

interTwin

D5.4 – Final DTE Infrastructure Software Release

Status: Under EC Review

Dissemination Level: public



**Funded by the
European Union**

Disclaimer: Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them


Abstract

Key Words Computing, Storage, HTC, Cloud, Data, Orchestration, policies

This deliverable provides the final Digital Twin Engine (DTE) Infrastructure Software Release. The document contains a detailed update of the DTE Infrastructure software products, services and components described previously on Deliverable 5.2. It describes the current status of the four main pillars developed within WP5, as well as all its constituent subcomponents.

Finally, the document presents the release notes and details future plans.



Document Description			
D5.4 – Final DTE Infrastructure software release			
Work Package number 5			
Document type	Deliverable		
Document status	Under EC Review	Version	1.0
Dissemination Level	Public		
Copyright Status	 <p>This material by Parties of the interTwin Consortium is licensed under a Creative Commons Attribution 4.0 International License.</p>		
Lead Partner	CERN		
Document link	https://documents.eji.eu/document/3947		
DOI	https://zenodo.org/records/14727089		
Author(s)	<ul style="list-style-type: none"> • Daniele Spiga (INFN) • Paul Millar (DESY) 0000-0002-3957-1279 • Liam Atherton (STFC UKRI) • Adrian Coveney (STFC UKRI) • Giacinto Donvito (INFN) • Alessandro Costantini (INFN) • Enrique Garcia (CERN) • Michał Orzechowski (Cyfronet) • Zdenek Sustr (CESNET) • Dijana Vrbanec (DESY) 		
Reviewers	<ul style="list-style-type: none"> • Andrea Manzi (EGI Foundation) • Matteo Bunino (CERN) 		
Moderated by:	Andrea Anzanello (EGI Foundation)		
Approved by	AMB		

Revision History			
Version	Date	Description	Contributors
V0.1	22/11/2024	ToC	Enrique Garcia (CERN), Daniele Spiga (INFN), Andrea Manzi(EGI Foundation)
v0.2	10/12/2024	updated section 3	Daniele Spiga (INFN), Michal Orzechowski (Cyfronet), Zdenek Sustr (CESNET), Adrian Coveney (STFC UKRI), Dijana Vrbanc (DESY), Giacinto Donvito (INFN), Alessandro Costantini (INFN)
v0.3	09/01/2025	Update introduction and abstract	Enrique Garcia (CERN)
v0.4	12/01/2025	final editing	Daniele Spiga (INFN)
v0.5	16/01/2025	internal review	Andrea Manzi (EGI Foundation), Matteo Bunino (CERN)
V0.6	21/01/2025	Version ready for QA	All Authors
V1.0	24/01/2025	Final	

Terminology / Acronyms	
Term/Acronym	Definition
DTE	The Digital Twin Engine developed by interTwin
FTS	The File Transfer Service. A software component maintained by a development team at CERN.
Rucio	A third-generation data management software component, maintained by the Rucio development team.
gfal	Grid File Access Library; software that provides an abstraction for POSIX-like access.
CVMFS	The CernVM File System provides a scalable, reliable and low-maintenance software distribution service.
K8s	Kubernetes. Container Orchestration Technology
LHC	Large Hadron Collider. A large scientific facility, located in Geneva, that supports high-energy particle physics research.
OIDC	OpenID-Connect. OpenID is an open standard and decentralised authentication protocol promoted by the non-profit OpenID Foundation. OpenID-Connect is the third generation of OpenID technology; a widely adopted solution that provides an authentication layer on top of the OAuth 2.0 authorization framework.
QoS	Quality of Service



D5.4 – Final DTE Infrastructure Software Release

NVMe	Non-volatile memory express is a new storage access and transport protocol for flash and next-generation solid-state drives (SSDs)
scp	Secure copy protocol (SCP) is a means of securely transferring computer files between a local host and a remote host or between two remote hosts.
UFTP	UNICORE FTP is a file transfer tool similar to Unix'
WLCG	The Worldwide LHC Computing Grid is a collaboration of resource (storage and computing) providers that collectively support key research. Although originally conceived to support analysis of LHC data, the collaboration has broadened to support related fields of scientific research.

Terminology / Acronyms: <https://confluence.egi.eu/display/EGIG>



Table of Contents

1 Introduction.....	9
1.1 Scope.....	9
1.2 Document Structure.....	9
2 WP5 Architecture.....	10
3 Components.....	12
3.1 Federated Compute.....	12
3.1.1 Release Notes.....	13
Monitoring System Components.....	15
VK tracing.....	16
3.1.2 Future Plans.....	18
3.2 Federated Data Infrastructure.....	18
3.2.1 Teapot.....	18
3.2.2 ALISE.....	19
3.2.3 FTS3.....	20
3.2.4 RUCIO.....	21
3.2.5 Rucio JupyterLab extension.....	22
3.2.6 Onedata S3.....	23
3.2.7 Release Notes.....	24
3.2.8 Future plans.....	25
3.3 Intelligent Providers Orchestration.....	25
3.3.1 AI Based Orchestrator.....	25
Release Notes.....	27
The Orchestrator has been updated with the following improvements.....	27
PaaS Dashboard.....	28
Federation Registry.....	28
Future Plans.....	29
Monitoring and Ranking system with integrated AI.....	29
Integration with InterLink offload approach.....	30
Data Streamer.....	30
3.4 Resource Accounting System.....	30
3.4.1 APEL Accounting.....	30
Release notes.....	31
Future plans.....	32
3.4.2 Kubernetes Usage Accounting Probe.....	32
Release notes.....	33
Future plans.....	33
4 Conclusions.....	34
5 References.....	35



Table of Figures

<u>Figure 1 - System landscape diagram (in the C4 model) of the DTE Infrastructure. This is a high-level view that highlights the user interactions with WP5</u>	12
<u>Figure 2 - Accounting Dashboard, with placeholder data to demonstrate how usage is presented</u>	32

Table of Tables

<u>Table 1 – List of infrastructure components released by task</u>	12
<u>Table 2 – Current status of the calls</u>	14
<u>Table 3 - List of Spans</u>	16



Executive summary

The objective of this deliverable (D5.4) is i) to provide the final Digital Twin Engine infrastructure software Release of the components described on the First DTE Infrastructure software release ([D5.2](#)), and ii) to describe the integration with other Work Packages and core components.

The document has been prepared by the experts who have developed and integrated the various subsystems into the final DTE infrastructure.

1 Introduction

1.1 Scope

This document describes the final architecture of the DTE inter Twin infrastructure, detailing the final software release of all the DTE infrastructure components as well as their future plans.

1.2 Document Structure

This document is organised in three sections. [Section 2](#) provides a recap of the technical design of the architecture for the Digital Twin Engine infrastructure with the aim to highlight the principal components that will be discussed in [Section 3](#). The latter introduces the list of infrastructure components under development and provides details on their functionalities, release notes and future plans.

Finally, [Section 4](#) summarises the status of integration of WP5 components with other components and work packages. It also provides an overview of the current status of the testbeds and pilot deployment.

2 WP5 Architecture

The architecture of the DTE Infrastructure has 4 main pillars as shown in [Figure 1](#). The Artificial Intelligence (AI) based orchestration together with the Federated Compute module are key components of WP5 designed to complement each other's features. The resource orchestration is aware of various cloud resource providers and possesses the capability to determine, at any given point in time, the 'best provider' for supporting the deployment of a service defined by WP6. After deploying a high-level service, the objective is to enable a Compute Federation in a heterogeneous environment. This, in turn, means fully exploiting any available resource, including possibly specialised hardware, to cater to specific workflows such as high performance instead of high throughput (HPC vs HTC). The interTwin WP5 proposes the offloading mechanism as a potential solution for this challenge. The purpose of offloading is to facilitate a seamless extension of services to a remote provider. In practice, this means that the offloading mechanism enables the distribution of a unit of work (i.e. a job, a JupyterLab, a function in a serverless system, or a containerized application) to various locations. The decision to adopt a container-based approach is fully compliant with requirements and discussions made with WP6.

Science does not only require access to computing resources in a distributed environment but also to effectively process data exploiting a distributed federation of storages. This is another important component of WP5. In practice we need to guarantee that scientific data required by Digital Twins are accessible by the service as well as by the offloaded process. This necessitates the implementation of components for effective data handling, management, and storage federation. The latter involves dealing with diverse backends both in terms of technology and policy, scattered across different locations. Consequently, we must develop software that implements abstraction layers to allow an easy interaction with a complex storage topology.

Finally, the last pillar of the DTE infrastructure is the resource accounting system, which plays a crucial role in ensuring fair allocation and utilisation of resources in a heterogeneous distributed environment. For this purpose, a centralised resource accounting system is necessary to aggregate usage records from across the infrastructure.

D5.4 – Final DTE Infrastructure Software Release

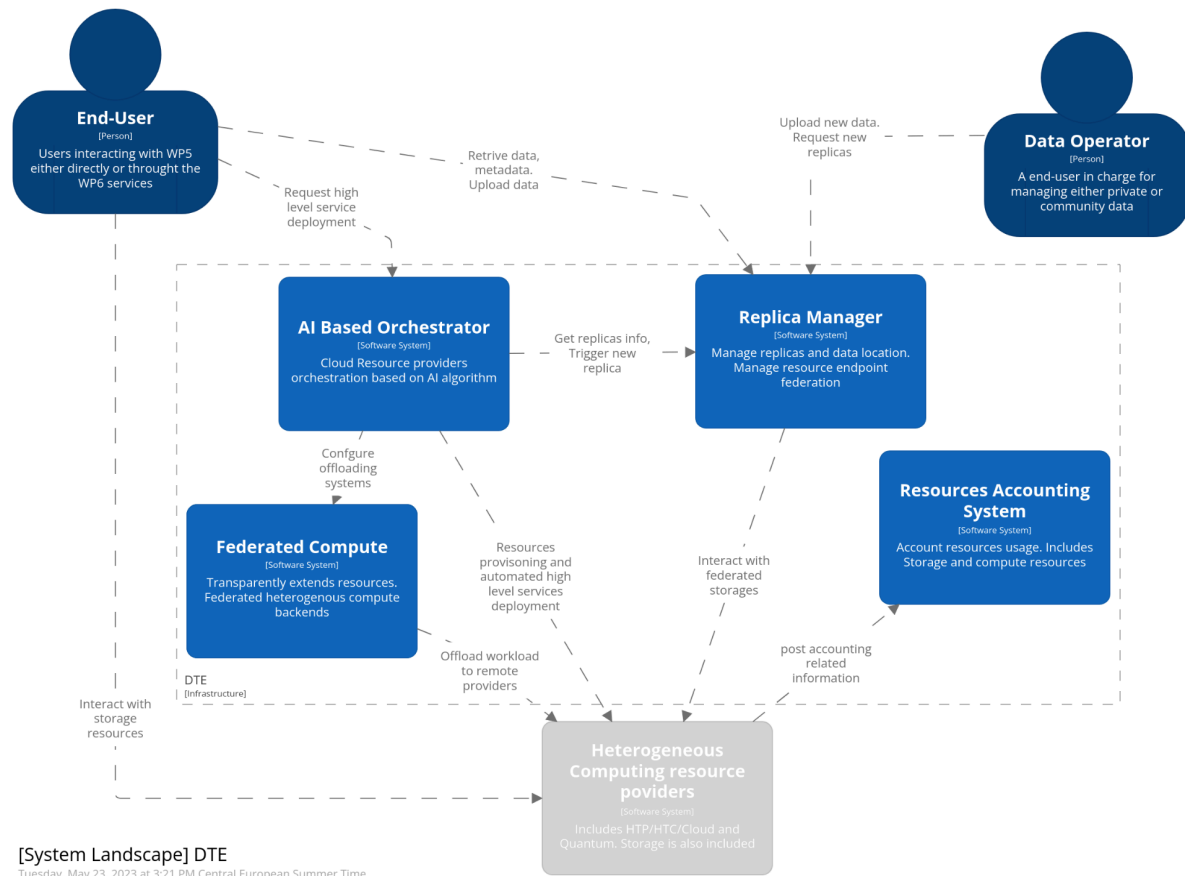


Figure 1 - System landscape diagram (in the C4 model) of the DTE Infrastructure. This is a high-level view that highlights the user interactions with WP5.

3 Components

The list of infrastructure components integrated and developed is shown in [Table 1](#). The design considerations are available in deliverable D5.3[R1].

Table 1 - List of infrastructure components released by task.

Task 5.1	Task 5.2	Task 5.3	Task 5.4
- interLink	- Teapot - Rucio - Rucio JupyterLab Extension - FTS - ALISE - Onedata S3	- APEL Accounting - Kubernetes Usage Accounting	- AI Based Orchestrator

3.1 Federated Compute

Component name	interLink
Description	<p>InterLink is an open-source service to enable transparent access to heterogeneous computing providers. It provides an abstraction for the execution of a Kubernetes pod on any remote resource capable of managing a Container execution lifecycle. The interLink component extends the Kubernetes Virtual Kubelet solution with a generic API layer for delegating pod execution on ANY remote backend. Kubernetes POD requests are digested through the API layer (e.g. deployed on an HPC edge) into batch job execution of a container.</p> <p>The API layer foresees a plugin structure to accommodate any possible backend integration.</p>
Value proposition	<p>Executing payloads in response to an external trigger like a storage event or a web server call.</p> <p>Frameworks for DAG¹ workflow management are usually well integrated with Kubernetes APIs.</p> <p>Users can exploit interLink via self-provisioned deployment (i.e. through already integrated high level services) or standalone Kubernetes deployment creating</p>

¹ <https://hazelcast.com/glossary/directed-acyclic-graph/>



	and deploying a simple container that can be scheduled on a remote system such as a Slurm batch on an HPC center. High Level services getting integrated are e.g.: Airflow/Kubeflow pipelines/Argo workflows/MLFlow, Jupyter notebooks
Users of the Component	Scientific users adopting Docker container-based workflow management or pipeline management systems.
User Documentation	https://github.com/interTwin-eu/interLink#fast forward-quick-start
Technical Documentation	https://github.com/interTwin-eu/interLink#digging-in-the-project
Responsible	Italian National Institute for Nuclear Physics (INFN) Contact point: Daniele Spiga - spiga@pg.infn.it
Licence	Apache 2.0
Source code	https://github.com/interTwin-eu/interLink

3.1.1 Release Notes

interLink is an open-source software product under development in the context of interTwin project. Its development started in order to provide an open-source solution capable of extending the container orchestration de-facto standard (kubernetes) to support offloading to any type of resource provider (Cloud/HTC/HPC) in a transparent manner, where little to no knowledge is required for the end user. The key objective of interLink is to enable a Kubernetes cluster to send containers/pods to a “virtual” node. This node seamlessly manages the entire lifecycle of the user's applications, whether on a remote server or, preferably, within an HPC batch queue.

Transparency is granted by the fact that it keeps exposing the very same experience of running a pod on cloud resources, thanks to the API layer that exposes the regular set of Kubernetes APIs.

The interLink project is organised in 3 major subcomponents. Namely:

- A Virtual Kubelet
- The InterLink API
- Plugins
- Monitoring system components

Below a detailed description per each subcomponent:

1. Virtual Kubelet

The Virtual Kubelet is based on the latest release provided by Kubernetes. It is responsible for the following actions:



- Attempting communication with the InterLink API to send a Service Account configuration. This configuration, utilised by the InterLink API, retrieves everything needed from the Kubernetes cluster for Plugins. The configuration is then stored in a temporary file and sent in the HTTP request body.
- Initiating a Ping call to InterLink every 10 seconds to verify the connection's availability. Upon unsuccessful 'restore', a new Service Account is generated and sent again.
- Managing basic Pod operation, such as adding, removing, querying pods to/from the cluster.

For every registered Pod the Virtual Kubelet sends a Create HTTP call to the InterLink API. In addition:

- Every 5 seconds, a Status HTTP call to the InterLink API is automatically issued to check every Pod's health.
- If a job executed (by a Pod) is terminated, Pods are removed

HTTP is implemented following the REST standard. The body of every call is the same: a marshalled (JSON type) list of Pod descriptors, which are stored in variables of type `v1.Pod`, a built-in go-client standard Kubernetes type. The current status of the calls is summarised in [Table 2](#).

Table 2 – Current status of the calls

Call	Outgoing URL
PingInterLink	InterLinkUrl:InterLinkPort/ping
SetKubeConfig	InterLinkUrl:InterLinkPort/setKubeCFG
Create	InterLinkUrl:InterLinkPort/create
Delete	InterLinkUrl:InterLinkPort/delete
Status	InterLinkUrl:InterLinkPort/status
getLogs	InterLinkUrl:InterLinkPort/getLogs

2. InterLink API

InterLink API is the core of the service. It is responsible for translating Virtual Kubelet's HTTP calls into standard, agnostic outputs for the plugin-based structure (Plugins). Below a description of the actions for each implemented call:

- *Ping*: allows the VK to check the InterLink API status



- *SetKubeConfig*: After receiving the configuration over the HTTP request body, it is stored inside `DataRootFolder/.kube/config` and set the `KUBECONFIG` env variable.
- *Create*: for each Pod registered to the Kubernetes cluster, the VK sends a Create Call to InterLink. In this phase, InterLink can retrieve all ConfigMaps, Secrets and EmptyDirs data. For all the containers in every single submitted Pod, secrets, ConfigMaps and, possibly, EmptyDirs are retrieved. All these values are then assembled together as a single struct of type `RetrievedPodData`. As a result, a JSON is sent over the HTTP request body to the Plugin.
- *Status/Delete*: The received body is just forwarded to the Plugin and any error is then handled and forwarded to the VK.
- *getLogs*: When receiving the `LogRequest`, there are many log options to satisfy, in any case the response is a byte array.

3. Plugins

The architecture is plugin-based and foresees a plugin for each distinct backend to be supported. For each API (VERB) the plugins perform specific operations based on what the actual backend is. Submitting a Pod to the cluster means the plugin will receive from InterLink the list of all related Secrets, ConfigMaps, EmptyDirs and the description of the Pod itself. Utilising this information, the plugin takes specific actions accordingly. Each Plugin accepts the three standard outgoing calls described above the InterLink API.

The plugins currently available are:

- `interlink-slurm-plugin`
 - A GO based plugin to connect Slurm managed batch system to interlink
- `interlink-kueue-plugin`
 - A Container plugin to connect kueue to interlink
- `interlink-htcondor-plugin`
 - A Python based plugin to connect HTCondor-CE to interlink
- `interlink-docker-plugin`
 - A python based plugin to connect any system with docker engine to interLink
- `interlink-unicore-plugin`
 - A Python based plugin to connect UNICORE API to interlink
- `interlink-arc-plugin`
 - a Python based plugin to connect ArcCE gateway to interLink

Monitoring System Components

To monitor the entire interLink stack, in particular the Virtual Kubelet (VK), an helm chart has been developed to ease the deployment steps. This is the first version of this component and has been developed in order to better support the operational aspects that relate to interLink.

The monitoring system is composed of the following two components:

- Grafana Tempo
- Grafana



Grafana Tempo (or simply Tempo) is an open-source distributed tracing backend developed by Grafana Labs designed to handle high-scale and high-volume distributed tracing data. The choice of Tempo was made because of its key features:

- **Scalability:** Tempo is designed to scale horizontally and handle millions of spans (a span is a single operation in a trace) per second. Moreover, it is capable of storing traces without requiring a database, instead leveraging object storage.
- **Simplicity:** Tempo does not index traces because it relies on Grafana for querying and visualizing traces. This approach reduces complexity and operational overhead.
- **Cost:** Tempo is cost-effective because it uses object storage for storing traces, which is cheaper than traditional databases.
- **Integration:** Tempo integrates with popular tracing protocols. One of them is OpenTelemetry, which is used by the Virtual Kubelet (VK) of the Interlink project.
- **Querying:** Tempo provides a query language that allows users to filter and aggregate traces. It also supports distributed sampling, which allows users to sample traces across services. The powerful query language is a key feature that enables users to extract insights from traces and it is one of the main reasons why Tempo was chosen for the monitoring system.

Grafana is an open-source platform for monitoring and observability that allows users to query, visualize, alert on, and understand metrics no matter where they are stored. It is used to create, explore, and share dashboards with teams and stakeholders. Grafana supports a wide range of data sources, including Tempo.

VK tracing

The Virtual Kubelet (VK) of the Interlink project is instrumented with OpenTelemetry to generate traces. OpenTelemetry is an open-source observability framework that provides APIs, libraries, agents, and instrumentation to collect telemetry data from applications and services. The traces generated by the VK are sent to Tempo, where they are stored and queried. Traces are generated by the VK when a request is made to the VK, and they contain information about the request, such as the details of the operation, the duration of the operation, and the services involved in the operation. A trace is a collection of spans, where each span represents a single operation in the trace. Spans are linked together to form a trace, which provides a complete view of the flow and performance of the operation. A span contains metadata, such as the name of the operation, the start and end time of the operation, and the service that generated the span.

The following table ([Table 3](#)) is a list of spans generated by the VK.

Table 3 - List of Spans

Span Name	Description	Attributes
-----------	-------------	------------



CreateHttpCall	Span that represents the HTTP call made by the VK to the Interlink API to create a pod.	pod.name, pod.namespace, start.timestamp, end.timestamp, duration, exitc.code
DeleteHttpCall	Span that represents the HTTP call made by the VK to the Interlink API to delete a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration, exitc.code
StatusHttpCall	Span that represents the HTTP call made by the VK to the Interlink API to get the status of a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration, exitc.code
LogHttpCall	Span that represents the HTTP call made by the VK to the Interlink API to get the logs of a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration, exitc.code
PingHttpCall	Span that represents the HTTP call made by the VK to the Interlink API check if the API is alive.	start.timestamp, end.timestamp, duration, exitc.code
CreatePodVK	Span that represents the call made by the VK to the Kubernetes API to create a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration
DeletePodVK	Span that represents the call made by the VK to the Kubernetes API to delete a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration
UpdatePodVK	Span that represents the call made by the VK to the Kubernetes API to update a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration
GetPodVK	Span that represents the call made by the VK to the Kubernetes API to get a pod.	pod.name pod.namespace, start.timestamp, end.timestamp, duration
GetPodStatusVK	Span that represents the call made by the VK to the	pod.name pod.namespace, start.timestamp,



	Kubernetes API to get the status of a pod.	end.timestamp, duration
GetPodsVK	Span that represents the call made by the VK to the Kubernetes API to get all pods.	start.timestamp, end.timestamp, duration

3.1.2 Future Plans

Feature wise the system is almost complete. The main priority for the upcoming period is on further integrating the support for the Datalake and data management interoperability.

The current logging capabilities are providing a wide set of information about the status of the Virtual Kubelets and this allows easier debugging in case of problems. However this is not enough and thus on the one side we plan to extend the management of the collected info while on the other side we plan to implement a full set of monitoring handles on the interLink API layer. The latter also includes the Plugin structure.

Additional improvements currently planned are related to the integration with ALISE service. These updates aim to address site resources access policies and improve the management of File System permissions on remote resources.

Furthermore during the last period of the project the plan is to implement extensive scale tests. This will be a key process and through all this we expect to bring the system to TRL 6.

3.2 Federated Data Infrastructure

3.2.1 Teapot

Component name	Teapot
Description	<p>Teapot is an easy-to-install edge service that a facility may deploy to provide remote access to their storage, in order to facilitate data ingress and egress. It provides the reference implementation of the teapot architecture: a scalable and technology agnostic approach to support data ingress and egress.</p> <p>FTS uses teapot to copy files from some external source into the facility's file storage, or to copy data from the facility's storage to some external destination. Rucio uses teapot to manage replicas at the facility: creating new</p>

	replicas with FTS or deleting cached data when the storage is under space pressure.
Value proposition	Unlike existing storage solutions that are compatible with Datalake, teapot does not require exclusive control of the underlying storage. Instead, it supports existing access methods. This allows facility admins to deploy teapot on top of existing storage solutions without disrupting established modes of access.
Users of the Component	<p>Teapot is a core component that most users do not access directly. Instead, teapot is used by any DT manager or user who is (directly or otherwise) managing data locality.</p> <p>One reason for transferring data is to satisfy the data requirements of some workflow when the required data is missing from the local storage of the facility that will run the workflow. Under these circumstances, the user that triggered the workflow is using the teapot software if either the source facility (hosting the data) or destination facility (requiring the data) has deployed the teapot software.</p>
User Documentation	https://github.com/interTwin-eu/teapot/blob/main/CONFIGURATION.md
Technical Documentation	n/a
Responsible	Deutsches Elektronen-Synchrotron (DESY). Contact point: Paul Millar < paul.millar@desy.de >
Licence	Apache 2.0
Source code	https://github.com/interTwin-eu/teapot

3.2.2 ALISE

Component name	ALISE
Description	Account Linking Service (ALISE) is a tool for linking a user's federated identity with their facility account. ALISE provides an automated procedure for users of a facility to register their federated identity.

Value proposition	<p>Most facilities have some account identity and access management (IAM) system. Among other things, this component is responsible for handling supported authentication, with typical features allowing passwords to be changed, handling forgotten passwords, and registering SSH public keys.</p> <p>Currently, most facilities have no support for OIDC (token-based) authentication. Therefore, their IAM solutions typically do not allow a user to register their OIDC identity.</p> <p>ALISE is an easy-to-deploy stand-alone service. By allowing users to register their OIDC identity, an ALISE instance allows sites to deploy other services that require OIDC/token-based authentication, and for those other services to identify users by their federated identity.</p> <p>The process to register a user's OIDC identity is needed only once per user. It requires no admin intervention.</p>
Users of the Component	All DT users that (directly or otherwise) make use of a facility that does not allow users to register their federated identity.
User Documentation	n/a
Technical Documentation	https://github.com/m-team-kit/alise/blob/master/README.md
Responsible	Karlsruhe Institute of Technology (KIT). Contact point: Marcus Hardt < hardt@kit.edu >
Licence	MIT
Source code	https://github.com/m-team-kit/alise/

3.2.3 FTS3

Component name	FTS3
Description	FTS3 is the software underlying a service responsible for globally distributing the majority of the LHC data across the WLCG infrastructure. It is a low-level data movement service, responsible for reliable bulk transfer of files from



	one site to another while allowing participating sites to control the network resource usage.
Value proposition	<p>Transferring large volumes of data between facilities requires a component that manages those transfers: monitoring their progress, cancelling transfers that have stalled or that take too long, and retrying failed transfers (where appropriate).</p> <p>Deploying FTS provides a common service for handling such transfers, allowing higher-level data management; e.g., bandwidth shaping links between facilities.</p>
Users of the Component	<p>FTS is a core component that most users do not access directly. Instead, FTS is used by any DT user or manager who is (directly or otherwise) managing data locality.</p> <p>FTS is used when a DT user or manager needs some data to be located at some facility and that data is not currently available.</p>
User Documentation	https://fts3-docs.web.cern.ch/fts3-docs/docs/cli.html
Technical Documentation	https://fts3-docs.web.cern.ch/fts3-docs/docs/developers.html
Responsible	CERN
Licence	Apache 2.0
Source code	https://gitlab.cern.ch/fts/fts3

3.2.4 RUCIO

Component name	Rucio
Description	<p>Built on more than a decade of experience, Rucio serves the data needs of modern scientific experiments.</p> <p>Large amounts of data, countless numbers of files, heterogeneous storage systems, globally distributed data centres, monitoring, and analytics. All coming together in a modular solution to fit your needs.</p> <p>Rucio provides a service that manages data locality. It provides a scalable solution for managing the dynamic</p>



	<p>locality of files in a heterogeneous, federated storage Datalake.</p> <p>Rucio will create new replicas when it identifies that data locality requirements are not being satisfied. It uses an FTS3 service to create those replicas by issuing requests and monitoring their progress.</p>
Value proposition	<p>When deployed, the Rucio software provides a service that allows a group of researchers to manage non-trivial amounts of data. For any given file, dataset, collection of files, or container (a collection of files and datasets), it provides information on where that data is currently available.</p> <p>It also supports dynamic, time-limited data placement, with data being made available for some period (e.g., to support some computational workflow).</p> <p>It optimises the use of available storage by operating a cache, assuming that data that was previously used is more likely to be needed in the future.</p> <p>Desired data locality is expressed in terms of declarative rules. These rules may be applied both to existing datasets and anticipated, future data.</p>
Users of the Component	In principle, all DT users that use the Datalake concept to manage their data are using Rucio. Depending on the use cases, DT users may interact directly with Rucio, or they may use Rucio via some intermediate service.
User Documentation	https://rucio.cern.ch/documentation/
Technical Documentation	https://rucio.cern.ch/documentation/
Responsible	CERN
Licence	Apache 2.0
Source code	https://github.com/rucio/rucio

3.2.5 Rucio JupyterLab extension

Component name	Rucio Jupyterlab extension
----------------	----------------------------



Description	The Rucio extension is a jupyterlab add-on that acts as an interactive user client, intended to ease data accessibility and data discoverability. It allows using certain Rucio capabilities like interactive data browsing and triggering data replicas to the storage volume attached to the jupyter server.
Value proposition	<p>The extension acts as an interface to the federated data infrastructure, aiming to simplify the interaction with the Rucio instance.</p> <p>Once a Jupyter session is spawned, the extension will be available on the left sidebar. The configuration to interact with the interTwin Rucio instance is executed in the background during the spawning of the session, allowing DT users to interact immediately with the Datalake.</p>
Users of the Component	Any DT user spawning a jupyter session on any remote resource compatible configured on the federated compute layer. The Rucio extension is compatible with the interTwin federated compute software stack (interLink), and therefore it can be installed on any jupyter-based container.
User Documentation	https://rucio.cern.ch/documentation/started/release-policy/#jupyterlab-extension
Technical Documentation	https://github.com/rucio/jupyterlab-extension/blob/master/CONFIGURATION.md
Responsible	CERN
Licence	Apache 2.0
Source code	https://github.com/rucio/jupyterlab-extension

3.2.6 Onedata S3

Component name	Onedata S3
Description	Onedata is a high-performance, distributed data management system designed for global infrastructures. It provides seamless access to heterogeneous storage resources and supports diverse use cases ranging from personal data management to large-scale scientific computations. Leveraging a fully distributed architecture,

	Onedata facilitates the creation of hybrid cloud environments that integrate private and public cloud resources.
Value proposition	<p>When deployed, the system enables users to collaborate, share, and publish data while supporting high-performance computations on distributed datasets via various interfaces, including POSIX-compliant native mounts, pyfs (Python filesystem) plugins, REST/CDMI APIs, and an S3 protocol (currently in beta).</p> <p>It supports on-the-fly replication, data pre-staging, data indexing, data caching and time-dependent cache cleanup.</p> <p>By supporting multiple types of storage backends, such as POSIX, S3, Ceph and OpenStack Swift, Onedata can serve as a unified virtual file system for multi-cloud environments.</p> <p>The rich API enables integration with existing systems and creation of complex data pipelines serving distributed workloads.</p>
Users of the Component	In the context of the interTwin Datalake, all users who need access to datasets available via Onedata, can access them via the Onedata S3 which is in turn used by Rucio.
User Documentation	https://onedata.org/#/home/documentation
Technical Documentation	https://onedata.org/#/home/documentation
Responsible	ACK Cyfronet AGH
Licence	Apache 2.0
Source code	https://github.com/onedata

3.2.7 Release Notes

This release represents the final release of the interTwin federated data management solution.

There are two external software components: FTS and Rucio. They are fully established projects, independent of the interTwin project. The software is production-ready, at TRL 9, and hardened with many years of production-critical use. Both projects have multiple deployments of their software, operated by different communities.



The ALISE software is currently in a development phase, under the aegis of interTwin. At the time of release, ALISE is TRL 4. The user-facing functionality of ALISE is mostly feature-complete; however, anticipated changes to the API imply that the necessary integration work (whereby a service uses ALISE to identify a user) should be considered experimental. Feedback from early adopters is encouraged, but any plans to deploy ALISE should be tempered by the anticipated changes to the API.

The teapot software has also been developed within the interTwin project. With this release, teapot is now TRL 6–7 and supports data transfer requirements of multiple, concurrent users. The per-user WebDAV instance management is automated, starting new services on demand, and terminating them if there is sufficient idle time. Finally the first version of the Onedata S3 component is released, allowing integration of Onedata technology in the interTwin federated data management solution.

3.2.8 Future plans

Some further improvements are planned for teapot. This includes integrating teapot with ALISE, to support automated identity management.

For ALISE, we anticipate possible improvements and stabilisation of the service-integration API, based on experience gained from integrating ALISE into various services. In addition, we plan to add support for client authentication in future versions of ALISE. This will limit access to the identity mapping information, providing this information to authorised services only.

As work continues with integrating the Datalake with various science use-cases, limitations may be found with the various components within the Datalake. Any such problems will be reported to the corresponding component's development and support teams. The members of interTwin will offer effort to fix such issues, should such capacity become available during the project's remaining lifetime.

3.3 Intelligent Providers Orchestration

3.3.1 AI Based Orchestrator

Component name	INDIGO PaaS Orchestrator
Description	<p>INDIGO – PaaS Orchestrator is the core component of the INDIGO PaaS layer. It collects high-level deployment requests and translates them into action to coordinate resources interacting with the underlying cloud infrastructures.</p> <p>It allows the provisioning of virtualized compute and storage resources on different Cloud Management Frameworks (like OpenStack, OpenNebula, AWS, MS</p>

	<p>Azure, Google Cloud, etc.). The PaaS orchestrator features advanced federation and scheduling capabilities. It ensures transparent access to heterogeneous cloud environments and the selection of resource providers based on criteria like user's SLAs, services availability, special hardware availability and data location.</p> <p>It manages deployment requests, expressed through templates written in TOSCA², the standard language for describing application topologies in cloud, and coordinates the deployment on the most suitable cloud site. To achieve this:</p> <ol style="list-style-type: none"> 1. it gathers SLAs, monitoring data and additional information from other platform services; 2. it asks the cloud provider ranker for a list of the best cloud sites.
Value proposition	<p>The INDIGO PaaS Orchestrator open source is a scientific gateway that allows users to easily access federated cloud systems, on top of which both simple and complex scientific virtual computational environments can be automatically deployed and configured.</p> <p>The provided dashboard implements a catalogue of services. When a user selects a specific service from the catalogue, the PaaS Orchestrator processes the inputs defined in the corresponding TOSCA template, completely automates the interaction with the backends, and hides the related complexity.</p> <p>The dashboard allows to simply define customised views of the Service Catalogue.</p>
Users of the Component	In principle, all DT users that use the clouds to deploy their high-level services. Depending on the use cases, DT users may interact directly with the PaaS Orchestrator via Dashboard interface.
User Documentation	https://indigo-dc.gitbook.io/indigo-paas-orchestrator/
Technical Documentation	https://indigo-dc.gitbook.io/indigo-paas-orchestrator/

² <http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html>



Responsible	INFN
Licence	Apache 2.0
Source code	https://github.com/indigo-dc/orchestrator

Release Notes

The Orchestrator collects all the information needed to deploy the virtual infra/service/job consuming other PaaS APIs. It depends on external services such as:

- SLAM Service: get the prioritised list of SLAs per user/group;
- Configuration Management DB: get the capabilities of the underlying IaaS platforms;
- Data Management Service: get the status of the data files and storage resources needed by the service/application;
- Monitoring Service: get the IaaS services availability and their metrics;
- CloudProviderRanker Service (Rule Engine): sort the list of sites on the basis of rules defined per user/group/use-case.

The orchestrator-dashboard Simple Graphical UI is a Python application built with the Flask microframework. Flask-Dance is used for OpenID-Connect/OAuth2 integration supporting the following functionalities:

- IAM authentication
- Display user's deployments
- Display deployment details, template, and log
- Delete deployment
- Create new deployment

The Orchestrator has been updated with the following improvements

- The possibility to manage the creation/deletion of IAM clients via Orchestrator. The INDIGO-IAM [INDIGO-IAM] client resource is completely managed by the Orchestrator: when a user requests the creation of a deployment requiring an INDIGO-IAM client, the Orchestrator creates the client, and when the user triggers the deletion of the deployment the Orchestrator deletes the corresponding INDIGO-IAM client;
- Added a parameter to force the deletion of a deployment. This may not delete some resources, for example IAM clients;
- Added a group authorization. Now the check is directly performed by the Orchestrator that stops the user's request when the used group is not allowed for the user;
- Added skipping monitoring workflow when the monitoring url is not specified;
- Metadata has been enriched with the information of preferred_username contained in the user's token

Additional developments have been performed to integrate the Orchestrator and the Federation Registry. In particular, the following main actions have been performed:

D5.4 – Final DTE Infrastructure Software Release

- Creation of the Data Type Objects (DTO) that defines the entities required by the Federation Registry (especially those returned by the API that the Orchestrator will contact);
- Creation of functions that map the entities returned from the Federation Registry to what the Orchestrator expects and that will be used in the other steps of the workflow.

Last release (v.4.0.0-RC.1):

<https://github.com/inf-n-datacloud/orchestrator/releases/tag/v4.0.0-RC.1>

PaaS Dashboard

It is a Flask³ application with a SQL database (MySQL) that enables users to interact easily with the services of the PaaS, particularly the Orchestrator, to create TOSCA-based deployments. The dashboard provides a user-friendly interface for managing and monitoring deployments.

The Dashboard has been updated with the following improvements:

- renewed graphics
- deployment ports management
- improved nodes management for cluster deployments
- integration with the Federation Registry as an alternative to the CMDB and the SLAT micro services
- added deployment status reset functionality
- bugs fixing

Latest release (v4.3.0-RC1):

<https://github.com/inf-n-datacloud/orchestrator-dashboard/releases/tag/v4.3.0-RC1>

Federation Registry

Efforts have been made to replace the Service Level Agreement Tool (SLAT) and the Configuration Management Database (CMDB) since they are outdated tools with vulnerabilities. The Federation Registry [FederationRegistry], that is a Python web application providing public REST API to inspect the configurations of the federated providers and map the resources available to each user group. Previously the automatic population of the CMDB was performed by the Cloud Info Provider (CIP), now the Federation Registry Feeder, a Python script, contacts the federated providers and populates the Federation Registry.

Latest release:

- Federation Registry (v.1.0.0-beta.2) -
<https://github.com/inf-n-datacloud/federation-registry/releases/tag/v1.0.0-beta.2>
- Federation Registry Feeder (v1.0.0-beta.5) -
<https://github.com/inf-n-datacloud/federation-registry-feeder/releases/tag/v1.0.0-beta.5>

³ <https://flask.palletsprojects.com/en/stable/>



Future Plans

The INDIGO PaaS Orchestrator will be extended to support EGI-Checkin as identity provider for the PaaS services that require the creation of a client. The development already introduced in Orchestrator has been done to accept different providers but a plugin is needed to support the entire workflow for EGI Check-in as well.

The INDIGO PaaS Orchestrator will be enhanced in order to support a monitoring and Metering System.

Monitoring and Ranking system with integrated AI

In the default configuration, the PaaS-Orchestrator determines the provider to which the deployment creation request is submitted based on an ordered list of providers, selected according to the group the user belongs to. This list is provided by the Cloud Provider Rankerservice, which applies a ranking algorithm using a limited set of metrics related to deployments and the Service Level Agreements defined for the providers. The INDIGO PaaS Orchestrator submits the deployment request to the first provider in the list, and in case of failure, it moves to the next provider until the list is exhausted.

The new AI-ranker service aims to improve the current ranking system and optimize resource usage by adopting Artificial Intelligence (AI) techniques. In particular, we want the AI-ranker, using a proper set of metrics and AI algorithms, to provide the Orchestrator with a list of ranked providers that aims to minimize deployment errors and the time required to create a deployment.

In this context, significant preparatory work was carried out to identify the most relevant metrics, as well as the sources from which these metrics can be obtained. The main sources we identified are: Orchestrator logs, Orchestrator Dashboard DB, and monitoring service.

Some of the metrics we identified are: user resource demands, user group resource quotas, and the current resource usage by user group for the allowed providers (in terms of CPU, RAM, volumes, FIPs).

The subsequent preparation of the dataset allowed us to study the use case and to identify and compare various AI techniques. The proposed approach involves the creation of two models: a model for the classification of deployment success/failure and a regression model for the deployment creation time. A combination of the outputs of the two models allows for the definition of an ordered list of providers that the orchestrator can use for submitting the deployments. Currently, the models that yielded the best scores are the RandomForest Classifier and Regressor. Moreover, we computed the feature importance score which allowed us to remove some of the useless features.

Then we started to work on the automatization of metrics collection, using Kafka as the main technology. To reduce the number of sources to contact, we moved the useful information stored in the Orchestrator Dashboard DB directly into the Orchestrator logs. This work will allow us to get a dataset for AI workflows simply by contacting Kafka.



We decided to use MLFlow⁴, also used by *itwinai* – the AI component from WP6, to store Machine Learning (ML) models and as the core technology to perform training and inference. Currently we are developing the following components that make up the AI-ranker.

- AI-ranker-registry: ML model registry that allows to store trained ML models and related metrics, compare them, and attach metadata.
- AI-ranker-training: a ML training script that is triggered once a new message is uploaded into the training queue of Kafka. When the training is completed, the AI-ranker-training saves the trained ML model into the AI-ranker-registry.
- AI-ranker-inference: a ML inference script that is triggered once a new message is uploaded into the inference queue of Kafka. The ML model used for inference is taken from the AI-ranker-registry.

We plan to release the entire AI-ranker service by July 2025.

Integration with InterLink offload approach

PaaS will be enhanced in order to allow deployment of k8s based service with embedded virtual kubelet setup. So PaaS deployed k8s based services are transparently configured to offload via interLink.

Data Streamer

Since the workflow manager used by the PaaS-Orchestrator is quite complex, we are evaluating the usage of Kafka as a data streamer between the multiple components that will be involved in the new architecture. The advantages of these services involves the high availability and robustness of the service. Moreover, we expect to easily integrate new services extending the number of topics.

The first use cases for this tool foresee:

- The collection of the results of the tests executed by rally probes on the providers for monitoring purposes
- The collection of the PaaS-Orchestrator deployments logs to build the training set that will be used by the new AI-Ranker service.

We plan to release a stable version of this service by July 2025

3.4 Resource Accounting System

3.4.1 APEL Accounting

Component name	APEL Accounting
----------------	-----------------

⁴ <https://mlflow.org/>



Description	The Accounting Repository stores compute (serial and parallel jobs), storage, and cloud resource usage data collected from resource centres within an e-infrastructure. Accounting information is gathered from distributed sensors into a central Accounting Repository, which is processed to generate summaries that are available through an Accounting Dashboard.
Value proposition	Aggregated statistics on resource usage available via a central dashboard to enable resourcing and funding decisions to be made.
Users of the Component	Resource centre admins; Research community managers
User Documentation	https://docs.egi.eu/providers/operations-manuals/man09_accounting_data_publishing/
Technical Documentation	https://github.com/apel/ssm/blob/dev/README.md
Responsible	STFC UKRI
Licence	Apache License, Version 2.0
Source code	https://github.com/apel/apel (client and server software) https://github.com/apel/ssm (messaging tool) https://github.com/apel/grafana-dashboards/ (dashboard)

Release notes

This release is an adaptation of the initial baseline release that used mostly existing sub-components. These include the SSM (Secure STOMP Messenger) messaging tool to exchange accounting records, the APEL accounting repository software, and the Grafana-based Accounting Dashboard. It has been modified to better suit it to interTwin requirements in terms of how data is presented. A demonstration with sample data is shown in [Figure 2](#).

The system has been deployed in a pre-production environment and pending interTwin usage data being sent and loaded from the other components and resource providers, will then be made externally visible in a production environment.

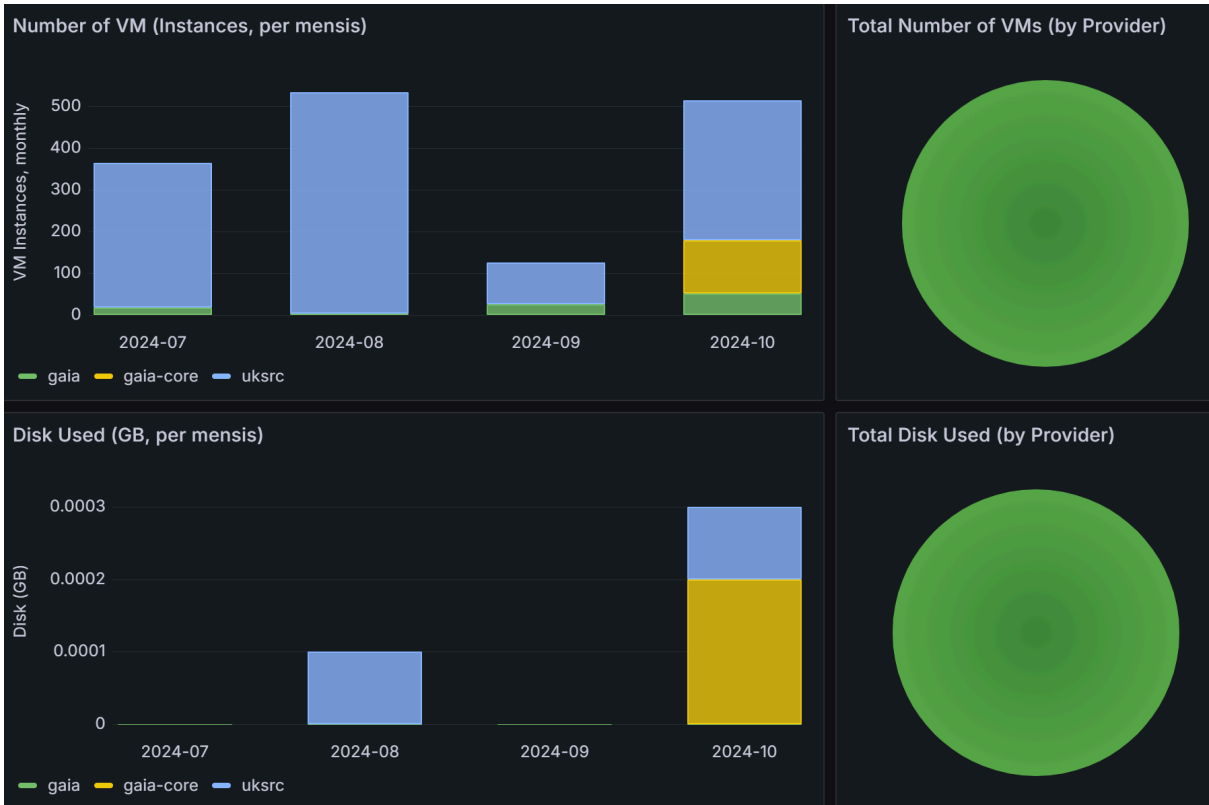


Figure 2 - Accounting Dashboard, with placeholder data to demonstrate how usage is presented.

Future plans

Work is ongoing to add EGI Check-in integration as the means of accessing the interTwin Accounting Dashboard. Once data is loaded and presentable from the Dashboard, feedback will be sought on its content and presentation which will then be used to inform future development of both the Dashboard and of the accounting probes, such as the Kubernetes usage accounting probe. As for the types of accounting collected by the accounting system, these will be extended to include GPUs and other accelerators.

3.4.2 Kubernetes Usage Accounting Probe

Component name	Kubernetes Usage Accounting Probe
Description	The tool extracts usage information from a local Prometheus database in a Kubernetes Cluster, and produces an OGF Cloud Usage Record that can be consumed by APEL. It is designed for regular execution at arbitrarily chosen intervals, to provide usage updates at selected granularity.
Value proposition	Usage statistics provided in sufficient detail to track, over time, the resource consumption by any supported user group or individual.
Users of the Component	Resource centre admins



User Documentation	N/A
Technical Documentation	https://github.com/interTwin-eu/kubernetes-usage-accounting/blob/master/README.md
Responsible	CESNET, EGI
Licence	ASL 2.0
Source code	https://github.com/interTwin-eu/kubernetes-usage-accounting

Release notes

This is a baseline release in the context of InterTwin.

Future plans

Further development will be reactive, based on experience with adoption at other sites.

4 Conclusions

The final release of WP5 software components includes all the packages used to build the DTE Infrastructure. The enhancements have been presented also highlighting the integration between various components. RUCIO and compute systems as well as ALISE and storage systems are two examples. The plan foresees further evolution in this respect.

The activities carried on in collaboration with resource providers as well as with scientific communities and with core and thematic modules have been a key to the evolution of the DTS software systems because of the gathered feedback during the early testing phases. The pilots deployed with the aim of validating the architecture and evolving the related software components currently involve almost all the sites that take part in WP5. In particular we are using two cloud providers (UKRI and GRNET) to instantiate k8s clusters from where we managed to federated both HTC (KBFI) and HPC (VEGA, Juelich, PSNC) both for compute via offloading and data via Datalake solutions. All these testbeds will keep playing a key role in the final part of the project where we plan not only to further consolidate the systems but also to implement scale tests to prove the robustness of the developed services.

5 References

Reference	
No	Description / Link
R1	interTwin D5.3 Final Architecture design update based on the first experiences Diego Ciangottini, Paul Millar, Dijana Vrbanec, Liam Atherton, Marica Antonacci, Daniele Spiga, Andrea Manzi, Renato Santana, David Kelsey, Adrian Coventry, & Shiraz Memon. DOI: https://zenodo.org/records/13710965

