EUROPEAN COMMISSION

HORIZON EUROPE PROGRAMME – TOPIC: HORIZON-CL5-2022-D2-01

FASTEST

Fast-track hybrid testing platform for the development of battery systems

Deliverable D6.4: LIMS integration with Digital Twin

Primary Author [Dr. Shuchen Liu]

Organization [FEV]

Date: [20.03.2026]

Doc. Version: [V1.0]

Funded by the European Union

UK Research and Innovation

Co-funded by the European Union and UKRI under grant agreements N° 101103755 and 10078013, respectively. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Climate, Infrastructure and Environment Executive Agency (CINEA). Neither the European Union nor CINEA can be held responsible for them

Document Control Information	
Settings	Value
Work package:	6 – Development of hybrid testing platform
Deliverable:	LIMS integration with Digital Twin
Deliverable Type:	R – Document, report
Dissemination Level:	PU - Public
Due Date:	30.11.2025 (Month 30)
Actual Submission Date:	30.03.2026
Pages:	< 30 >
Doc. Version:	V1.0
GA Number:	101103755
Project Coordinator:	Bruno Rodrigues | ABEE (bruno.rodrigues@avestaholding.com)

Formal **Reviewers**		
Name	Organization	Date
Koen Vanden Boer	**FM**	27.02.2026
Nuno Marques	**INEGI**	03.03.2026
Igor Mele	**UL**	17.2.2026
Batuhan Cinar	**SURREY**	18.03.2026

Document History			
Version	Date	Description	Author
0.1	19.01.2026	First Draft before review	Murat Bayraktar (FEV) Dr. Shuchen Liu (FEV) Peter May (FEV)
0.2	13.03.2026	Modification after review FM, INEGI, UL	Murat Bayraktar (FEV) Dr. Shuchen Liu (FEV)
0.3	20.03.2026	Modification after review Surry	Dr. Shuchen Liu (FEV)
1.0	20.03.2026	First Submission	Murat Bayraktar (FEV) Dr. Shuchen Liu (FEV) Peter May (FEV)

Project Abstract

Current methods to evaluate Li-ion batteries safety, performance, reliability and lifetime represent a remarkable resource consumption for the overall battery R&D process. The time or number of tests required, the expensive equipment and a generalized trial-error approach are determining factors, together with a lack of understanding of the complex multiscale and multi-physics phenomena in the battery system. Besides, testing facilities are operated locally, meaning that data management is handled directly in the facility, and that experimentation is done on one test bench.

The FASTEST project aims develop and validate a fast-track testing platform able to deliver a strategy based on Design of Experiments (DoE) and robust testing results, combining multi-scale and multi-physics virtual and physical testing. This will enable an accelerated battery system R&D and more reliable, safer and long-lasting battery system designs. The project's prototype of a fast-track hybrid testing platform aims for a new holistic and interconnected approach. From a global test facility perspective, additional services like smart DoE algorithms, virtualized benches, and digital twin (DT) data are incorporated into the daily facility operation to reach a new level of efficiency.

During the project, FASTEST consortium aims to develop up to TRL 6 the platform and its components: the optimal DoE strategies according to three different use cases (automotive, stationary, and off-road); two different cell chemistries, 3b and 4 solid-state (oxide polymer electrolyte); the development of a complete set of physics-based and data driven models able to substitute physical characterization experiments; and the overarching Digital Twin architecture managing the information flows, and the TRL 6 proven and integrated prototype of the hybrid testing platform.

LIST OF ABBREVIATIONS, ACRONYMS AND DEFINITIONS

Acronym	Name
ACR	Azure Container Services
AKS	Azure Kubernetes Services
DoE	Design of Experiment
DT	Digital Twin
FMU	Functional Mock-Up Unit
IoT	Internet of things
IP	Internet Protocol
JSON	Javascript Object Notation
JVM	Java Virtual Machine
LIMS	Laboratory Inventory Management System
MQTT	Message Queuing Telemetry Transport
MQTT	Message Queuing Telemetry Transport
POC	Proof Of Concept
RAM	Random Access Memory
SFTP	Secure File Transfer Protocol
UI	User Interface
UUT	Unit Under Test
UUID	Unit under test ID
XiL	X in the loop

LIST OF TABLES

LIST OF FIGURES

Table of Contents

1. EXECUTIVE SUMMARY

Deliverable D6.4 focuses on the process integration digital twin into Laboratory Information Management System (LIMS), as well as the cloud-based and physical-based battery tests. It explains how the data flows between them to carry out a near real time simulation according to the end user preferred schedule. A clear description of how critical information such as the Unit Under Test (UUT) and simulation metrics are exchanged between the interacting components is laid out.

In the current testing facility landscape, operations are often managed locally, with data management and storage handled directly within the facility. The LIMS is introduced as a central component of the FASTEST project to streamline the management of testing facilities. The efficient use of available resources is highlighted as crucial for overall efficiency improvement.

D6.4 specifies the integration mechanisms for the two main DT–LIMS contact points: requesting and receiving model assets (FMU plus interconnection configuration) and publishing near real-time metrics back to DT. The adopted pattern forwards validated test requests to DT via MQTT, enables secure DT-to-LIMS model transfer via SFTP into Azure Blob Storage using private connectivity, and triggers orchestration automatically through Event Grid. A Durable Functions pipeline then performs file validation and preparation, links to DoE and schedule optimization stages where applicable, and dispatches execution instructions to virtual and physical benches; during execution, telemetry is shared via an MQTT broker such that DT and other subscribers can consume live results.

Finally, the deliverable reports initial integration testing with project partners to validate the secure model exchange and the end-to-end connectivity needed for subsequent hybrid workflows.

2. OBJECTIVES

Purpose of this work task T6.4 is the integration of LIMS system with the digital twin to retrieve critical information such as UUT and correct models for each requested test simulation. To ensure a functional system that includes both digital twin and LIMS, the system architecture needs to be designed carefully alongside the required communication protocols. Nevertheless, design decisions must be made regarding the FMU model generation, data management and workflow orchestration. The objective of this deliverable is to explain how these decisions were made, explain the reasoning behind them and showcase integration test results done with the partners from Digital Twin and Physical Test Bench sides.

3. INTRODUCTION

The following chapter focuses on laying the foundation of integration concept between LIMS and DT systems from a technical overview. First, the purpose of the integration is explained, followed up by the problem description, technical requirements and the proposed solution in the form of a cloud architecture design. Detailed descriptions on the deployment method of each service used in our architecture supports the context mentioned.

3.1 Purpose of Integrating Digital Twin with LIMS

The purpose of integrating the digital twin with LIMS is to ensure holistic functionality of the systems, specifically the part where simulation models are exchanged in form of (.FMU) between LIMS and DT. Following the user lifecycle design principle, there are two main points of contact between DT and LIMS. One is where LIMS retrieves the correct models from DT, the other is where DT receives simulation metrics in near real-time basis from LIMS. We examined the technical requirements to ensure functionality and seamless integration of the mentioned two points and implemented them in our system accordingly.

3.2 Problem description

In this working package, we are focusing on finding and implementing the technical requirements for a smooth integration between DT and LIMS under the mentioned contact points. The first contact point is where the end-user chooses the battery type, test type, UUTID and their preferred date of simulation from LIMS dashboard. This data is forwarded to DT where the correct (.FMU) model and parameters will be chosen and forwarded back to LIMS. DT is serving as a model registry in this workflow, and we need to send the test request in a communication protocol that is robust and executes in a timely manner. Additionally, LIMS need to save the received models for each requested test on the cloud in safe and organized manner.

The second contact point is where DT reads simulation metrics in near real-time manner to draw them in a dashboard at DT system. For that we must specify the metrics structure, level of details, sending frequency and the communication protocol used to send them.

3.3 Requirements

The described problems require the followings:

- A standard JSON format to follow when sending and receiving simulation metrics
- A robust communication protocol for sharing these metrics with digital twin partners
- A topic tree that specifies on what topic the data should be exchanged when using MQTT
- Implement the correct database structure to ensure end users send only what DT expects from LIMS UI in terms of test names.

4. System architecture & Communication protocols

This section focuses on illustrating the architecture implemented to enable the technical requirements for integrating LIMS and DT on Azure Cloud.

4.1 Integration Architecture

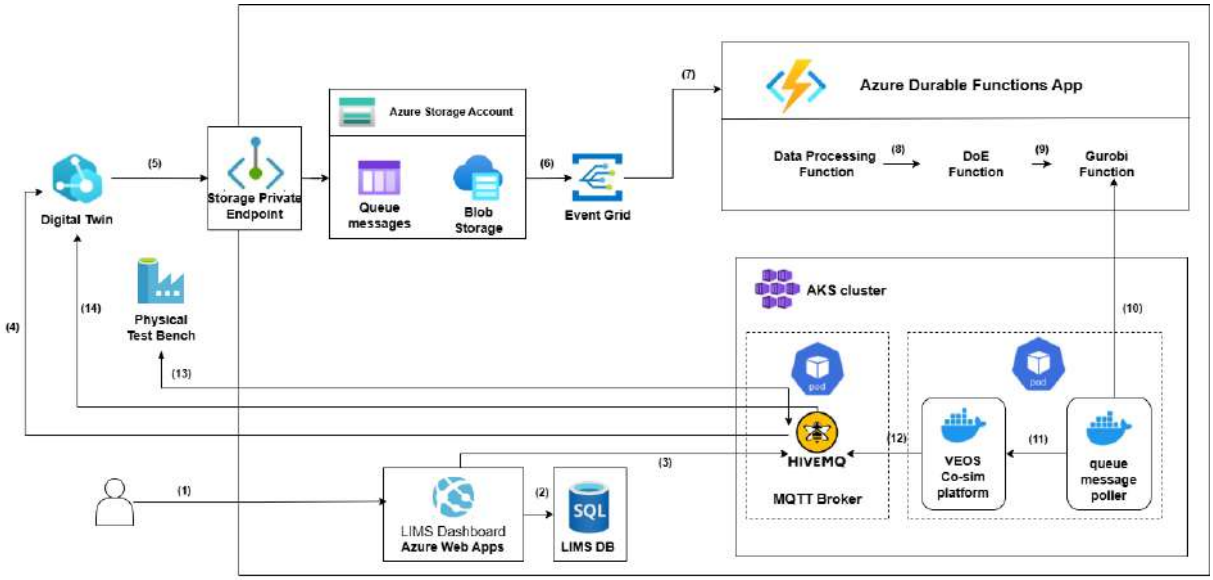

Figure 1 LIMS Azure Infrastructure for connecting virtual and physical test benches, DoE and Digital Twin

In Figure 1, an architecture diagram illustrates the integration between the digital twin and LIMS through Azure cloud services. Because of the scope of this deliverable, we focus on the interaction between digital twin and LIMS, this interaction starts with the end user requesting a battery test from LIMS dashboard. The user starts by defining a set of configurations for the requested test. These configurations are:

- preferred schedule of the test
- UUT ID
- Use case
- Relevant Standards
- Test Names

The available test names are dynamically filtered based on the user's selections. LIMS queries the connected SQL database to retrieve compatible test names matching the selected UUT ID, schedule constraints, use case, and relevant standards. After retrieving and listing the available choices for test names, the end user submits the test request. LIMS forwards this request via MQTT protocol to the digital twin. Digital twin acknowledges the request and selects a suitable (.FMU) model from its registry alongside its inter-communications JSON file based on the

test requirements. Digital twin uploads these files to LIMS Azure storage account under blob storage via a private endpoint for secure data exchange. Event Grid monitors the blob storage and triggers the LIMS pipeline when new files are uploaded. The LIMS pipeline consists of three sequential stages orchestrated by Azure Durable Functions:

1. **Data Processing**: Validates and prepares files from the digital twin
2. **DoE Execution**: Runs Design of Experiments algorithms for parameter optimization
3. **Gurobi Optimization**: Calculates the optimal test schedule.

The result of this schedule is forwarded to both the virtual and physical simulation test benches via Azure queue messages and MQTT respectively. As the virtual test bench container (VEOS co-simulation platform) is deployed inside a Kubernetes Cluster in AKS, a sidecar container deployed in the same Kubernetes pod polls the Azure queue for new test requests. Upon detection, it uses Linux namespace entry (nsenter) to execute commands in the VEOS container's namespace, triggering the simulation at the scheduled time. During the tests (both physical and virtual), the metrics are shared via MQTT using HiveMQ broker deployed at LIMS infrastructure.

Table 1 List of interaction paths among LIMS components

Path	Title	Description	Protocol	From	To
1	User test request	End user initiates battery test request and defines test configurations (schedule, UUT ID, use case, standards, names)	HTTPS	End user	LIMS Dashboard (Azure Web App)
2	Database Query	LIMS queries database to retrieve available test names based on user's configuration selections	SQL	LIMS Dashboard	LIMS SQL DB
3+4	Test Request Forward	LIMS forwards validated test request to Digital Twin for FMU model selection and file preparation	MQTT	LIMS Dashboard	Digital Twin
5	File Upload to Storage	Digital Twin uploads selected FMU model files and inter_connection.json to Azure Blob Storage via secure private endpoint	sFTP	Digital Twin	Azure Storage Account (Blob Storage)
6	EventGridd Monitor	Event Grid continuously monitors Blob Storage for new file upload events	Event Subscription	Azure Storage Account (Blob Storage)	Event Grid
7	Pipeline Trigger	Event Grid triggers the Azure Durable Functions App when new files are detected, initiating the data processing pipeline	EventGrid Event	Event Grid	Azure Durable Functions App
8	Data Processing activity	Durable Functions orchestrator executes first activity function Data Processing to validate and moves files from pool to simulation folders	Internal Function Call	File Monitor Function	Data processing function
9	DoE activity	DoE Function executes Design of Experiments algorithms	Internal Function Call	DoE	Gurobi
10	Gurobi schedule optimization activity	Gurobi Function calculates optimal test schedules and propagates results to virtual or	Internal Function Call	Gurobi	VEOS + Physical test bench

		physical test benches via queue messages & MQTT respectively			
11	Queue Message for VEOS and execute command accordingly	Sidecar container uses polls the queue message for new test requests. Then, it uses Linux namespace entry (nsenter) to execute test simulation commands in the VEOS co-simulation platform container	Linux IPC (nsenter)	Sidecar container	VEOS Co-simulation platform
12	Simulation executed and propagated near-real-time results	The co-simulation platform VEOS runs the simulation and shares results via MQTT to DT and physical test bench	MQTT	HiveMQ broker	DT & Physical Test bench
13	Physical test bench	A bidirectional flow where physical test bench receives test start command from HiveMQ broker and sends back results	MQTT	Physical test bench	HiveMQ broker
14	Digital Twin consumes near-real time metrics from running virtual simulations	Digital twin receives the metrics from the running test simulations in near real-time manner via MQTT	MQTT	VEOS	Digital Twin

4.2 Technology Stack Documentation

The LIMS platform leverages a modern, cloud-native technology stack built on Azure infrastructure. The user-facing LIMS Dashboard is developed as an ASP.NET MVC web application utilizing Bootstrap 5 for responsive design providing an intuitive interface for test configuration and workflow monitoring. The backend orchestration layer is implemented using Azure Durable Functions written in Python 3.11, deployed to Azure Functions (Premium Plan) in the West Europe region, managing the sequential execution of computational workflows including file validation, Design of Experiments (DoE) algorithms, and Gurobi-based schedule optimization. The virtual simulation environment utilizes VEOS Co-Simulation Platform deployed as a Docker container within an Azure Kubernetes Service (AKS) cluster, accompanied by a Python-based queue listener service running as a sidecar container in the same pod, which polls Azure Storage Queues for test execution commands and triggers simulations using Linux namespace entry (`nsenter`) for inter-container communication. Data persistence and exchange are managed through Azure Blob Storage with private endpoint connectivity, while event-driven triggers leverage Azure Event Grid for serverless workflow initiation. The architecture integrates with external systems via MQTT protocol through a HiveMQ broker deployed in AKS for real-time metrics collection from both physical and virtual test benches, with all components instrumented using Azure Application Insights for comprehensive telemetry and observability.

4.3 MQTT Broker Configuration & Message Schemas

The LIMS platform employs HiveMQ 4.37.0 as the MQTT broker, deployed within the Azure Kubernetes Service (AKS) cluster using a Helm chart configuration with high availability through a two-node replica set. The broker is configured with the "allow-all" extension enabled for simplified connectivity during internal development phase within a controlled network perimeter, utilizing the standard MQTT protocol on port 1883 for basic communication and reserving port 8883 for production grade TLS-secured connections. The HiveMQ Control Center is accessible on port 8080 with session affinity configured using ClientIP to ensure consistent administrative access. The deployment allocates 1 CPU core and 2048MB of memory per node, with Java Virtual Machine (JVM) tuned to utilize 50% of available RAM for both initial and maximum heap sizes through the configuration. Prometheus metrics are exposed on port 9399 at the root path for comprehensive monitoring and observability, with optional Service Monitor integration configured for 15-second scrape intervals. The broker operates with information level logging and supports both standard MQTT connections for telemetry data exchange between physical test benches, virtual VEOS simulations, and the LIMS dashboard.

As the MQTT protocol is used mainly for sharing simulation outcomes with subscribed clients, a standard message format needs to be defined for both input and output operations. The below format is agreed to be the message scheme sent from LIMS to DT after the end user chooses the test configurations from LIMS dashboard and during the test run inside the co-simulation platform VEOS. The below message will help DT determine the correct (.FMU) model for the requested test. The MQTT message to be sent follows the JSON structure shown in figure 2.

```
{
  "test_name": "Electrical - Overcharge",
  "test_UUID": "",
  "test_UUT": "Gen3b - Cell",
  "test_bench": "Virtual",
  "use_case": "Automotive",
  "variables": {
    "test_procedures": [],
    "custom_profile": {"Name":"current",
                       "type":"input",
                       "unit":"mA",
                       "values":[{"timestamp":0,
                                  "value":"1.0"},
                                  {"timestamp":1,
                                  "value":"2.0"},
                                  {"timestamp":2,
                                  "value":"3.0"},
                                  }],
  "FMU_signal_connections":[{"InSignalReference":"FMU _04/Port_C",
                             "OutSignalReference":"FMU _03/Port_E"}],
                     },
  "test_date": "2024-08-05 10:00:00",
  "status": {
    "value": "In Progress",
    "last_update": "2024-08-05 11:00:00"
  }
}
```


Figure 2 MQTT message structure sent from LIMS to DT

5. Digital Twin Interface Integration

The LIMS dashboard is the central place where end users interact with LIMS platform to request and track test requests run on either virtual or physical test benches. The dashboard makes use of SignalR to show real-time updates from the test status using the MQTT broker.

5.1 User Interface inputs

The dashboard is mainly composed of 3 pages as homepage "Dashboard", "Test Requests" an "New Requests". The homepage shown in figure 3 allows an overview of the test requests divided into total requests, submitted, running, completed and failed. The stats shown help both end-users and developers to observe the success rate of test requests.

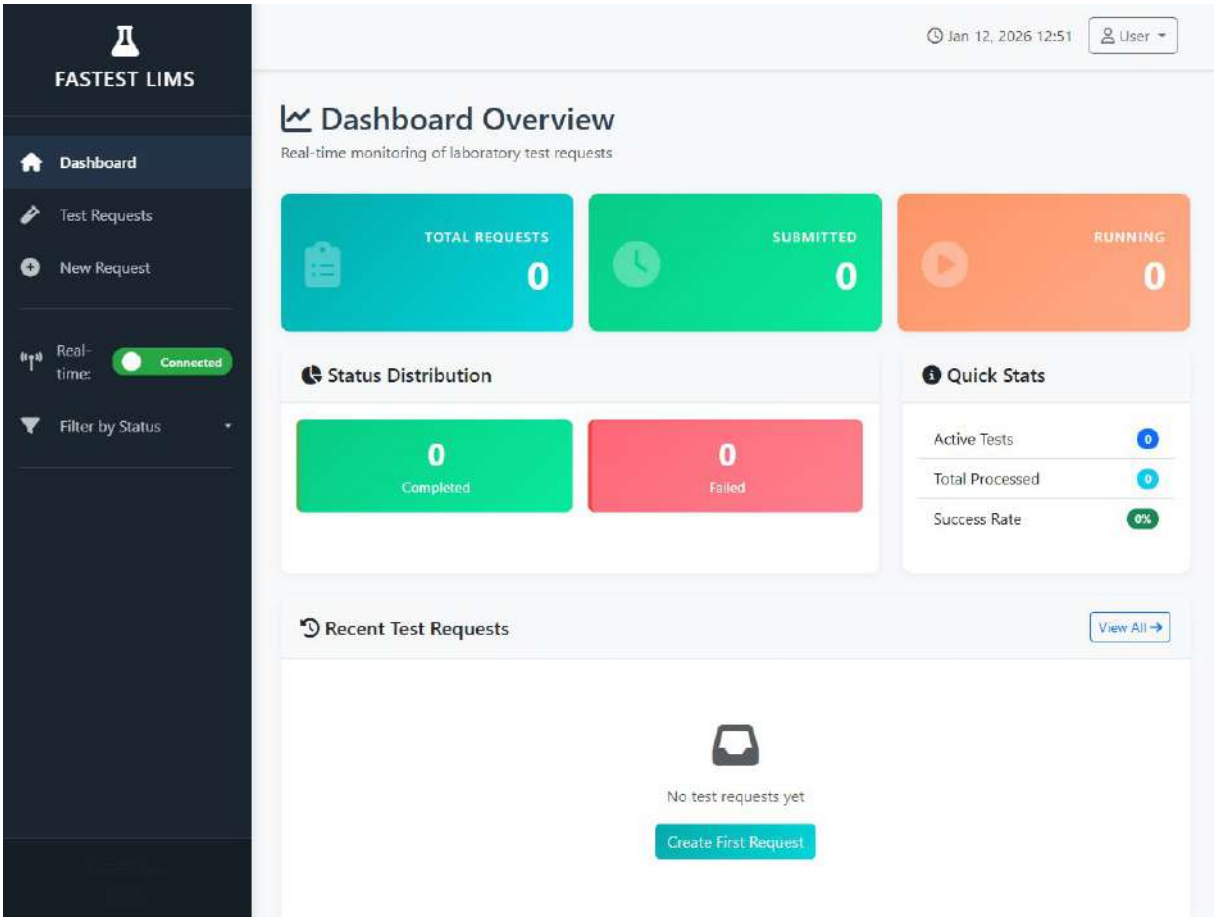

Figure 3 LIMS Dashboard Overview

The "Test Requests" page is where previously initiated test requests are shown with ability to see detailed progress statuses. In case no previous test requests are found, the users can directly create a new request on the page "New Request".

This page, shown in figure 4, shows the required information needed to initiate a test request. This information includes:

- Requester Name: for identification purposes
- Requester ID: for identification purposes
- Preferred test schedule: for test scheduling algorithm to optimize best suiting schedule near the preferred date.
- Unit Under Test ID:
    - Options available:
        - Generation 3b Battery cell/module/pack
        - Generation 4 Battery cell/module/pack
    - Purpose: for choosing a suitable test names
- Use case:
    - Options available:
        - Automotive
        - Stationary
        - Off-Road
    - Purpose: for choosing a suitable test names
- Standard:
    - Options available:
        - UL1642
        - IEC62660
        - IEC62660-2:2020-07
        - IEC62660-3:2022
        - IEC62619
        - UL1973:2022-02-25
        - UL9540
        - UN38.3
        - ISO12405.4
        - IEC62281
    - Purpose: for choosing a suitable test names
- Test Name:
    - Options available: the options are taken from the deliverable D1.1 on Figures 11 and 12. These are basically all the test names studied to be implemented under the scope of this project. These tests are

filtered according to the chosen values of the preceding values on the page (UUTID, use case, standard)

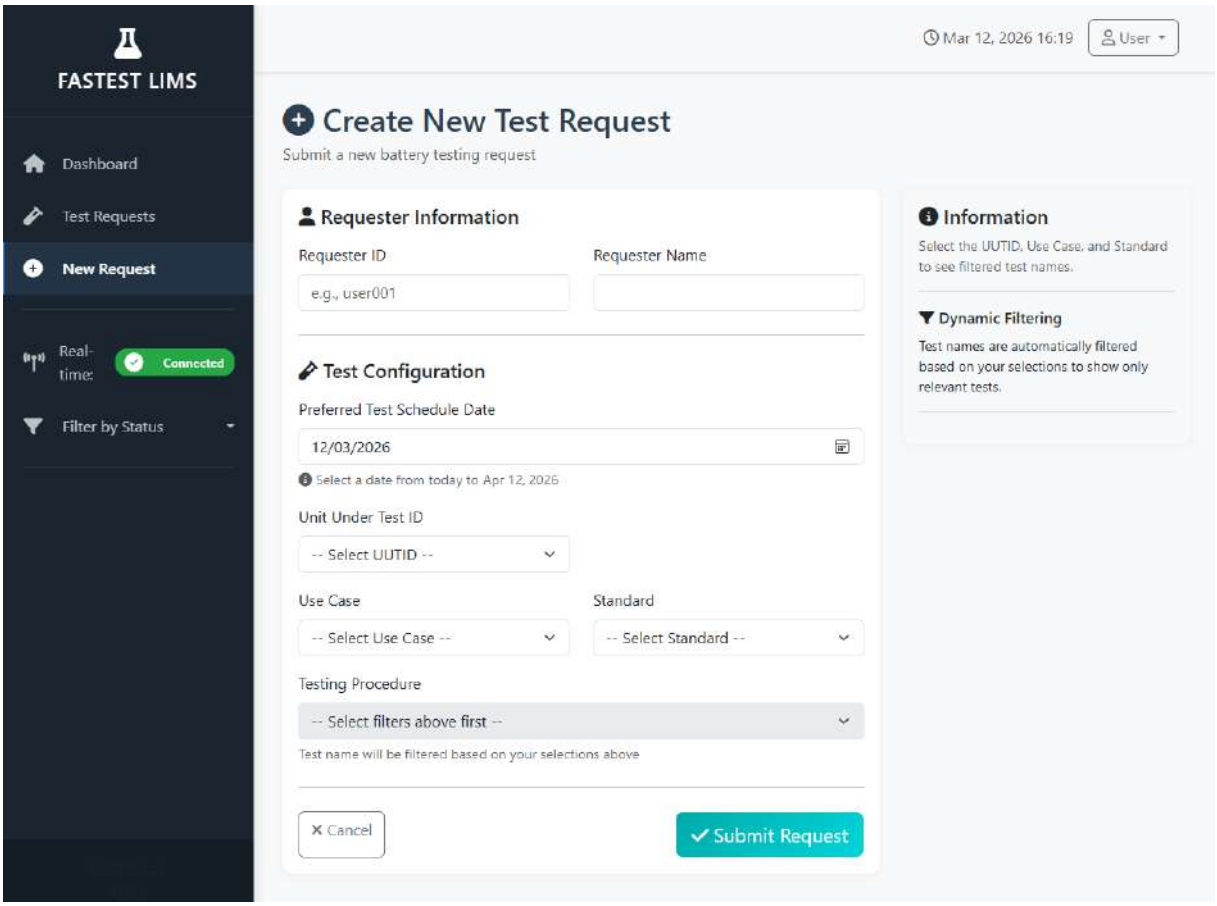

Figure 4 LIMS interface for test request submission

6. Workflow Orchestration & Data Management

This section provides comprehensive documentation of the Azure Durable Functions-based orchestration pipeline that coordinates the sequential execution of computational optimization workflows for battery testing. The orchestration system manages the end-to-end lifecycle from file ingestion through Design of Experiments (DoE) processing to optimal schedule generation using Gurobi optimization algorithms.

6.1 Durable Function Pipeline Documentation and data exchange pattern

The Azure Durable Functions App serves as the orchestration backbone of the LIMS pipeline, providing stateful workflow management for the battery test optimization process.

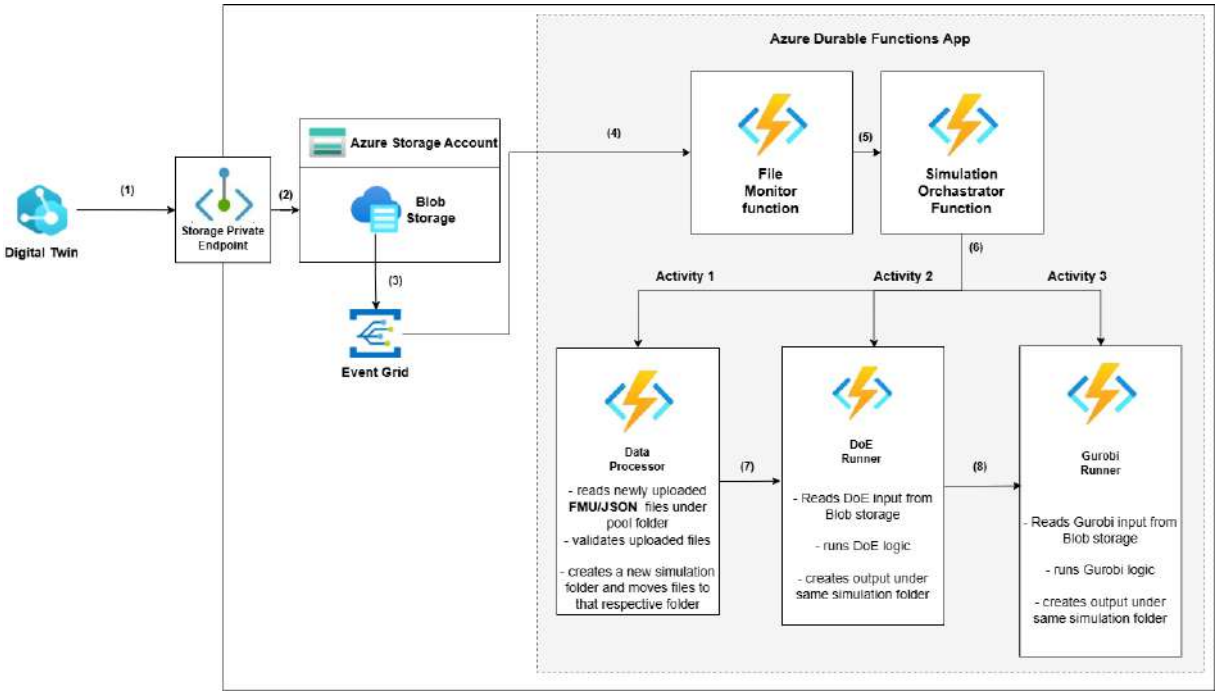

Figure 5 Detailed look at the LIMS pipeline

As illustrated in Figure 5, the pipeline implements a three-stage sequential processing architecture where each stage represents a distinct computational workload with specific input/output requirements and execution characteristics. The pipeline architecture is designed to handle long-running computational workflows that can span from several minutes to hours, depending on the complexity of the optimization problems. Unlike traditional serverless functions that are limited by execution timeouts, the Durable Functions framework provides a durable execution model that persists workflow state to Azure Storage, enabling the orchestration to survive function host restarts, scaling events, and infrastructure updates without losing progress. The pipeline consists of five primary components that work together to implement the complete workflow.

- File Monitor Function (EventGrid trigger)
- Simulation Orchestrator Function
- Activity 1: Data processor
- Activity 2: DoE runner
- Activity 3: Gurobi runner

The File Monitor serves as the entry point for the pipeline, automatically triggered when FMU model files arrive in the blob storage "pool" container. This function extracts blob metadata from the Event Grid event, validates the payload, and initializes a new orchestration instance. The function returns an HTTP 202 (Accepted) response with the orchestration instance ID for tracking purposes. Execution completes within seconds as it performs no computational work.

Simulation Orchestrator function:

The Orchestrator function coordinates the execution of three sequential activities while maintaining workflow state. It manages activity invocations in strict sequential order, implements automatic retry logic for transient failures, and propagates metadata between activities. The orchestrator uses a replay-based execution model where the function is re-executed from the beginning each time an activity completes, with orchestration history stored in Azure Storage Table to reconstruct state.

Activity 1 (Data processor):

The Data Processor validates uploaded files and prepares the folder structure for subsequent processing stages. It first checks the pool container for required files (minimum 1 FMU file + inter_connection.json), validating file completeness before proceeding. Upon validation, it creates a timestamped simulation folder using the format fastest_simulation_YYYYMMDD and moves files from pool to the structured directory {simulation_folder}/fastest-models/fmus/. The UUID will be appended to the name of each newly generated folder at a later stage of the project to assist in detecting the FMU uploaded by DT. The activity validates FMU file integrity, creates the DoE input folder at {simulation_folder}/doe-input/, and cleans up the pool container by deleting processed files. The activity returns comprehensive metadata including the workflow ID, simulation folder path, FMU path, DoE input path, list of moved files, and validation results (file counts, validation status). This metadata serves as the coordination mechanism for subsequent activities.

Activity 2 (DoE runner):

The DoE Runner executes Design of Experiments algorithms to generate optimal parameter configurations for battery testing. It reads FMU files and the interconnection JSON from blob storage using paths provided by Activity 1, then executes DoE algorithms to optimize parameter space coverage while minimizing the number of required test runs. The activity generates test configurations covering the factorial design space and writes DoE results to {simulation_folder}/doe-output/.

Activity 3 (Gurobi):

The Gurobi Runner calculates the optimal test execution schedule using Gurobi's mixed-integer linear programming (MILP) solver. It reads DoE experiment configurations from blob storage, formulates a MILP optimization problem with the objective to minimize total test duration (makespan) while respecting resource constraints and temporal restrictions. The solver considers resource capacity limits, working hours constraints, and setup/teardown time requirements to generate a feasible schedule that maximizes equipment utilization efficiency.

After optimization completes, the activity writes the schedule to {simulation_folder}/schedule-output/ and sends a queue message to the VEOS queue to trigger test execution. The final output of the optimization is actual optimized start date of the test, that might be the same as requested, and the virtual machine name to run the test on. This architecture leverages Durable Functions' orchestrator pattern to guarantee exactly-once execution semantics, automatic retry logic for transient failures, and the ability to track long-running workflows without maintaining persistent HTTP connections, making it ideal for managing the multi-stage, computationally demanding battery test optimization pipeline.

Data exchange pattern followed in the pipeline implements a hybrid strategy that balances orchestration efficiency with data durability. Data is exchanged on two layers, one responsible for small data (metadata exchange) and the other (Payload Exchange) is responsible for larger data files like the (.FMU) models

Metadata Exchange: Small JSON objects (typically <10KB) containing workflow metadata pass directly between activities through the orchestration context. This includes workflow IDs, folder paths, file counts, and validation flags. Metadata is serialized and stored in orchestration history, enabling the azure managed durable function replay under the hood without re-reading blob storage.

Payload Exchange: Larger binary and structured data (FMU files, DoE results, optimization outputs) is stored in and retrieved from Azure Blob Storage. Activities read from and write to blob storage using paths provided in metadata, ensuring data persists beyond function execution lifetime.

This approach provides several benefits:

- Activities don't need to know implementation details of other activities
- Large datasets don't traverse the orchestration layer
- Data persists beyond function lifetime
- There's clear separation between control flow (metadata) and data flow (blob storage).

6.2 Event-Driven Trigger Configuration

The pipeline trigger mechanism leverages Azure Event Grid, a fully managed event routing service that enables event-driven, reactive programming patterns. Event Grid provides reliable event delivery with built-in retry logic and dead-letter queue support, ensuring that no file upload events are lost even during system maintenance or scaling events. The service monitors the blob storage container

and automatically invokes the pipeline when files arrive from the Digital Twin system. To add guardrails when uploading files, we activated subject filter. When activated with specific file extension and blob path, only those caught files will be accepted and would trigger the event on EventGrid.

6.3 Activity Functions Specification

All activity functions are hosted under the function's app with the following deployment settings:

- Runtime: Python 3.11
- Hosting Plan: Premium Plan
- Region: West Europe
- Managed Identity: System-assigned managed identity for Azure resource authentication

6.4 Blob Storage Structure & Naming Conventions

As we use Azure Blob Storage as the central storage where we exchange inputs and outputs with the digital twin, an organized folder directory is needed to separate inputs from outputs and persist data over time with clear indications on the time of the executed simulation, and it is setup as below:

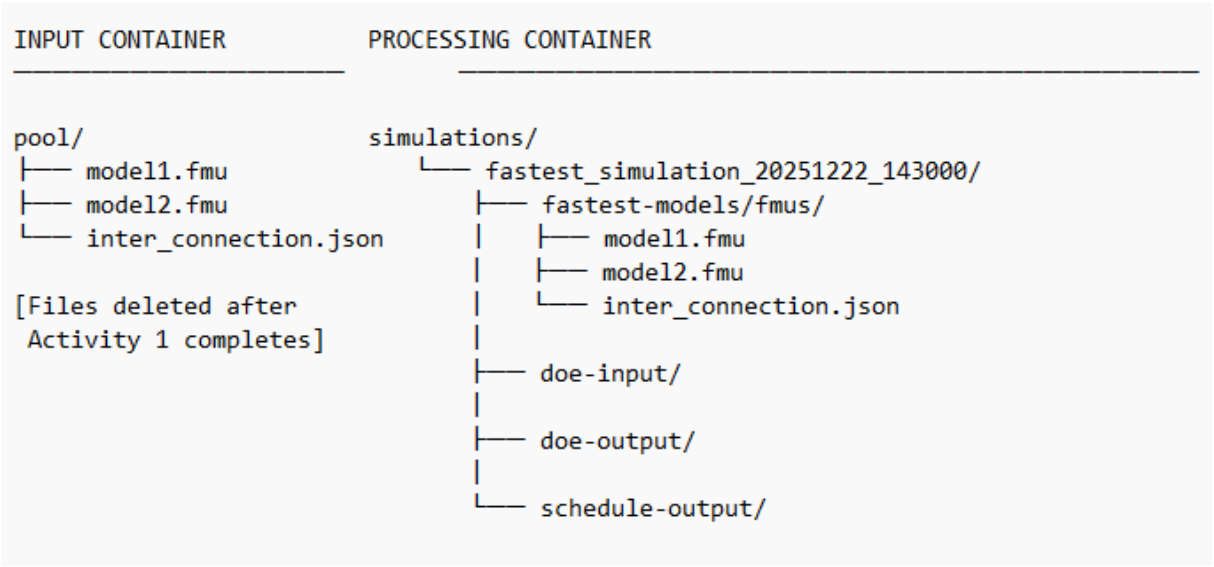

Figure 6 LIMS folder and file structure

The "pool" folder is designed to accept inputs from DT at any time for any kind of simulation. The EventGrid is setup to catch (.FMU) and inter_connection.json files. After being uploaded, the FileMonitor Azure function inside the LIMS pipeline will create a new simulation folder with a timestamp. Then subsequently, the first activity function in the LIMS pipeline called "Data Processor" will move them to

that newly created folder and delete the contents under "pool" folder to accept new files for the DT next upload

6.5 SQL Database and Schemas

As the database type used for this project is SQL based, the database tables scheme is a topic of importance because it determines how the data is stored and retrieved under the scope of this project. For this project, we utilize Azure SQL server with a database called "lims-platform". The database compute type is set to serverless with a 30GB storage and 1vCPU that is scalable upon requirements.

The database mainly is constructed of the following tables:

- UUTIDs
- UseCases
- TestRequests
- TestNames
- TestNameApplicability

UUTIDs:

- Purpose: A lookup table that includes the individual to-be tested battery IDs of types cell/module/pack. It dictates what test names will be available at the end of the form.
- Schema:

UUTIDs		
int	Id	PK (primary key)
nvchar	Code	UK (unique key)
nvchar	Descripition	
bit	IsActive	
Datetime2	CreatedAt	

UseCases:

- Purpose: A lookup table that includes the individual test use cases to be chosen by end user. It dictates what test names will be available at the end of the form.
- Schema:

UseCases

int	Id	PK (primary key)
nvchar	Name	UK (unique key)
nvchar	Descripition	
bit	IsActive	
Datetime2	CreatedAt	

RelevantStandards:

- Purpose: A lookup table that includes the individual test standards that the test shall follow. This dictates what test names are available at the end of the form.
- Schema:

RelevantStandards		
int	Id	PK (primary key)
nvchar	Code	UK (unique key)
nvchar	Name	
nvchar	Description	
bit	IsActive	
Datetime2	CreatedAt	

TestNames:

- Purpose: A lookup table that includes the actual test names, called test names, that will run in the virtual and physical test benches.
- Schema:

RelevantStandards		
int	Id	PK (primary key)
nvchar	Code	UK (unique key)
nvchar	Name	
nvchar	Description	
bit	IsActive	
Datetime2	CreatedAt	

TestNameApplicability:

- Purpose: A many-to-many junction table that dictates what test names accepts what UUTIDs, use case and standards.
- Schema:

TestingNameApplicability		
int	Id	PK (primary key)
int	TestNameId	FK (foreign key)
Int	StandardId	FK
int	UseCaseId	FK
int	UUTIDId	FK
bit	IsApplicable	FK
Datetime2	CreatedAt	

TestRequests:

- Purpose: A table that preserves the history of test requests in details.
- Schema:

TestRequests		
int	Id	PK (primary key)
nvchar	RequestNumber	UK (unique key)
nvchar	RequesterId	
Nvchar	RequesterName	
Datetime2	PreferredTestScheduleDate	
int	UUTIDId	FK (foreign key)
int	UseCaseId	FK
int	StandardId	FK
int	TestNameId	FK
int	Status	
nvchar	WorkflowId	
nvchar	SimulationFolder	
nvchar	FmuBlobPath	
nvchar	SimulationResultsPath	
int	SimulationProgress	
nvchar(MAX)	TestMetricsJson	
Datetime2	CreatedAt	

Datetime2	UpdatedAt	
Datetime2	FmuReadyAt	
Datetime2	DoeCompleteAt	
Datetime2	ScheduleReadyAt	
Datetime2	SimulationStartedAt	
Datetime2	SimulationCompleteAt	
Datetime2	CompletedAt	
Datetime2	FailedAt	

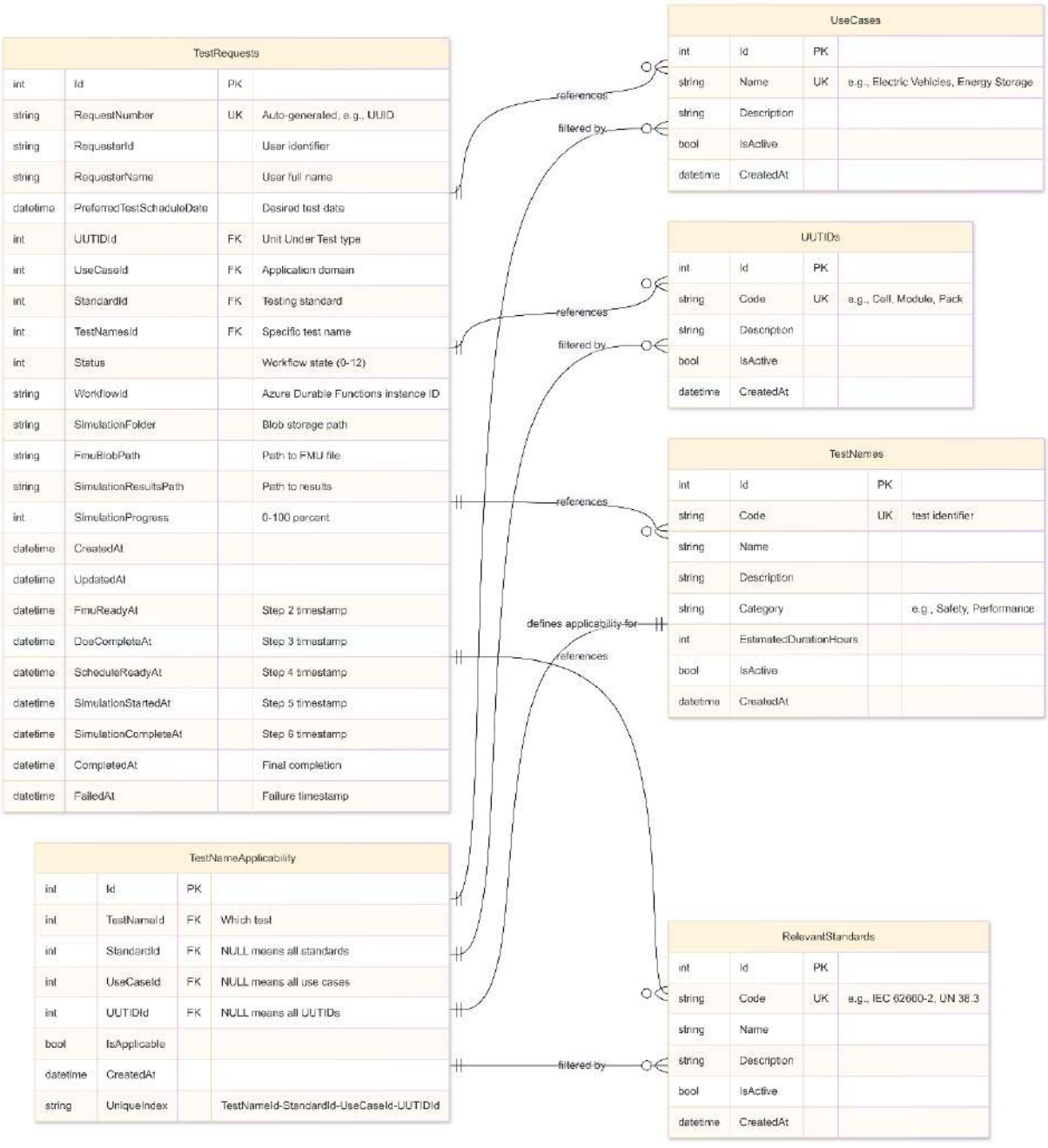

Figure 7 LIMS SQL database for test scheduling

The above figure is an ERD diagram describing the entities relationship between the database tables. Understanding the variables types, relations and intentions is a crucial part of creating an accurate common understanding between all partners.

7. Testing & Validation

The test with our partner DT included mainly two sessions that cover the contact points with them:

1- Model files upload from DT to LIMS
2- Test results sharing from LIMS to DT

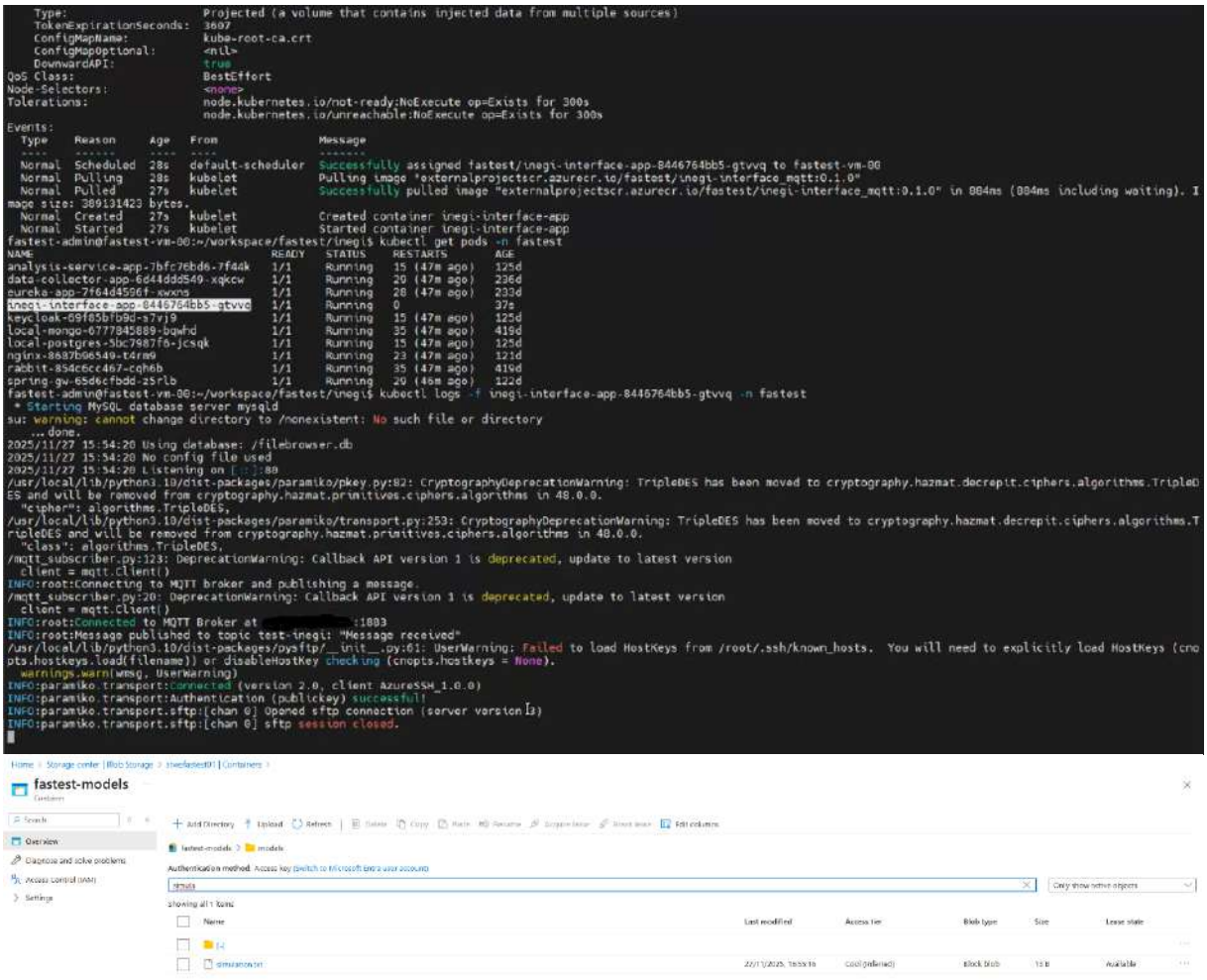

Figure 8 Test result LIMS accessing DT simulation files via sFTP

The first test included the case where the correct simulation model is chosen from DT in form of a (.FMU) file and is shared with LIMS. The means of communication used here is file upload to LIMS Azure Blob Storage through sFTP protocol. For security purposes, an Azure Private Endpoint is deployed at DT side, and it targets our own azure storage account for secure access. This way we enable private communication between different tenants/subscriptions on Azure. To upload a file from DT to LIMS storage account, a special user was created by LIMS alongside an SSH key. DT used these username/ssh key to connect and upload their model. As seen from the above screenshots, the file upload using sFTP was successfully implemented.

Regarding the second test, we were able to successfully exchange data via MQTT protocol with our partners at DT (COMAU). The test took place after whitelisting IPs from the partners into our infrastructure firewall. The core concept of this test is to establish a bridge-mode connection between our HiveMQ broker and the partners' RabbitMQ broker. Once bridge connection is setup with the correct parameters (remote broker's IP, authentication credentials, topic), messages flow from our co-simulation client to the DT through the established bridge as shown in the graph below.

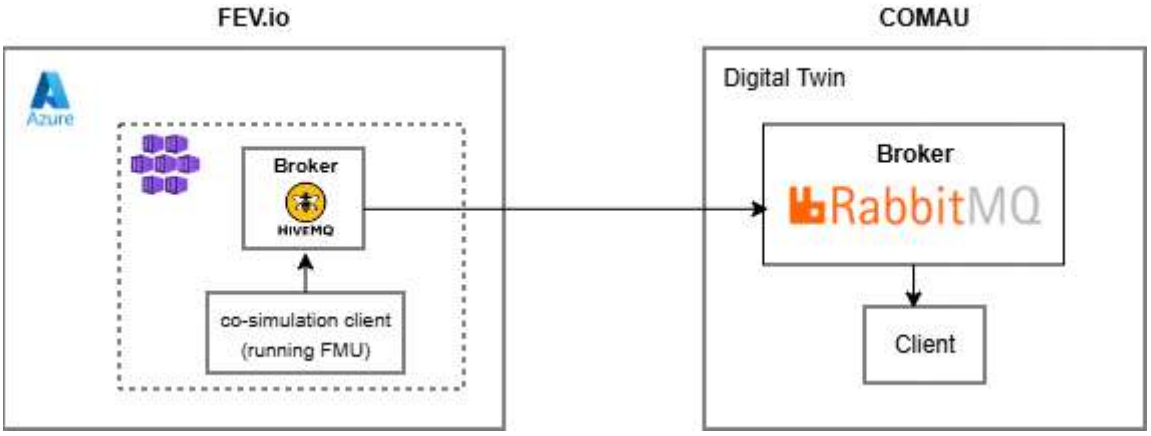

Figure 9 Setup of the connection test with DT

The reason why we don't initiate direct connection between clients is due to client security where their IPs are not exposed to public connections as the broker is published. Test success evidence is as below:

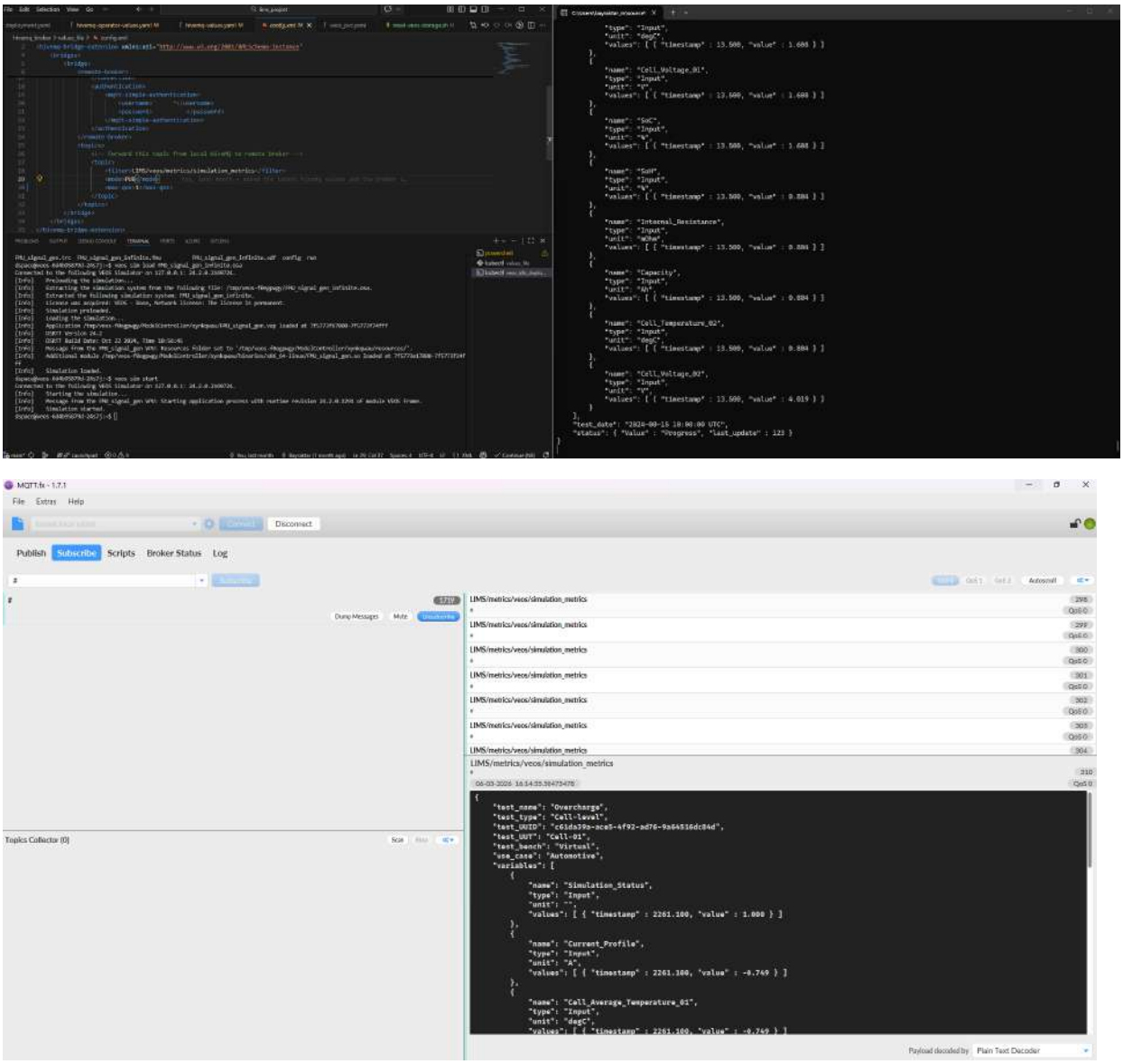

Figure 10 Test result LIMS forwarding test results to DT via MQTT

8. Conclusion

This deliverable established the technical integration concept and implementation for connecting the FASTEST LIMS with the DT to support hybrid battery test workflows, with LIMS acting as the orchestration layer for user-driven test requests, model retrieval, workflow triggering, and monitoring. Within the scope of T6.4, the work specifies the two primary DT–LIMS contact points: LIMS requesting and receiving the correct simulation model assets (FMU and interconnection description) and DT consuming simulation metrics from LIMS in a near real-time manner.

The orchestration workflow is described as a sequential pipeline (data processing/validation, DoE execution, and Gurobi-based schedule optimization) that propagates execution commands to virtual and physical benches via Azure queues and MQTT, while publishing results/metrics through the HiveMQ broker for consumption by DT and other subscribers.

Initial integration testing with DT partners validated the secure DT-to-LIMS model delivery mechanism, demonstrating successful upload of FMU model files into the LIMS storage account using the defined sFTP approach (including the private endpoint setup and SSH-key-based access). This provides a concrete proof point for one of the key DT–LIMS interfaces required for the wider hybrid workflow. The tests under the scope of this deliverable were successfully executed and required evidence is collected and documented. During the phase of these tests all necessary infrastructure updates including firewall whitelisting for MQTT test and cross-tenant Azure private endpoints deployment for the sFTP test.

9. BIBLIOGRAPHY

HiveMQ. (2024, 10 30). *www.hivemq.com*. Retrieved from www.hivemq.com: https://www.hivemq.com/solutions/the-best-mqtt-broker-for-azure/

Microsoft. (2024, 10 30). *App Service - Build and Host Web Pages*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/app-service

Microsoft. (2024, 10 30). *Application Gateway*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/application-gateway/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure Container Apps*. Retrieved from Microsoft Azure: https://azure.microsoft.com/de-de/products/container-apps/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure Functions*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/functions/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure SQL Database*. Retrieved from Microsoft Azure cloud: https://azure.microsoft.com/en-us/products/azure-sql/database/?msockid=333b0623c62160732cc912b7c75961f7#Features

Microsoft. (2024, 10 30). *Enable or disable SSH File Transfer Protocol (SFTP) support in Azure Blob Storage*. Retrieved from Microsoft Azure: https://learn.microsoft.com/en-us/azure/storage/blobs/secure-file-transfer-protocol-support-how-to?tabs=azure-portal

Sidna, J., Amine, B., Abdallah, N., & El Alami, H. (2020). Analysis and evaluation of communication Protocols for IoT Applications. *Proceedings of the 13th international conference on intelligent systems: theories and applications*, (pp. 1-6).