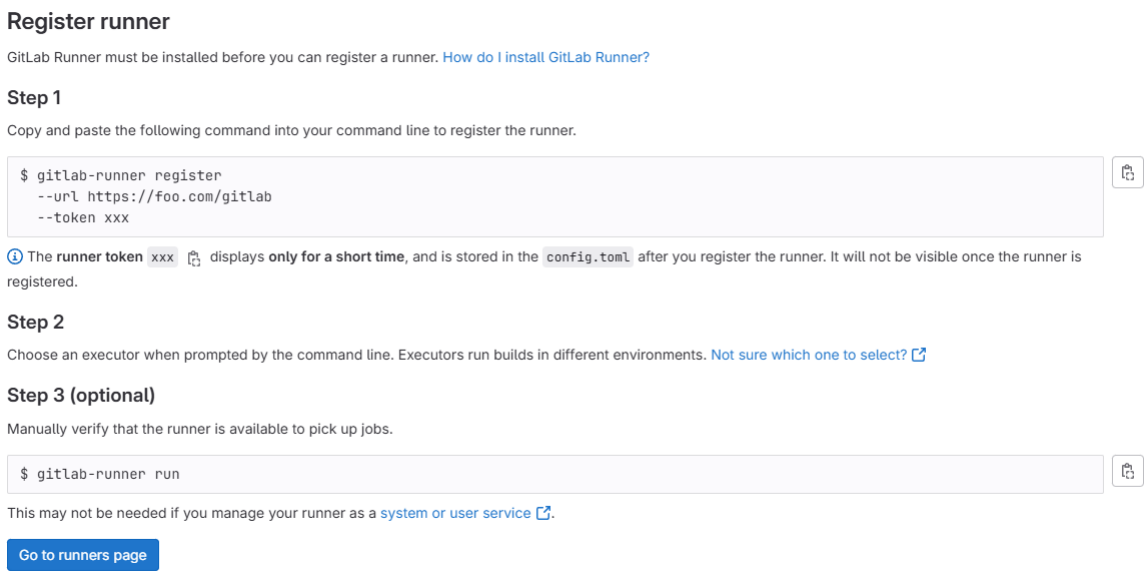
First, obtain the token that registers the runner with the GitLab instance. Open the GitLab instance (remote or local) and follow the steps for the chosen runner scope:

Runner Scope	Steps
Instance Runner	1. On the **Admin** dashboard, navigate to **CI/CD > Runners**. 2. Select **New instance runner**.
Group Runner	1. On the **DTaaS** group page, navigate to **Settings > CI/CD > Runners**. 2. Ensure the **Enable shared runners for this group** option is enabled. 3. On the **DTaaS** group page, navigate to **Build > Runners**. 4. Select **New group runner**.
Project Runner	1. On the **DTaaS** group page, select the project named after the GitLab username. 2. Navigate to **Settings > CI/CD > Runners**. 3. Select **New project runner**.

Each scope then shows a page for creating the runner:

1. Under **Platform**, select the Linux operating system.
2. Under **Tags**, add a `linux` tag.
3. Select **Create runner**.

The following screen appears:



Save the generated runner authentication token.

CONFIGURING THE RUNNER

The runner compose files are in `deploy/services/runner`:

Scenario	Compose file	Variables it reads
Localhost	`compose.runner.local.yml`	`DTAAS_DIR`
Server	`compose.runner.server.yml`	`DTAAS_DIR`, `SERVER_DNS`

Both files mount the TLS certificate of the GitLab host from `${DTAAS_DIR}/deploy/docker/certs/...`. That directory no longer exists in the repository, so edit the certificate volume in the compose file to point to the certificate in use, for example `deploy/dtaas/`

Copyright © 2022 - 2026 The INTO-CPS Association

`docker/secure-server_with_integrated-gitlab/certs/fullchain.pem` for the integrated package. In the server scenario, remove the certificate volume if GitLab does not use a self-signed certificate.

The runner must be registered with the GitLab instance before the two can communicate. `deploy/services/runner/runner-config.toml` contains this template:

```
concurrent = 20

[[runners]]
  name = "dtaas-runner-1"
  description = "dtaas-runner-1"
  url = "https://intocps.org/gitlab/" # Edit this
  token = "xxx" # Edit this
  executor = "docker"
  request_concurrency = 10
  [runners.docker]
    tls_verify = false
    image = "ruby:2.7"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    volumes = ["/cache"]
    network_mode = "host" # Disable this in secure contexts
```

1. Set `url` to the URL of the GitLab instance.
2. Set `token` to the runner authentication token obtained earlier.
3. In the server installation scenario, remove the line `network_mode = "host"`.

The GitLab documentation lists the advanced configuration options.

START THE GITLAB RUNNER

Run these commands from the DTaaS repository directory. Each pair starts and then stops the `gitlab-runner` container.

1. Localhost Installation

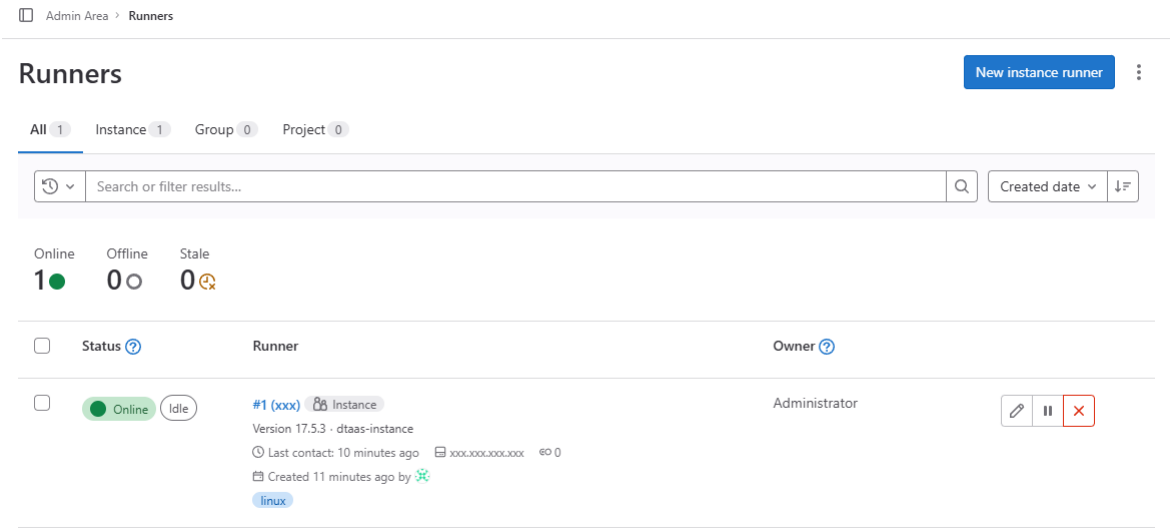
```
export DTAAS_DIR=$(pwd)
docker compose -f deploy/services/runner/compose.runner.local.yml up -d
docker compose -f deploy/services/runner/compose.runner.local.yml down
```

2. Server Installation

```
export DTAAS_DIR=$(pwd)
export SERVER_DNS=intocps.org
docker compose -f deploy/services/runner/compose.runner.server.yml up -d
docker compose -f deploy/services/runner/compose.runner.server.yml down
```

The runner inside the container starts automatically. To check that it works, open the page where the runner was created. An Instance Runner, for example, looks like this:

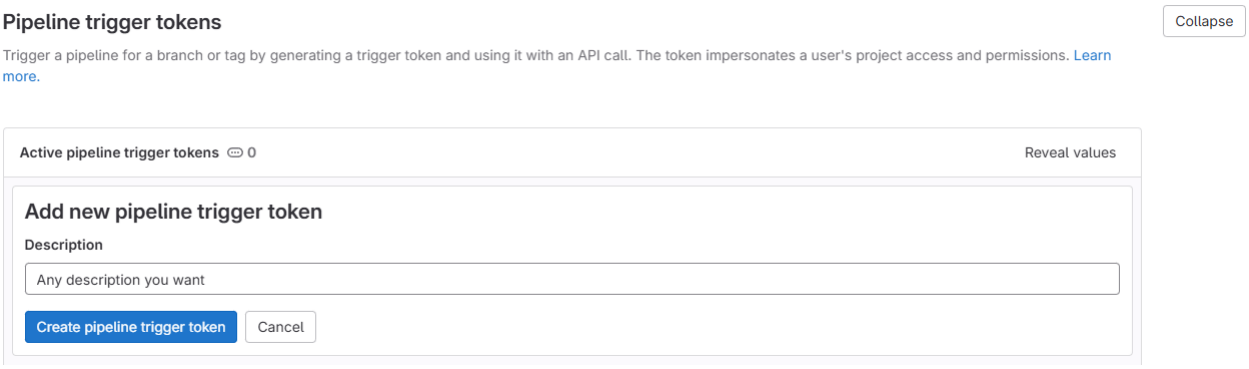
Copyright © 2022 - 2026 The INTO-CPS Association



The GitLab runner is now ready to accept jobs from the GitLab instance.

PIPELINE TRIGGER TOKEN

The Digital Twins Preview Page uses the GitLab API, which requires a Pipeline Trigger Token. Open the project in the **DTaaS** group and navigate to **Settings > CI/CD > Pipeline trigger tokens**. Add a new token with any description.

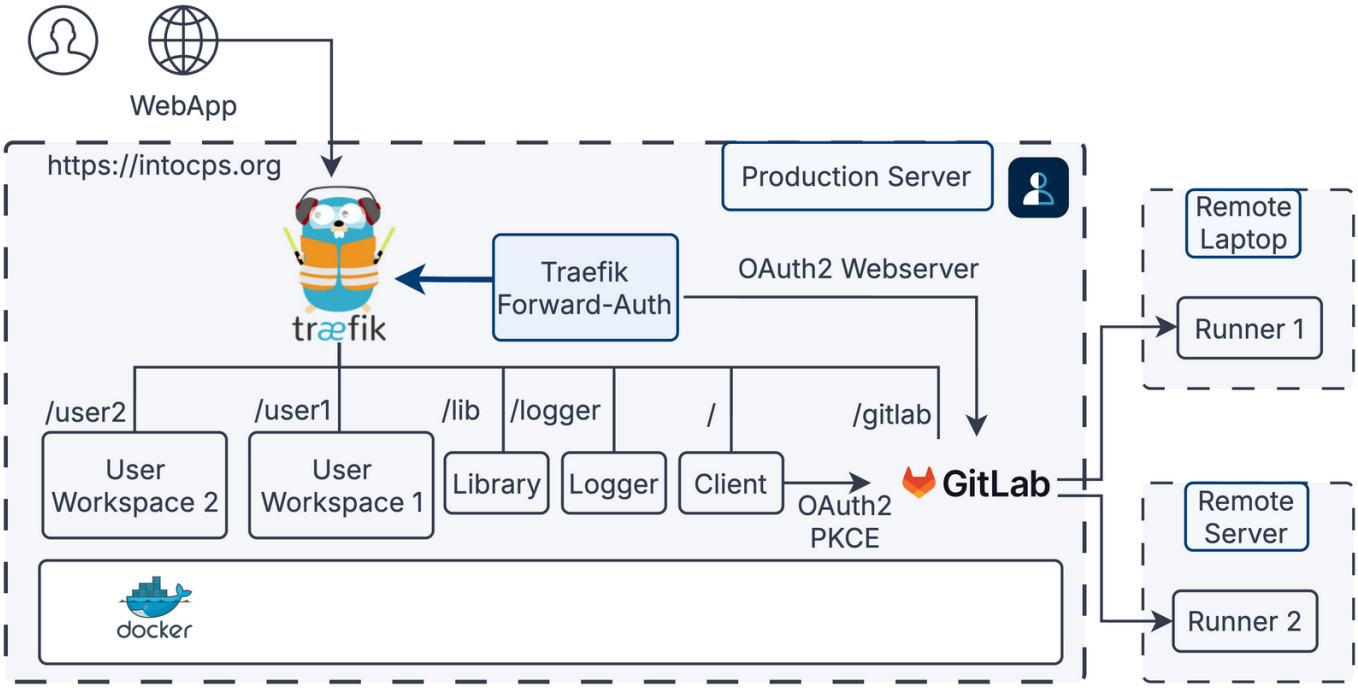


The Digital Twins Preview Page can now manage and execute digital twins.

Copyright © 2022 - 2026 The INTO-CPS Association

Setting up GitLab Runners with Docker on Windows for DTaaS

This guide sets up and configures GitLab runners with Docker on Windows for the DTaaS platform. The diagram shows the intended installation.



A path such as `/user1` is the URL route of that service, for example `https://intocps.org/user1`. The thick arrow is the forward-auth check: Traefik asks traefik-forward-auth to authorise each request before it routes it.

STEP-BY-STEP SETUP PROCESS

1. Install GitLab Runner

Download and install GitLab Runner for Windows from GitLab's official download page.

```
# Navigate to your download directory
cd C:\path\to\download\folder

# Run the GitLab Runner
.\gitlab-runner.exe install

# Start the service
.\gitlab-runner.exe start
```

2. Getting a token

To obtain the GitLab token, open the project page (e.g. https://dtaas-digitaltwin.com/gitlab/dtaas/USERNAME) and:

1. Go to **Settings > CI/CD > Runners**.
2. Press **New Project Runner**.
3. Add the tag `linux`.
4. Leave the other settings unchanged and press **Create Runner**.
5. GitLab displays the token. Save it.

Copyright © 2022 - 2026 The INTO-CPS Association

3. Register the Runner for DTaaS

Register the runner with the URL of the GitLab instance and the token:

```
# Register the runner for DTaaS
.\gitlab-runner.exe register --url "https://intocps.org/gitlab" --token ""

# When prompted, enter:
# - Name: [Your machine name or any preferred name]
# - Executor: docker
# - Default Docker image: ruby:2.7
# - Tags: linux
```

DTaaS digital twins need a Linux environment: their pipelines run shell scripts with commands such as `chmod +x`.

4. Configure the config.toml

Most of the setup happens in the `config.toml` file. It is usually at `C:\Users\YourUsername\.gitlab-runner\config.toml`, or in the directory where the gitlab-runner executable was installed and run.

DTaaS Configuration for Windows Hosts

The recommended configuration for running DTaaS digital twins on Windows is:

```
concurrent = 1
check_interval = 0
connection_max_age = "15m0s"
shutdown_timeout = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "Some name"
  url = "https://dtaas-digitaltwin.com/gitlab"
  id = 3
  token = "YOUR_RUNNER_TOKEN"
  token_obtained_at = 2025-03-17T19:50:25Z
  token_expires_at = 0001-01-01T00:00:00Z
  executor = "docker"
  [runners.custom_build_dir]
    enabled = false
  [runners.cache]
    MaxUploadedArchiveSize = 0
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.feature_flags]
    FF_NETWORK_PER_BUILD = false
  [runners.docker]
    tls_verify = false
    image = "ruby:2.7"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache"]
    shm_size = 0
    network_mtu = 0
  tags = ["linux"]
```

5. Restart the Runner

Restart the runner to apply the configuration:

```
.\gitlab-runner.exe stop
.\gitlab-runner.exe start
```

VERIFYING THE SETUP

Check that the runner is configured correctly:

```
.\gitlab-runner.exe verify
```

A successful check prints something like:

```
Verifying runner... is valid        runner=YourRunnerToken
```


Copyright © 2022 - 2026 The INTO-CPS Association

UNDERSTANDING THE DTAAS PROJECT SETUP

DTaaS runs digital twins as follows:

1. Digital twins live in the `digital_twins` directory.
2. Each digital twin has lifecycle scripts (create, execute, terminate, clean).
3. The GitLab CI/CD pipeline triggers these scripts based on user actions.
4. The scripts need a Linux environment.

COMMON ERRORS

GitLab CI/CD pipelines on Windows with Docker can fail with these errors:

```
ERROR: Failed to remove network for build
ERROR: Job failed: invalid volume specification: "c:\\cache"
```

The usual causes are:

1. An incorrect Docker executor configuration
2. Windows-style paths, which Docker does not accept
3. A mismatch between the runner's executor type and the pipeline requirements

Use the Docker (Linux) executor even on a Windows host, because the digital twin scripts are written for Linux, and write volume paths in Linux form (`/cache`).

Copyright © 2022 - 2026 The INTO-CPS Association

4.6 Independent Packages

4.6.1 Independent Packages

The DTaaS development team publishes reusable packages that together form the complete DTaaS application.

The packages are published on github, npmjs, and docker hub repositories.

The packages on github are published more frequently but are not user tested. The packages on npmjs and docker hub are published at least once per release. Regular users should use the packages from npmjs and docker hub.

The packages are:

Package Name	Description	Documentation for	Availability
dtaas-web	React web application	Useful only for DevOps features. The workspace features are not available in the standalone package.	docker hub and github
libms	Library microservice	npm package	npmjs and github
		container image	docker hub and github
logger-ms	Workflow logger microservice	npm package	npmjs and github
		container image	docker hub and github
runner	REST API wrapper for multiple scripts/ programs	npm package	npmjs and github
workspace	User workspace container image	workspace localhost and workspace secure server	docker hub and `intocps/workspace:*` images referenced by deployment packages

Copyright © 2022 - 2026 The INTO-CPS Association

4.6.2 Workspace Deployment Scenarios

Workspace deployments cover how users reach their workspaces and which identity provider signs them in.

Available Workspace Scenarios

Scenario	Purpose
Dex localhost	Single-user local workspace deployment
Keycloak secure server	Multi-user secure server deployment with OIDC

Writes From the Client, and Jupyter XSRF Protection

The Building Models page writes a converted model back into the user's library through the workspace's own Jupyter Contents API. This is the only place where the web client writes to a workspace.

The Jupyter server can guard writes with a token that it sets as the `_xsrf` cookie and expects echoed in an `X-XSRFToken` header. **The workspace image DTaaS ships sets no such cookie and accepts the write.** The client sends the header when the cookie is readable and leaves it out when it is not. Writes from the client depend on this deployment assumption.

On a workspace that enables the protection, the cookie is `HttpOnly`, so scripts cannot read it, the header is omitted and every write is refused. The page still draws the model; it converts the model again on the next visit instead of loading a stored file. The refusal is logged to the browser console, so the two cases can be told apart.

If you enable XSRF protection on a workspace and want conversions to persist, produce the `.glb` files outside the browser with `ifc-to-glb` from the ifc-utils repository. This is also the better route for a large model.

Notes

- Use the workspace scenarios when you mainly need workspace authentication and per-user routes.
- The DTaaS package deployments include the full DTaaS web platform. See localhost, secure multi-user and secure multi-user with integrated GitLab.

Copyright © 2022 - 2026 The INTO-CPS Association

4.6.3 Library Microservice

Host Library Microservice

The **lib microservice** is a simplified file manager that serves files over GraphQL and HTTP APIs. It lists directory contents, and it uploads and downloads files.

This page explains how to install the npm package of the library microservice and run it as a standalone service.

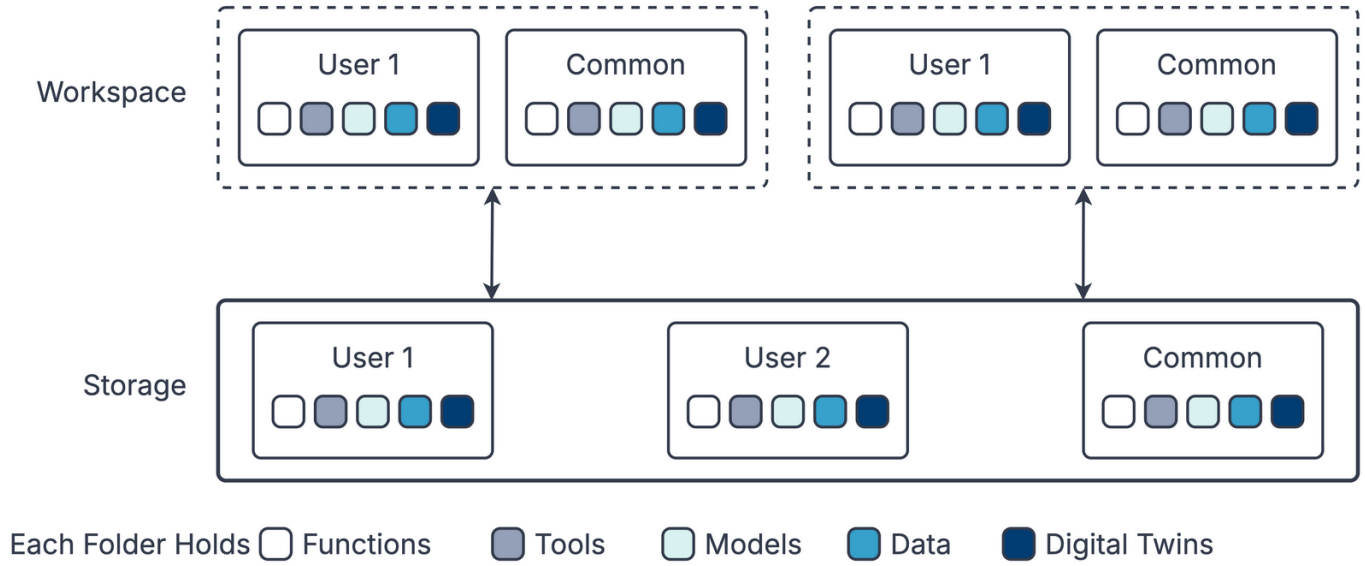
SETUP THE FILE SYSTEM

Outside the DTaaS Platform

The package can be used independently of the DTaaS. In this case no particular file structure is needed; any directory will do.

Inside the DTaaS Platform

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



A skeleton file structure is available in the DTaaS codebase. Copy it to create a file system for users.

INSTALL

The npm package is available in GitHub packages registry and on npmjs. **Prefer the package on npmjs over GitHub**.

Install the package with one of the two following commands.

npmjs

```
sudo npm install -g @into-cps-association/libms  # requires no login
```

GitHub

```
# requires login
sudo npm config set @into-cps-association:registry https://npm.pkg.github.com
sudo npm install -g @into-cps-association/libms
```

The *github package registry* asks for a username and password. The username is the GitHub username and the password is a GitHub personal access token with the *read:packages* scope.

Copyright © 2022 - 2026 The INTO-CPS Association

USE

Display help.

```
$libms -h
Usage: libms [options]

The lib microservice is a file server. It supports file transfer
over GraphQL and HTTP protocols.

Options:
  -c, --config <file>  provide the config file (default libms.yaml)
  -H, --http <file>    enable the HTTP server with the specified config
  -h, --help           display help for libms
```

Neither option is mandatory.

See configuration for the configuration conventions. By convention the configuration is saved in a `libms.yaml` file. If `-c` is not given, **libms** looks for `libms.yaml` in the working directory from which it is run. To run **libms** with that default file, run

```
$libms
```

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

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

If the configuration file has another name, for example `libms-config.yaml`, run

```
$libms -c "config/libms-config.yaml"
```

Press `Ctrl+C` to halt the application. 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.

Protocol Support

libms supports the GraphQL protocol by default. It can also serve files to a browser over HTTP. Enable this with the `-H http.json` flag; the sample http config can be used.

```
$nohup libms [-H http.json] & disown
```

The usual file upload and download options then become available.

SERVICE ENDPOINTS

The GraphQL URL: `localhost:PORT/lib`

The HTTP URL: `localhost:PORT/lib/files`

The service API documentation is available on user page.

Copyright © 2022 - 2026 The INTO-CPS Association

Host Library Microservice

The **lib microservice** is a simplified file manager that serves files over GraphQL and HTTP APIs. It lists directory contents and transfers files to and from users.

This page explains how to run the library microservice as a standalone docker container.

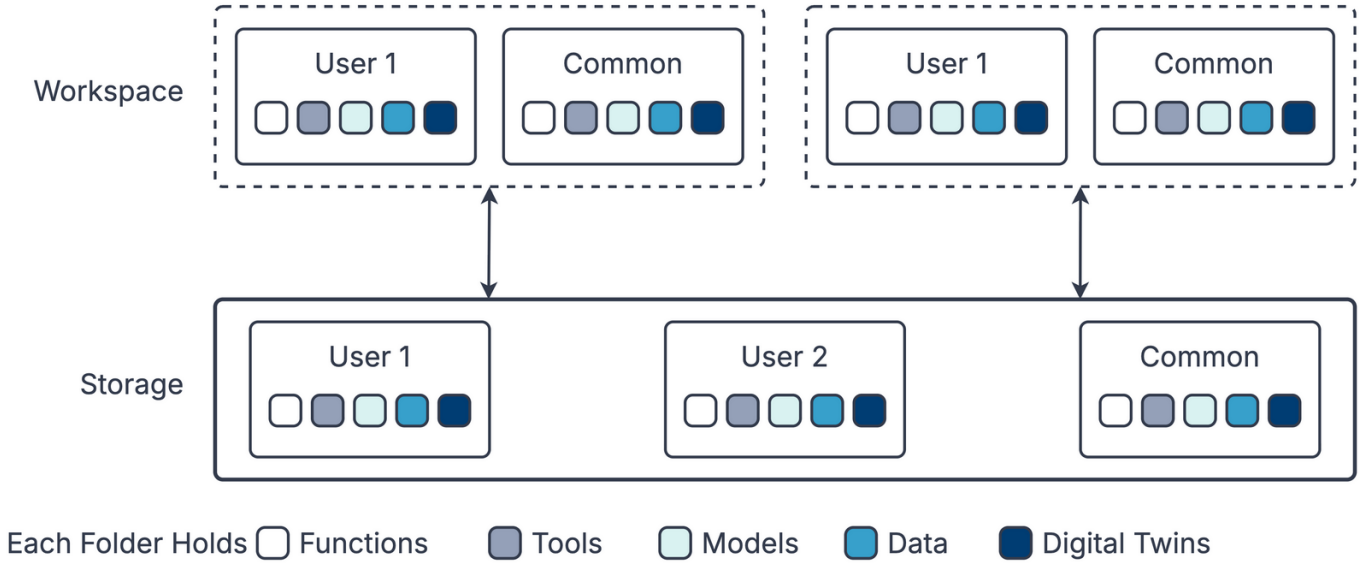
SETUP THE FILE SYSTEM

Outside the DTaaS Platform

The package can be used independently of the DTaaS. In this case no particular file structure is needed. A directory named `files`, placed in the directory from which `compose.lib.yml` will be run, is enough.

Inside the DTaaS Platform

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



A skeleton file structure is available in the DTaaS codebase. Copy it to create a file system for users. Name the directory `files` and place it in the directory from which `compose.lib.yml` will be run.

The container runs as uid 1000 and writes into this directory, so on Linux hosts give the directory to that user:

```
sudo chown -R 1000:1000 files
```

USE

Use the docker compose file to start the service.

```
# To bring up the container
docker compose -f compose.lib.yml up -d
# To bring down the container
docker compose -f compose.lib.yml down
```

SERVICE ENDPOINTS

The GraphQL URL: `localhost:4001/lib`

The HTTP URL: `localhost:4001/lib/files`

The service API documentation is available on user page.

Copyright © 2022 - 2026 The INTO-CPS Association

4.6.4 Logger Microservice

Logger Microservice NPM Package

The logger microservice package is `@into-cps-association/logger-ms`. Install it from npmjs:

```
npm install -g @into-cps-association/logger-ms
```

Start it with `logger-ms`. It reads `logger.yaml` from the working directory if the file exists; use `-c <file>` to give another configuration file.

To build and start it from source instead, run these commands in `servers/logger/`:

```
yarn install
yarn build
yarn start
```

By default, the logger binds to `127.0.0.1:4003` and accepts unauthenticated ingest. Set `LOGGER_AUTH_TOKEN` for non-browser producers that can send bearer headers. The bundled browser client uses `navigator.sendBeacon`, which cannot set custom headers, so enforce browser ingest authentication at the reverse proxy.

For the full configuration reference, see `servers/logger/README.md`.

Copyright © 2022 - 2026 The INTO-CPS Association

Host Logger Microservice

The **logger microservice** ingests workflow log events produced by the DTaaS client and stores them as JSON Lines (`.jsonl`) files.

This page explains how to run the logger microservice as a standalone docker container.

SETUP

Place a `logger.yaml` configuration file in the directory from which `compose.logger.yml` will be run. Download the sample configuration and use it as a template:

```
curl -o logger.yaml \
  https://raw.githubusercontent.com/INTO-CPS-Association/DTaaS/feature/distributed-demo/servers/logger/config/logger.yaml.sample
```

The captured events go to a `logs` directory in the same location. The logger runs as the `node` user (uid 1000), so create the directory and, on Linux hosts, give it to uid 1000:

```
mkdir logs
sudo chown -R 1000:1000 logs
```

If the `logs` directory does not exist, Docker creates it as `root`. Images before 0.1.2 cannot write into such a directory.

USE

Use the docker compose file to start the service.

```
# To bring up the container
docker compose -f compose.logger.yml up -d
# To bring down the container
docker compose -f compose.logger.yml down
```

SERVICE ENDPOINTS

The health check URL: `localhost:4003/logger/health`

The event ingestion URL: `localhost:4003/logger`

The configuration options are described in the npm package page and the service is described on the user page.

Copyright © 2022 - 2026 The INTO-CPS Association

4.7 Guides

4.7.1 Install DTaaS on Localhost (GUI)

This guide runs DTaaS on localhost through a graphical user interface (GUI). It suits a single user who wants to run DTaaS on their own computer.

Two installation scenarios are available:

Scenario	Auth Provider	External Account Required
DTaaS Localhost	GitLab OAuth	Yes (gitlab.com)
Workspace Localhost	Dex (local)	No

Requirements

- Docker Desktop or Docker Engine with the Compose plugin
- Portainer Community Edition (setup below)

Download Packages

Download the package for your scenario:

1. **dtaas-localhost-xx.zip** for the DTaaS localhost installation, or
2. **workspace-dex-localhost-xx.zip** for the Workspace localhost installation

> **Tip**
>
> The guide uses Linux-style paths such as `/Users/username/DTaaS`. On Windows use an equivalent path, for example `C:\DTaaS`.

Start Portainer

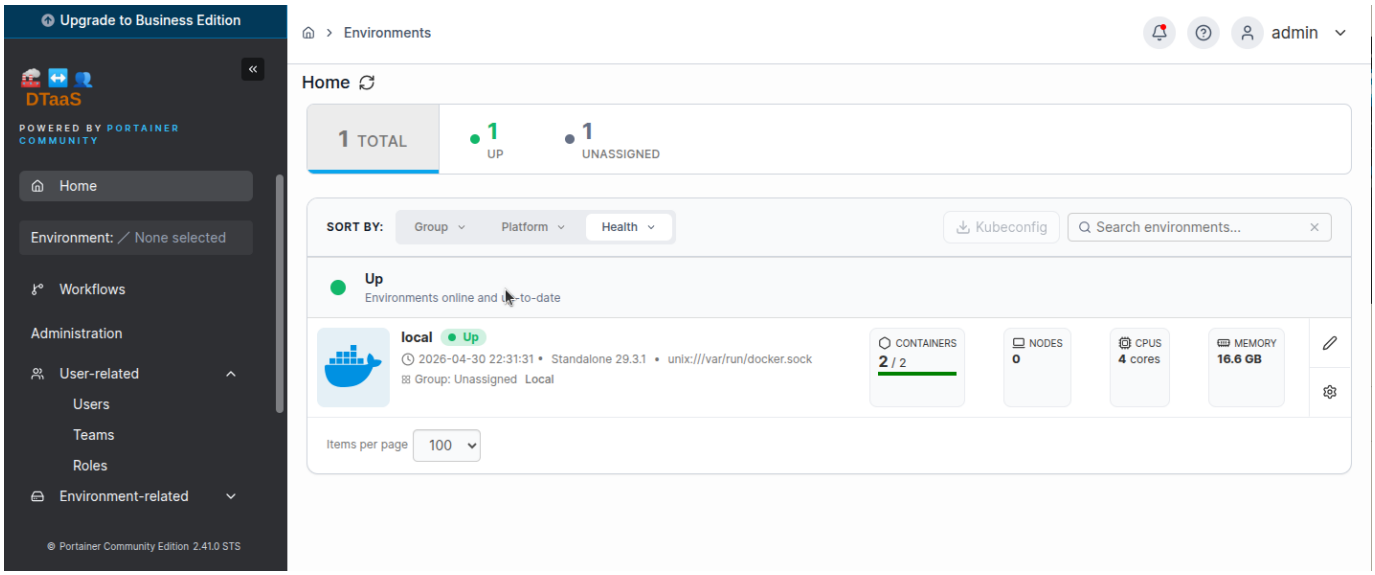
Portainer Community Edition provides a graphical interface for managing Docker containers at `https://localhost:9443`.

Follow the official Portainer CE documentation or run these commands:

```
docker volume create portainer_data
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v portainer_data:/data portainer/portainer-ce:2.21.4
```

Open `https://localhost:9443` and complete the Initial Setup to create an administrator account.

Copyright © 2022 - 2026 The INTO-CPS Association

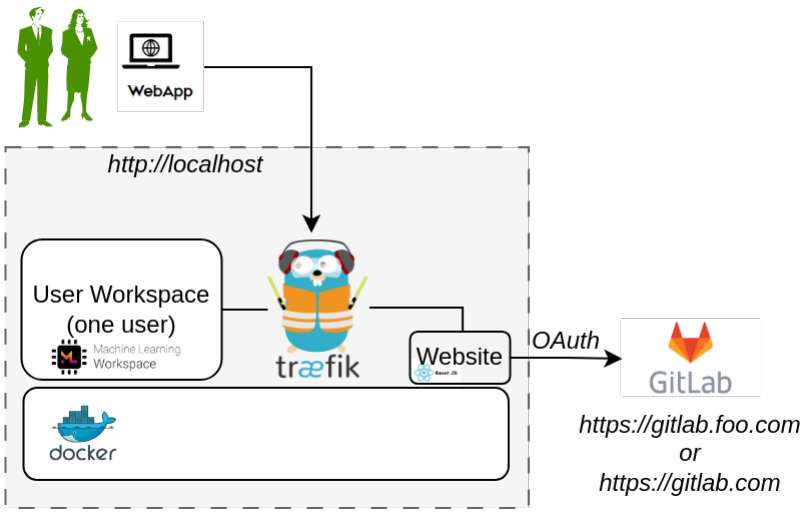


Tip

To restart Portainer later, run `docker start portainer`.

1. DTaaS Localhost

Unzip the **dtaas-localhost-xx.zip** package, which contains the installation files. This scenario uses **GitLab OAuth** for authentication and needs a GitLab account on https://gitlab.com.



1.1 CONFIGURATION

1.2 Create Configuration Files

Copy the example files:

```
cp config/.env.example config/.env
cp config/client.js.example config/client.js
```


Copyright © 2022 - 2026 The INTO-CPS Association

1.3 Environment Variables

Edit `config/.env`:

Variable	Example	Description
`DEFAULT_USER`	`user1`	Workspace path prefix and folder name
`COMPOSE_PROJECT_NAME`	`dtaas`	Docker Compose project name

1.4 Client Configuration

This step is optional. If the localhost installation needs its own OAuth 2.0 application, follow these steps to create one. Then edit `config/client.js` and set the OAuth application credentials from the GitLab account:

Variable	Example	Description
`REACT_APP_CLIENT_ID`	*(OAuth app id)*	GitLab OAuth application ID
`REACT_APP_AUTH_AUTHORITY`	`https://gitlab.com/`	OAuth authority URL

1.5 Create User Workspace

Create a workspace directory for the user set in `DEFAULT_USER`:

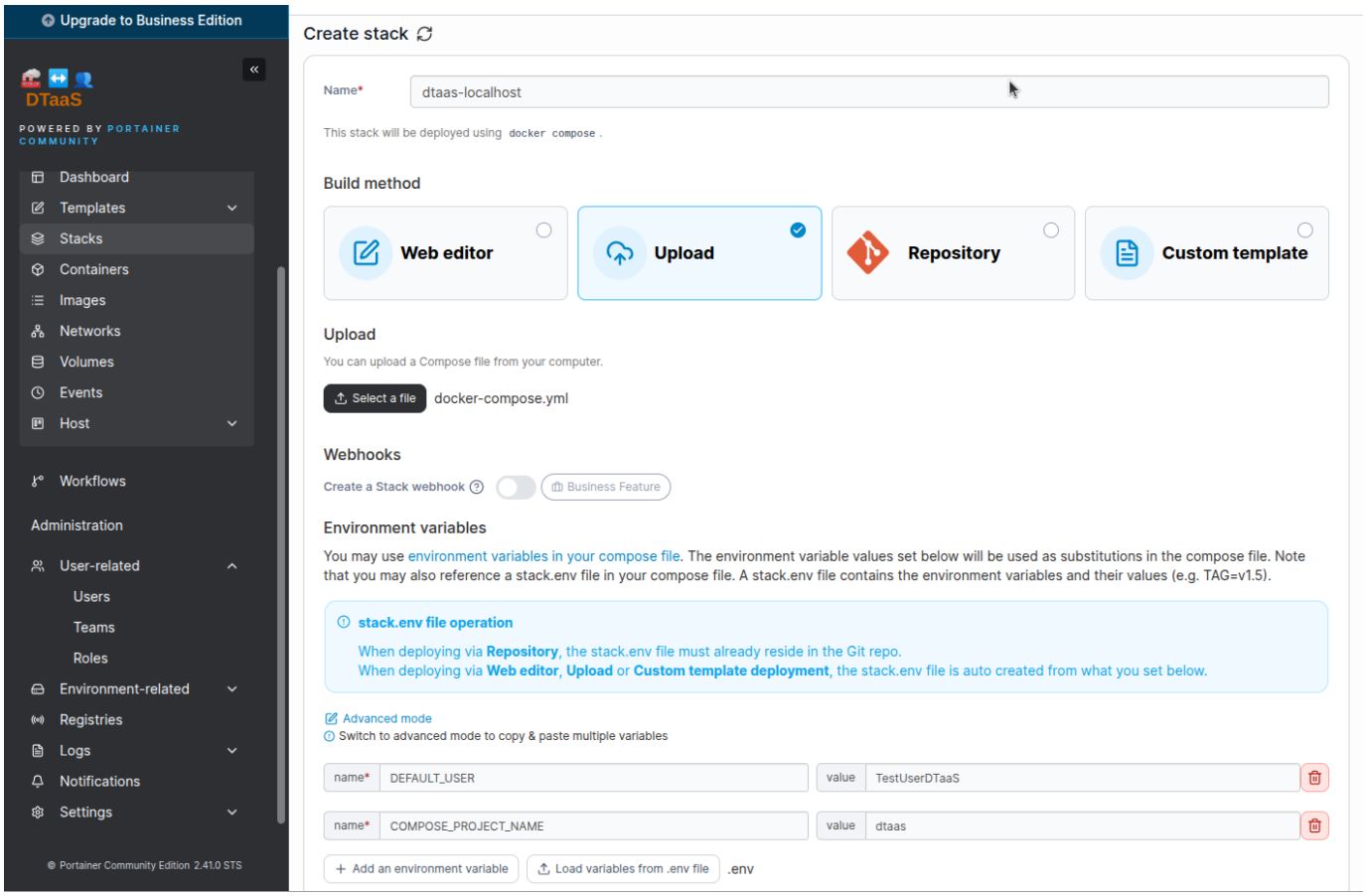
```
cp -R files/template files/<DEFAULT_USER>
sudo chown -R 1000:100 files/*
```

The logger microservice writes its events to `logs/logger`. Create the directory and give it to uid 1000, the user that runs the logger:

```
mkdir -p logs/logger
sudo chown -R 1000:1000 logs/logger
```


Copyright © 2022 - 2026 The INTO-CPS Association

1.6 CREATE THE PORTAINER STACK

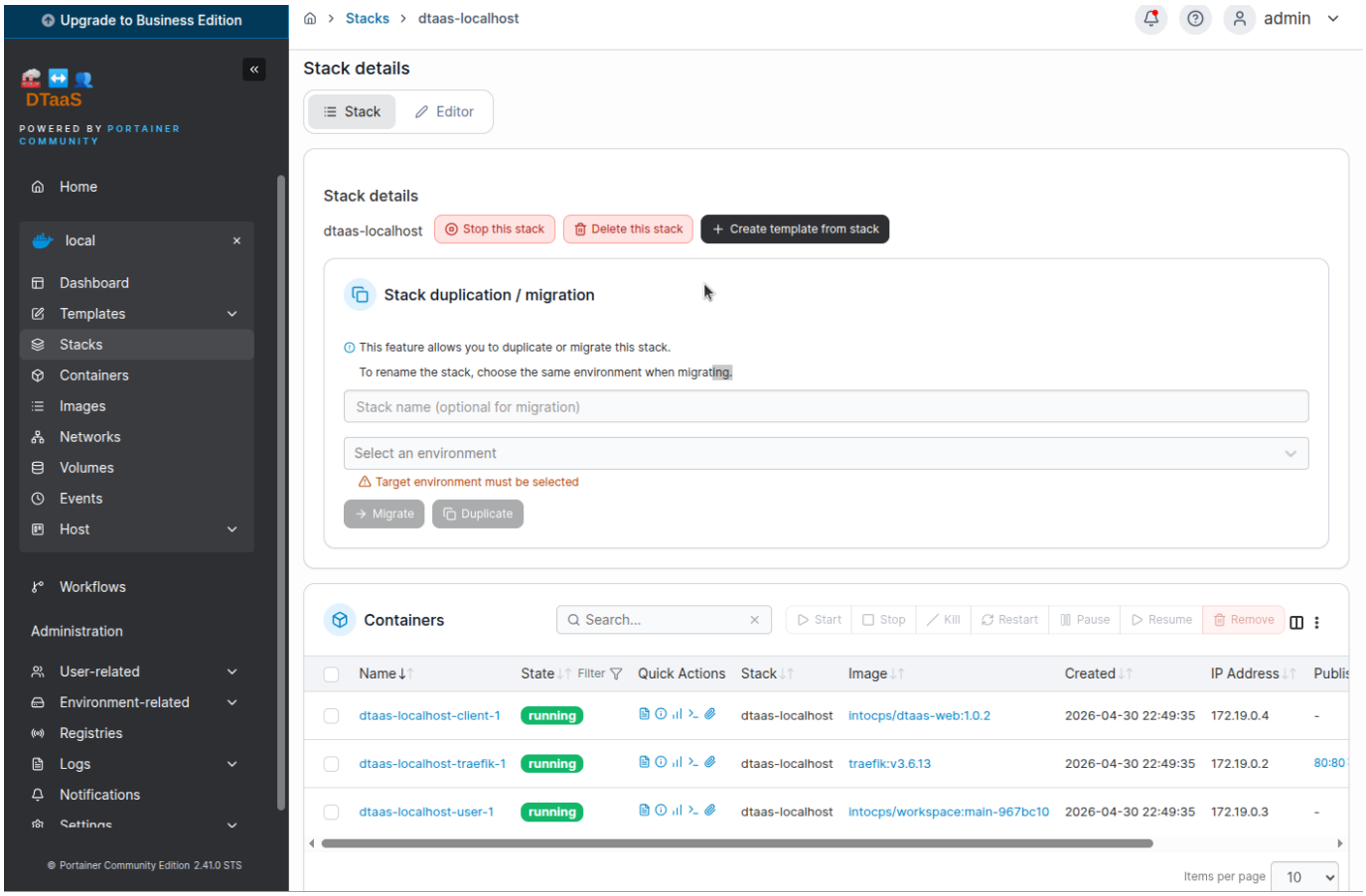


1. Navigate to **Stacks** and click **Add Stack**.
2. Name the stack, for example `dtaas-localhost`.
3. Select the **Upload** build method.
4. Upload the compose file (`docker-compose.yml`).
5. Load the environment file (`config/.env`).

> **Tip**
>
> If the `.env` file is not visible, select **All Files** in the file explorer dialog.

Click **Deploy the stack**. Portainer creates a stack with this name, which looks like this:

Copyright © 2022 - 2026 The INTO-CPS Association



1.7 USE

Open http://localhost in a web browser and sign in with the configured GitLab credentials.

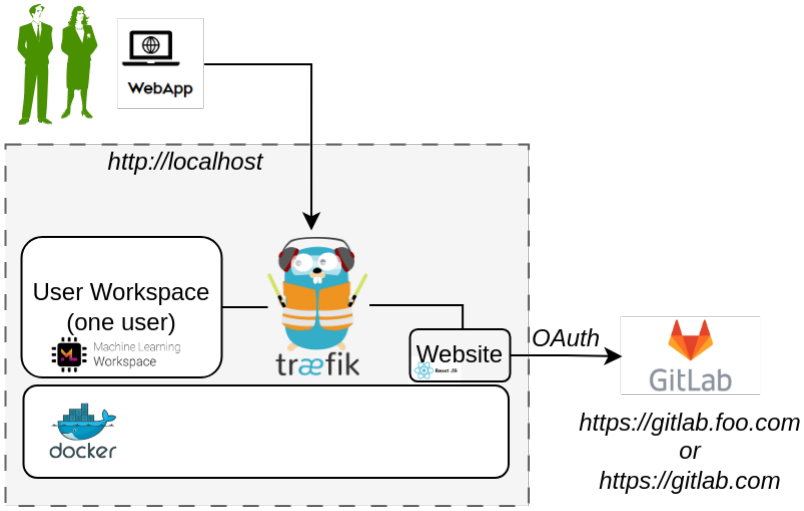
1.8 LIMITATIONS

This scenario does not include the library microservice or backend forward-auth.

2. Workspace Localhost

Unzip the **workspace-dex-localhost-xx.zip** package, which contains the installation files.

This scenario uses **Dex** as a local identity provider. It needs no external account and comes with default credentials.



Copyright © 2022 - 2026 The INTO-CPS Association

2.1 CONFIGURATION

Copy the example files:

```
cp config/.env.example config/.env
cp config/dex-config.yaml.example config/dex-config.yaml
```

2.2 Environment Variables

Edit `config/.env`:

Variable	Example	Description
`COMPOSE_PROJECT_NAME`	`dtaas`	Docker Compose project name
`DEFAULT_USER`	`user`	Default user login profile for Dex

Tip

The local login users are defined in `config/dex-config.yaml`.

2.3 CREATE THE PORTAINER STACK

1. Navigate to **Stacks** and click **Add Stack**.
2. Name the stack, for example `workspace-localhost`.
3. Select the **Upload** build method.
4. Upload the compose file (`docker-compose.yml`).
5. Load the environment file (`config/.env`).

Click **Deploy the stack**.

2.4 USE

Open http://localhost in a web browser. Sign in with the default Dex credentials:

- **Email:** `user@intocps.org`
- **Password:** `user`

2.5 LIMITATIONS

- This scenario does not include the library microservice.
- DevOps features are not available.
- See the custom user instructions to change the default credentials.

References

Image sources: Traefik logo, reactjs, gitlab

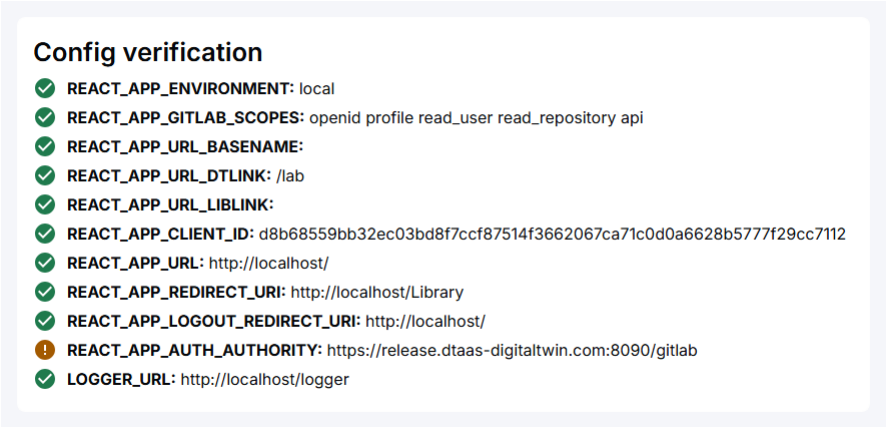
Copyright © 2022 - 2026 The INTO-CPS Association

4.7.2 Check Client Configuration

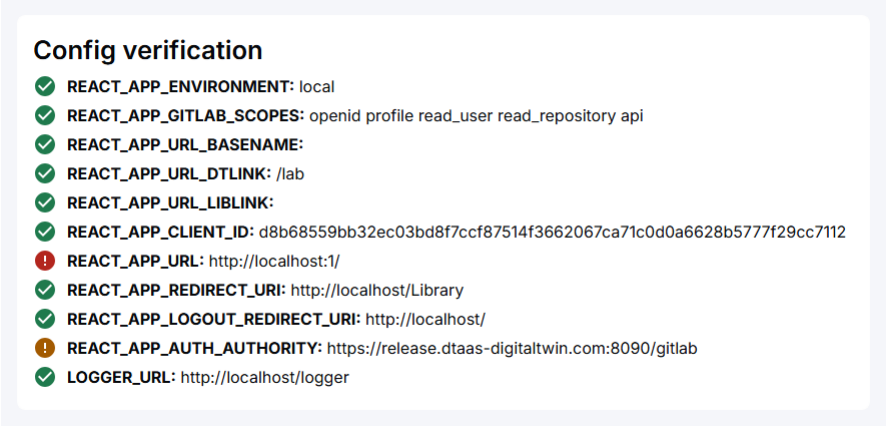
A common error is an incorrect configuration of the React website. The client has a checklist page that verifies its configuration.

Visit: `https://<SERVER_DNS>/config/developer`

A correct configuration shows the following result.



An incorrect client website URL shows the following result.



Copyright © 2022 - 2026 The INTO-CPS Association

4.7.3 Make Common Assets Read Only

Why

Some installations need to stop users from changing the common assets. To make the common area read only, change the volume definitions of the user workspace services.

> **Note**
>
> Make these changes before installing the application.

How

To make the common assets read only, change the user services in the secure-server `docker-compose.yml` file as follows.

```
...
user1:
  ....
  volumes:
    - ./files/common:/workspace/common:ro
  ....

user2:
  ....
  volumes:
    - ./files/common:/workspace/common:ro
  ....
```

The `:ro` suffix at the end of the line makes the common assets read only.

To apply the same restriction to new users, make the same change to any additional `userX` service blocks in `deploy/dtaas/docker/secure-server/docker-compose.yml`. Users added with the DTaaS CLI get their volumes from the `users.server.yml` and `users.server.secure.yml` templates in the installation directory, so add `:ro` to the `/workspace/common` volume there as well.

Copyright © 2022 - 2026 The INTO-CPS Association

4.7.4 Renewing LetsEncrypt Certificates

LetsEncrypt certificates expire every three months and must be renewed. This guide covers renewal for the current DTaaS package layouts.

Prerequisites

- Administrative access to the host
- `certbot` installed
- Access to Docker commands

Step 1: Renew certificates

```
sudo certbot renew --dry-run
sudo certbot renew
```

CLI shortcut for generated deployments

If the DTaaS CLI created the deployment (`dtaas deployment generate`) and `[common.security].certs-src` in `dtaas.toml` points to the directory holding the renewed `fullchain.pem` / `privkey.pem` (e.g. `/etc/letsencrypt/live/your-domain.com`), one command replaces the manual copy and restart below:

```
dtaas platform update --certs --output-dir /path/to/deployment
```

The command validates the renewed pair: it must parse, the key must match the certificate, and neither the leaf nor any intermediate may have expired. It then stops `traefik` , swaps the new files into the deployment's `certs/` directory (restoring the previous pair if anything goes wrong), restarts `traefik` and waits for it to come back up. On POSIX hosts it also restores `0600` on the private key; on Windows it prints a warning, because file permissions cannot be enforced there. If validation fails, the live certificates stay untouched. See the DTaaS CLI page for details.

The manual steps below apply to the `deploy/dtaas/docker/*` package layouts and the services project.

Step 2: Deploy to DTaaS server package

For the secure server package (`deploy/dtaas/docker/secure-server`):

```
sudo cp /etc/letsencrypt/live/your-domain.com/fullchain.pem \
  /path/to/DTaaS/deploy/dtaas/docker/secure-server/certs/fullchain.pem
sudo cp /etc/letsencrypt/live/your-domain.com/privkey.pem \
  /path/to/DTaaS/deploy/dtaas/docker/secure-server/certs/privkey.pem
```

For the integrated package (`deploy/dtaas/docker/secure-server_with_integrated-gitlab`):

```
sudo cp /etc/letsencrypt/live/your-domain.com/fullchain.pem \
  /path/to/DTaaS/deploy/dtaas/docker/secure-server_with_integrated-gitlab/certs/fullchain.pem
sudo cp /etc/letsencrypt/live/your-domain.com/privkey.pem \
  /path/to/DTaaS/deploy/dtaas/docker/secure-server_with_integrated-gitlab/certs/privkey.pem
```

Step 3: Restart Traefik in DTaaS package

Server package:

```
cd /path/to/DTaaS/deploy/dtaas/docker/secure-server
docker compose --env-file config/.env up -d --force-recreate traefik
```


Copyright © 2022 - 2026 The INTO-CPS Association

Integrated GitLab package:

```
cd /path/to/DTaaS/deploy/dtaas/docker/secure-server_with_integrated-gitlab
docker compose --env-file config/.env up -d --force-recreate traefik
```

Step 4: Deploy to services project (if used)

If the platform services run from a project generated by `dtaas-services`, refresh that project's `certs/` directory. Set `CERTS_SRC` in `config/services.env` to the certbot directory that holds the renewed certificates, then run from the project directory:

```
cd /path/to/services-project
sudo -E env PATH="$PATH" dtaas-services host setup
```

`host setup` copies the newest certificate pair from `CERTS_SRC` into `certs/` as `fullchain.pem` and `privkey.pem`, and rebuilds the per-service certificate files and permissions.

Step 5: Restart affected services project containers

```
cd /path/to/services-project
docker compose -f compose.services.yml --env-file config/services.env up -d --force-recreate
```

If the ThingsBoard or GitLab compose stacks run in the same project:

```
docker compose -f compose.thingsboard.yml --env-file config/services.env up -d --force-recreate
docker compose -f compose.gitlab.yml --env-file config/services.env up -d --force-recreate
```

Verification

```
curl -I https://your-domain.com
```

Check the logs of the `traefik` service if needed.

Copyright © 2022 - 2026 The INTO-CPS Association

4.7.5 Self-Signed Certificates

This guide configures DTaaS to work with self-signed TLS certificates. Internal servers that are not reachable from the public Internet need them, because they cannot get certificates from public certificate authorities such as Let's Encrypt.

Background

When DTaaS is installed with a self-signed certificate, the `traefik-forward-auth` service must trust that certificate in order to complete the OAuth 2.0 token exchange with the GitLab instance. Otherwise, authentication fails with a certificate error, and the logs of the `traefik-forward-auth` service contain this error:

```
level=error msg="Code exchange failed with provider"
error="Post https://intocps.org/gitlab/oauth/token:
x509: certificate signed by unknown authority"
```

Prerequisites

- Administrative access to the host
- A DTaaS installation using the secure server package

Step 1: Update the DNS Configuration

Containers in the secure-server package use the host's DNS configuration. If the GitLab instance uses an internal DNS name that the public Internet cannot resolve, add the correct nameserver to `/etc/resolv.conf` on the Docker host so that the host can resolve it. Ask the IT department for the correct values. For example:

```
search client.intocps.org
nameserver 10.20.25.125
```

Step 2: Create Local TLS Certificates with mkcert

Install `mkcert` if it is not already present, preferably with the OS package manager:

```
# Debian / Ubuntu 22.04+
sudo apt install mkcert

# macOS (via Homebrew)
brew install mkcert
```

Alternatively, download the binary from the mkcert releases page and verify its SHA-256 checksum against the value published on that page before installing:

```
wget https://github.com/FiloSottile/mkcert/releases/download/v1.4.4/mkcert-v1.4.4-linux-amd64
# Verify the checksum from https://github.com/FiloSottile/mkcert/releases/tag/v1.4.4
sha256sum mkcert-v1.4.4-linux-amd64
sudo mv mkcert-v1.4.4-linux-amd64 /usr/local/bin/mkcert
sudo chmod +x /usr/local/bin/mkcert
```

Create a local root CA and server certificates:

```
mkcert -install
mkcert "intocps.org" "*.intocps.org" "localhost" "127.0.0.1" "::1"
cp ~/.local/share/mkcert/rootCA.pem rootCA.crt
```

Replace `intocps.org` with the actual hostname of the DTaaS installation.

Copyright © 2022 - 2026 The INTO-CPS Association

Step 3: Build a Custom traefik-forward-auth Image

The `traefik-forward-auth` container must trust the local root CA. Create a `Dockerfile` in the same directory as `rootCA.crt`:

```
# Stage 1: prepare the root CA certificate
FROM alpine:latest AS cert-builder
RUN apk add --no-cache ca-certificates
COPY rootCA.crt /usr/local/share/ca-certificates/
RUN update-ca-certificates

# Stage 2: copy the updated certificate bundle into the final image
FROM thomseddon/traefik-forward-auth:latest
COPY --from=cert-builder /etc/ssl/certs/ca-certificates.crt \
     /etc/ssl/certs/ca-certificates.crt
CMD ["traefik-forward-auth"]
```

Build the image:

```
docker build -t traefik-forward-auth-local:latest .
```

Step 4: Use the Custom Image

In `deploy/dtaas/docker/secure-server/docker-compose.yml`, replace the image reference `image: thomseddon/traefik-forward-auth:latest` with:

```
image: traefik-forward-auth-local:latest
```

Step 5: Recreate the forward-auth Container

```
docker compose --env-file config/.env \
  up -d --force-recreate traefik-forward-auth
```

External GitLab with Self-Signed Certificates

If the GitLab OAuth provider is hosted on a separate server (for example, `gitlab.intocps.org`) and also uses a self-signed certificate, generate that certificate using the same `mkcert` root CA:

```
mkcert "gitlab.intocps.org" "localhost" "127.0.0.1" "::1"
```

The custom `traefik-forward-auth` image already trusts the `mkcert` root CA, so the OAuth token exchange succeeds.

Notes

- Certificates created with `mkcert` are trusted only on the machine where `mkcert -install` was run. Other clients that access the DTaaS installation must also import `rootCA.crt` into their trust store.
- If the server has a valid public TLS certificate, this guide is not needed. Use the standard Let's Encrypt renewal guide instead.

Copyright © 2022 - 2026 The INTO-CPS Association

5. Frequently Asked Questions

5.1 Abbreviations

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

5.2 General Questions

What is the DTaaS platform?

The DTaaS platform is a software platform for creating and running digital twins. The features page gives an overview of what DTaaS can do.

What is the scope and current capabilities of the DTaaS platform?

1. DTaaS is a web-based interface for invoking the tools needed to work with one or more DTs.
2. DTaaS lets users run DTs in their private workspaces. These user workspaces are based on the Ubuntu 20.04 operating system.
3. DTaaS can help create reusable DT assets only if the DT asset authoring tools run in the Ubuntu 20.04 xfce desktop environment.
4. From the DTaaS platform's perspective, DTs are executables. Users are free to work with DTs in whatever way suits them. DTaaS suggests creating DTs from reusable assets and offers a suggested structure for DTs. The examples show the DTaaS workflow in more detail. This suggested workflow is not mandatory.
5. DTs can run as services with a REST API from within user workspaces, which makes service-level DT composition possible.

What can not be done inside the DTaaS platform?

1. DTaaS does not help install DTs obtained from external sources.
2. Windows-only tools cannot run inside DTaaS workspaces.

Is there any fundamental difference between commercial solutions like Ansys Twin Builder and the DTaaS platform?

Commercial DT platforms like *Ansys Twin Builder* integrate models, simulation and sensors tightly. This leaves fewer choices in DT design and implementation, and it brings vendor lock-in. DTaaS, on the other hand, lets users split a DT into reusable assets and combine these assets flexibly.

Is licensed software such as Matlab provided?

Proprietary and commercially licensed software is not available on the platform by default. However, each user has a private workspace based on a Linux xfce desktop environment. Users can install proprietary and commercially licensed software in their workspaces. A screencast shows Matlab Simulink running within the DTaaS platform. Licensed software installed by one user is not available to other users.

Copyright © 2022 - 2026 The INTO-CPS Association

5.3 Digital Twin Assets

Can the DTaaS platform be used to create new DT assets?

The core purpose of DTaaS is to help users create DTs from assets already available in the library. Create Library Assets However, users can use the services available in their workspace to install asset authoring tools there. 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 a workspace are only available to that user.

5.4 Digital Twin Models

Can the DTaaS platform create new DT models?

DTaaS is not a model creation tool, and it does not create digital twin models itself. It can still help users create them: model creation tools can be placed inside DTaaS, and any Linux desktop or terminal tool can run inside DTaaS. Thus, models can be created inside DTaaS and executed using tools that run on Linux. Windows-only tools cannot run in DTaaS.

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

DTaaS by itself does not produce any models. It provides a platform and an ecosystem of services that let digital twins run as services. Since each user has a Linux OS at their disposal, they can also run digital twins that have a graphical interface. In short, the DTaaS platform is neither a modelling nor a simulation tool. If such tools are required, they must be brought onto the platform. For example, if Matlab is needed, the user must bring the licensed Matlab software.

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

The DTaaS platform is agnostic to the kind of models used. DTaaS can run all kinds of models, including behavioural and data models. As long as the models and matching solvers can run on a Linux OS, they can be used in DTaaS. In some cases, models and solvers (tools) are bundled together to form monolithic DTs. The DTaaS platform can run such DTs as well. DTaaS does not provide dedicated solvers. If a solver can be installed in the workspace, the platform does not need to provide one.

Does it support XML-based representation and ontology representation?

Not at present. **The team is looking for users who need this capability. Given concrete requirements and an example, the team can discuss how to realise it in DTaaS**.

5.5 Communication Between Physical Twin and Digital Twin

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

At an abstract level, data flows from the physical entity to the digital entity and back to the physical entity. The person designing the digital entity decides how this communication happens. DTaaS provides communication services that make this communication easier. The InfluxDB and RabbitMQ services hosted on DTaaS may be used for two-way communication between digital and physical entities. RabbitMQ has its MQTT plugin enabled, so it also works as an MQTT broker.

Copyright © 2022 - 2026 The INTO-CPS Association

How is a physical entity such as shape, size, weight, structure, or chemical attributes measured using DTaaS? Is any specific technology used in this case?

The measurements are made at the physical twin and then sent to the digital twin. A digital twin platform like DTaaS can only facilitate the communication of these measurements from the physical twin. DTaaS provides InfluxDB and RabbitMQ (which also acts as an MQTT broker) for this purpose. These are among the most widely used services for digital twin communication. DTaaS also permits the use of other communication technologies and services hosted elsewhere on the Internet.

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

The DTaaS platform cannot tell whether data is static or dynamic. It can store names, units and any other text description of the quantities of interest (weight of batter, voltage output, etc.). It can also store the data sent by the physical twin. The user has to make the distinction between static and dynamic data. Only metadata can reveal more about the nature of the data. A tool may help in very specific cases, but it still needs metadata. If a person makes this distinction, the need for metadata goes down but does not go away entirely. Some manufacturer-supported DT platforms tightly integrate data and model. In such cases, the tool itself manages the metadata. DTaaS is a generic platform for running digital twins. If a tool runs on a Linux desktop or command line, DTaaS can support it. The tool (e.g., Matlab) can then manage the metadata requirements itself.

5.6 Digital Twin DevOps Automation

Can a DT execute forever?

The web UI imposes a 10-minute timeout on an execution. Users can also terminate an execution manually. The best option is for a DT to run a terminate script, which ends the execution and returns the logs.

Is the DT execution really scalable?

The capacity of the DT execution infrastructure depends on GitLab and on the compute power available to the GitLab runners. GitLab itself does not limit the number of runners.

If many GitLab runners are attached to a GitLab repository, which one is used?

This is indeterminate. The order and location of a DT's execution cannot be relied on.

5.7 Data Management

Can the DTaaS platform collect data directly from sensors?

Yes, via the platform services.

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 platform provides InfluxDB, PostgreSQL, RabbitMQ, MQTT, MongoDB and ThingsBoard services. Both the physical twin and the digital twin can use these services to communicate. IoT (time-series) data can be collected using the InfluxDB and MQTT broker services. InfluxDB has a user interface for analysing the collected data. Users can also upload their own data files into DTaaS.

Copyright © 2022 - 2026 The INTO-CPS Association

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

Yes, via the platform services.

Which transmission protocol does the DTaaS platform allow?

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

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

Information from multiple sources can be stored. The InfluxDB service hosted on DTaaS has the Influx / Flux query language for sensor fusion, analysis and inference.

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

Graphical, geometric and image visualisation. If the visualisation needs specific licensed software, the user must provide the licence for it. DTaaS does not support AR/VR.

5.8 Platform Native Services on the DTaaS Platform

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

This is the job of a digital twin. If a ready-to-use digital twin exists for this purpose, DTaaS lets others use it. Anomaly detection can be done with platform services such as Grafana, ThingsBoard and InfluxDB.

Copyright © 2022 - 2026 The INTO-CPS Association

5.9 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 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
Microsoft Azure DT	Commercial Cloud	H	H	H	M	H
AWS IOT Greengrass	Open source commercial	H	H	H	M	H
Eclipse Ditto	Open source	M	H	M	H	H
Asset Administration Shell	Open source	H	H	L	H	M
PTC Thingworx	Commercial	H	H	H	H	H
GE Predix	Commercial	M	H	H	M	L
The DTaaS Platform	Open source	H	H	L	L	M

```
Adapted by Tanusree Roy from Tables 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 confusion comes from the fact that different DT platforms (Azure DT, GE Predix) provide different kinds of DT capabilities. GE Predix can run all kinds of models natively, even next to (on) the PTs. The Azure DT service cannot do this natively. The user must integrate it with other Azure services or third-party services to get the capabilities that GE Predix provides natively in one interface. The right platform therefore depends on the specific requirements.

5.10 GDPR Concerns

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

The DTaaS platform does not store any personal information about users. It stores only usernames to identify users, and these usernames do not contain enough information to deduce a user's true identity.

Copyright © 2022 - 2026 The INTO-CPS Association

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

The multi-user server installation requires an HTTPS-terminating reverse proxy between the users and the DTaaS platform installation. Administrators can install HTTPS certificates into the application. The codebase can produce an HTTPS-enabled application, and users can also install their own certificates obtained from certificate authorities such as Let's Encrypt.

What security measures does the cloud provider offer?

The DTaaS platform can be installed on a corporate server hosted behind network firewalls, so that only permitted user groups have network access and physical access to the server.

How is user access controlled and authenticated?

The multi-user server installation of DTaaS has two levels of authorisation. The first level is server-side OAuth 2.0 authorisation through Traefik forward-auth over a secure HTTPS connection. The second level is the OAuth 2.0 PKCE authorisation flow in the web client. A GitLab instance provides both. DTaaS does not store the account and authorisation information of users.

Does the 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?

The platform has two roles: administrator and user. The administrator manages the user roles.

Copyright © 2022 - 2026 The INTO-CPS Association

6. Developer

6.1 Guide

6.1.1 Contributors Guide

This guide describes how to contribute to the Digital Twin as a Service (DTaaS) project, from opening an issue to creating a pull request (PR), review and merge. Please read the Code of Conduct before contributing.

Project Goals

The DTaaS platform supports the creation, management and execution of Digital Twins (DTs) through a service-oriented architecture that promotes reuse of DT assets [1]. This documentation explains the software design and development processes of the project to members of the development team.

Additional resources:

- Developer Slides
- Developer Overview Video
- DTaaS DevOps Workflows
- Research Paper

Codebase Overview maps the directories of the monorepo.

Development Environment

The recommended development setup is the devcontainer defined in `.devcontainer/devcontainer.json`. It runs the containerised environment from `developer/devenv/`, which holds every toolchain needed to build and test the projects. **Reopen in Container** runs `developer/devenv/init.sh` on the host first. Outside VS Code, run that script once and follow `developer/devenv/README.md`.

Development Workflow

Contributors follow this procedure:

1. Fork the main repository into a personal GitHub account.
2. Configure Qlty, Codecov and SonarQube for the fork. Codecov does not need a secret token for public repositories.
3. Use Node.js 26 and Python 3.14.
4. Follow the Fork, Branch, PR workflow.
5. Develop in the fork and open a PR from the working branch to the `feature/distributed-demo` branch of the fork. The PR triggers the GitHub Actions, Qlty, SonarQube and Codecov checks.
6. Fix every issue the automated checks report.
7. Once the checks pass, open a PR to the `feature/distributed-demo` branch of the upstream DTaaS repository.
8. Project administrators or maintainers review and merge the PR.

Each PR should either satisfy a well-defined user story or improve code quality.

Local Testing lists the build and test commands to run before opening a PR.

Copyright © 2022 - 2026 The INTO-CPS Association

Coding Agents and Editors

The project makes extensive use of coding agents. The primary usage scenarios include:

- Co-development assistance
- Code review support
- Draft pull requests for prototyping ideas

Every coding agent reads the same instructions: `AGENTS.md` for the general coding guidelines and `PROJECT.md` for the repository-specific ones. Task-specific instructions are Agent Skills in `.agents/skills/`. Coding Agents describes these files and the available skills.

Contributors should disclose code generated by coding agents, particularly when such code embeds significant programming logic.

The following practices are strictly prohibited:

- Contributing unknown or unreviewed code
- Failing to understand the implications of generated code

Summary: Contributors must fully understand their contributions and should not delegate critical thinking to coding agents.

Code Quality

Code quality is checked by:

- **Static Analysis**: Qlty reports linting issues. Install the Qlty CLI and run `qlty check --no-fail --sample 5 --no-formatters` to list them locally. Qlty analyses only the files that differ from the default branch (`feature/distributed-demo`).
- **Security Scans**: SonarQube reports code quality and security issues.
- **Test Coverage**: Codecov collects coverage reports.
- **Continuous Integration**: every GitHub Actions workflow must pass.

QLTY

Qlty performs static analysis, linting and style checks.

New issues appear on the PR page. The issues and code sections of the Qlty dashboard list the rest.

Qlty Cloud skips pull requests opened from forks, because GitHub withholds its token from them. The `Qlty` workflow (`.github/workflows/qlty.yml`) runs the Qlty CLI on every pull request instead. It fails on new actionlint findings in changed workflows, and reports dependency advisories (osv-scanner) and code smells in the job log and summary without failing.

Contributions should not introduce new quality issues. Fix any that appear, using the suggestions from Qlty. Add an ignore flag only in exceptional cases.

CODECOV

Codecov tracks test coverage for the whole project. The testing documentation describes the testing practices and workflows.

GITHUB ACTIONS

The project defines several GitHub Actions workflows. Every pull request and direct commit must pass all of them. Secrets for GitHub Actions lists the secrets and variables a fork needs for the publishing jobs.

References

[1] Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

Copyright © 2022 - 2026 The INTO-CPS Association

6.2 System

6.2.1 System Overview

The Digital Twin as a Service (DTaaS) platform supports the whole digital twin (DT) lifecycle. Users create, configure, execute and share digital twins built from reusable assets[1]. The architecture follows established principles for realising digital twins in practice[2] and also supports use cases such as runtime verification of autonomous systems[3].

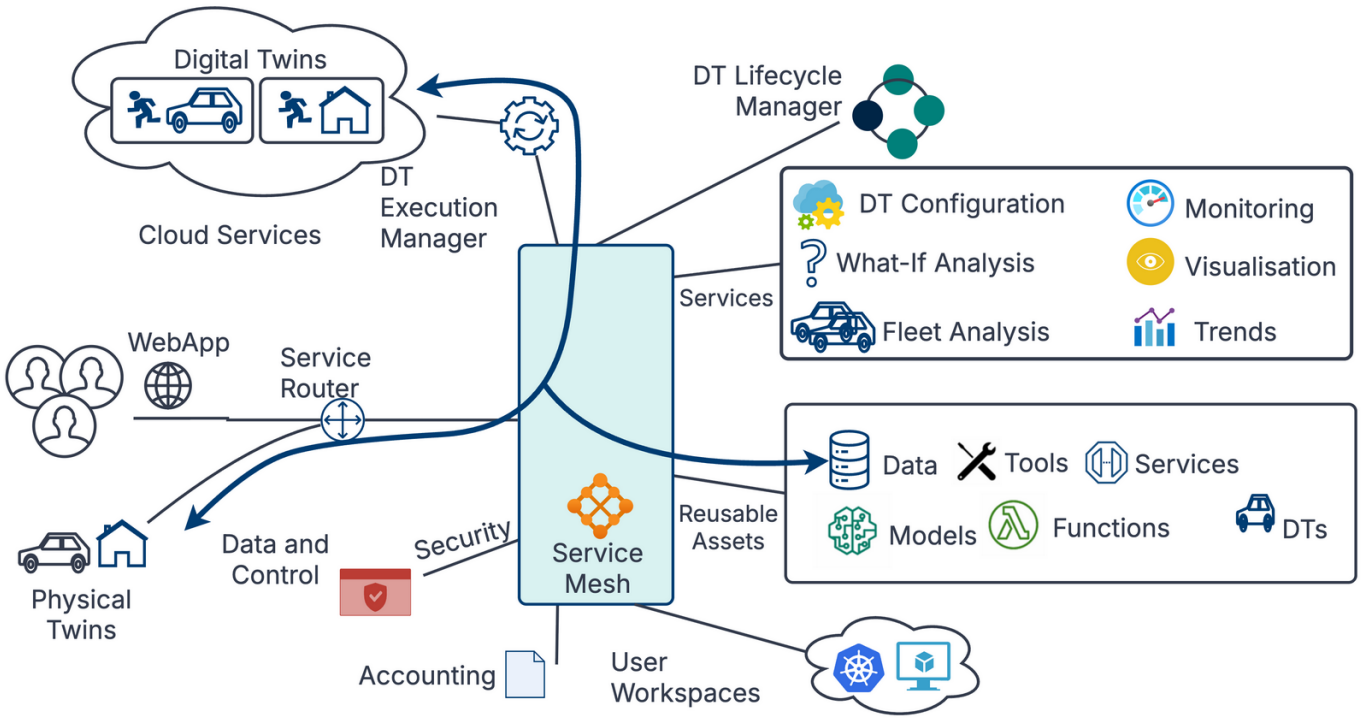
User Requirements

The platform provides the following capabilities:

1. **Author**: create the assets of a DT on the platform itself. This requires software frameworks and tools built for authoring DT assets.
2. **Consolidate**: consolidate the available DT assets and authoring tools so that users can navigate the library of reusable assets. This requires support for discovering available assets.
3. **Configure**: select and configure DTs. This also requires validation of a given configuration.
4. **Execute**: provision computing infrastructure on demand to execute a DT.
5. **Explore**: interact with a DT and explore the results stored inside and outside the platform. Exploration may lead to analytical insights.
6. **Save**: save the state of a DT that is already executing. This is needed to save and re-spawn DTs on demand.
7. **Services**: integrate DTs with on-platform or external services that 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 DTaaS software platform.



Copyright © 2022 - 2026 The INTO-CPS Association

SYSTEM COMPONENTS

Users interact with the software platform through a web application. The service router is the single point of entry for direct access to platform services and controls user access to the microservice components. The service mesh provides microservice discovery, load balancing and authorization.

Further microservices manage DT reusable assets, platform services, the DT lifecycle, DT execution, accounting and security. The microservices are complementary and composable, and together they meet the core requirements of the system.

The microservices responsible for the user requirements are:

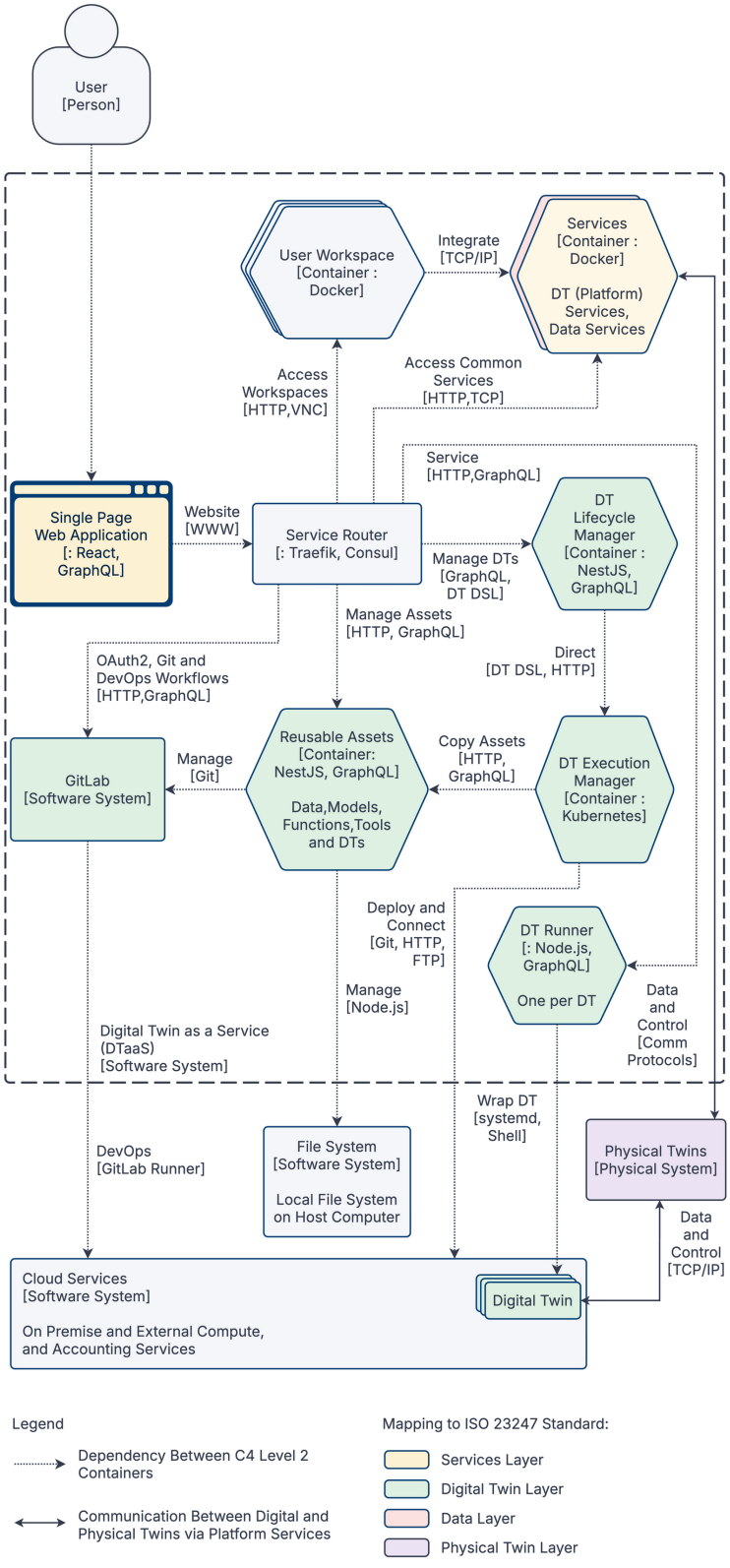
1. **The security microservice** implements role-based access control (RBAC) in the platform.
2. **The accounting microservice** keeps track of the live status of the platform, DT assets and infrastructure usage. It must enforce any licensing and usage restrictions. Accounting is a prerequisite for commercialising the platform. Because the platform makes heavy use of external infrastructure and resources, the accounting microservice needs to interface with the accounting systems of those external services.
3. **User Workspaces** are virtual environments in which users perform lifecycle operations on DTs. These environments are either Docker containers or virtual machines that give users a desktop interface.
4. **Reusable Assets** are the parts from which DTs are created. The assets page explains them further.
5. **Services** are available to all DTs and users of the DTaaS platform. Services build on DTs and provide user interfaces to users.
6. **DT Execution Manager** provides virtual and isolated execution environments for DTs. It also provisions cloud resources dynamically.
7. **DT Lifecycle Manager** manages the lifecycle operations on all DTs. It directs the *DT Execution Manager* to execute, save and terminate DTs.

The web application can also carry domain-specific pages that come from separate npm packages. Client Extensions describes the current state and the proposed extension architecture.

Several of these components are not implemented yet; the table below gives their status. The architecture review describes the system as implemented.

The C4 diagram below shows the DTaaS architecture in more detail.

Copyright © 2022 - 2026 The INTO-CPS Association



Copyright © 2022 - 2026 The INTO-CPS Association

The table maps the architectural components to related pages in the documentation.

System Component	Doc Page(s)
Service Router	Traefik Gateway
Web Application	React Web Application
Reusable Assets	Library Microservice
Workflow Logs	Logger Microservice
Digital Twins and DevOps	Integrated GitLab
Platform Services	Third-party Services (MQTT, InfluxDB, RabbitMQ, Grafana, MongoDB, PostgreSQL, and ThingsBoard)
DT Lifecycle Manager	Not available yet
Security	GitLab client OAuth 2.0 and server OAuth 2.0
Digital Twins as Services	DT Runner
Accounting	Not available yet
Execution Manager	Not available yet

References

Font sources: fileformat

[1]: Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.

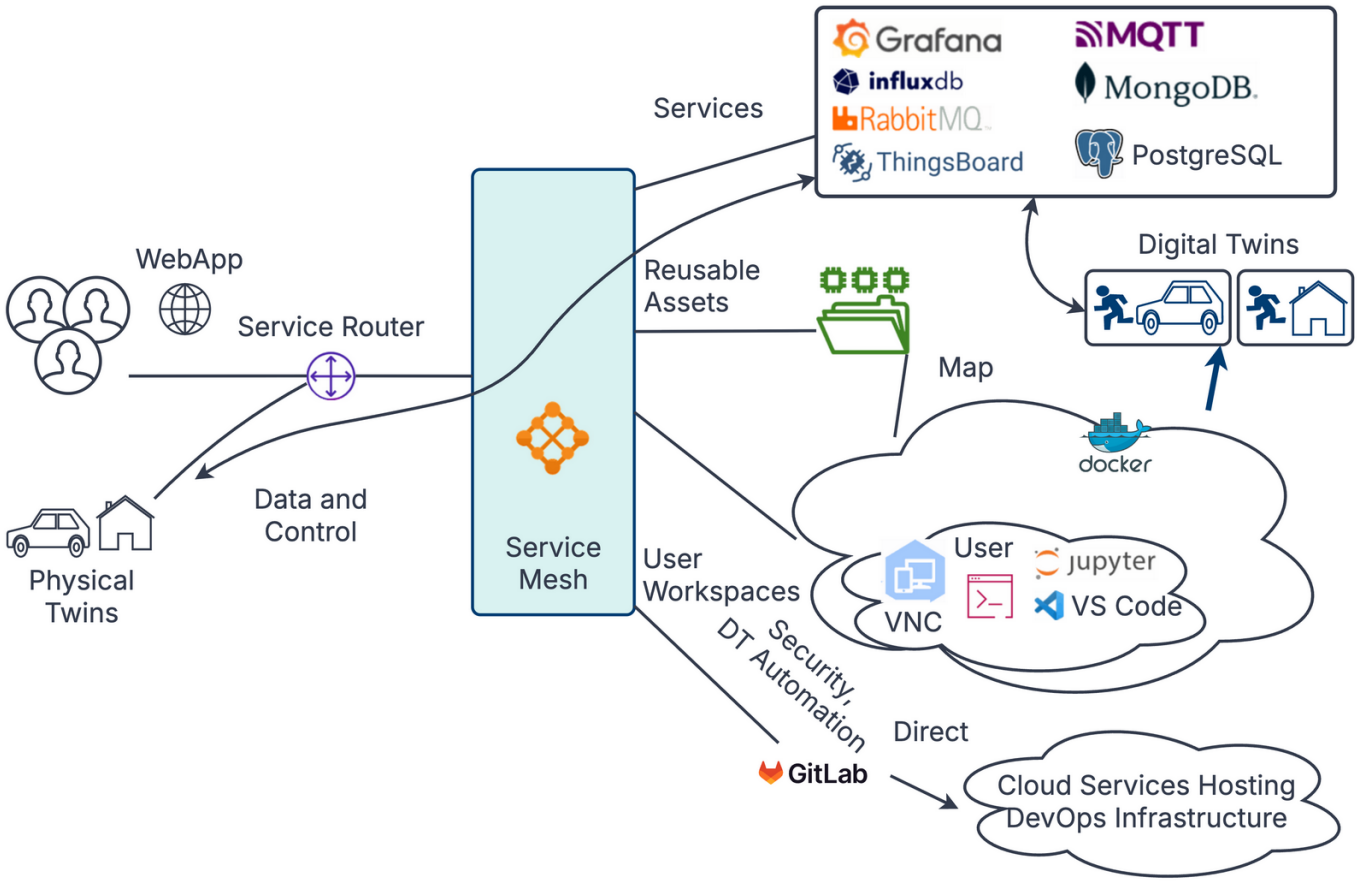
[2]: Talasila, Prasad, et al. "Realising digital twins." The engineering of digital twins. Cham: Springer International Publishing, 2024. 225-256.

[3]: Kristensen, Morten Haahr, et al. "Runtime Verification of Autonomous Systems Utilizing Digital Twins as a Service." 2024 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C). IEEE, 2024.

Copyright © 2022 - 2026 The INTO-CPS Association

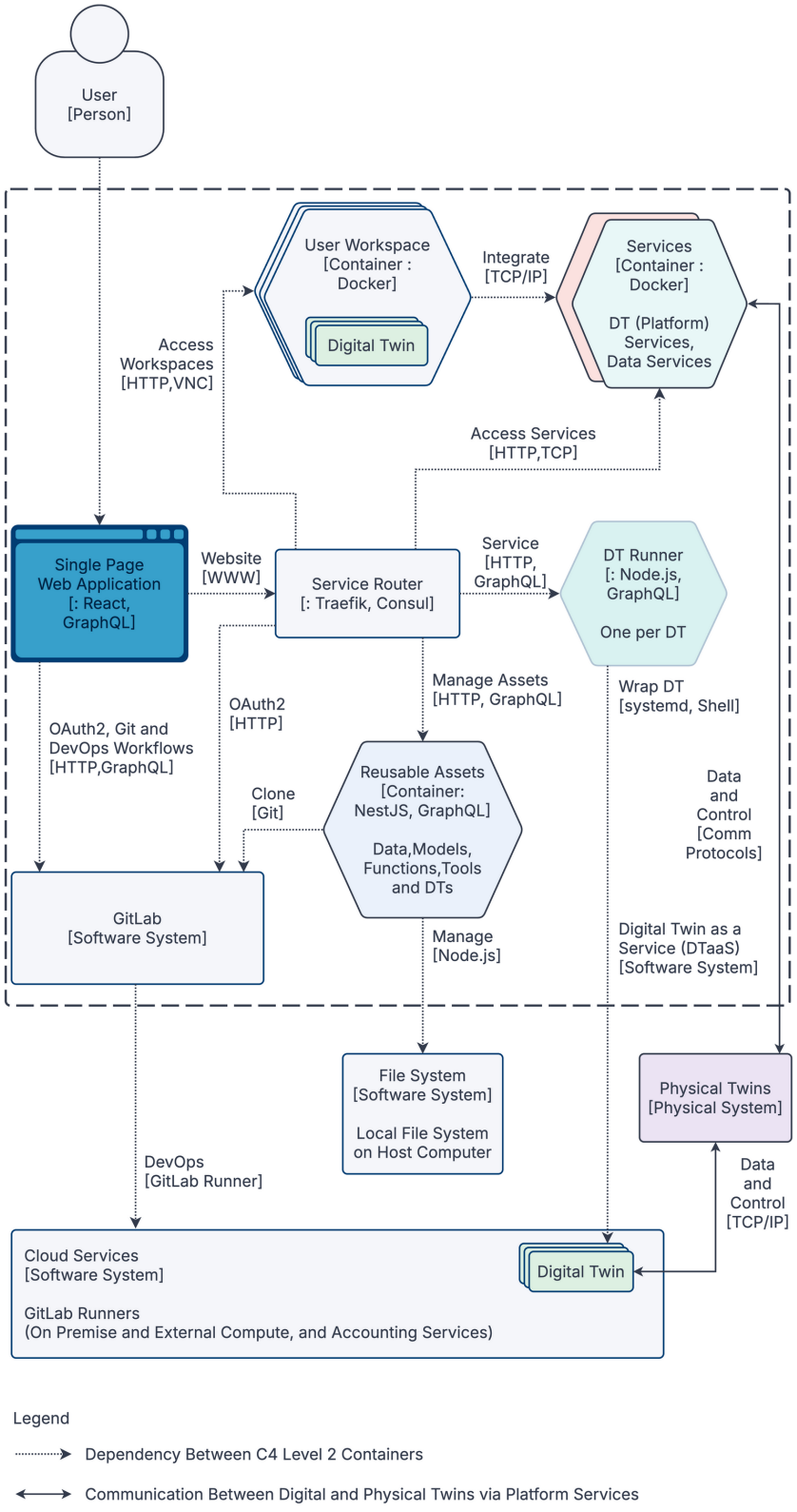
6.2.2 Current Status

DTaaS is a monorepo under active development. It has deployment scenarios meant for production use, and work continues on execution automation, reusable assets and contributor tooling.



The C4 diagram shows the same status in more detail.

Copyright © 2022 - 2026 The INTO-CPS Association



Platform Snapshot

- React 19 client, built with Vite, with a typed backend abstraction over GitLab APIs.
- NestJS library, runner and logger microservices.
- `@into-cps-association/dt-automation`, an npm library with the data, state and GitLab operations for creating, managing and executing DTs.

Copyright © 2022 - 2026 The INTO-CPS Association

- `@into-cps-association/bim-kit`, maintained in the `ifc-utils` repository, for the Building Models page.
- `gitlab-common`, a Python library shared by the two CLIs.
- Python CLIs for DTaaS administration (`dtaas`) and platform services management (`dtaas-services`).
- Scenario-based admin documentation for DTaaS and workspace deployments.

Security and Access Model

Two authorization patterns are in use:

- GitLab-centred OAuth flows for DTaaS package scenarios.
- Keycloak-centred OIDC flows for secure workspace scenarios.

Traefik and its forward-auth middleware protect routes and mediate access at the gateway.

Workspaces and Assets

Workspace deployments give each user an isolated environment and share assets between users. Asset management currently uses file-system workflows and the library microservice, which is still evolving.

Current Engineering Focus

- Improve scenario automation and configuration guidance.
- Strengthen test coverage and CI reliability across packages.
- Continue improving DevOps execution visibility in the client UI.
- Evaluate the proposed client extension architecture, which adds domain-specific pages through npm packages.
- Keep package publishing and release flows consistent across npm, Docker, and Poetry-based tools.

Contributions that improve reproducibility, documentation quality and integration between components are welcome.

Copyright © 2022 - 2026 The INTO-CPS Association

6.2.3 Client Extensions

The web client uses npm packages for parts of its behaviour, and some of these packages are maintained outside this repository. This page lists them, describes how the Building Models page is wired into the client, and summarises the modular architecture proposed for domain-specific pages.

Packages Used by the Client

Package	Source	Role in the client
`@into-cps-association/dt-automation`	`lib/dt-automation/` in this repository	Data, state and GitLab operations for creating, managing and executing DTs. The client takes it from npm (`^0.2.0` in `client/package.json`).
`@into-cps-association/bim-kit`	ifc-utils	IFC to glTF conversion in the browser, the binding manifest and the viewer of the Building Models page. The client pins version `0.1.1`.

The client page describes the classes of `dt-automation`, and Publish Packages describes how it is released.

The Building Models Page

The domain logic of the Building Models page lives in `bim-kit`. The client holds three wiring files in `client/src/route/bim/`:

- `Bim.tsx` places the page in the DTaaS layout and hands the viewer the library URL of the signed-in user.
- `library.ts` names the folder that holds the models, `common/models`.
- `persistGeometry.ts` writes a model converted in the browser back to the library, so that each model is converted once.

A change to how a model is drawn reaches DTaaS as a new version of `bim-kit`. The page is still wired into the client by hand, in the route table (`src/routes.tsx`), the menu (`src/page/MenuItems.tsx`), the icon registry (`src/components/appIcons.tsx`) and `package.json`. A page for another domain would have to edit the same files.

Proposed Modular Architecture

The modular architecture described here is a proposal. It is not merged into the client, and `client/package.json` does not depend on the SDK.

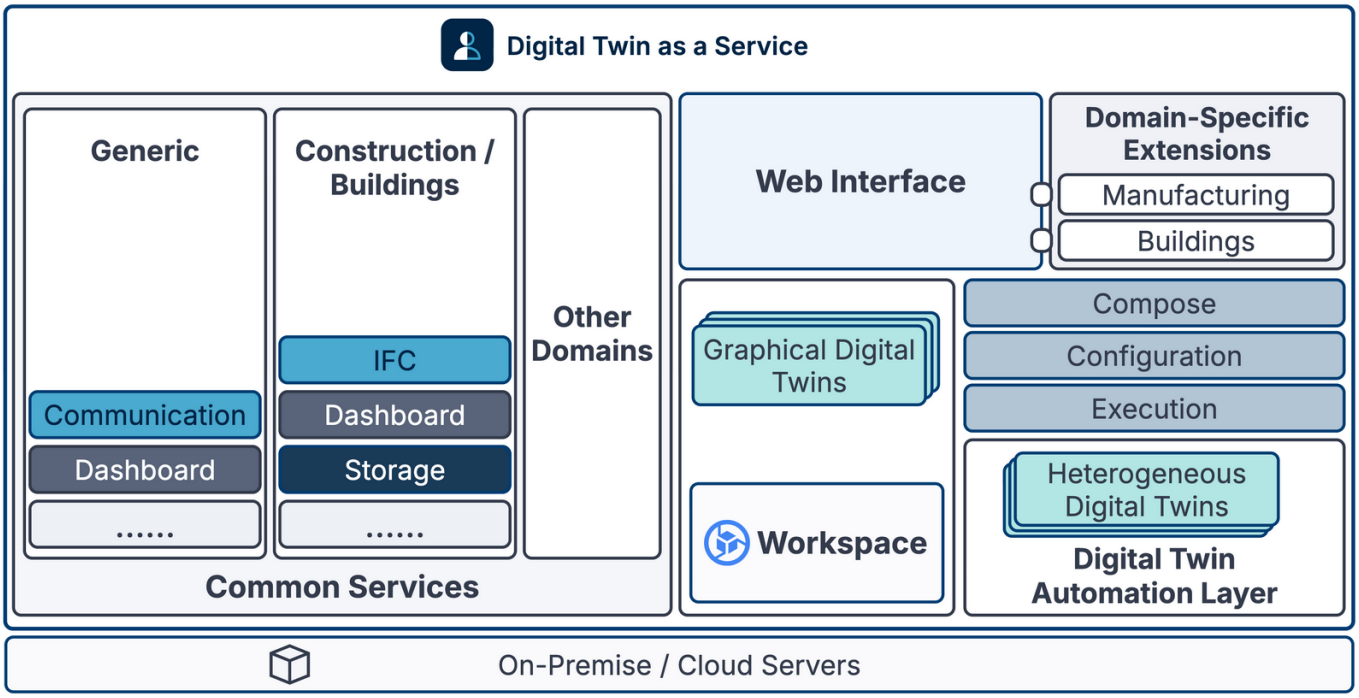
The dtaas-sdk repository holds `@into-cps-association/dtaas-sdk`, which defines a contract between the web client (the host) and domain extensions. In the proposal:

- A domain extension is an npm package. It declares what it contributes, such as pages, navigation entries, digital twin tabs, asset previews, anchor kinds, converters, encoding presets and substrates, as one `DtaasExtension` object.
- The host passes platform services to each extension through a `HostServices` object: authentication, the library, the workspace contents API, GitLab, live signals, the visualisation service, user interface helpers, logging, settings and configuration. An extension does not import client internals.
- The client lists the extensions compiled into a build in one file, `src/extensions.ts`. A deployment can disable a compiled-in extension in `env.js` with `REACT_APP_EXTENSIONS_DISABLED`.
- A common visualisation core, `dtaas-visualisation`, holds the parts that do not depend on the domain: live data transport, the temporal store, anchor resolution, encoding rules and the standard substrates.
- `bim-kit` is moved onto the contract, and the wiring files in `client/src/route/bim/` move into it.

The SDK declares a contract revision (`sdk: 1`) that is separate from its npm version, and a host built on the SDK refuses an extension built for another revision. The `examples/bim` folder of the SDK repository ports `bim-kit` 0.1.1 onto the contract as a private package.

Copyright © 2022 - 2026 The INTO-CPS Association

The figure shows the direction of this architecture. Domain-specific extensions add pages to the web interface, and the common services are grouped into a generic set and a set for each domain, such as IFC for construction and buildings. Of the two extensions drawn, the Buildings page exists in the client today, as a hand-wired route.



The SDK repository documents the proposal in detail:

- Modular architecture proposal
- Writing an extension
- Integrating the SDK into a host
- The `visualisation.json` schema
- The bim-kit port

Copyright © 2022 - 2026 The INTO-CPS Association

6.2.4 DTaaS Research Overview

This document summarises the research behind the DTaaS platform for code contributors who need to understand the design rationale, the architectural decisions and the intended evolution of the system.

1. Introduction

A digital twin is a software representation of a physical asset that stays connected to its real-world counterpart through data exchange. The connection supports monitoring, analysis and prediction throughout the asset lifecycle. Digital twin platforms provide shared infrastructure for creating, managing and executing digital twins at scale.

DTaaS (Digital Twin as a Service) is an open-source platform that treats digital twins as compositions of reusable assets. The platform organises assets into four categories: data, models, functions (methods and scripts) and services. Users compose these assets into digital twin configurations that can be executed, monitored and evolved independently.

DT asset relationship

The platform targets operability: creating, configuring, executing, monitoring and evolving twins in long-running environments. This sets DTaaS apart from simulation-only tools and pure modelling frameworks.

2. Requirements

Eight requirements drive the platform design:

1. **Author**: create and edit digital twin assets in isolated workspaces
2. **Consolidate**: combine assets from different sources into coherent twin configurations
3. **Configure**: set parameters for execution without modifying asset source code
4. **Execute**: run digital twin configurations on demand or continuously
5. **Explore**: browse available assets, twins and execution results
6. **Save**: persist configurations, results and intermediate state
7. **What-if Analysis**: evaluate alternative scenarios by varying parameters
8. **Collaborate**: share assets and twins across users and organisations

These requirements apply across domains. Validated case studies include food fermentation process control, indoor climate management for firefighter training and structural health monitoring of civil infrastructure.

3. System Architecture

System Architecture

The architecture divides the platform into four concern areas:

- **Gateway and authentication**: a Traefik reverse proxy with OAuth 2.0 / OpenID Connect for route protection and user identity management
- **Execution management**: backend services that trigger, monitor and stop digital twin runs via GitLab CI/CD pipelines
- **Reusable-asset handling**: a library microservice that serves assets from a local file system or git repositories and exposes asset discovery and retrieval APIs
- **Workspace isolation**: per-user JupyterLab containers with private file systems for authoring and running assets

System Architecture C4 Diagram

The current codebase implements these areas as follows. The React client handles user interaction and itself triggers and monitors the GitLab CI/CD pipelines that execute digital twins. Traefik manages routing and TLS. The library microservice serves reusable assets, and the DT runner executes commands inside a workspace. The architecture review describes this implemented architecture in detail.

Copyright © 2022 - 2026 The INTO-CPS Association

Implementation Status

4. User View

DTaaS user view

Users see DTaaS as a single web interface for workspace operations, asset browsing, digital twin execution and result exploration. Each user works in their own isolated workspace and shares assets through the common library. The platform supports both class-level twin definitions (reusable templates) and instance-level twins (bound to specific physical assets).

DT class and instance

5. Lifecycle-Driven Engineering

Lifecycle Phases on DTaaS

Lifecycle coverage is a core requirement. The platform supports the create, manage, execute and stop phases, with iterative reconfiguration between them. For this reason the contracts between the client and the backend cover both file management and execution tracking.

The lifecycle model assumes that digital twins evolve: models are updated as physical assets change, new data sources are integrated, and execution configurations are refined based on operational experience.

6. DevOps and Federation

DevOps with GitLab

GitLab-backed DevOps automation is a core platform capability. Repository structure, pipeline triggers and OAuth configuration are part of the platform contract and not only deployment details. The platform supports two execution modes, continuous (long-running services) and one-off (batch analysis jobs), and GitLab CI/CD pipelines manage both.

Related recordings:

- DTaaS DevOps workflows
- DTaaS and DevOps

Federated DTaaS

The federation extension lets multiple DTaaS instances discover and reuse each other's assets across organisational boundaries. Federated digital twins can be organised as composites (assembled from assets on different instances), fleets (coordinated groups) or hierarchies (parent-child relationships).

Related recording:

- Federated DTaaS video

User interactions in Federated DTaaS

7. Application: Structural Health Monitoring

SHM DT Workflow

Structural health monitoring (SHM) of civil infrastructure shows DTaaS used to integrate specialised engineering workflows. In this domain, digital twin assets include sensor data acquisition modules (accelerometers, distributed optical sensors), operational modal analysis functions, finite element model updating procedures and fatigue-based methods for estimating remaining useful life.

The SHM workflow composes these assets into a pipeline. Sensor data flows through modal analysis to extract structural properties. These feed into model updating, which calibrates a finite element model, and the calibrated model drives damage prognosis and remaining useful life estimation. Domain specialists build such workflows from reusable building blocks and keep their existing tools.

Copyright © 2022 - 2026 The INTO-CPS Association

Guidance for Contributors

When implementing or reviewing DTaaS changes:

1. Map the change to one of the concern areas above (architecture, lifecycle, DevOps/federation, domain application, monitoring).
2. Verify the change preserves the intended lifecycle and composability behaviour.
3. Check whether the change shifts architecture boundaries (gateway/auth, execution manager, reusable-asset layer, workspace isolation).
4. Document any intentional departure from design assumptions.

Anti-patterns to avoid:

- Hard-coding one modelling or tooling workflow where the design expects heterogeneity.
- Collapsing layered responsibilities into UI code for short-term convenience.
- Treating DevOps and federation behaviour as optional.

References

The research behind DTaaS is documented in the following papers.

1. **Composable Digital Twins on Digital Twin as a Service Platform** defines the platform architecture, asset model, lifecycle phases and system requirements. (`Talasila, Prasad, et al. "Composable digital twins on Digital Twin as a Service platform." Simulation 101.3 (2025): 287-311.`)
2. **Realising Digital Twins** positions DTaaS in the wider digital twin field, with a comparative survey of existing frameworks, cloud deployment strategies and fleet management concepts. (`Talasila, Prasad, et al. "Realising digital twins." The engineering of digital twins. Cham: Springer International Publishing, 2024. 225-256.`)
3. **Towards Federated Digital Twin Platforms** extends DTaaS with multi-instance federation, DevOps integration and asset reuse across organisations. (`Frasheri, Mirgita, Prasad Talasila, and Vanessa Scherma. "Towards federated digital twin platforms." arXiv preprint arXiv:2505.04324 (2025).`)
4. **Structural Health Monitoring of Engineering Structures Using Digital Twins** applies DTaaS to vibration-based SHM with modal analysis and model updating workflows. (`Talasila, P., et al. "Structural health monitoring of engineering structures using digital twins: A digital twin platform approach." International Conference on Experimental Vibration Analysis for Civil Engineering Structures. Cham: Springer Nature Switzerland, 2025.`)

Copyright © 2022 - 2026 The INTO-CPS Association

6.2.5 Code Review

Architectural and Design Review

This section contains an independent architectural and design review of the DTaaS codebase, performed against the `feature/distributed-demo` branch (commit `e7f9dd6`, July 2026). The findings describe the codebase at that commit.

SCOPE AND METHOD

The review covers the five deliverable packages of the monorepo, and the deployment scenarios that bind them together (the last row):

Package	Path	Technology
React client	`client/`	React 19, Redux Toolkit, oidc-client-ts
Library microservice	`servers/lib/`	NestJS, GraphQL (Apollo), cloudcmd
DT Runner	`servers/execution/runner/`	NestJS, REST, execa
DTaaS CLI	`cli/`	Python, click, python-on-whales
Services manager	`deploy/services/cli/`	Python, click (dtaas-services)
Deployment scenarios	`deploy/dtaas/docker/`, `deploy/vagrant/`	Traefik, docker compose

The review followed a standard system design review framework: requirements traceability, recovery of the high-level architecture from code and deployment artifacts, deep dives into each component, and an analysis of scale, reliability and security across components. Each finding was checked against the source files, and each claim in these documents cites the file it was observed in.

DOCUMENTS

1. Architecture Review: the architecture as implemented, recovered from code and compose files, with data flows and trade-off analysis.
2. Design Review: component-level review of the client, microservices, CLIs and deployment tooling.
3. Systemic Issues: issues that recur across components and releases, with evidence.
4. Improvement Proposals: prioritised, incremental recommendations with migration paths and explicit trade-offs.

CHANGES SINCE THE REVIEW

The findings describe commit `e7f9dd6`; only factual slips about that commit were corrected in place. These statements in them no longer describe the current branch:

- *Packages.* Three packages added since then are not reviewed: the logger microservice (`servers/logger/`), the DT automation library (`lib/dt-automation/`) and the shared GitLab Python library (`lib/gitlab_common/`).
- *Services.* The platform now runs three backend services: the library microservice, the DT Runner and the logger microservice.
- *Image pins.* The docker compose files now pin `intocps/dtaas-web:1.7.1` and `intocps/libms:0.5.13` (at the review: `1.0.2` and `0.5.9`), and add `intocps/logger-ms`.
- *Observability.* The logger microservice exposes `GET /logger/health` and writes workflow events as JSON Lines (`logs/workflow-logs.jsonl` by default). The library microservice and the runner still have no health endpoint.
- *CI.* The `workspace.yml` workflow is gone, and `logger-ms.yml`, `dt-automation.yml`, `gitlab-common.yml`, `docs-publish.yml` and `qlty.yml` have been added.
- *Resource limits.* Users added through the DTaaS CLI get CPU, memory and process limits from the `[common.resources]` section of `dtaas.toml` (`cli/src/templates/users.resources.yml`). The finding on missing limits applies to the users defined directly in the base compose files.

Copyright © 2022 - 2026 The INTO-CPS Association

SUMMARY VERDICT

DTaaS has a clear product vision, unusually good documentation for a research platform and substantial test investment in the client. The main architectural risks are concentrated in four areas:

- the gap between the documented and the implemented architecture;
- the delegation of core platform logic (DT lifecycle, execution, state) to the browser and to GitLab;
- a perimeter-only security model in which internal services trust the network;
- a static per-user deployment model that limits scale.

None of these is fatal. All four can be addressed incrementally, and the improvements document proposes concrete paths.

Copyright © 2022 - 2026 The INTO-CPS Association

Architecture Review (As Implemented)

The architecture page describes the intended architecture. This page documents the architecture that the code and deployment artifacts realised at the time of the review, because the two differ in important ways; see systemic issue SI-1.

REQUIREMENTS TRACEABILITY

The architecture page lists eight user requirements. Their implementation status, recovered from code, is:

Requirement	Status	Implemented by
Author	Implemented	User workspaces (containers), integrated GitLab
Consolidate	Implemented	Library microservice, client library pages
Configure	Partial	Client editors (`client/src/route/digitaltwins/editor`, Monaco)
Execute	Partial	GitLab CI pipelines triggered from the browser; DT Runner
Explore	Partial	Client DevOps pages, IndexedDB execution history
Save	Not implemented	None
Services	Implemented	dtaas-services CLI (MQTT, InfluxDB, Grafana, ...)
Share	Partial	GitLab group/project visibility; common library files

Components named in the architecture but absent from the codebase are the DT Lifecycle Manager, DT Execution Manager, Accounting microservice, Security microservice (RBAC) and a service mesh. The architecture page's own component table already marks three of these "Not available yet", and the review found no partial implementations of any of them.

HIGH-LEVEL ARCHITECTURE (RECOVERED)



Copyright © 2022 - 2026 The INTO-CPS Association

Three facts define this architecture, and the intended-architecture diagram shows none of them:

1. **GitLab is the de facto platform backend.** It provides identity (OAuth), asset and DT storage (repositories), the execution engine (CI pipelines and runners) and authorisation boundaries (group and project membership). Evidence: `lib/dt-automation/src/gitlab/` implements pipeline start, cancel and log fetching against the GitLab REST API, and `deploy/dtaas/docker/*/docker-compose.yml` wires `traefik-forward-auth` to GitLab OAuth endpoints.
2. **The browser is the DT lifecycle manager.** Orchestration runs client-side: resolving projects, fetching trigger tokens, starting pipelines, polling status and collecting logs (`lib/dt-automation/src/gitlab/execution/pipelinePolling.ts`, `instance.ts`). Execution history is persisted in the browser's IndexedDB (`client/src/database/executionHistoryDB.ts`), so the platform itself holds no record of DT executions.
3. **Security is perimeter-only.** Traefik routes by path prefix and `traefik-forward-auth` authenticates at the edge. Behind the perimeter, no service authenticates its callers. The library microservice exposes GraphQL and cloudcmd without authentication checks (`servers/lib/src/` contains no auth middleware), and the DT Runner executes configured commands for any caller (`servers/execution/runner/src/app.controller.ts`).

DATA FLOWS

DT execution flow (primary flow): when the user clicks Execute, the client resolves the GitLab project ID and trigger token, starts a GitLab CI pipeline with variables, polls the pipeline and job status, fetches the job logs and writes the results to browser IndexedDB. The platform's servers take no part in this loop; a network observer sees only browser-to-GitLab traffic.

Asset flow: library assets live either on the file system mounted into `libms` (local backend) or in git repositories (git backend, `servers/lib/src/files/git/git-files.service.ts` using isomorphic-git). The client queries them over GraphQL and renders file trees; cloudcmd provides direct file management.

User provisioning flow: an admin runs the Python CLI (`cli/`), which rewrites docker compose files from templates to add or remove per-user workspace services, then reconciles the running deployment via python-on-whales. Users are compose services, not data.

TRADE-OFF ANALYSIS OF THE CURRENT ARCHITECTURE

The current shape was chosen for real benefits. This section states what each choice costs.

Delegating the backend to GitLab gave the team a large amount of functionality (auth, storage, CI, permissions, web IDE) for almost no implementation cost, which is a rational choice for a small research team. The price is hard vendor coupling, an execution model constrained to CI pipeline semantics, and a security model bounded by what GitLab OAuth scopes can express. The "backend abstraction" in `lib/dt-automation/src/` has exactly one implementation, and its interfaces leak GitLab concepts such as pipelines and trigger tokens.

Putting lifecycle logic in the client avoided building and operating a lifecycle service. The price is that there is no headless API (a CLI or a scheduled job cannot execute a DT without a browser), state dies with the browser profile, trigger tokens are handled in browser memory, and every open tab generates its own polling load.

Perimeter-only security kept the microservices simple. The price is that a single misconfigured Traefik rule, or any foothold inside the docker network, reaches unauthenticated services, including one whose purpose is to execute shell commands.

Users as compose services made multi-user hosting possible with plain docker compose. The price is O(users) compose definitions, a restart of the stack topology to add users, no per-user resource quotas in the base scenarios, and a hard ceiling well below "as a Service" scale.

SCALE AND RELIABILITY POSTURE

The platform services hold no state that matters for load (libms is stateless over mounted files and the runner runs per workspace), so the practical limits are the single Traefik instance, the single GitLab instance and the per-user container model. The single points of failure are Traefik, forward-auth and GitLab. The platform has no health, metrics or alerting support: no `/health` endpoints, no metrics export and no structured audit log of user actions (see SI-8). This is tolerable at current deployment sizes (tens of users per host), and it is the first thing to revisit if institutional deployments grow.

Copyright © 2022 - 2026 The INTO-CPS Association

Design Review (Per Component)

This page collects findings for individual packages. Problems that cut across packages are in Systemic Issues.

REACT CLIENT (`CLIENT/`)

Strengths. The client is the most mature package. It has about 16 kLOC of source and about 28 kLOC of tests (unit, integration and Playwright end-to-end tests under `client/test/`), a documented internal architecture (`lib/dt-automation/src/ARCHITECTURE.md`), typed Redux slices, and runtime configuration injection via `env.js` that lets one build serve many deployments.

Findings.

1. *The backend abstraction leaks.* `BackendInterface` and the builder/strategy design described in `ARCHITECTURE.md` promise swappable backends, but GitLab is the only implementation, and the abstractions expose GitLab concepts directly: the `Pipeline` and `TriggerToken` types (`lib/dt-automation/src/interfaces/execution.ts`) and the `getTriggerToken` methods of `GitlabAPI` (`lib/dt-automation/src/gitlab/backend.ts`) and `GitlabInstance` (`lib/dt-automation/src/gitlab/instance.ts`), along with pipeline refs. A second backend (for example Gitea) could not be written against these interfaces without changing them. Either commit to GitLab and simplify, or lift the abstraction to lifecycle verbs (`start/stop/status/logs`) and push the pipeline vocabulary down into the GitLab adapter.
2. *Too much platform logic lives here.* Pipeline orchestration, polling, log collection, retries and status interpretation (`gitlab/execution/`) are platform behaviour rather than presentation. The consequences are detailed in SI-3.
3. *Execution and measurement history are browser-local.* `src/database/executionHistoryDB.ts` and `measurementHistoryDB.ts` persist to IndexedDB. History is lost on another machine, in another browser or after clearing the profile, and users cannot share it, even though the platform otherwise emphasises sharing.
4. *Parallel `util/` and `utils/` directories* (`src/util/`, `src/utils/`) and separate store trees (`client/src/store/` and `lib/dt-automation/src/store/`) blur module boundaries. This is minor, but this kind of drift spreads.
5. *Benchmark and measurement task code ships with the product client.* `lib/dt-automation/src/gitlab/measure/tasks/` (scenarios with multiple identical DTs and with different runners) is experiment tooling embedded in the production bundle. It belongs in a separate package, or at least behind a build flag.

LIBRARY MICROSERVICE (`SERVERS/LIB/`)

Strengths. The service is small and focused (about 600 LOC of source, with more test code than source). It has a clean NestJS module layout with a config service and a files module whose two swappable providers (local and git) sit behind `files-service.factory.ts`. This strategy pattern does not leak, unlike the client's.

Findings.

1. *No authentication or authorisation of its own.* The GraphQL API and the embedded cloudcmd file manager trust the perimeter entirely. cloudcmd allows file modification and upload, so behind a single mis-scoped Traefik rule it becomes an unauthenticated file manager over the shared asset store.
2. *`mock-fs` is declared as a production dependency* (`package.json` `dependencies`). Every production install and Docker image pulls in this test-only file system mocking library. It should be in `devDependencies`.
3. *Several API surfaces without a stated contract.* The service exposes GraphQL (files resolver), cloudcmd's own HTTP/WebSocket API and git operations via isomorphic-git. The documentation does not say which surface is the supported integration contract; `API.md` and `HTTP.md` describe the surfaces separately.

DT RUNNER (`SERVERS/EXECUTION/RUNNER/`)

Strengths. The runner is minimal by design (about 400 LOC): a whitelist of permitted commands from `runner.yaml`, Zod-validated DTOs and execa-based process management. The whitelist model is the right instinct.

Copyright © 2022 - 2026 The INTO-CPS Association

Findings.

1. *No caller authentication.* Any process that can reach the port can run any whitelisted command and read the execution history. The runner relies entirely on network placement. A bearer-token check, even a static shared secret from its YAML config, would be a cheap and meaningful hardening step.
2. `Queue` *is not a queue.* `queue.service.ts` appends every command it receives to an array; `activeCommand()` returns the last element, nothing is ever dequeued, and `checkHistory()` returns the whole array. It is an unbounded execution log with a misleading name. Memory grows with every command, and the type does not express how concurrent commands are handled (is a second command rejected or queued?). Rename it, bound it and make the concurrency policy explicit.
3. *Silent failure channel.* `newCommand` reduces every failure to one boolean, and the controller maps it to a generic `invalid command` response. The caller cannot tell "not whitelisted" from "spawned but exited non-zero" or "spawn failed".

DTAAS CLI (`CLI/`) AND SERVICES MANAGER (`DEPLOY/SERVICES/CLI/`)

Strengths. The new CLI (commit `e7f9dd6`) consolidates user and configuration management with templates (`dtaas.toml`, generated compose files), validation, and reconcile and uninstall flows. This is the right direction for reducing the deployment matrix. dtaas-services has its own test tree for each service.

Findings.

1. *Two Python CLIs with overlapping missions.* `cli/` (platform installation, users) and `deploy/services/cli/` (third-party services, including its own GitLab compose template) both template compose files and wrap docker, and they are packaged and released separately. The boundary between them is historical rather than conceptual, and admins must learn both.
2. *Compose-file rewriting as the user model.* Users are managed by editing YAML service definitions. This works at current scale, but it couples user management to file-format details and requires file system access to the deployment host. There is no API that an admin UI could call. (Accepted for now; see improvement I-7 before investing here.)

DEPLOYMENT SCENARIOS (`DEPLOY/`)

Findings.

1. *Four docker scenarios, Vagrant and two CLIs* form the real installation matrix (localhost, server, secure-server, secure-server with GitLab, and a single Vagrant machine in `deploy/vagrant/` with optional developer provisioning). Each scenario has its own compose file and CONFIG.md, and drift between them is already visible (see the next point). Each new scenario adds to the testing and documentation burden.
2. *Image version drift and unpinned images.* The docker compose files pin versions of the `intocps/dtaas-web` and `intocps/libms` images that can lag behind the latest releases on Docker Hub. `traefik-forward-auth` uses the `:latest` tag of a third-party image whose upstream maintenance has stalled. The release process does not update the compose files (see SI-7).
3. *No resource limits on user workspaces.* Workspace containers in the base scenarios have no CPU or memory limits, so one user can starve a shared host. The intended architecture assigns these accounting and quota functions to the accounting service, which has not been built.

DOCUMENTATION AND CI

The documentation tree is a real asset: per-package DEVELOPER.md files, admin scenario guides, a test pyramid description and MkDocs publishing. CI runs per package (client, lib-ms, runner, Python CLIs, docs, workspace) with Codecov upload and SonarCloud badges. The gaps: no integration pipeline exercises a composed deployment end to end, and quality gates differ between packages. The client has extensive end-to-end tests, the runner and libms have unit, integration and end-to-end Jest tests but no browser-level tests, and STATUS.md does not show coverage for the Python packages.

Copyright © 2022 - 2026 The INTO-CPS Association

Systemic Issues

These issues cut across packages. They recur in several packages, survive individual refactorings and shape what the platform can become. Each is stated with evidence and consequences; the remedies are in Improvement Proposals.

SI-1: DOCUMENTED VS IMPLEMENTED ARCHITECTURE

The architecture page presents a microservice architecture with a service mesh, security (RBAC), accounting, a DT lifecycle manager and an execution manager. None of these five components exists in any form in the codebase, and the implemented system (edge auth, three services and GitLab) has a very different security and data topology. The page's own component table admits three gaps ("Not available yet"), but the prose and diagrams above it describe the planned system as if it were current.

Consequence: new contributors and adopters evaluating the platform form a wrong model of the system, and design discussions argue from components that do not exist. The architecture that is actually implemented, which is defensible on its own terms, is not written down anywhere. The current-status page partly compensates but does not describe the topology.

SI-2: GITLAB IS A HARD DEPENDENCY, NOT A BACKEND CHOICE

Identity, asset storage, DT execution and authorisation all resolve to GitLab. The client's backend abstraction has one implementation whose interface leaks GitLab vocabulary (trigger tokens, pipelines). Forward-auth is configured against GitLab OAuth URLs, the DevOps integration assumes GitLab CI semantics, and even the services CLI ships a GitLab compose template. There is no seam where a different provider could be inserted without touching every layer.

Consequence: every GitLab breaking change is a platform breaking change. Institutions that run GitHub, Gitea or plain git cannot adopt DTaaS without also adopting GitLab. The execution model is permanently constrained to CI pipeline semantics (batch jobs, per-commit refs, runner scheduling), even where DTs need long-running or interactive execution.

SI-3: THE BROWSER IS THE PLATFORM

DT lifecycle orchestration, pipeline polling, log aggregation and the only persistent record of executions (IndexedDB) live in the React client. The platform's own servers neither mediate nor observe DT executions.

Consequence: no headless automation (CLI, cron job, external system) can run a DT. Execution history is stored per browser profile and cannot support auditing, accounting or the "Save" requirement. Closing a tab orphans in-flight orchestration, and secrets (trigger tokens) pass through browser memory. The unbuilt Lifecycle Manager (SI-1) also becomes harder to introduce, because its logic keeps accumulating client-side in TypeScript coupled to React state.

SI-4: PERIMETER-ONLY TRUST

A single enforcement point (Traefik with forward-auth) protects every service. libms (including cloudcmd file management), the runner (command execution) and the workspace containers accept any request that reaches them. Authorisation works at the granularity of path prefixes. The forward-auth component is a third-party image pulled at `:latest`.

Consequence: one routing misconfiguration, one compromised container on the docker network or one forward-auth regression exposes the entire platform, including arbitrary (whitelisted) command execution in workspaces. The design has no defence in depth, and nothing in the codebase currently provides a second layer that could be turned on.

SI-5: STATIC, FILE-EDITED USER MODEL

Users are docker compose services that the CLI generates from templates. Provisioning requires access to the host file system and a reconciliation of the stack, and per-user resources are unbounded in the base scenarios.

Consequence: scale is capped by compose-file size and the capacity of a single host. Onboarding cannot be self-service or API-driven. Quotas and accounting (a stated prerequisite for commercialisation) have no enforcement point. This is the concrete form of the missing Execution Manager.

SI-6: SPRAWLING DEPLOYMENT AND CONFIGURATION MATRIX

The matrix spans four docker scenarios, each with integrated or external GitLab, plus Vagrant variants. Two Python CLIs both generate compose files, and there are at least four configuration mechanisms (`env.js` runtime injection, `.env` files, YAML service configs, `dtaas.toml`). The documentation covers each cell, CI does not, and drift between cells is already visible.

Consequence: maintenance effort grows with the matrix rather than with features. Every release must be reasoned through N scenarios by hand, and users on the less common paths find the breakage first.

Copyright © 2022 - 2026 The INTO-CPS Association

SI-7: RELEASE ENGINEERING GAPS

Compose scenarios pin superseded image versions (`dtaas-web:1.0.2` against the released 1.3.0; `libms:0.5.9` against 0.5.11). One security-critical image floats on `:latest`. A test library (`mock-fs`) ships in the libms production dependencies. Versions across npm, Docker Hub, PyPI and the compose files are coordinated by hand.

Consequence: installing from the repository and installing the release give different systems. The floating tag creates supply-chain exposure, and misplaced dependencies bloat images and widen the attack surface.

SI-8: NO OBSERVABILITY PLANE

No platform service exposes health or metrics endpoints, there is no structured audit log of user or DT activity, and the admin docs give no monitoring guidance. The only operational signals are container logs and GitLab's own UI.

Consequence: operators learn about failures from user reports, accounting (SI-5) has no data source, and SLO-style operation of an institutional deployment is not possible.

INTERDEPENDENCY MAP

The issues depend on each other. SI-2 and SI-3 together block the Lifecycle and Execution managers whose absence makes up SI-1. SI-5 and SI-8 together block accounting. SI-4 is cheap to fix only while the number of services is small, that is, before fixing SI-1 adds services. The improvement proposals are prioritised in this dependency order:

```
SI-3 (browser logic) ──► fix first: creates the server-side seam
    │
    ├──► enables server-side history/audit ──► SI-8, then SI-5 quotas
    │
    └──► shrinks GitLab surface area ──► makes SI-2 tractable
SI-4 (perimeter trust) ── fix in parallel: cheapest now
SI-6/SI-7 (matrix, releases) ── continuous reduction, CI-enforced
SI-1 (docs) ── fix immediately: writing, not engineering
```


Copyright © 2022 - 2026 The INTO-CPS Association

Improvement Proposals

These proposals are prioritised and incremental. Each names the systemic issues it addresses, sketches a migration path that avoids a big-bang rewrite, and states the trade-off it accepts. The priorities are **P1** (do next; unblocks the rest), **P2** (do after the P1 seams exist) and **P3** (continuous or opportunistic).

I-1 (P1): INTRODUCE A MINIMAL DT LIFECYCLE SERVICE

Addresses SI-1, SI-3; enables I-2, I-4, I-7.

Create the first real increment of the documented DT Lifecycle Manager: a small NestJS service (matching the team's existing skills and the libms and runner idioms) that owns the verbs `start / stop / status / logs / history` for DT executions. Its first version implements them by calling GitLab exactly as the client does today. The orchestration code in `lib/dt-automation/src/gitlab/execution/` serves as a de facto specification and can be ported almost mechanically.

Migration path: (1) stand the service up behind Traefik next to libms; (2) port the pipeline start, poll and log logic from the client; (3) switch the client's `BackendInterface` implementation to call the lifecycle service instead of GitLab, removing gitbeaker from the browser bundle; (4) move trigger tokens server-side. The client's existing abstraction is the right cut point, and this is where building it pays off.

Trade-off: one more service to deploy and operate, and the direct path from browser to GitLab, which currently keeps that traffic off the platform, becomes platform traffic. This is accepted because every further platform capability (headless API, history, accounting, quotas) needs this seam.

I-2 (P1): SERVER-SIDE EXECUTION HISTORY

Addresses SI-3, SI-8; prerequisite for accounting.

Give the lifecycle service a persistent store (SQLite per deployment is enough at first; the schema in `client/src/database/executionHistoryDB.ts` is a starting point) and record every execution with user, DT, timestamps and outcome. IndexedDB can stay as an offline cache, but the server becomes the source of truth. This also produces the audit trail that SI-8 notes is missing.

Trade-off: this introduces the platform's first stateful service, so the admin docs need a note on backup and restore. This is accepted, because IndexedDB cannot serve as the system of record for a multi-user platform.

I-3 (P1): SECOND SECURITY LAYER BEHIND THE PERIMETER

Addresses SI-4.

Three cheap, independent steps:

1. Validate the identity issued by forward-auth at each service. Forward-auth already forwards `X-Forwarded-User`; services should reject requests without it, and the Traefik edge should strip the header so that it cannot be spoofed from outside.
2. Add a shared-secret or JWT check to the DT Runner. It executes commands, so it must not be open.
3. Replace `thomseddon/traefik-forward-auth:latest` with a maintained, version-pinned equivalent (for example oauth2-proxy), or at least pin a digest.

A small shared NestJS guard package keeps this uniform across libms, the runner and the new lifecycle service.

Trade-off: slightly more configuration per service and a migration note for existing deployments. There is no architectural cost; the current design simply omits this defence in depth.

I-4 (P2): RESHAPE THE BACKEND ABSTRACTION AROUND LIFECYCLE VERBS

Addresses SI-2.

After I-1, the GitLab vocabulary (pipelines, trigger tokens, refs) lives in exactly one adapter inside the lifecycle service. Define the provider interface there in DT terms (`execute(dt, config)`, `status(execution)`, `logs(execution)`) and keep GitLab as the reference implementation. Do **not** build a second provider speculatively. The goal is an honest seam, so that a Gitea, GitHub or local-executor provider becomes a bounded project when a concrete adopter needs one.

Trade-off: the abstraction has to be maintained without a second implementation to validate it. Deriving the interface from verbs already in use, rather than from anticipated generality, reduces this risk.

Copyright © 2022 - 2026 The INTO-CPS Association

I-5 (P2): COLLAPSE THE DEPLOYMENT MATRIX

Addresses SI-6, SI-7.

Declare the new DTaaS CLI (`cli/`) the single supported entry point for platform deployment, and turn the four docker scenario directories into generated artifacts (or examples explicitly marked as frozen) instead of parallel copies maintained by hand. Remove the overlap between `cli/` and `deploy/services/cli/`, either by making the services CLI a dependency that the platform CLI invokes or by merging the packages. In CI, add one composed smoke test: bring up the localhost scenario via the CLI, log in, list library assets and run one trivial DT. This single job catches the cross-package breakage that per-package workflows cannot detect.

Trade-off: scenario flexibility narrows and some existing installation habits break, so a deprecation cycle of one release is needed. This is accepted because the matrix is the project's largest hidden maintenance cost.

I-6 (P2): RELEASE-TIME CONSISTENCY CHECKS

Addresses SI-7.

Automate what is currently coordinated by hand. Add a release workflow step that (1) rewrites image tags in the compose scenarios (generated, after I-5) to the versions being released; (2) fails on any `:latest` tag; (3) runs `npm pkg`-level checks such as "no test libraries in `dependencies`". Moving `mock-fs` to devDependencies in libms can happen immediately, independent of the rest. These steps take hours of CI work.

I-7 (P3): QUOTAS AND THE PATH TO THE EXECUTION MANAGER

Addresses SI-5; depends on I-1, I-2.

In the near term, add CPU and memory limits to the workspace container templates. This takes one line per service and protects shared hosts immediately. In the medium term, once the lifecycle service records executions (I-2), quota enforcement has both a data source and an enforcement point. "Users as data" (an API-driven provisioner wrapping the docker API, replacing compose-file rewriting) then becomes the first real slice of the Execution Manager. Defer any discussion of Kubernetes until a deployment outgrows one host; compose with limits is the right tool at current scale.

I-8 (P3): TRUTH-FIRST DOCUMENTATION AND ADRS

Addresses SI-1.

Restructure the architecture page into "Implemented architecture" (the recovered diagram in the architecture review can seed it) and "Target architecture", and make the status column of the component table prominent. Record the load-bearing decisions that are currently implicit (GitLab as backend, perimeter auth, users as services, browser-side orchestration) as short ADRs. The repository has no `adr/` directory today. ADRs let future contributors tell decisions made for stated reasons from ones that accumulated. This proposal costs only writing time and can begin immediately.

OBSERVABILITY NOTE (CROSS-CUTTING)

Fold this into I-1 and I-3 instead of running a separate project: every service gains a `/health` endpoint (NestJS Terminus is one dependency), and the lifecycle service logs structured events. Grafana is already in the list of supported services, so pointing it at the platform's own signals closes SI-8 with tooling that admins already deploy.

WHAT TO REVISIT AS THE SYSTEM GROWS

These signals should trigger a re-evaluation of decisions that this review otherwise endorses:

- more than about 50 workspaces on a host, or any multi-host deployment (revisit compose-based provisioning, I-7);
- a concrete non-GitLab adopter (fund the second provider, I-4);
- commercial accounting requirements (the SQLite store of I-2 becomes Postgres and event-shaped);
- a need for interactive or long-running DT execution. Then the CI pipeline execution model itself becomes the constraint, not just where it runs. That is a larger redesign and should get its own ADR when it arrives.

Copyright © 2022 - 2026 The INTO-CPS Association

6.3 Technical Concepts

6.3.1 OAuth 2.0 and Keycloak Summary

DTaaS uses OAuth 2.0/OIDC at two levels:

- Browser-side login for the web client.
- Gateway-side protection for backend routes and workspaces.

Two Deployment Patterns

1. GITLAB-CENTRED OAUTH

The DTaaS packages that use GitLab as the identity provider follow this pattern.

- The web client authenticates users through the OAuth/OIDC settings in its runtime config.
- `traefik-forward-auth` handles gateway authorization.
- GitLab applications provide the client ID and client secret values that DTaaS components use.

2. KEYCLOAK-CENTRED OIDC

Workspace secure-server deployments use this pattern.

- Keycloak provides the realm, users and OIDC clients.
- `traefik-forward-auth` validates sessions and tokens and enforces route access.
- Workspace routes stay protected when users access different sub-paths.

Control Plane vs Data Plane

- Control plane: provider setup, secret management and callback URI configuration.
- Data plane: the request flow through Traefik and forward-auth before it reaches DTaaS.

Core Runtime Values

The secure-server OIDC variables include:

- `KEYCLOAK_ISSUER_URL`
- `KEYCLOAK_CLIENT_ID`
- `KEYCLOAK_CLIENT_SECRET`
- `OAUTH_SECRET`

GitLab-based scenarios set the equivalent OAuth authority and client values in their own configuration files.

Debugging Login Failures

When debugging a login failure, first identify which identity provider pattern is active. Most failures come from a mismatch in one of these:

- Redirect/callback URL.
- Client ID and secret.
- Hostname and TLS assumptions.
- User whitelist or route policy in the forward-auth config.

Check the configuration first, then the gateway and provider logs.

Copyright © 2022 - 2026 The INTO-CPS Association

6.3.2 OAuth 2.0 Summary

The Authentication Microservice (Auth MS) follows the OAuth 2.0 RFC specification. This page summarises OAuth 2.0 and how the DTaaS platform uses it.

Entities

OAuth 2.0, when used to verify user identity, has three main entities:

- **The User:** the entity whose identity the platform is trying to verify. In DTaaS, this is the user of the DTaaS software.
- **The Client:** the entity that wants to know or verify the identity of a user. In DTaaS, this is the Auth MS, initialised with a GitLab application. Do not confuse it with the DTaaS frontend website, which the rest of the documentation also calls the client.
- **The OAuth 2.0 Identity Provider:** the entity that lets the client know the identity of the user. In DTaaS, this is GitLab. Users usually already have a protected account with this provider. The account is registered under a unique key, such as an email ID or username, and is usually password protected so that only its owner can log in. After logging in, the user is asked to approve sharing their profile information with the client. If they approve, the client gets access to the user's email ID, username and other profile information, which it uses to verify the user's identity.

Note: In general, the authorization server (which requests user approval) and the resource (user identity) provider can be two different servers. In DTaaS, the GitLab instance handles both through different API endpoints. The concepts stay the same, so this page talks about three entities: the User, the OAuth 2.0 Client and the GitLab instance.

THE OAUTH 2.0 CLIENT

Many platforms can initialise an OAuth 2.0 client. DTaaS does it by creating an "application" in GitLab. The client does not have to be initialised on the same platform as the identity provider; the two are separate concerns. The DTaaS OAuth 2.0 client is a GitLab instance-wide application, and its configuration has two main elements:

- **Redirect URI**: the URI to which users are redirected after they approve sharing information with the client.
- **Scopes:** the types and levels of access the client has to the user's profile. DTaaS needs only the `read_user` scope, which allows access to the user's profile information for identity verification.

Creating the GitLab application generates a Client ID and a Client Secret. Any application that holds these credentials becomes an OAuth 2.0 Client, so the Client Secret must never be shared. The DTaaS Auth MS uses the Client ID and Client Secret to act as an OAuth 2.0 Client and follow the OAuth 2.0 workflow to verify user identity.

OAuth 2.0 Workflows

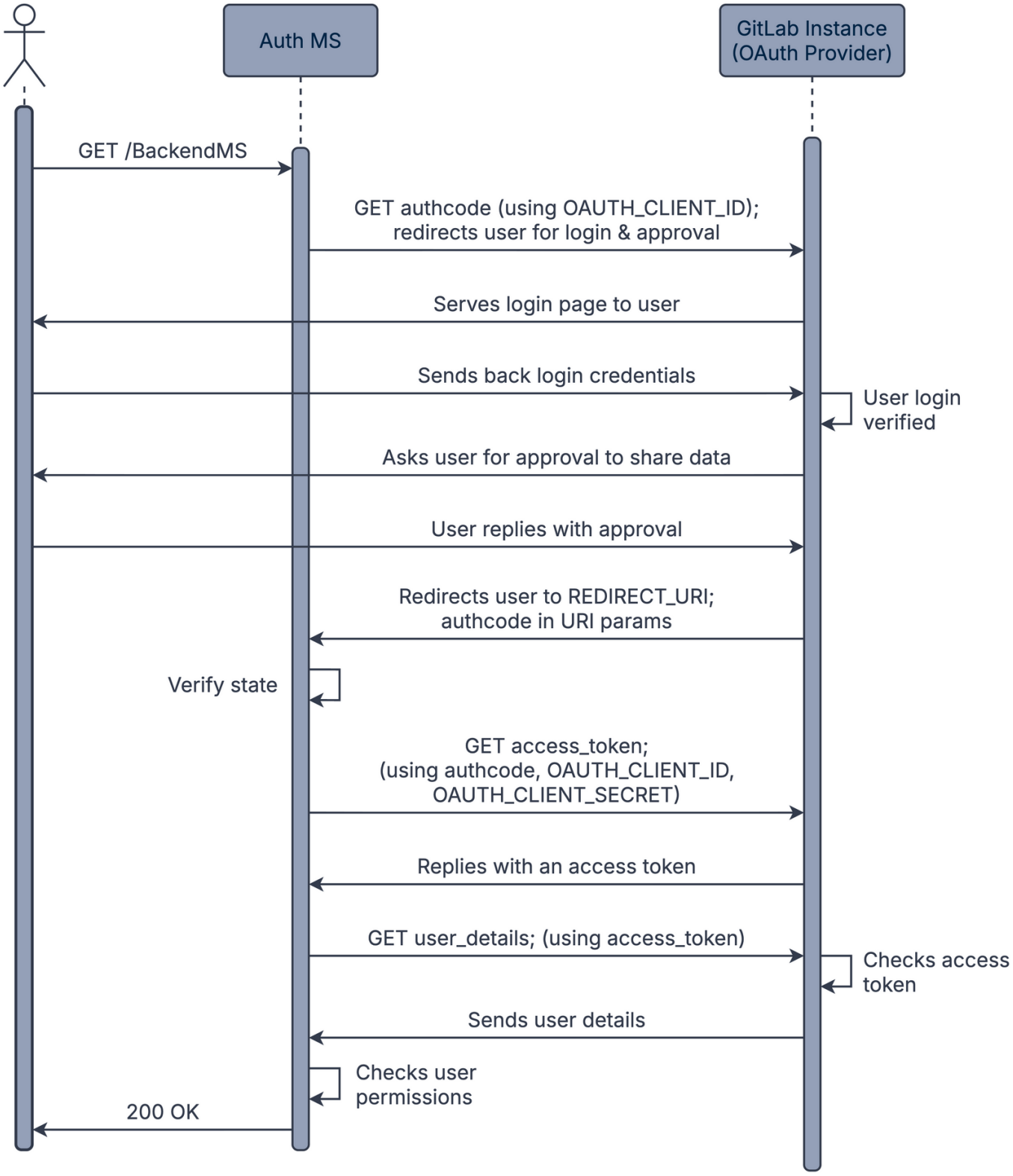
The DTaaS platform uses two OAuth 2.0 flows.

OAUTH 2.0 AUTHORIZATION CODE FLOW

In this flow, the client exchanges an authorization code for access tokens. The DTaaS Auth MS, which secures all backend DTaaS services, uses this flow.

The Client (Auth MS) starts the workflow whenever it needs to verify a user's identity. It sends an authorization request to GitLab and tries to obtain an access token, which lets it fetch user information. With that information, the Auth MS verifies the user's identity and decides whether the user may access the requested resource.

Copyright © 2022 - 2026 The INTO-CPS Association



The diagram shows the requests from the Auth MS to the OAuth 2.0 provider in abbreviated form. The Auth MS implementation documentation explains the DTaaS workflow in detail.

OAUTH 2.0 PKCE (PROOF KEY FOR CODE EXCHANGE) FLOW

PKCE is an extension to the OAuth 2.0 Authorization Code Flow that adds security for public clients, which cannot store client secrets securely. It protects against attacks such as authorization code interception.

The DTaaS client website uses the PKCE flow for login. The Auth0 documentation describes this flow in more detail.

Copyright © 2022 - 2026 The INTO-CPS Association

6.3.3 System Design of DTaaS Authorization Microservice

DTaaS needs backend authorization to protect its backend services and user workspaces. This page describes the system design of the DTaaS Auth Microservice, which provides it.

Requirements

The Auth MS only has to answer requests of the general form "Is User X allowed to access /BackendMS/example?".

If the GitLab OAuth 2.0 provider verifies the user's identity AND the user is allowed to access the requested microservice or action, the Auth MS responds with a 200 (OK) code and lets the request pass through the gateway to the microservice or server.

If identity verification through GitLab OAuth 2.0 fails OR the user is not permitted to access the requested resource, the Auth MS responds with a 40X (NOT OK) code and stops the request.

Forward Auth Middleware in Traefik

Traefik lets you attach middlewares to the routes configured in it. A middleware intercepts requests on a route and can analyse or modify them before passing them on to the services. Traefik's ForwardAuth middleware delegates authentication to an external service. If the external authentication server responds with a 2XX code, the middleware acts as a proxy and lets the request through to the service. If the external server responds with any other code, the middleware drops the request and returns that response code to the user.



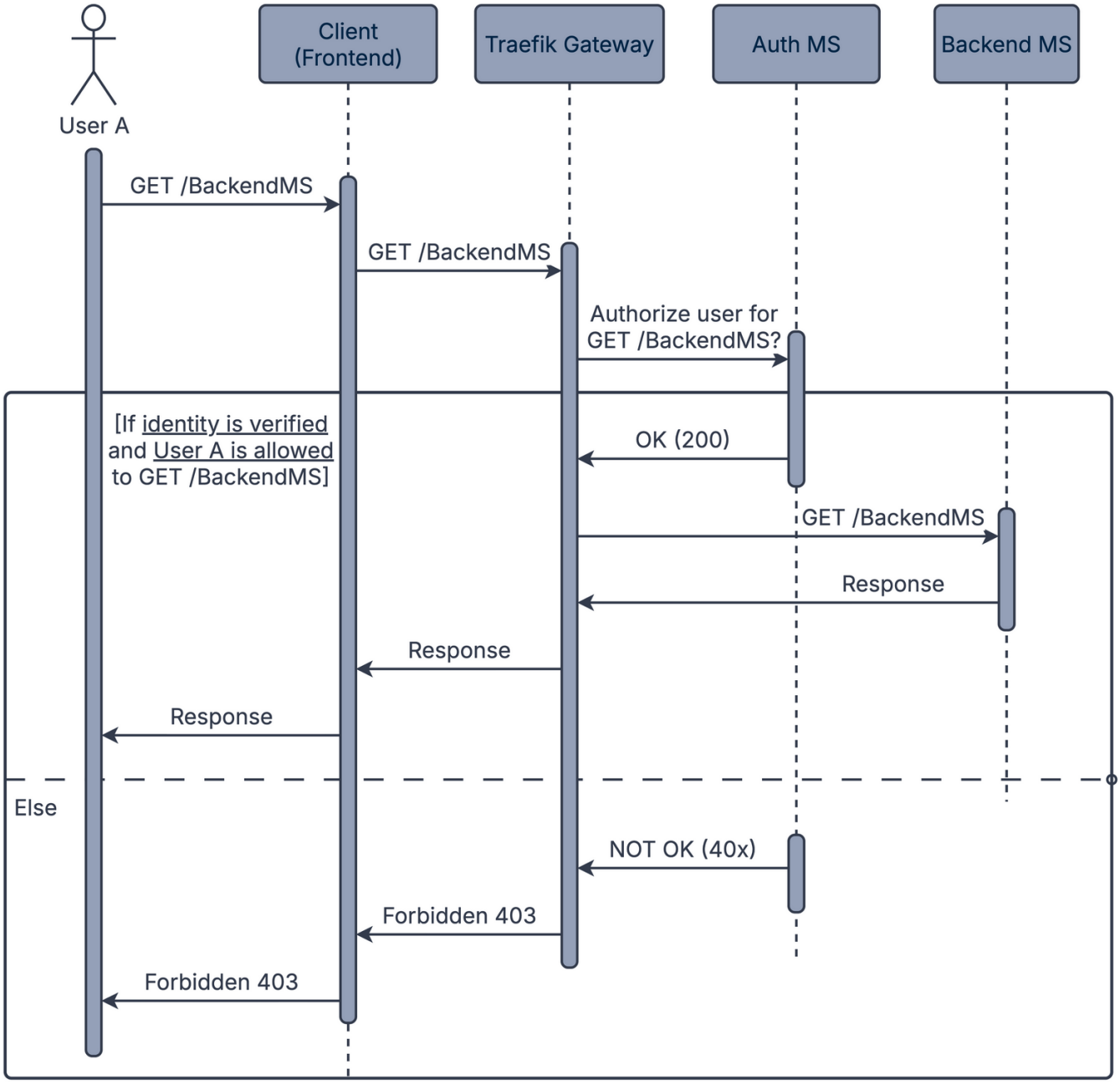
(source: Traefik documentation)

An Auth Microservice can therefore be added to the existing gateway and DTaaS system by making it the external authentication server for ForwardAuth middlewares. These middlewares can be attached to any route that needs authentication. DTaaS attaches it to all routes, because the platform requires at least identity verification for every request through the gateway.

Auth MS Design

The sequence diagram shows how the integrated Auth MS works.

Copyright © 2022 - 2026 The INTO-CPS Association



- The user makes every request through the React website, the frontend of the DTaaS platform.
- The request goes through the Traefik gateway, where the route's ForwardAuth middleware intercepts it.
- The middleware asks the Auth MS whether this request from this user should be allowed.
- The Auth MS (the auth server) verifies the user's identity using OAuth 2.0 with GitLab, and checks whether the user may make this request.
- If the user is verified and allowed to make the request, the auth server responds with 200 OK to the Traefik gateway (more precisely, to the middleware in Traefik).
- Traefik then forwards the request to the service. Any response from the service passes back along the chain to the user.
- If the user is not verified or not allowed to make the request, the auth server responds with a 40X code to the Traefik gateway.
- Traefik then drops the request and tells the client that the request was forbidden, passing on the auth server's response code.

Copyright © 2022 - 2026 The INTO-CPS Association

6.3.4 Auth Microservice

This page describes the workflow and implementation of the DTaaS Auth Microservice. Read the System Design and the OAuth 2.0 summary first.

Workflow

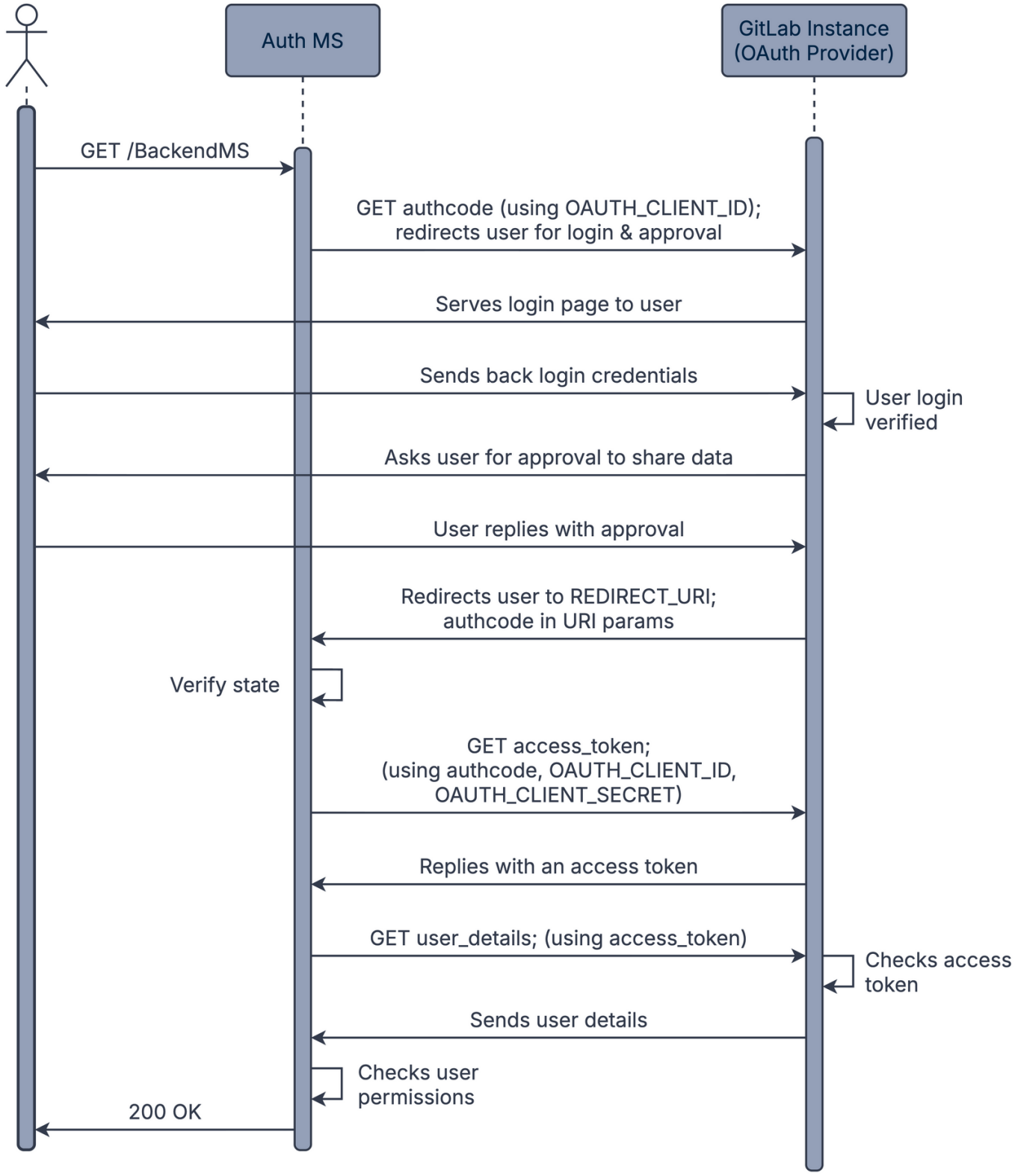
USER IDENTITY USING OAUTH 2.0

This discussion uses the following constants:

- CLIENT ID: the OAuth 2.0 Client ID of the Auth MS.
- CLIENT SECRET: the OAuth 2.0 Client Secret of the Auth MS.
- REDIRECT URI: the URI to which the user is redirected after approving sharing of information with the client.
- STATE: a random string that identifies a specific "GET authcode" request.
- AUTHCODE: the single-use authorization code that the OAuth 2.0 provider (the GitLab instance) returns in response to "GET authcode" after the user approves.

Assume also that DTaaS uses a dedicated GitLab instance at https://gitlab.intocps.org (instead of https://intocps.org).

Copyright © 2022 - 2026 The INTO-CPS Association



A successful OAuth 2.0 workflow has the following steps:

- The user requests a resource, say *GET/BackendMS*.
- The Auth MS intercepts this request and starts the OAuth 2.0 process.
- The Auth MS sends an authorization request to the GitLab instance.

Copyright © 2022 - 2026 The INTO-CPS Association

The diagram abbreviates this request as *GET/authcode*. The actual request (a user redirect) looks like:

```
https://gitlab.intocps.org/oauth/
authorize?
response_type=code&
client_id=OAUTH_CLIENT_ID&
redirect_uri=REDIRECT_URI&
scope=read_user&state=STATE
```

Here `gitlab.intocps.org/oauth/authorize` is the GitLab endpoint that handles authorization code requests.

The query parameters include the expected response type, which is fixed as `code`, meaning that an authorization code is expected. The other parameters are the client ID, the redirect URI, the scope (`read_user` here) and the state, the random string that identifies the request.

- The OAuth 2.0 provider redirects the user to the login page, where the user logs into their protected account with their username or email ID and password.
- The OAuth 2.0 provider then asks the user to approve or deny sharing the requested information with the Auth MS. The user must approve for authentication to succeed.
- After approval, the GitLab instance redirects the user to the REDIRECT URI, in this form:

```
REDIRECT_URI?code=AUTHCODE&state=STATE
```

The REDIRECT URI is the one set when the OAuth 2.0 client was initialised, which is also the one the Auth MS sent in the "GET authcode" request. The GitLab instance adds the query parameters: the AUTHCODE that the Auth MS requested, and the STATE, which is the same random string as in the "GET authcode" request.

- The Auth MS reads these query parameters and checks that STATE matches the random string it sent in the "GET authcode" request. This confirms that the AUTHCODE answers its own request.
- The Auth MS exchanges the single-use AUTHCODE for an access token. The access token can be used more than once but expires after a set time. For the exchange, the Auth MS makes another request to the GitLab instance. The sequence diagram abbreviates it as *GET/access_token*. The actual request is:

```
POST https://gitlab.intocps.org/oauth/token,
parameters = 'client_id=OAUTH_CLIENT_ID&
client_secret=OAUTH_CLIENT_SECRET&
code=AUTHCODE&
grant_type=authorization_code&
redirect_uri=REDIRECT_URI'
```

For most OAuth 2.0 providers, exchanging an authorization code for a token is a POST request. The https://gitlab.intocps.org/oauth/token endpoint handles token exchange requests. The POST request carries the client ID, the client secret, the AUTHCODE and the redirect URI. The grant type is always `authorization_code`, which says that an authorization code is being exchanged for an access token.

- The GitLab instance exchanges a valid AUTHCODE for an access token and sends it to the Auth MS. An example response:

```
{
  "access_token": "d8aed28aa506f9dd350e54",
  "token_type": "bearer",
  "expires_in": 7200,
  "refresh_token":"825f3bffb2544b976633a1",
  "created_at": 1607635748
}
```

The `access_token` field holds the string to use as an access token in the headers of requests for user information. The `token_type` field is usually `bearer`. The `expires_in` field gives the number of seconds the token stays valid, and `created_at` is the Unix timestamp at which it was created. The `refresh_token` field holds a string that can extend the access token's lifetime, but the

Copyright © 2022 - 2026 The INTO-CPS Association

Auth MS does not use it: when an access token expires, the Auth MS requests a new one. Below, TOKEN stands for the access token string from this response.

- The Auth MS now has an access token that it can use to retrieve the user's information. If the Auth MS already held a valid access token for this user, it would skip the steps above, which exist only to obtain that token.
- The Auth MS makes a final request to the GitLab instance, abbreviated as *GET user_details* in the sequence diagram. The actual request is:

```
GET https://gitlab.intocps.org/api/v4/user
"Authorization": Bearer <TOKEN>
```

Here, https://gitlab.intocps.org/api/v4/user is the API endpoint that returns user information. The request needs an authorization header with a valid access token; TOKEN is the access token that the Auth MS holds.

- The GitLab instance verifies the access token and, if it is valid, responds with the user's information, including the username and email ID. An example response:


```
{
  "id": 8,
  "username": "UserX",
  "name": "XX",
  "state": "active",
  "web_url": "http://gitlab.intocps.org/UserX",
  "created_at": "2023-12-03T10:47:21.970Z",
  "bio": "",
  "location": "",
  "public_email": null,
  "skype": "",
  "linkedin": "",
  "twitter": "",
  "organization": "",
  "job_title": "",
  "work_information": null,
  "followers": 0,
  "following": 0,
  "is_followed": false,
  "local_time": null,
  "last_sign_in_at": "2023-12-13T12:46:21.223Z",
  "confirmed_at": "2023-12-03T10:47:21.542Z",
  "last_activity_on": "2023-12-13",
  "email": "UserX@localhost",
  "projects_limit": 100000
}
```


The important fields in this response are `email` and `username`. Each is unique to a user, so either one identifies the user.

- The Auth MS reads candidate key fields such as `email` and `username` from the response. The Auth MS now knows who the user is.

CHECKING USER PERMISSIONS - AUTHORIZATION

The Auth MS also enforces access policies for DTaaS resources. Some resources or microservices may need to be restricted to certain users. For example, /BackendMS/user1 might be open only to the user with username user1, and /BackendMS/group3 only to users with an email ID in the @gmail.com domain. The Auth MS has to impose such restrictions. This takes two steps:

- First, the Auth MS must know and trust the user's identity, that is, believe that the user is who they claim to be. The previous section achieves this.
- Second, the Auth MS checks this identity against a set of rules or a database of allowed users to decide whether the user may access the requested resource.

The second step needs, for every service, either a set of rules that defines which users may access it, or a database of the user identities allowed to access it. The rules or database use the user identity, here the email ID or username, to make the decision. Rules are written against the form of the username or email ID, for example as a regular expression. A database uses the user identity as its key, and a simple lookup decides whether the user is allowed or denied.

In the sequence diagram, the Auth MS has a self-request marked "Checks user permissions" after it receives the user identity from the GitLab instance. At this point the Auth MS compares the user's identity with the rules or database for the requested

Copyright © 2022 - 2026 The INTO-CPS Association

service. If the identity has access to the resource, the Auth MS responds with 200 OK. Authentication has then succeeded and the user can access the resource.

Note: the Auth MS and the user never communicate directly. The Auth MS sends and receives all requests and responses through the Traefik gateway. It is the external server of the route's ForwardAuth middleware and talks only to that middleware. If authentication succeeds, the gateway forwards the request to the resource when it receives the 200 OK; otherwise it drops the request and returns the error code to the user.

Implementation

TRAEFIK-FORWARD-AUTH

The Auth MS is the open-source thomseddon/traefik-forward-auth, configured for DTaaS.

traefik-forward-auth is published as a docker.io image and runs as a Docker container, so it brings no dependency management issues. It is added to Traefik routers as a middleware server, so it needs Traefik to work, as well as the services whose access it controls. Traefik, the traefik-forward-auth service and the other services run together as a stack of Docker containers, and most of the setup is in the `docker-compose.yml` file.

Configuring the Auth MS takes three steps.

- Configure the traefik-forward-auth service. First set the environment variables. Because GitLab is the provider, the generic-oauth provider configuration is used. The required variables include the OAuth 2.0 Client ID, Client Secret and scope, as well as the API endpoints for getting an AUTHCODE, exchanging the code for an access token and getting user information.

Also create a router for the REDIRECT URI path, with traefik-forward-auth itself as its middleware. After approval, the user is sent to the REDIRECT URI; this router lets the gateway pass that request to the Auth service for the token exchange. The configuration also defines the ForwardAuth middleware, which the design above requires, and a load balancer for the service. A conf file holding the selective authorization rules (described later) is mounted as a volume. This follows the suggested configuration. The following is added to the Docker services:

```
traefik-forward-auth:
  image: thomseddon/traefik-forward-auth:latest
  volumes:
    - <filepath>/conf:/conf
  environment:
    - DEFAULT_PROVIDER=generic-oauth
    - PROVIDERS_GENERIC_OAUTH_AUTH_URL=https://gitlab.intocps.org/oauth/authorize
    - PROVIDERS_GENERIC_OAUTH_TOKEN_URL=https://gitlab.intocps.org/oauth/token
    - PROVIDERS_GENERIC_OAUTH_USER_URL=https://gitlab.intocps.org/api/v4/user
    - PROVIDERS_GENERIC_OAUTH_CLIENT_ID=OAUTH_CLIENT_ID
    - PROVIDERS_GENERIC_OAUTH_CLIENT_SECRET=OAUTH_CLIENT_SECRET
    - PROVIDERS_GENERIC_OAUTH_SCOPE=read_user
    - SECRET=a-random-string
    # INSECURE_COOKIE is required if
    # not using a https entrypoint
    - INSECURE_COOKIE=true
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.redirect.entryPoints=web"
    - "traefik.http.routers.redirect.rule=PathPrefix(`/_oauth`)"
    - "traefik.http.routers.redirect.middlewares=traefik-forward-auth"
    - "traefik.http.middlewares.traefik-forward-auth.forwardauth.address=http://traefik-forward-auth:4181"
    - "traefik.http.middlewares.traefik-forward-auth.forwardauth.authResponseHeaders=X-Forwarded-User"
    - "traefik.http.services.traefik-forward-auth.loadbalancer.server.port=4181"
```

- Add the traefik-forward-auth service to the backend services as a middleware.

To do this, add the following lines to the docker-compose configuration of each service:

```
    - "traefik.http.routers.<service-router>.rule=Path(`/<path>`)"
    - "traefik.http.routers.<service-router>.middlewares=traefik-forward-auth"
```


Copyright © 2022 - 2026 The INTO-CPS Association

This creates a router for the route and attaches the auth middleware to it.

- Finally, set user permissions by writing rules in the conf file. Each rule has a name (its identifier) and the route on which it applies. A rule also has an action, either `auth` or `allow`. With `allow`, requests on the route skip even the OAuth 2.0 identification. With `auth`, requests on the route require OAuth 2.0 user identification and follow the sequence diagram. For `auth` rules, the Auth MS retrieves the user information and uses the user's email ID as the identity. Two kinds of restriction can be configured on this identity:
- Whitelist: a list of user identities (email IDs in DTaaS) that may access the route.
- Domain: a domain (for example, gmail.com). Only email IDs in that domain (for example, johndoe@gmail.com) may access the route.

Setting either property on an `auth` rule gives selective access control over the route. If an `auth` rule sets neither, the OAuth 2.0 process still runs and retrieves the user identity, but every known user (every user who completes OAuth 2.0) may access the resource.

DTaaS currently uses only whitelist rules.

The rules can be used in three ways, shown below with the lines to add to the conf file.

- No Auth: serves the `Path(`/public`)` route with an `allow` rule.

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

- User specific: serves the `Path(`/user1`)` route with a rule that allows only the `user1@localhost` identity.

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

- Common Auth: serves the `Path(`/common`)` route with a rule that requires OAuth 2.0 (`action=auth`) but allows every valid, known user identity.

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


Copyright © 2022 - 2026 The INTO-CPS Association

6.3.5 GitLab CI/CD Infrastructure

The files in the Library are stored in a Git repository, so DTaaS uses GitLab's parent-child pipelines to run them. A parent pipeline starts another pipeline, the child pipeline, in the same project. GitLab's documentation on CI/CD Pipelines describes pipelines in more detail.

CI/CD Pipelines

Continuous Integration (CI) and Continuous Deployment (CD) are two parts of the DevOps methodology.

CI means integrating code changes into a shared repository often. Each integration triggers automated builds and tests, so problems are detected early. Changes are checked quickly, which reduces integration problems.

CD automates the release process: code changes are tested and prepared for deployment automatically. Teams using CD can deploy updates quickly and reliably. Together, CI/CD automates the software delivery pipeline and removes most or all of the manual work needed to move a code change from a commit to a production environment. The process covers compilation, testing (unit, integration and regression), deployment and infrastructure provisioning.

GitLab explains CI/CD practices in more detail in this article.

A CI/CD pipeline is a series of automated processes that carry out CI and CD. Once started, a pipeline runs without manual intervention.

GitLab covers the whole DevOps lifecycle in one application, including everything CI/CD needs. Its pipeline documentation was the main reference for this design.

A pipeline is made of jobs and stages. *Jobs* define the tasks to carry out, and *stages* define the order in which jobs run. When all jobs in a stage succeed, the pipeline moves on to the next stage. If any job fails, the pipeline stops.

When a pipeline starts, its jobs are distributed among the available runners.

GitLab runners are agents of the GitLab Runner application. They execute jobs according to their configuration and the available resources. Runners can run on virtual machines, in containers or on physical servers, and can be managed locally or in the cloud.

DTaaS uses this parent-child pipeline setup to execute the digital twins stored in a user's GitLab repository.

> **Note**
>
> The recommended way to modify these pipelines is through the Digital Twins Preview Page.

Parent Pipeline

The parent pipeline is the top-level element. It has a single stage, `triggers`, which triggers the child pipelines.

The `.gitlab-ci.yml` file in the user repository holds the triggers for the DTs. Each trigger belongs to one DT and becomes active when the API call provides the matching value of the `DTName` variable. The `RunnerTag` variable sets a custom runner tag that executes each job in the DT's pipeline.

For one parent pipeline execution, the generated `DTName` rules must activate exactly one trigger job. The client treats zero downstream pipelines as a transitional state and rejects multiple downstream pipelines as a configuration error; it never picks the first result in API order.

Copyright © 2022 - 2026 The INTO-CPS Association

The keywords used in the pipeline configuration are:

- `image`: the Docker image, such as `fedora:41`, that provides the environment for the pipeline.
- `stages`: the phases of the pipeline, such as `triggers`, which run in order.
- `trigger`: starts another pipeline, using the configuration from an external file.
- `include`: imports configuration from another file.
- `rules`: conditions for running a job, based on variables or states.
- `if`: a condition within `rules` that decides, from the value of a variable, whether a job runs.
- `when`: when the job runs, such as `always`.
- `variables`: variables, like `RunnerTag`, passed to the pipeline.

Here is an example YAML file that registers a trigger for a DT named `mass-spring-damper`:

```yaml
image: fedora:41

stages:
  - triggers

trigger_mass-spring-damper:
  stage: triggers
  trigger:
    include: digital_twins/mass-spring-damper/.gitlab-ci.yml
  rules:
    - if: '$DTName == "mass-spring-damper"'
      when: always
  variables:
    RunnerTag: $RunnerTag
```

Digital Twin Structure

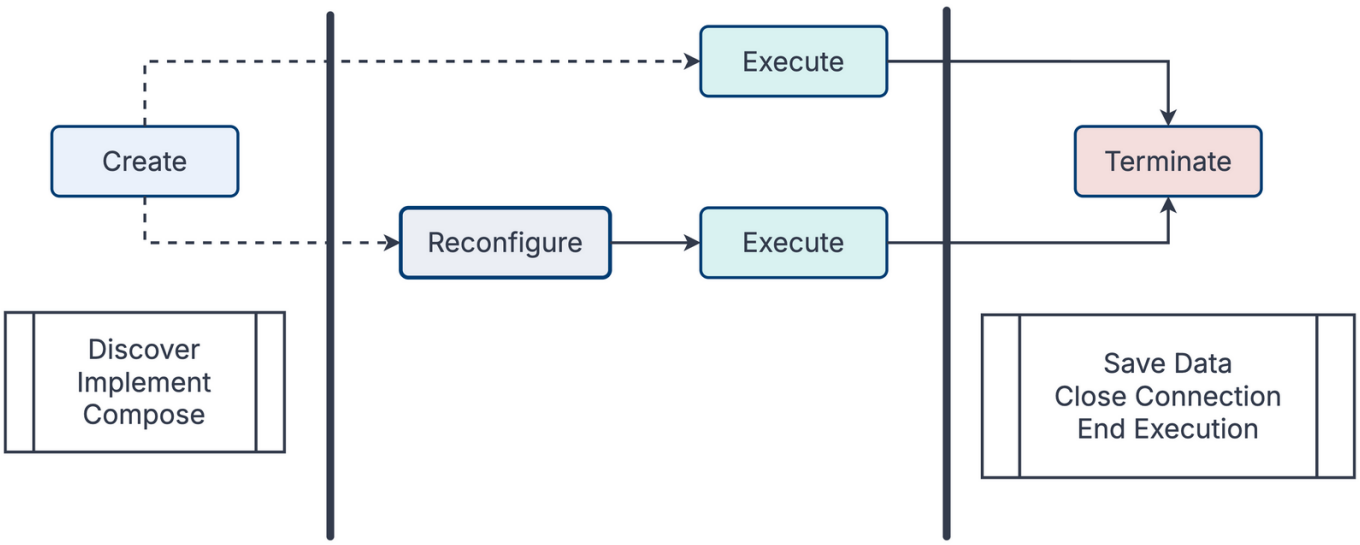
The `digital_twins` folder contains DTs built by one or more users. They are meant to be flexible enough to reconfigure for specific use cases.

For example, the dtaas/user1 repository on gitlab.com has a `digital_twins` directory with a `hello_world` example. Its file structure is:

```
hello_world/
├── lifecycle/
│   ├── clean
│   ├── create
│   ├── execute
│   └── terminate
├── .gitlab-ci.yml
└── description.md
```

The `lifecycle` directory contains four files, `clean`, `create`, `execute` and `terminate`, which are simple BASH scripts. They correspond to stages in a digital twin's lifecycle.

Copyright © 2022 - 2026 The INTO-CPS Association



Child Pipelines

Each DT folder contains a child pipeline that automates the DT's lifecycle. Each child pipeline ignores the image given in the parent pipeline and uses the image set in its own YAML configuration, or Ruby's default image if none is set.

The keywords used in a child pipeline, based on GitLab's CI/CD YAML syntax reference, are:

1. `stage` : the steps of the pipeline, such as create, execute and clean, which run in a fixed order.
2. `script` : the commands to run at each step, for example changing directories, modifying permissions or running lifecycle scripts.
3. `tags` : which runner runs the jobs, giving control over where and how they run.

For the `mass-spring-damper` DT, the stages create, execute and terminate the DT simulation, then clean up the environment so it is ready for the next execution. Here is an example configuration that defines `create`, `execute` and `clean` in the child pipeline:

```
image: ubuntu:20.04

stages:
    - create
    - execute
    - clean

create_mass-spring-damper:
    stage: create
    script:
        - cd digital_twins/mass-spring-damper
        - chmod +x lifecycle/create
        - lifecycle/create
    tags:
        - $RunnerTag

execute_mass-spring-damper:
    stage: execute
    script:
        - cd digital_twins/mass-spring-damper
        - chmod +x lifecycle/execute
        - lifecycle/execute
    tags:
        - $RunnerTag

clean_mass-spring-damper:
    stage: clean
    script:
        - cd digital_twins/mass-spring-damper
        - chmod +x lifecycle/terminate
        - chmod +x lifecycle/clean
        - lifecycle/terminate
    tags:
        - $RunnerTag
```


Copyright © 2022 - 2026 The INTO-CPS Association

Direct API Calls

A GitLab DevOps pipeline can be triggered through an API call with a pipeline trigger token created on the GitLab instance. The call needs these values:

1. `<trigger_token>`: the user's GitLab trigger token.
2. `<digital_twin_name>`: the name of the DT (e.g. mass-spring-damper).
3. `<runner_tag>`: the tag of the GitLab runner the user wants to use.
4. `<project_id>`: the ID of the GitLab project, shown on the project overview page.

The example below sets the `DTName` variable to the DT name and the `RunnerTag` variable to the GitLab Runner tag, and runs the pipeline on the `main` branch:

```
curl --request POST \
  --form "token=<trigger_token>" \
  --form ref=main \
  --form "variables[DTName]=<digital_twin_name>" \
  --form "variables[RunnerTag]=<runner_tag>" \
  "https://intocps.org/gitlab/api/v4/projects/<project_id>/trigger/pipeline"
```

Reference

Vanessa Scherma, Design and implementation of an integrated DevOps framework for Digital Twins as a Service software platform, Master's Degree Thesis, Politecnico Di Torino, 2024.

Copyright © 2022 - 2026 The INTO-CPS Association

6.3.6 Testing

Fundamental Concepts in Software Testing

DEFINITION OF SOFTWARE TESTING

Software testing investigates the quality of a software product across different scenarios. It can also be defined as the process of verifying and validating that a program works as expected and meets the business and technical requirements that guided its design and development.

IMPORTANCE OF SOFTWARE TESTING

Testing finds defects introduced during the different development phases and checks that the product works as expected in the expected scenarios. A stronger test suite gives more confidence in the product. Tests also let developers make incremental changes to the source code and check that those changes do not break existing functionality.

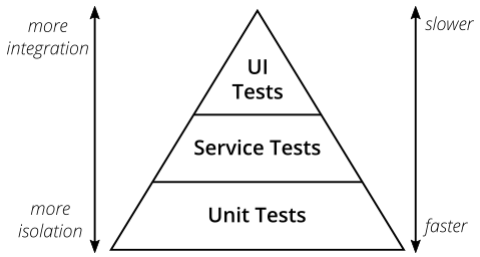
TEST DRIVEN DEVELOPMENT (TDD)

Test Driven Development (TDD) is a development process built on repeating a very short cycle. The developer first writes an automated test case, which initially fails, that defines a desired improvement or new function. They then write the minimum code needed to pass that test, and finally refactor the new code to acceptable standards. TDD is closer to specification than to validation: it gives a method for thinking through requirements or design before writing functional code.

BEHAVIOUR DRIVEN DEVELOPMENT (BDD)

Behaviour Driven Development (BDD) grew out of TDD. It keeps the practice of writing tests first, but its tests describe behaviour instead of verifying a unit of implementation. Development and management teams get a shared set of tools and a shared process for working together. BDD usually relies on a simple domain-specific language (DSL) with natural language constructs (e.g., English-like sentences) that express behaviour and expected outcomes. The Mocha and Cucumber testing libraries are built around BDD.

Testing Workflow



(Ref: Ham Vocke, The Practical Test Pyramid)

The DTaaS project follows the test pyramid shown above. Testing of a new feature starts with isolated tests and moves towards full integration. The types of tests are explained below, in the order in which they should be performed.

UNIT TESTS

Unit testing tests the individual units or components of a piece of software. Its aim is to isolate a section of code and verify that it is correct.

Ideally, each test case is independent of the others. Substitutes such as method stubs, mock objects and spies help test a module in isolation.

Benefits of Unit Testing

- Unit tests increase confidence when changing and maintaining code. When good unit tests run every time the code changes, defects introduced by the change are found quickly.
- Code designed with fewer interdependencies, to make unit testing easier, suffers fewer unintended effects from changes.
- Fixing a defect found during unit testing costs less time, effort and money than fixing one found at a higher level.

Copyright © 2022 - 2026 The INTO-CPS Association

Unit Tests in DTaaS

Each DTaaS component uses its own technology stack, so the unit test packages differ between components. The client uses Jest and React Testing Library, and the NestJS servers use Jest. The Python CLIs and libraries use pytest. The build and test command for each project is listed in `PROJECT.md` in the repository root.

INTEGRATION TESTS

Integration testing combines individual software modules and tests them as a group. The DTaaS project uses an integration server for both software development and integration testing.

The existing integration tests work at the component level. Integration tests between components have not been implemented yet and are deferred to future work.

END-TO-END TESTS

Testing code changes through the end-user interface shows whether the code has the intended effect for users. End-to-End tests in DTaaS require a working setup.

The client has end-to-end tests written with Playwright. The library microservice, the runner and the logger microservice have end-to-end tests run with Jest (`yarn test:e2e`).

FEATURE TESTS

A software feature is a change to the system that adds new functionality or modifies existing functionality. New features should be tested when they are implemented, along with checks that they do not break existing features.

The DTaaS project does not have feature tests yet. Cucumber is planned for implementing them.

References

1. Arthur Hicken, Shift left approach to software testing
2. Justin Searls and Kevin Buchanan, Contributing Tests wiki.
3. This wiki has a good explanation of TDD and test doubles.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4 Codebase

6.4.1 Codebase Overview

DTaaS is a monorepo. It holds the web client, the backend services, shared libraries, deployment packages, admin tooling, documentation and release scripts.

Top-level layout

```
DTaaS/
├── .agents/skills/        # Agent Skills for coding agents
├── .claude/skills/        # Claude Code stubs for the skills
├── .github/               # Workflows, issue templates, Dependabot configuration
├── cli/                   # Python admin CLI package
├── client/                # React and TypeScript frontend
├── deploy/                # Scenario-based deployment packages and services CLI
├── developer/             # Development container and Docker workflow
├── docs/                  # MkDocs source documentation
├── lib/                   # Shared libraries
│   ├── dt-automation/     # Digital twin automation npm package
│   └── gitlab_common/     # GitLab Python library for the CLIs
├── script/                # Automation for docs, environment setup, and releases
└── servers/               # Backend services
    ├── execution/runner/  # Runner microservice
    ├── lib/               # Library microservice
    └── logger/            # Logger microservice
```

Directory roles

Directory	Purpose
`.agents/`	Agent Skills; see Coding Agents.
`cli/`	Python package for administrative user and workspace operations.
`client/`	Main DTaaS web application with route, store, model, and test layers.
`deploy/`	Installable DTaaS, workspace, runner, and platform-services packages.
`developer/`	The `devenv` development container and the `check` Docker workflow.
`docs/`	Documentation source used by `mkdocs.yml` and `mkdocs-github.yml`.
`lib/`	The `dt-automation` npm package and the `gitlab_common` Python library.
`script/`	Helper scripts for docs publishing, environment setup, and release tasks.
`servers/`	NestJS services: the library, runner and logger microservices.
`.github/`	CI workflows, repository templates and Dependabot configuration.

Codebase entry points

The pages in this section follow the parts of the monorepo:

- Client for the React frontend structure and test commands.
- DevOps Framework and Client DevOps Integration for GitLab-backed digital twin lifecycle flows.
- DTaaS CLI for the Python admin CLI package.
- DTaaS Services for the platform-services provisioning CLI.
- Library Microservice and Runner Microservice for backend service implementation details.
- Publish Packages for package-release workflows.

Copyright © 2022 - 2026 The INTO-CPS Association

The logger microservice (`servers/logger/`) has no page in this section yet. Its design notes are in `servers/logger/ARCHITECTURE.md` and `servers/logger/DEVELOPER.md` .

Agent and automation files

Coding agents read `AGENTS.md` , which holds the general coding guidelines, and `PROJECT.md` , which holds the build and test commands, verification steps, shipping cycle and pull request checks specific to this repository. `CLAUDE.md` imports both for Claude Code.

Task-specific instructions are Agent Skills in `.agents/skills/` , with a stub for each in `.claude/skills/` for Claude Code. `.github/copilot-instructions.md` points GitHub Copilot to `AGENTS.md` and `PROJECT.md` . See Coding Agents.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.2 React Client

The DTaaS web client is a React and TypeScript single page application. Users compose digital twins, browse assets, edit configuration and run digital twins through GitLab pipelines from it.

Current Stack

- React 19
- React Router 7
- Redux Toolkit 2
- Material UI 9
- Vite 8 for the development server and production build
- Gitbeaker REST client for GitLab APIs
- Jest 30 and React Testing Library for unit and integration tests, Playwright for end-to-end tests

Current Source Layout

```
client/
├── src/
│   ├── index.tsx
│   ├── AppProvider.tsx
│   ├── routes.tsx
│   ├── components/
│   ├── database/
│   ├── page/
│   ├── route/
│   ├── store/
│   ├── theme/
│   ├── util/
│   └── utils/
├── test/
│   ├── __mocks__/
│   ├── unit/
│   ├── integration/
│   └── e2e/
├── config/
├── public/
├── index.html
├── vite.config.ts
└── package.json
```

The `route/` directory holds one folder per page, including `bim/` for the Building Models page. The Material UI theme and design tokens live in `theme/`.

The shared backend package lives under `lib/dt-automation/` and the client consumes it from npm as `@into-cps-association/dt-automation`. The Building Models page comes from `@into-cps-association/bim-kit`, which is maintained in the ifc-utils repository. Client Extensions describes both packages and the proposed extension architecture.

Backend Integration Layer

The `dt-automation` package wraps the GitLab-backed operations.

The main classes are:

- `GitlabAPI`: low-level GitLab REST wrapper.
- `GitlabInstance`: project and trigger-token context.
- `DigitalTwin`: DT-level operations and execution history integration.
- `LibraryAsset`: access to reusable library assets.

`commitMultipleActions` batches file updates, so one commit can carry several create, update and delete operations.

Copyright © 2022 - 2026 The INTO-CPS Association

Build and Test Commands

Run from `client/`:

```
yarn syntax
yarn build
yarn config:test
yarn test:unit
yarn test:int
yarn test:e2e
```

Use the `test:unit` and `test:int` scripts rather than bare `jest`. The scripts pass `--setupFilesAfterEnv`, which installs the `react-redux` mock that many tests need.

Notes for Contributors

- Keep GitLab-specific logic in backend implementation modules.
- Keep route and view components focused on presentation and user workflows.
- Prefer typed interfaces in `lib/dt-automation/src/interfaces/` when extending API contracts.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.3 DevOps Framework

The DevOps framework lets the DTaaS client manage the lifecycle of a Digital Twin through the GitLab API and GitLab CI/CD pipelines. With it the client:

- discovers DT definitions and reusable assets in repositories,
- creates and updates DT files in Git repositories,
- triggers pipelines that execute DTs,
- collects pipeline and job status and logs to show in the UI.

Where the Code Lives

The code is in `lib/dt-automation/src/`. The main modules are:

- `gitlab/backend.ts`: GitLab API wrapper.
- `gitlab/instance.ts`: backend session and project context.
- `digitalTwin.ts`: DT-level operations and execution tracking.
- `interfaces/backendInterfaces.ts`: backend contracts and API shapes.
- `util/`: execution history and file-management helpers.

Client DevOps Integration describes these modules in more detail.

GitLab API Support

The GitLab backend implementation supports:

- pipeline trigger and cancellation,
- repository file create, edit and delete operations,
- group and project discovery,
- pipeline job listing and log retrieval,
- pipeline status polling,
- batching several file changes into one commit with `commitMultipleActions`.

With the batched commit, the user sees DT creation or an update as one atomic change.

Execution Flow

1. `GitlabInstance` resolves the project IDs and the trigger token.
2. The UI operation calls DT logic in `DigitalTwin`.
3. `DigitalTwin` uses the backend APIs to prepare files and trigger the pipeline.
4. The execution history is updated and shown in the UI.
5. Job logs and statuses are fetched and rendered for diagnosis.

Pipeline Contract

The client triggers pipelines with two variables: `DTName`, the selected digital twin, and `RunnerTag`, the tag of the GitLab runner that should execute it. The GitLab CI pipelines must read these variables so that each DT runs on the intended runner.

Practical Guidance

- Prefer backend interface methods over ad hoc API calls in UI code.

Copyright © 2022 - 2026 The INTO-CPS Association

- Use the batched commit method when updating several files.
- Keep execution history updates in the utility modules so that status reporting stays consistent.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.4 Client DevOps Integration

This page describes how the React client manages digital twins through GitLab.

Scope

The DevOps integration in the client:

- discovers Digital Twin and library assets in GitLab repositories,
- creates, updates and deletes DT files,
- triggers and stops CI/CD pipeline executions,
- collects pipeline status and logs for the execution views.

Current Module Map

The implementation lives in `lib/dt-automation/src/`:

- `gitlab/backend.ts`: `GitlabAPI`, the concrete `BackendAPI` implementation.
- `gitlab/instance.ts`: `GitlabInstance`, which holds the project context and trigger token.
- `gitlab/execution/`: pipeline polling, status checks and log fetching.
- `digitalTwin.ts`: DT lifecycle operations and execution hooks.
- `DTAssets.ts`, `fileHandler.ts`, `libraryAsset.ts`, `libraryManager.ts`: file and asset management primitives.
- `util/digitalTwinPipelineExecution.ts`: start and stop execution paths.
- `util/digitalTwinExecutionHistory.ts`: execution history and log updates.
- `state/`: Redux state for execution history and status views.
- `interfaces/backendInterfaces.ts`: backend contracts and shared types.

Architectural Roles

Context, API access and DT domain behaviour sit in separate classes.

- `GitlabAPI` calls the GitLab REST API through `@gitbeaker/rest` (files, pipelines, groups and projects, jobs, logs, multi-action commits).
- `GitlabInstance` resolves `projectId` and `commonProjectId`, retrieves the trigger token and exposes accessors for execution data.
- `DigitalTwin` combines DT asset and file operations with pipeline execution and execution-history updates.
- `DTAssets`, `LibraryAsset`, `LibraryManager` and `FileHandler` handle low-level file and repository interactions.

UI code therefore does not depend on GitLab implementation details.

Execution Flow

1. A session creates and initializes `GitlabInstance`.
2. The UI performs DT actions through `DigitalTwin` and the backend utility modules.
3. Pipeline start and stop calls go through `GitlabInstance` and `GitlabAPI`.
4. Status polling and job log fetching update the execution history state.
5. UI components render current and past execution results.

Copyright © 2022 - 2026 The INTO-CPS Association

File Operations and Commit Strategy

The client supports both per-file operations and batched commits.

- Single-file operations use `createRepositoryFile`, `editRepositoryFile` and `removeRepositoryFile`.
- Multi-file updates should use `commitMultipleActions` to produce one commit when scaffolding or reconfiguring a DT.

A batched commit keeps a lifecycle update atomic and avoids a long trail of single-file commits.

Main Contracts

The `BackendAPI` contract includes:

- pipeline trigger, cancel and status,
- pipeline job listing and job log retrieval,
- repository file create, read, update and delete, and tree listing,
- group and project discovery,
- multi-action commits (`commitMultipleActions`).

When extending DevOps behaviour, update the interface contracts first, then the GitLab implementation and its call sites.

Contributor Guidance

- Keep GitLab-specific details inside `lib/dt-automation/src/gitlab/`.
- Keep route and page components focused on presentation and user interaction.
- Reuse the utility and state modules for execution history instead of adding ad hoc polling to UI components.
- Add or update unit and integration tests when changing backend contracts.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.5 Library Microservice

The Library Microservice gives access to workspace files in two ways: a GraphQL API, and an HTTP file server that users can open in a web browser. It reads files from the local file system or from GitLab, and its API follows the GitLab GraphQL API. Both users and digital twins use it to reach the reusable assets in the platform library.

> **Warning**
>
> This microservice is still under heavy development. It is not yet a satisfactory replacement for the file server currently in use.

Architecture and Design

The microservice is built with the NestJS framework and Apollo GraphQL. It uses the Strategy pattern: a factory picks one of several file storage backends (local file system or GitLab). The GraphQL API of the library microservice is meant to be compatible with the GitLab GraphQL service.

Package Structure

```
servers/lib/
├── src/                          # Application source code
│   ├── main.ts                   # Application entry point
│   ├── bootstrap.ts              # NestJS bootstrap configuration
│   ├── app.module.ts             # Root application module
│   ├── schema.gql                # Auto-generated GraphQL schema
│   ├── types.ts                  # GraphQL type definitions
│   ├── graphql-options.ts        # Apollo GraphQL module options
│   ├── config/                   # Configuration module
│   │   ├── config.module.ts      # Configuration DI module
│   │   ├── config.service.ts     # Configuration service
│   │   ├── config.interface.ts   # Configuration interface
│   │   ├── config.model.ts       # Configuration model
│   │   └── util.ts               # Configuration utilities
│   ├── files/                    # Files module (core functionality)
│   │   ├── files.module.ts       # Files DI module
│   │   ├── files.resolver.ts     # GraphQL resolver
│   │   ├── files-service.factory.ts  # Backend factory
│   │   ├── interfaces/           # Service interfaces
│   │   │   └── files.service.interface.ts
│   │   ├── local/                # Local filesystem backend
│   │   │   ├── local-files.module.ts
│   │   │   └── local-files.service.ts
│   │   └── git/                  # GitLab backend
│   │       ├── git-files.module.ts
│   │       └── git-files.service.ts
│   ├── enums/                    # Enumeration definitions
│   │   └── config-mode.enum.ts   # Backend mode enum
│   └── cloudcmd/                 # CloudCmd file browser integration
│       └── cloudcmd.ts
├── test/                         # Test suites
└── dist/                         # Compiled output
```

Main Components

GRAPHQL RESOLVER

The `FilesResolver` class exposes two GraphQL queries:

Query	Purpose
`listDirectory`	Returns directory contents (files and folders)
`readFile`	Returns file content as raw text

Copyright © 2022 - 2026 The INTO-CPS Association

FILE SERVICE INTERFACE

The `IFilesService` interface defines the contract for all file backends:

```
interface IFilesService {
  listDirectory(path: string): Promise<Project>;
  readFile(path: string): Promise<Project>;
  getMode(): CONFIG_MODE;
  init(): Promise<any>;
}
```

BACKEND IMPLEMENTATIONS

Backend	Purpose
`LocalFilesService`	Accesses files from local filesystem
`GitFilesService`	Accesses files from GitLab repository

FACTORY PATTERN

`FilesServiceFactory.create()` picks the backend whose `getMode()` matches the `mode` setting in the configuration file (`local` or `git`). `files.module.ts` calls it once, when NestJS builds the `FILE_SERVICE` provider, and the resolver receives the chosen service through dependency injection. An unknown mode stops the service with an `Invalid MODE` error.

UML Diagrams

CLASS DIAGRAM

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

    class FilesServiceFactory {
    +create(configService: IConfig, fileServices: IFilesService[]): IFilesService$
    }

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

    class Config {
    +getMode(): string
    +getLocalPath(): string
    }

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

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

SEQUENCE DIAGRAM

The factory has already chosen `LocalFilesService` at startup, so a request goes straight from the resolver to the backend.

```
sequenceDiagram
    actor Client
    actor Traefik

    box LightGreen Library Microservice
    participant FR as FilesResolver
    participant IFS as IFilesService
    participant LFS as LocalFilesService
    end

    participant FS as Local File System DB
```


Copyright © 2022 - 2026 The INTO-CPS Association

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

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

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

    LFS ->> FS : Access filesystem under the configured local path
    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

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

    deactivate FR
```

Build and Test

Run from `servers/lib/`:

```
yarn build && yarn test:all
```

`yarn test:nocov` also runs `test/cloudcmd`, which needs a configured `.env` and a running libms under pm2.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.6 Runner Microservice

The Runner microservice executes scripts for digital twins in the DTaaS platform. It accepts HTTP requests to run pre-configured commands and returns the execution status and logs to the caller.

Package Structure

```
servers/execution/runner/
├── src/                          # Application source code
│   ├── main.ts                   # Application entry point
│   ├── app.module.ts             # Root NestJS module
│   ├── app.controller.ts         # REST API controller
│   ├── execa-manager.service.ts  # Command execution manager
│   ├── execa-runner.ts           # Command runner implementation
│   ├── queue.service.ts          # Command queue service
│   ├── runner-factory.service.ts # Runner instance factory
│   ├── validation.pipe.ts        # Request validation pipe
│   ├── config/                   # Configuration module
│   │   ├── commander.ts          # CLI argument parsing
│   │   ├── config.interface.ts   # Configuration interface
│   │   ├── configuration.service.ts  # Configuration service
│   │   └── util.ts               # Configuration utilities
│   ├── dto/                      # Data transfer objects
│   │   └── command.dto.ts        # Command request DTO
│   └── interfaces/               # Interface definitions
│       ├── command.interface.ts  # Command types
│       └── runner.interface.ts   # Runner interface
├── api/                          # API documentation
├── config/                       # Sample configuration files
├── test/                         # Test suites
└── dist/                         # Compiled output
```

Architecture and Design

The microservice is built with the NestJS framework and runs commands with the `execa` library. A queue holds the execution requests, and a factory creates the runner instances.

CORE COMPONENTS

```
classDiagram
    class AppController {
        -manager: ExecaManager
        +getHistory(): ExecuteCommandDto[]
        +newCommand(dto): void
        +cmdStatus(): CommandStatus
    }

    class ExecaManager {
        -commandQueue: Queue
        -config: Config
        +newCommand(name): [boolean, Map]
        +checkStatus(): CommandStatus
        +checkHistory(): ExecuteCommandDto[]
    }

    class Queue {
        -queue: Command[]
        +enqueue(command): void
        +checkHistory(): ExecuteCommandDto[]
        +activeCommand(): Command
    }

    class RunnerFactory {
        +create(command): Runner
    }

    class ExecaRunner {
        -command: string
        -stdout: string
        -stderr: string
        +run(): Promise~boolean~
        +checkLogs(): Map
    }

    class Config {
        -configValues: ConfigValues
        +loadConfig(options): void
        +permitCommands(): string[]
        +getPort(): number
        +getLocation(): string
    }
```


Copyright © 2022 - 2026 The INTO-CPS Association

```
    AppController --> ExecaManager : uses
    ExecaManager --> Queue : uses
    ExecaManager --> Config : uses
    ExecaManager --> RunnerFactory : uses
    RunnerFactory --> ExecaRunner : creates
```

Main Components

REST API CONTROLLER

The `AppController` exposes three endpoints:

Endpoint	Method	Purpose
`/`	POST	Submit a new command for execution
`/`	GET	Get status of the current command
`/history`	GET	Retrieve execution history

EXECUTION MANAGER

The `ExecaManager` service runs each command in these steps:

1. Adds the command to the queue, marked `invalid`.
2. Checks the command against the list of permitted commands.
3. For a permitted command, creates a runner through the factory and runs the script from the configured location.
4. Marks the command `valid` if the script succeeds, and returns the status and logs.

COMMAND QUEUE

The `Queue` service keeps the execution history and tracks the active command. Each command is stored with its execution status (`valid` or `invalid`).

RUNNER INTERFACE

The `Runner` interface defines the contract for command executors:

```
interface Runner {
  run(): Promise<boolean>;
  checkLogs(): Map<string, string>;
}
```

CONFIGURATION

The service reads its configuration from a YAML file (see `config/runner.yaml.sample`) with these settings:

Setting	Purpose
`port`	HTTP server port
`location`	Base directory for executable scripts
`commands`	Allowlist of permitted command names

Security Considerations

The runner only executes commands that are listed in the configuration file. A request for any other command is recorded as `invalid` and not run, which prevents arbitrary command execution.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.7 Command Line Interface

The Command Line Interface (CLI) lets administrators manage a DTaaS installation from the shell. It covers the whole deployment lifecycle: generating and validating the configuration, generating deployment files for each supported scenario, installing and uninstalling a deployment, updating certificates and configuration in place, and adding, removing, pausing and resuming users.

The user guide is the admin CLI page, and the full command reference is in `cli/README.md`. This page describes the internal structure for contributors.

Command Surface

Every command follows the grammar `dtaas <noun> <verb>`. The four nouns are listed in workflow order.

```
dtaas
├── config
│   ├── generate                  # write dtaas.toml + users.csv
│   ├── validate                  # check dtaas.toml, report all errors
│   └── reconcile                 # report or fix registry drift
├── deployment
│   └── generate --type X         # generate a deployment package
├── platform
│   ├── install                   # docker compose up of a deployment
│   ├── uninstall                 # tear down (optionally user files)
│   ├── update                    # --certs and/or --config, in place
│   ├── status                    # state of the installation
│   ├── stop / start              # core services only
│   └── pause / resume            # core services only
└── user
    ├── add                       # provision user workspaces
    ├── delete                    # deprovision user workspaces
    ├── status                    # state of additional users
    └── pause / stop / resume     # suspend or resume single users
```


Copyright © 2022 - 2026 The INTO-CPS Association

Package Structure

The CLI package is organised as follows:

```
cli/
├── src/                          # Application source code
│   ├── __init__.py               # Package initialization
│   ├── cmd.py                    # Root 'dtaas' group, wires the nouns
│   ├── cmd_config.py             # 'config' commands
│   ├── cmd_deployment.py         # 'deployment' commands
│   ├── cmd_platform.py           # 'platform install/uninstall/update'
│   ├── cmd_lifecycle.py          # 'platform status/stop/start/...'
│   ├── cmd_user.py               # 'user add/delete/status'
│   ├── cmd_user_lifecycle.py     # 'user pause/stop/resume'
│   ├── cmd_options.py            # Shared Click options
│   ├── cmd_utils.py              # Shared command helpers
│   ├── cmd_user_utils.py         # User input resolution
│   ├── cmd_user_emails.py        # Emails given but not applied
│   ├── cmd_user_passwords.py     # Initial GitLab passwords
│   ├── cmd_deploy_utils.py       # Deployment command helpers
│   ├── pkg/                      # Implementation modules
│   │   ├── build.py              # Vendors shared code into src/
│   │   ├── certs.py              # Certificate placement
│   │   ├── cert_update.py        # In-place certificate rotation
│   │   ├── cert_validate.py      # Certificate pair validation
│   │   ├── config.py             # dtaas.toml parsing and access
│   │   ├── config_update.py      # In-place configuration re-apply
│   │   ├── config_validate.py    # Configuration validation rules
│   │   ├── config_validate_gitlab.py # [gitlab] validation rules
│   │   ├── constants.py          # Constant definitions
│   │   ├── deploy.py             # Install/uninstall orchestration
│   │   ├── deploy_config.py      # Scenario configuration mapping
│   │   ├── _deploy_data.py       # Per-scenario substitution specs
│   │   ├── gitlab/               # Optional GitLab user provisioning
│   │   ├── lifecycle.py          # Platform status/stop/start/pause
│   │   ├── messages.py           # Shape of user command output
│   │   ├── project.py            # Template copying for 'generate'
│   │   ├── registry.py           # User registry (JSON)
│   │   ├── registry_csv.py       # users.csv parsing
│   │   ├── state.py              # Runtime state cache
│   │   ├── user_files.py         # User file cleanup on uninstall
│   │   ├── users.py              # User add handler
│   │   ├── users_compose.py      # Per-user compose generation
│   │   ├── users_delete.py       # User delete handler
│   │   ├── users_gitlab.py       # GitLab account and token setup
│   │   ├── users_gitlab_records.py # Saved tokens and markers
│   │   ├── users_gitlab_targets.py # Users to provision in GitLab
│   │   ├── users_lifecycle.py    # Per-user pause/stop/resume
│   │   ├── users_utils.py        # User management helpers
│   │   ├── utils.py              # File I/O and utilities
│   │   └── validators.py         # Shared validation helpers
│   └── templates/                # dtaas.toml, users.csv and compose
│       └── ...                   # templates for user workspaces
├── tests/                        # Test suites and fixtures
├── pyproject.toml                # Poetry project configuration
└── README.md                     # Command reference
```

Architecture and Design

The CLI is a Python package that uses the Click framework for command parsing. Each noun is a Click group in its own `cmd_<noun>.py` module, and `cmd.py` attaches the groups to the root `dtaas` group. Command modules parse options, load and validate the configuration, and call functions in `pkg/`. Docker operations go through python-on-whales.

`dtaas.toml` is the single source of truth. Every command reads it through the `Config` class in `pkg/config.py`, and `platform update --config` re-applies it to an installed deployment so that the deployment does not drift from it.

The CLI also keeps two files of its own. `dtaas.users.registry.json` (`pkg/registry.py`) records the additional users added with `user add`, and `.dtaas.state.json` (`pkg/state.py`) caches what is known about the provisioned user containers.

GitLab account provisioning in `pkg/gitlab/` is built on the `gitlab_common` library. `src/pkg/build.py` vendors it from `lib/gitlab_common/` into the source tree, so run it before running the tests:

```
poetry run python src/pkg/build.py && poetry run pytest
```


Copyright © 2022 - 2026 The INTO-CPS Association

MAIN MODULES

Module	Purpose
`cmd.py`	Root group; attaches the four noun groups
`cmd_user.py`	User add/delete/status command definitions
`pkg/config*.py`	TOML parsing, validation, in-place updates
`pkg/project.py`	Template copying (`config generate` , `deployment generate`)
`pkg/deploy.py`	Install/uninstall orchestration
`pkg/lifecycle.py`	Platform status, stop, start, pause, resume
`pkg/cert_*.py`	Certificate validation and rotation
`pkg/users*.py`	User workspace lifecycle
`pkg/registry.py`	Registry of additional users
`pkg/state.py`	Runtime state of user containers
`pkg/utils.py`	YAML/TOML I/O, template substitution

Sequence Diagram

The following diagram shows how `dtaas user add` provisions a user:

```
sequenceDiagram
    actor Admin
    participant CLI as Click CLI
    participant Config as pkg/config.py
    participant Registry as pkg/registry.py
    participant Users as pkg/users.py
    participant Utils as pkg/utils.py
    participant Docker as Docker Compose
    participant FS as File System

    Admin ->> CLI: dtaas user add alice --email ...
    activate CLI

    CLI ->> Config: Config()
    activate Config
    Config ->> Utils: import_toml("dtaas.toml")
    Utils -->> Config: config data
    deactivate Config

    CLI ->> Registry: stage the new user
    CLI ->> Users: add_users(config_obj, start_only, passwords)
    activate Users

    Users ->> Registry: load_registry()
    Registry -->> Users: user list

    Users ->> Config: get_server_dns() / get_path()
    Config -->> Users: server DNS, installation path

    Users ->> Config: get_resource_limits() / get_tls()
    Config -->> Users: resource limits, TLS setting

    Users ->> FS: Copy workspace files for each user
    Users ->> Users: generate compose service entries

    Users ->> Utils: export_yaml(compose)
    Utils ->> FS: Write user compose file

    Users ->> Docker: docker compose up -d (new users only)
    Docker -->> Users: Containers started

    Users -->> CLI: None (success)
    deactivate Users

    CLI -->> Admin: "Users added successfully"
    deactivate CLI
```


Copyright © 2022 - 2026 The INTO-CPS Association

Error Handling Pattern

The CLI handles errors in two ways:

1. Most modules under `src/pkg/` raise exceptions.
2. `config.py` and `utils.py` return `(result, Exception | None)` pairs, and callers check the error before using the result.

The command modules turn both kinds of error into `click.ClickException`, so the user sees a short message instead of a traceback. Test commands with Click's `CliRunner` rather than `subprocess`.

Resource Limits Configuration

Administrators can limit the resources each user container may use, so that one user cannot take over a shared server. The limits are set in the `[common.resources]` section of `dtaas.toml`. The `set_limits` flag turns enforcement on and off. When it is `false`, user containers are created without limits and the other fields are ignored. The defaults come from the `dtaas.toml` template:

Parameter	Description	Default
`cpus`	CPU core allocation	`4`
`mem_limit`	Memory limit	`4G`
`pids_limit`	Maximum process count	`4960`
`shm_size`	Shared memory size	`512m`

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.8 DTaaS Services Codebase

This page describes the code of the `dtaas-services` CLI, which sets up and runs the DTaaS platform services.

Location and Purpose

Source package:

- `deploy/services/cli`

The CLI:

- generates a services project structure that is ready to run,
- sets up certificates and service permissions,
- installs, starts, stops, restarts, removes and cleans service containers,
- runs the post-start setup for ThingsBoard and GitLab,
- creates users and resets service passwords from the CSV and environment configuration.

Runtime Command Surface

Every command follows the grammar `dtaas-services <noun> <verb>`. The entry point is `dtaas_services/cmd.py`, which registers the nouns in install order:

- `project generate`
- `host setup`
- `service install`, `service start`, `service stop`, `service restart`, `service status`, `service remove`, `service clean`
- `user add`
- `user reset-password`

The old top-level spellings (`generate-project`, `setup`, `start` and so on) are registered by `commands/aliases.py`. They print a deprecation warning and forward to the `<noun> <verb>` command.

Poetry script mapping (`pyproject.toml`):

```
[tool.poetry.scripts]
dtaas-services = "dtaas_services.cmd:services"
```


Copyright © 2022 - 2026 The INTO-CPS Association

Package Layout

```
deploy/services/cli/
├── dtaas_services/
│   ├── cmd.py
│   ├── commands/
│   │   ├── project_ops.py
│   │   ├── host_ops.py
│   │   ├── service_ops.py
│   │   ├── install_helpers.py
│   │   ├── user_ops.py
│   │   ├── aliases.py
│   │   └── utility.py
│   ├── pkg/
│   │   ├── build.py
│   │   ├── config.py
│   │   ├── cert.py
│   │   ├── template.py
│   │   ├── force_overwrite.py
│   │   ├── docker_utils.py
│   │   ├── formatter.py
│   │   ├── sanitize.py
│   │   ├── utils.py
│   │   ├── password_store.py
│   │   ├── lib/
│   │   │   ├── manager.py
│   │   │   ├── docker_executor.py
│   │   │   ├── status.py
│   │   │   ├── cleanup.py
│   │   │   ├── initialization.py
│   │   │   └── utils.py
│   │   └── services/
│   │       ├── mongodb.py
│   │       ├── rabbitmq.py
│   │       ├── influxdb/
│   │       ├── postgres/
│   │       ├── thingsboard/
│   │       └── gitlab/
│   ├── gitlab_common/      # vendored from lib/gitlab_common (gitignored)
│   │   ├── client.py
│   │   ├── errors.py
│   │   ├── messages.py
│   │   ├── users.py
│   │   └── validators.py
│   └── templates/
├── tests/
├── README.md
└── pyproject.toml
```

Layered Design

COMMAND LAYER

The files in `dtaas_services/commands/` define the Click commands and the high-level flows:

- `project_ops.py`: project scaffold generation (`project generate`).
- `host_ops.py`: TLS certificate and permission setup (`host setup`).
- `service_ops.py`: the service verbs (`install`, `start`, `stop`, `restart`, `status`, `remove`, `clean`).
- `install_helpers.py`: selector validation and the ThingsBoard and GitLab install flows used by `service install`.
- `user_ops.py`: user provisioning and password reset.
- `aliases.py`: deprecated top-level spellings that warn and forward to their `<noun> <verb>` replacement.

CORE LIBRARY LAYER

The files in `dtaas_services/pkg/` hold reusable logic:

- `config.py`: environment and configuration loading.
- `template.py`: project scaffolding and template copying.
- `force_overwrite.py`: what `project generate --force` may replace, what it must never replace, and copies that preserve permissions.
- `cert.py`: certificate placement and normalization.
- `password_store.py`: tracks the current admin password of each supported service.
- `lib/*`: compose command execution and service state management.

Copyright © 2022 - 2026 The INTO-CPS Association

SERVICE MODULES LAYER

`dtaas_services/pkg/services/` implements service-specific behaviour:

- `gitlab/` : health checks, root password setup, PAT creation, OAuth application setup and GitLab user provisioning.
- `thingsboard/` : setup, sysadmin and tenant workflows, and user setup from the credentials file.
- `influxdb/` , `rabbitmq.py` , `mongodb.py` , `postgres/` : permissions, readiness checks and account bootstrap routines.

SHARED GITLAB LAYER

`gitlab_common` holds GitLab operations that do not depend on the deployment (`get_gitlab_client` , `validate_user_row` , `create_user` , `create_user_pat` , `revoke_user_pats` , `with_hint`). Every function takes explicit arguments and changes no environment, filesystem, console or process-global state.

Its single source of truth is `lib/gitlab_common/` , a standalone Poetry package with its own tests. Each consumer **copies** it into its own tree instead of depending on it:

```
lib/gitlab_common/gitlab_common/     <-- edit and test here
| copied by dtaas_services/pkg/build.py |
        v
dtaas_services/gitlab_common/        <-- gitignored, never committed
```

Run `poetry run python -m dtaas_services.pkg.build` before testing or packaging. CI does this through `run-build-script: true` . It is a plain script, not a Poetry build hook, because a hook makes poetry-core emit a platform-specific wheel instead of `py3-none-any` . Copying the code instead of using a path dependency keeps `dtaas-services` and the DTaaS CLI independent of each other. It also keeps local paths out of the published wheel metadata, which PyPI rejects.

Deployment-specific glue stays in `pkg/services/gitlab/` : URL derivation from `GITLAB_PORT` and `HOSTNAME` , token-file persistence, docker health checks and Rich console output. Two consequences are deliberate:

- `pkg/services/gitlab/_api.py` suppresses urllib3's `InsecureRequestWarning` when `SSL_VERIFY=false` . `gitlab_common` does not, so importing the shared module never silences warnings for unrelated consumers.
- `gitlab_common` defaults Personal Access Tokens to least-privilege repository-only scopes. `pkg/services/gitlab/users.py` asks for the broader `api` scope explicitly through `PatOptions` .

Configuration Inputs

A generated project is driven by:

- `config/services.env` : ports, hostnames, SSL behaviour and service credentials.
- `config/credentials.csv` : user list (`username,password,email`).
- TLS certificates copied into the generated `certs/` layout.

The main GitLab-related environment variables are `HOSTNAME` , `GITLAB_PORT` , `SSL_VERIFY` and `GITLAB_ROOT_NEW_PASSWORD` .

Typical Workflow

1. Generate project files:
2. `dtaas-services project generate`
3. Edit configuration:
4. `config/services.env`
5. `config/credentials.csv`
6. Set up certs and permissions:
7. `dtaas-services host setup`
8. Start selected services:
9. `dtaas-services service start [-s ...]`

Copyright © 2022 - 2026 The INTO-CPS Association

10. Run post-install setup where needed:
11. `dtaas-services service install -s thingsboard`
12. `dtaas-services service install -s gitlab`
13. Provision users:
14. `dtaas-services user add`

Testing and Quality

The package uses pytest with strict markers, and test discovery starts in `deploy/services/cli/tests`.

Run the checks from `deploy/services/cli`:

```
poetry install
poetry run python -m dtaas_services.pkg.build
CI=true poetry run pytest --ignore=tests/system_tests
```

The build step vendors `gitlab_common`; without it the tests cannot be collected. The CLI checks for root before it validates arguments, so a non-root run needs `CI=true`. The tests in `tests/system_tests` call `docker` and are excluded here.

Contributor Notes

- Keep command modules thin and put business logic in `pkg/`.
- Add service-specific behaviour in `pkg/services/<service>/` or peer modules, not in generic command handlers.
- Commands should print messages the user can act on, and should not leave partial side effects without reporting them.
- When adding service operations, include tests under `tests/test_services` and command-level tests under `tests/test_commands`.

Copyright © 2022 - 2026 The INTO-CPS Association

6.4.9 Publish Packages

DTaaS is a monorepo that publishes npm packages, Docker images, Python packages and installation packages.

Package Channels

- npm packages (public): `@into-cps-association/*`, published to both GitHub Packages and the npm registry
- Docker images: GitHub Container Registry and Docker Hub
- Python packages: the DTaaS CLI and the platform-services CLI on PyPI
- Installation packages: zip files attached to each GitHub release by the `Release Packages` workflow

JavaScript/TypeScript Packages

The package roots are:

- `servers/lib` -> `@into-cps-association/libms`
- `servers/execution/runner` -> `@into-cps-association/runner`
- `servers/logger` -> `@into-cps-association/logger-ms`
- `lib/dt-automation` -> `@into-cps-association/dt-automation`
- `client` -> `@into-cps-association/dtaas-web`, which is published as a Docker image rather than an npm package

The workflow of each package (`lib-ms.yml`, `runner.yml`, `logger-ms.yml` and `dt-automation.yml`) first runs the lint, build and test jobs. It then calls the reusable `publish-npm.yml` workflow once for GitHub Packages and once for the npm registry. `publish-npm.yml` runs `yarn install`, `yarn build` and `yarn publish --access public` with the registry token passed in as the `NPM_TOKEN` secret.

Publishing happens on a push or manual run on the `feature/distributed-demo` branch. The logger workflow also publishes from any `feature/*` or `release-v*` branch.

The `dt-automation` workflow also checks the tarball contents and installs the packed tarball into an empty project before publishing.

Private Registry Workflow (Development)

Use a private registry such as Verdaccio to test publish and unpublish behaviour before a public release.

A typical local flow:

```
docker run -d --name verdaccio -p 4873:4873 verdaccio/verdaccio
npm adduser --registry http://localhost:4873
npm set registry http://localhost:4873/
```

Docker Artifacts

All Dockerfiles for the published images live in `developer/check/`. The reusable workflows `docker-build.yml`, `docker-ghcr.yml` and `docker-dockerhub.yml` build them. The `client.yml`, `lib-ms.yml` and `logger-ms.yml` workflows publish the `dtaas-web`, `libms` and `logger-ms` images for `linux/amd64` and `linux/arm64`. Each platform is built on a native runner and pushed by digest; a final job tags one multi-platform manifest that points at both. Deployment-level compose definitions are under `deploy/`.

When changing runtime dependencies, check that the images build and the deployment scenarios start before tagging a release.

Copyright © 2022 - 2026 The INTO-CPS Association

Python Packages

Two Poetry-based CLIs are published to PyPI:

- `cli` (`dtaas`), by the `CLI for DTaaS` workflow (`python-cli.yml`)
- `deploy/services/cli` (`dtaas-services`), by the `Platform Services CLI` workflow (`platform-services-cli.yml`)

Both call the reusable `python-reusable.yml` workflow for tests and builds, and publish on a push or manual run on the `feature/distributed-demo` branch. The `lib/gitlab_common` library is tested by the `gitlab-common.yml` workflow and is not published on its own.

Use Poetry for versioning and keep the lock files up to date so that builds are reproducible.

Installation Packages

The installation packages attached to each GitHub release are built by the `Release Packages` workflow (`.github/workflows/release-packages.yml`) with `script/release-packages.sh`. Each package is `<package>-<version>.zip`, holding every tracked file of its source directory under `deploy/dtaas/docker/` or `deploy/workspace/`.

The workflow builds and verifies the packages whenever `deploy/dtaas/`, `deploy/workspace/`, the script or the workflow changes, and keeps them for two days as one `release-packages` artifact of the workflow run. Publishing a GitHub release also attaches them to the release. Do not edit or re-zip them by hand; change the source directory instead.

To build and check the packages locally:

```
git lfs install --local
git lfs pull --include="deploy/dtaas/**,deploy/workspace/**"
script/release-packages.sh build dev dist
script/release-packages.sh verify dev dist
```

`git lfs install --local` sets up the Git LFS filters. Without them, the fetched images look like uncommitted changes and the script refuses to build.

To replace the packages of an existing release, check out its tag, build the packages with the script from the default branch into a fresh directory and upload them:

```
git switch --detach v1.0
git lfs install --local
git lfs pull --include="deploy/dtaas/**,deploy/workspace/**"
git show origin/feature/distributed-demo:script/release-packages.sh > /tmp/release-packages.sh
rm -rf dist-1.0
bash /tmp/release-packages.sh build 1.0 dist-1.0
bash /tmp/release-packages.sh verify 1.0 dist-1.0
gh release upload v1.0 dist-1.0/*-1.0.zip --clobber --repo INTO-CPS-Association/DTaaS
```

`--clobber` replaces only assets with the same name. Releases up to v1.0 carry packages without a version in their name, such as `dtaas-localhost.zip`, so delete those after the upload:

```
for name in dtaas-localhost dtaas-server dtaas-secure-server \
  dtaas-secure-server-integrated-gitlab workspace-dex-localhost \
  workspace-keycloak-production; do
  gh release delete-asset v1.0 "$name.zip" --yes --repo INTO-CPS-Association/DTaaS
done
```

Release Guidance

- Bump versions deliberately, and only for the packages that changed.
- Prefer the automated CI publishing where it exists.
- After publishing, install the package and run it once to check that it works.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5 Contributions

6.5.1 Contributor Covenant Code of Conduct

Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

Our Standards

Examples of behaviour that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behaviour include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behaviour and will take appropriate and fair corrective action in response to any behaviour that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behaviour may be reported to the community leaders responsible for enforcement by opening a new issue in this repository. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Copyright © 2022 - 2026 The INTO-CPS Association

Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. CORRECTION

Community Impact: Use of inappropriate language or other behaviour deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behaviour was inappropriate. A public apology may be requested.

2. WARNING

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behaviour. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. TEMPORARY BAN

Community Impact: A serious violation of community standards, including sustained inappropriate behaviour.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. PERMANENT BAN

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behaviour, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5.2 Coding Agents

Coding agents read their instructions from files in the repository. Every agent reads the same files, so the rules are written once.

Instruction files

File	Holds	Read by
`AGENTS.md`	General coding guidelines	Codex, OpenCode, GitHub Copilot and most other agents
`PROJECT.md`	Build and test commands, verification, shipping, pull request checks, code standards and per-project conventions	Every agent, through `AGENTS.md`
`CLAUDE.md`	Imports `AGENTS.md` and `PROJECT.md`	Claude Code
`.github/copilot-instructions.md`	A pointer to `AGENTS.md` and `PROJECT.md`	GitHub Copilot

Change the rules in `AGENTS.md` or `PROJECT.md`; the other files only point to them.

Skills

Task-specific instructions are Agent Skills. Each skill is a directory `.agents/skills/<name>/` holding a `SKILL.md` file and, where needed, scripts. Codex and OpenCode find skills in `.agents/skills/`.

Claude Code reads skills from `.claude/skills/` only. Each skill therefore has a stub `.claude/skills/<name>/SKILL.md` with the same frontmatter, whose body sends the agent to the canonical file. A symbolic link would check out as a plain file on Windows clones without symlink support. The `Lint Miscellaneous Scripts` workflow fails when a stub's frontmatter differs from its canonical skill.

The repository provides these skills:

Skill	Use it to
`upgrade-npm-packages`	Upgrade npm dependencies behind a ten-day release cooldown; see Upgrade NPM Packages
`capture-docs-screenshots`	Retake the user documentation screenshots from a running installation
`update-docs-diagrams`	Edit, restyle and export the documentation diagrams
`review-pull-request`	Check that a pull request passes every quality gate before merge
`check-code-quality`	Review the code quality of one component
`plan-before-coding`	Analyse the codebase and requirements before implementing
`architecture-design`	Design an architecture and its diagrams and documentation
`react-frontend-engineer`	Work on the React client
`playwright-tester`	Explore the website and write Playwright tests
`technical-debt-remediation-plan`	Plan the remediation of technical debt
`critical-thinking`	Challenge assumptions behind a proposed solution
`mentor`	Guide an engineer rather than writing the code for them

Seven of these skills are adapted from awesome-copilot under the MIT licence; see `.github/REFERENCES.md`.

Copyright © 2022 - 2026 The INTO-CPS Association

ADDING A SKILL

1. Create `.agents/skills/<name>/SKILL.md`. Its frontmatter needs `name`, equal to the directory name, and a `description` that says what the skill does and when to use it. The agent decides whether to load the skill from the description alone.
2. Create `.claude/skills/<name>/SKILL.md` with the same frontmatter and a body that points to the canonical file.
3. Add the skill to the table above.
4. Validate both directories with skills-ref: `skills-ref validate .agents/skills/<name>`.

Using coding agents

Contributors use coding agents for co-development, code review and draft pull requests. Disclose code that a coding agent generated, particularly when it carries significant program logic. Understand every line you contribute: review generated code before submitting it, and keep the critical thinking for yourself.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5.3 Local Testing

This guide gives a local test loop for contributors working in the DTaaS monorepo. The authoritative build and test command for each project is in the "Build and test commands" section of `PROJECT.md`; the commands below follow it. The development container in `developer/devenv/` has every toolchain they need.

Recommended Order

1. Run syntax and lint checks for the changed component.
2. Run fast local tests (unit or focused integration).
3. Run broader test suites before opening a pull request.
4. Build the docs if markdown or navigation changed.

Client (React + TypeScript)

Run from `client/`:

```
yarn syntax
yarn build
yarn config:test
yarn test:unit
yarn test:int
```

`yarn config:test` copies the test configuration into `public/env.js` and `build/env.js`. Use the `yarn test:*` scripts rather than bare `jest`: they load the setup files that the tests depend on.

For end-to-end checks:

```
yarn test:e2e
```

The end-to-end suite logs in through GitLab, so it needs a reachable GitLab instance configured in `client/test/.env`. GitHub Actions does not run it; run it locally before opening a pull request that changes client behaviour. See `client/test/README.md`.

Library Microservice (NestJS)

Run from `servers/lib/`:

```
yarn syntax
yarn build
yarn test:all
```

For API-level checks of the cloudcmd integration, use `yarn test:http`. It needs a configured `.env` file and starts the library microservice under pm2.

Runner Microservice (NestJS)

Run from `servers/execution/runner/`:

```
yarn syntax
yarn build
yarn test
```

`yarn test:unit` and `yarn test:int` run the unit and integration tests separately.

Copyright © 2022 - 2026 The INTO-CPS Association

Logger Microservice (NestJS)

Run from `servers/logger/`:

```
yarn syntax
yarn test
```

DT Automation Library (TypeScript)

Run from `lib/dt-automation/`:

```
yarn syntax
yarn test:unit
```

The package has no `test` script.

DTaaS CLI (Python)

Run from `cli/`:

```
poetry install
poetry run python src/pkg/build.py
poetry run pytest
```

The build step copies the shared code from `lib/` into the package; the tests cannot be collected without it.

GitLab Common Library (Python)

Run from `lib/gitlab_common/`:

```
poetry install
poetry run pytest
```

Platform Services CLI (Python)

Run from `deploy/services/cli/`:

```
poetry install
poetry run python -m dtaas_services.pkg.build
CI=true poetry run pytest --ignore=tests/system_tests
```

The CLI checks for root before it validates arguments, so non-root runs need `CI=true`. The tests in `tests/system_tests` call `docker`.

Documentation Validation

Run from the repository root:

```
python -m pytest script/docs
python script/docs/docs.py build
```

On Linux, to validate the full docs build including the PDF export:

```
python script/docs/docs.py build --pdf
```

Practical Tip

When changing several components, validate each package on its own first. Failures then stay local and are quicker to debug, and unrelated test errors do not get in the way.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5.4 Docker Workflow for DTaaS

This document describes how the Docker files are built and used for developing and installing the DTaaS platform.

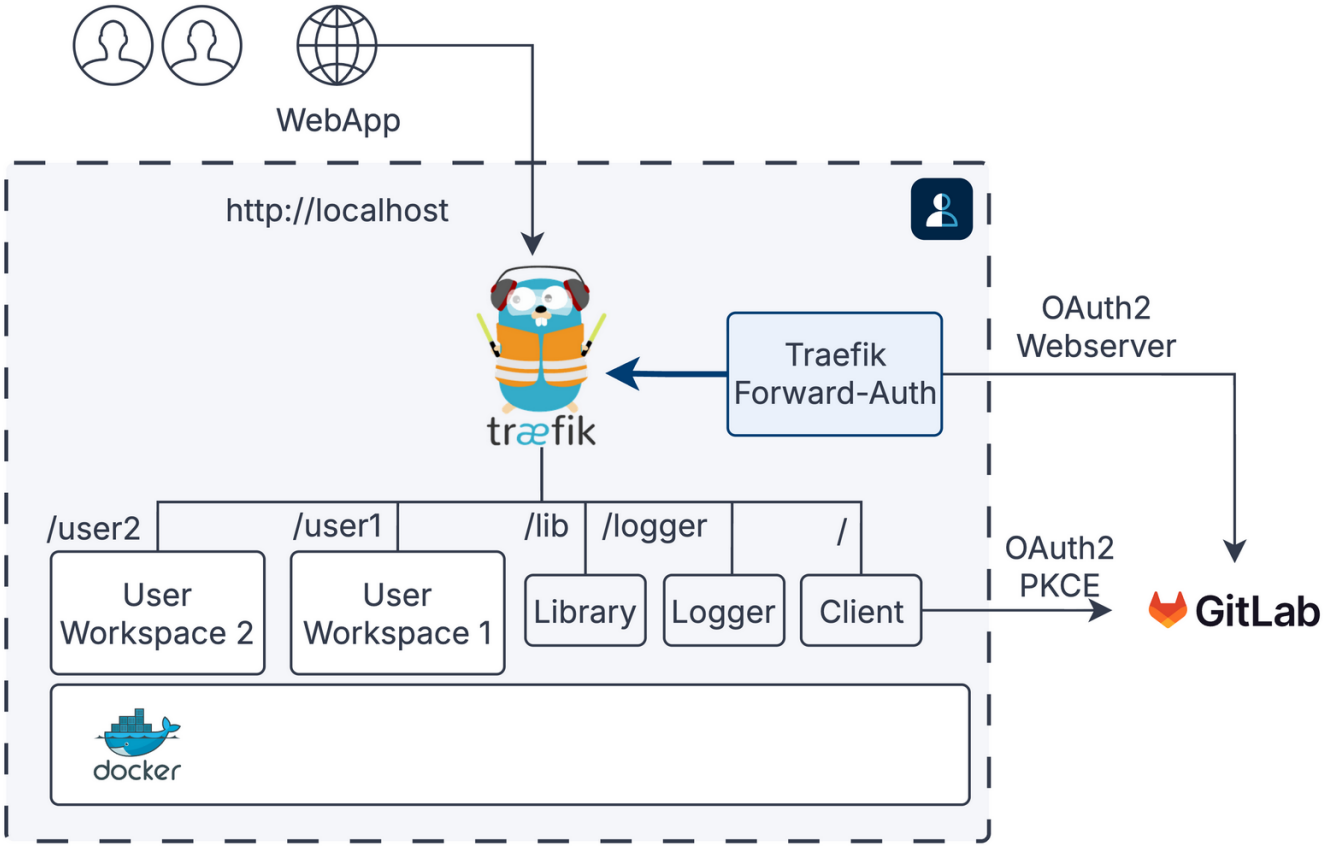
NOTE: The Docker workflows need a local Docker CE installation.

Run

The `developer/` directory holds two separate things:

- `developer/check/` checks that DTaaS *functions* on a developer machine. Follow the instructions in `developer/check/README.md` to start a localhost instance. It is an end-to-end test of the codebase as it exists in the local git directory.
- `developer/devenv/` is the containerised *developer environment*: the Node, Python and documentation toolchains needed to work on the code. See `developer/devenv/README.md` .

The diagram shows the localhost instance that `developer/check/` starts: two user workspaces, the library and logger microservices and the client behind Traefik, with forward-auth signing users in through GitLab.



CI Dockerfile Validation

A pull request that modifies `client/**`, `servers/lib/**`, `servers/logger/**` or the relevant Dockerfiles triggers a Docker build check. The check uses the reusable workflow `.github/workflows/docker-build.yml` and runs in parallel with the test jobs. It builds

Copyright © 2022 - 2026 The INTO-CPS Association

each image for both `linux/amd64` and `linux/arm64`, each on a native runner (`ubuntu-latest` and `ubuntu-24.04-arm`), so an image that fails to build on either platform fails the check.

Workflow	Dockerfile validated	Notes
`.github/workflows/client.yml`	`developer/check/client.dockerfile`	Full multi-stage build inside Docker
`.github/workflows/client.yml`	`developer/check/client.built.dockerfile`	Validates the publish image Dockerfile
`.github/workflows/lib-ms.yml`	`developer/check/libms.dockerfile`	Full multi-stage build inside Docker
`.github/workflows/logger-ms.yml`	`developer/check/logger.dockerfile`	Full multi-stage build inside Docker

The Docker build checks also run on pushes. The push triggers of `.github/workflows/client.yml` and `.github/workflows/lib-ms.yml` cover only the source code directories (`client/**` and `servers/lib/**`).

The publish jobs of these two workflows run on pushes to, and manual runs on, the `feature/distributed-demo` branch. Because the push trigger covers only the source code directories, a change to a Dockerfile alone does not re-publish an already published package version.

In all three workflows, the Docker publish jobs wait for the Docker build check to pass on both platforms. A publish that has started is never cancelled by a newer push; only superseded pull request runs are.

The logger microservice workflow (`.github/workflows/logger-ms.yml`) also runs the image from `developer/check/logger.dockerfile` on amd64 and checks its health endpoint. It publishes that image on pushes to `feature/*` and `release-v*` branches, and its push trigger includes that Dockerfile.

Publish Docker Images

Build and publish the Docker images. This step is needed only to publish images to Docker Hub.

The CI workflows publish every image for both `linux/amd64` and `linux/arm64`. The `docker build` commands below build only for the platform of the machine they run on. To publish both platforms by hand, create a builder that supports multi-platform output once with `docker buildx create --use`, then run `docker buildx build --platform linux/amd64,linux/arm64 --push` with the same `-t` and `-f` arguments instead of `docker build` and `docker push`. The platform that differs from the host is built under QEMU emulation, which is several times slower. Docker Desktop and Rancher Desktop provide the emulation; on a plain Linux Docker Engine, register it first with `docker run --privileged --rm tonistiigi/binfmt --install arm64`.

> **Warning**
>
> Only project maintainers publish images. Other developers can skip this step.

The DTaaS development team publishes reusable packages, which together form the complete DTaaS application.

The packages are published on github, npmjs and docker hub.

The packages on github are published more often but are not user tested. The packages on npmjs and docker hub are published at least once per release. Regular users should use the packages from npm and Docker Hub.

Copyright © 2022 - 2026 The INTO-CPS Association

The packages are:

Package Name	Description	Availability
dtaas-web	React web application	docker hub and github
libms	Library microservice	npmjs and github
		docker hub and github
logger-ms	Logger microservice	npmjs and github
		docker hub and github
runner	REST API wrapper for multiple scripts/programs	npmjs and github
dt-automation	Data and business-logic layer of the client	npmjs and github

REACT WEBSITE

```
docker build -t intocps/dtaas-web:latest -f ./developer/check/client.built.dockerfile .
docker tag intocps/dtaas-web:latest intocps/dtaas-web:<version>
docker push intocps/dtaas-web:latest
docker push intocps/dtaas-web:<version>
```

For example, to tag version **0.3.1**, use

```
docker tag intocps/dtaas-web:latest intocps/dtaas-web:0.3.1
```

To test the React website container on localhost, use

```
docker run -d \
  -v ${PWD}/client/config/local.js:/dtaas/client/build/env.js \
  -p 4000:4000 intocps/dtaas-web:latest
```

LIBRARY MICROSERVICE

The Dockerfile of the library microservice has a `VERSION` argument, which selects the package version to install from http://npmjs.com.

```
docker login -u <username> -p <password>
docker build -t intocps/libms:latest -f ./developer/check/libms.npm.dockerfile .
docker push intocps/libms:latest
docker build --build-arg="VERSION=<version>" \
  -t intocps/libms:<version> -f ./developer/check/libms.npm.dockerfile .
docker push intocps/libms:<version>
```

For example, to tag version 0.3.1, use

```
docker build --build-arg="VERSION=0.3.1" \
  -t intocps/libms:0.3.1 -f ./developer/check/libms.npm.dockerfile .
```

To test the library microservice on localhost, use

```
docker run -d -v ${PWD}/files:/dtaas/libms/files \
  -p 4001:4001 intocps/libms:latest
```

LOGGER MICROSERVICE

The logger microservice image is built from the source code with `logger.dockerfile`. Use the version in `servers/logger/package.json` as the image tag.

```
docker login -u <username> -p <password>
docker build -t intocps/logger-ms:latest -f ./developer/check/logger.dockerfile .
docker tag intocps/logger-ms:latest intocps/logger-ms:<version>
docker push intocps/logger-ms:latest
docker push intocps/logger-ms:<version>
```


Copyright © 2022 - 2026 The INTO-CPS Association

For example, to tag version 0.1.2, use

```
docker tag intocps/logger-ms:latest intocps/logger-ms:0.1.2
```

To test the logger microservice on localhost, use

```
mkdir logs
sudo chown -R 1000:1000 logs
docker run -d -v ${PWD}/logs:/dtaas/logger/logs \
  -p 4003:4003 intocps/logger-ms:latest
```

The captured events are written to `logs/workflow-logs.jsonl`.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5.5 Secrets for GitHub Actions

The publishing jobs of the GitHub Actions need the following secrets from Docker Hub:

Secret Name	Explanation
`DOCKERHUB_SCOPE`	Username or organisation name on Docker Hub
`DOCKERHUB_USERNAME`	Username on Docker Hub
`DOCKERHUB_TOKEN`	API token to publish images to Docker Hub, with `Read`, `Write` and `Delete` permissions

They also need this secret from npmjs:

Secret Name	Explanation
`NPM_TOKEN`	Token to publish npm packages to the default npm registry.

Publishing to GitHub Packages uses the built-in `GITHUB_TOKEN`. The test jobs upload coverage with an optional `CODECOV_TOKEN` secret, which public repositories do not need.

The following repository variables are optional. When set, the npm package is published under the given name instead of the name in its `package.json`:

Variable Name	Package
`PACKAGE_NAME`	Library microservice (`servers/lib`)
`NPM_RUNNER_PACKAGE_NAME`	Runner (`servers/execution/runner`)
`NPM_LOGGER_PACKAGE_NAME`	Logger microservice (`servers/logger`)

Add the secrets in the GitHub Secrets Setting of the fork, and the variables under **Settings > Secrets and variables > Actions > Variables**.

Copyright © 2022 - 2026 The INTO-CPS Association

6.5.6 Publish NPM packages

The DTaaS monorepo contains several npm packages.

Default npm registry

The default registry for npm packages is npmjs. Public packages are published to **npmjs**.

GitHub Actions publish the library microservice (`lib-ms.yml`), the runner (`runner.yml`), the logger microservice (`logger-ms.yml`) and the `@into-cps-association/dt-automation` library (`dt-automation.yml`). Each workflow calls the reusable `.github/workflows/publish-npm.yml` workflow.

A package can also be published by hand:

```
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 version of the runner package is published to this registry for each release of the DTaaS platform.

GitHub npm registry

The GitHub Actions of the project also publish the packages to the GitHub npm registry. Installing from this registry requires a personal access token from GitHub.

Private Registry

SETUP PRIVATE NPM REGISTRY

Publishing to npmjs is public and cannot be undone, so set up a private npm registry for local development. A private registry lets you publish and unpublish packages freely.

Verdaccio is the recommended registry. The following commands 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
```

Open `http://localhost:4873` in a browser and log in with the user credentials to see the published packages.

Publish to private npm registry

To publish a package to the local registry, run:

```
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` is updated accordingly. See the verdaccio docs for more information.

To unpublish a package, for example `@into-cps-association/runner@0.0.2` , run:

```
npm unpublish --registry http://localhost:4873/ \
  @into-cps-association/runner@0.0.2
```


Copyright © 2022 - 2026 The INTO-CPS Association

To install or uninstall this utility for all users, run:

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

USE THE PACKAGES

Packages in the private npm registry are used like regular packages installed from npmjs.

For example, to use the `@into-cps-association/runner@0.0.2` package, run:

```
sudo npm install --registry http://localhost:4873 \
  -g @into-cps-association/runner
runner # launch the digital twin runner
```


Copyright © 2022 - 2026 The INTO-CPS Association

6.5.7 Upgrade NPM Packages

A malicious npm release is usually found and pulled from the registry within days. The project therefore waits before taking any newly published package version.

- Dependabot waits **three days**, through `cooldown: default-days: 3` in `.github/dependabot.yml`.
- Upgrades made by hand bypass Dependabot. Run them through the `upgrade-npm-packages` agent skill or its script, which wait **ten days**.

Agent skill

The `upgrade-npm-packages` skill in `.agents/skills/` runs the script, explains every change, runs the project's build and tests, and pins back any upgrade that breaks them. Ask a coding agent to upgrade the npm packages of a project to use it. Coding Agents explains how skills are organised.

Usage

The script needs Node.js 24 or later, npm 7 or later and yarn 1. It has no npm dependencies of its own. To run it without an agent:

```
node .agents/skills/upgrade-npm-packages/scripts/npm_pkg_upgrade.js client/package.json
```

The script takes the path to one `package.json` and performs these steps:

1. Computes the cutoff date, ten days before now.
2. For every entry in `dependencies`, `devDependencies` and `optionalDependencies`, picks the newest stable, non-deprecated version published on or before the cutoff that stays within the range the spec already allows. The range prefix (`^`, `~` or none) is kept.
3. Writes the new versions into the `package.json`.
4. Deletes the `yarn.lock` beside it and generates a fresh one.

The range follows npm's meaning of the prefix, and an exact version is treated like `^`:

Spec	Stays below
`^6.0.3` or `6.0.3`	`7.0.0`
`^0.18.9`	`0.19.0`
`~30.4.2`	`30.5.0`

The output lists every changed package as `upgrade` or `DOWNGRADE`, and every package left alone as `skip`, with the reason. A line starting with `newer` names a version outside the range, such as a new major release, which the script does not apply. Take such a version by editing the `package.json` and running the script again.

Packages published from this repository

Packages in the `@into-cps-association/` scope, such as `@into-cps-association/dt-automation`, are built and published from this repository, so the cooldown does not apply to them. The script keeps their versions as written in the `package.json`, even when the release is only days old. Their provenance is tracked separately; see the issue on npm provenance attestations.

Copyright © 2022 - 2026 The INTO-CPS Association

How the new lockfile honours the cooldown

Yarn 1 has no setting to hold back recently published versions, and a fresh `yarn install` resolves every transitive dependency to its newest version. The script therefore builds the lockfile in an empty temporary directory, away from any existing `node_modules`:

1. `npm install --package-lock-only --before=<cutoff>` resolves the whole dependency tree, transitive dependencies included, as it stood at the cutoff. The exempt `@into-cps-association/` packages are left out of this step.
2. A second `npm install --package-lock-only`, without `--before`, adds the exempt packages. It keeps every version the first step locked. Any package this step adds is printed as `no cooldown: <name>@<version>` for review.
3. `yarn_lock_from_npm.js` converts the `package-lock.json` into a `yarn.lock` holding exactly the versions npm chose.
4. `yarn install --frozen-lockfile` checks that yarn accepts the lockfile without resolving anything itself. The lockfile is then copied beside the `package.json`.

`yarn import` is not used. It reads an existing `node_modules` in preference to the `package-lock.json`, and it re-resolves some entries from the live registry, which lets recent versions back in.

Run `yarn install` afterwards to install the new versions.

Review the result

The script applies the cooldown mechanically. Review the diff before committing it.

- **Downgrades.** A version newer than the cutoff is replaced by the newest older version.
- **Newer versions outside the range.** Review each `newer` line. A major release can break the build; for example, TypeScript 7 does not work with `ts-jest`, so the client stays on TypeScript 6.
- **Deliberate pins.** A `~` range keeps a package on its minor release. For example, `jest` in `servers/lib` stays on `~30.4.2`; see `servers/lib/DEVELOPER.md`.
- **Skipped entries.** Ranges such as `>=1.0.0`, `file:` and `npm:` aliases, prerelease versions and packages missing from the registry are left unchanged.

Then run the build and test commands for the project; see `PROJECT.md`.

Linting and tests

The `Lint and Test Node Scripts` job in `.github/workflows/lint-scripts.yml` checks the skill's `scripts/` directory with eslint and prettier, runs its unit tests, and checks every skill's Claude Code stub. The linters are installed only in CI, at exact versions and without a `package.json`. To run the same checks locally:

```
cd .agents/skills/upgrade-npm-packages/scripts
npm install --no-save --no-package-lock --ignore-scripts \
  eslint@10.11.0 @eslint/js@10.0.1 prettier@3.9.9
npx eslint .
npx prettier --check .
node --test ./*.test.js
```


Copyright © 2022 - 2026 The INTO-CPS Association

6.5.8 Vagrant Deployment

This page summarises the Vagrant setup in this repository. The full instructions are in `deploy/vagrant/README.md`.

Purpose

The Vagrant setup installs DTaaS in a virtual machine, so it runs on any operating system that supports Vagrant. The same machine can serve as a reproducible development and evaluation environment.

Host Requirements

Recommended host capacity:

- 16 GB RAM
- 8 vCPUs
- 50 GB available disk space

Required tooling:

- Vagrant
- vagrant-disksize plugin

Provisioning Modes

`deploy/vagrant` has two provisioning scripts:

- `user.sh` for the user installation (default)
- `developer.sh` for additional developer tooling

To enable developer provisioning, uncomment the `developer.sh` line in `deploy/vagrant/Vagrantfile`. Vagrant may time out while provisioning both scripts; in that case, run `developer.sh` from inside the virtual machine.

Basic Workflow

Run the following from the `deploy/vagrant` directory:

```
vagrant up
vagrant ssh
sudo bash /vagrant/route.sh
```

`route.sh` sets a cron job inside the virtual machine that removes the conflicting default route.

Configuration

The file `deploy/vagrant/config.json` sets the hostname, MAC address, disk size, VM name, memory and number of CPUs of the virtual machine.

Related

- DTaaS installation scenarios
- DTaaS package deployments: `docs/admin/dtaas/`
- Workspace deployments
- Platform services

Copyright © 2022 - 2026 The INTO-CPS Association

7. Known Issues in the Software

If you find a bug, please open an issue.

7.1 Third-Party Software

Some known issues come from third-party software included in the DTaaS platform. They are listed below.

7.2 GitLab

The GitLab OAuth 2.0 authorisation service has no way to sign out of a third-party application. After a user logs out of the DTaaS platform, GitLab still shows the user as signed in. The next time the user clicks the **Sign In** button on the DTaaS platform page, the GitLab login page is not displayed. Instead, the user goes directly to the **Library** page. Close the browser window after use to avoid this. Alternatively, open the linked GitLab instance (ex: `https://intocps.org/gitlab`) and sign out there. To exit the DTaaS platform completely, the user has to sign out of both the DTaaS platform and GitLab.

The same applies to the Workspace installation.

Copyright © 2022 - 2026 The INTO-CPS Association

8. Thanks

8.1 Funding Sources

The following projects have funded this project.

- Sustainable Water-based Cooling in Megacities (SWiM) - a project sponsored by the Grundfos Foundation.
- Security for the Digital Twin as a Service Platform - a project sponsored by the Thomas B. Thriges Foundation.
- CP-SENS (Cyber-Physical Sensing for Machinery and Structures) - a Grand Solution project funded by the Innovation Fund Denmark under the grant agreement 2081-00006B.
- DIGITbrain (Digital Twins for Manufacturing SMEs) - a project funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 952071.
- Digital Twins for Cyber-Physical Systems (DiT4CPS) - a project sponsored by the Grundfos Foundation.

8.2 Contributions

Contribution	Contributors
Mass Spring Damper	Prasad Talasila
Water Tank Fault Injection	Henrik Ejersbo and Mirgita Frasheri
Water Tank Model Swap	Henrik Ejersbo and Mirgita Frasheri
Desktop Robotti with RabbitMQ	Mirgita Frasheri
Water Treatment Plant and OPC-UA	Lucia Royo and Alejandro Labarias
Three Water Tanks with DT Manager Framework	Santiago Gil Arboleda
Flex-Cell with Two Industrial Robots	Santiago Gil Arboleda
Incubator	Morten Haahr Kristensen
Firefighters in Emergency Environments	Lars Vosteen and Hannes Iven
Mass Spring Damper with NuRV Runtime Monitor	Alberto Bonizzi
Incubator with NuRV Runtime Monitor	Alberto Bonizzi and Morten Haahr Kristensen
Incubator with NuRV Runtime Monitor Service	Valdemar Tang
Water Tank Fault Injection with NuRV Runtime Monitor	Alberto Bonizzi
Incubator Co-Simulation with NuRV Runtime Monitor FMU	Morten Haahr Kristensen
Incubator with NuRV Runtime Monitor FMU as Service	Valdemar Tang and Morten Haahr Kristensen
Incubator with NuRV Runtime Monitor as Service	Morten Haahr Kristensen and Valdemar Tang
Developer and example documentation	Astitva Sehgal
Documentation theme and diagrams	Jady Pamella Barbacena da Silva
Building Models page and bim-kit library	Jady Pamella Barbacena da Silva
Questions that ended up in the FAQ	Tanusree Roy and Farshid Naseri

Copyright © 2022 - 2026 The INTO-CPS Association

8.3 Developers

See the list of code contributors on GitHub.

8.4 Documentation

The platform is described in this paper:

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. In 2023 IEEE Smart World Congress (SWC), pp. 1-8. IEEE. doi:10.1109/SWC57546.2023.10448890. Preprint: arXiv:2305.07244.

Copyright © 2022 - 2026 The INTO-CPS Association

9. License

9.1 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-CPS 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-CPS 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
```


Copyright © 2022 - 2026 The INTO-CPS Association

```
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.
```


Copyright © 2022 - 2026 The INTO-CPS Association

```
"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.
```


Copyright © 2022 - 2026 The INTO-CPS Association

```
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 ---
```


Copyright © 2022 - 2026 The INTO-CPS Association

9.2 Third Party Software

The DTaaS platform uses a range of third-party software components. Each component remains subject to its upstream licence terms.

9.2.1 Deployment and Runtime Software

The core deployment stack currently references the following software:

Software Package	Usage	Licence
Docker CE	mandatory	Apache 2.0
Traefik	mandatory	MIT
Traefik Forward Auth	optional	MIT
GitLab CE	optional	MIT
GitLab Runner	optional	MIT
Dex	optional	Apache 2.0
Keycloak	optional	Apache 2.0
RabbitMQ	optional	MPL 2.0
MongoDB	optional	SSPL v1
Grafana	optional	AGPL v3
InfluxDB	optional	Apache 2.0 / MIT
PostgreSQL	optional	PostgreSQL Licence
ThingsBoard	optional	Apache 2.0

9.2.2 Development Environments

In addition to runtime software, the development and documentation workflows use the following tools.

Software Package	Usage	Licence
Node.js	mandatory	Node.js licence
npm	mandatory	Artistic Licence 2.0
Yarn	optional	BSD 2-Clause
Material for MkDocs	mandatory	MIT
JupyterLab	optional	BSD 3-Clause
MicroK8s	optional	Apache 2.0

Copyright © 2022 - 2026 The INTO-CPS Association

9.2.3 Client Packages With Copyleft Terms

Most client dependencies are permissively licensed and are declared in `client/package.json`. Two are named here because their terms carry obligations that a redistributor has to meet, and a name in a lock file is not a notice.

Software Package	Usage	Licence
@into-cps-association/bim-kit	mandatory	INTO-CPS Association Public Licence
web-ifc	mandatory	MPL 2.0

`web-ifc` is the IFC geometry kernel, and it reaches the client through `bim-kit`, which embeds its WebAssembly build so that a deployment serves no extra file. Embedding it does not change its terms.

MPL 2.0 is file-level copyleft and is compatible with redistributing DTaaS under the INTO-CPS Licence. What it requires of anyone shipping a build:

- **The source of the MPL-covered files has to be available** to whoever receives that build. Upstream is linked above, and the version in use is the one `client/yarn.lock` resolves for `web-ifc`.
- **The notice has to travel with the artifact.** Embedding the kernel inside a JavaScript chunk makes its provenance invisible in the built output, which is the reason this table exists instead of a pointer to `package.json`.
- **A modified MPL file stays MPL.** `bim-kit` uses `web-ifc` as a dependency and does not fork it, so the boundary is the package boundary.

9.2.4 Package Dependencies

Additional third-party dependencies for client, servers, CLI, and tooling are declared in:

- `client/package.json`
- `servers/**/package.json`
- `cli/pyproject.toml`
- `deploy/services/cli/pyproject.toml`
- `script/docs/mkdocs-requirements.txt`

Upstream dependency licences should be consulted in those manifests when detailed licence auditing is required.

Copyright © 2022 - 2026 The INTO-CPS Association