TRE-FX

Technical Documentation - Primary Implementation

UK Research and Innovation

HDRUK Health Data Research UK

ADRUK Data-driven change

- **Title**: TRE-FX Technical Documentation - Primary Implementation
- **Date**: 2024-01-18
- **Authors**: Tom Giles, Jonathan Couldridge, Sam Cox, Daniel Lea, Vasiliki Panagi, Simon Thompson, Philip Quinlan
- **Cite as**: https://doi.org/10.5281/zenodo.10376658
- **Abstract**: This report documents the primary TRE-FX implementation, which uses a microservice architecture, to be deployed with layers outside and inside the Trusted Research Environments.

Primary TRE-FX implementation

This project wanted to illustrate that each of the components in this Microservice architecture is interchangeable with equivalent components. The Primary TRE-FX implementation is the main implementation to be installed within the TREs as part of the TRE-FX project. It has been built to interact with mature TRE services. It has an API based architecture that consists of a number of microservices running on three separate VMs. Alongside the TRE Controller and Workflow Executor this deployment relies on several additional components, including a Keycloak server and a Intermediary store that serves as a place to put the crates for job requests, outputs of executions,

and final results crates.

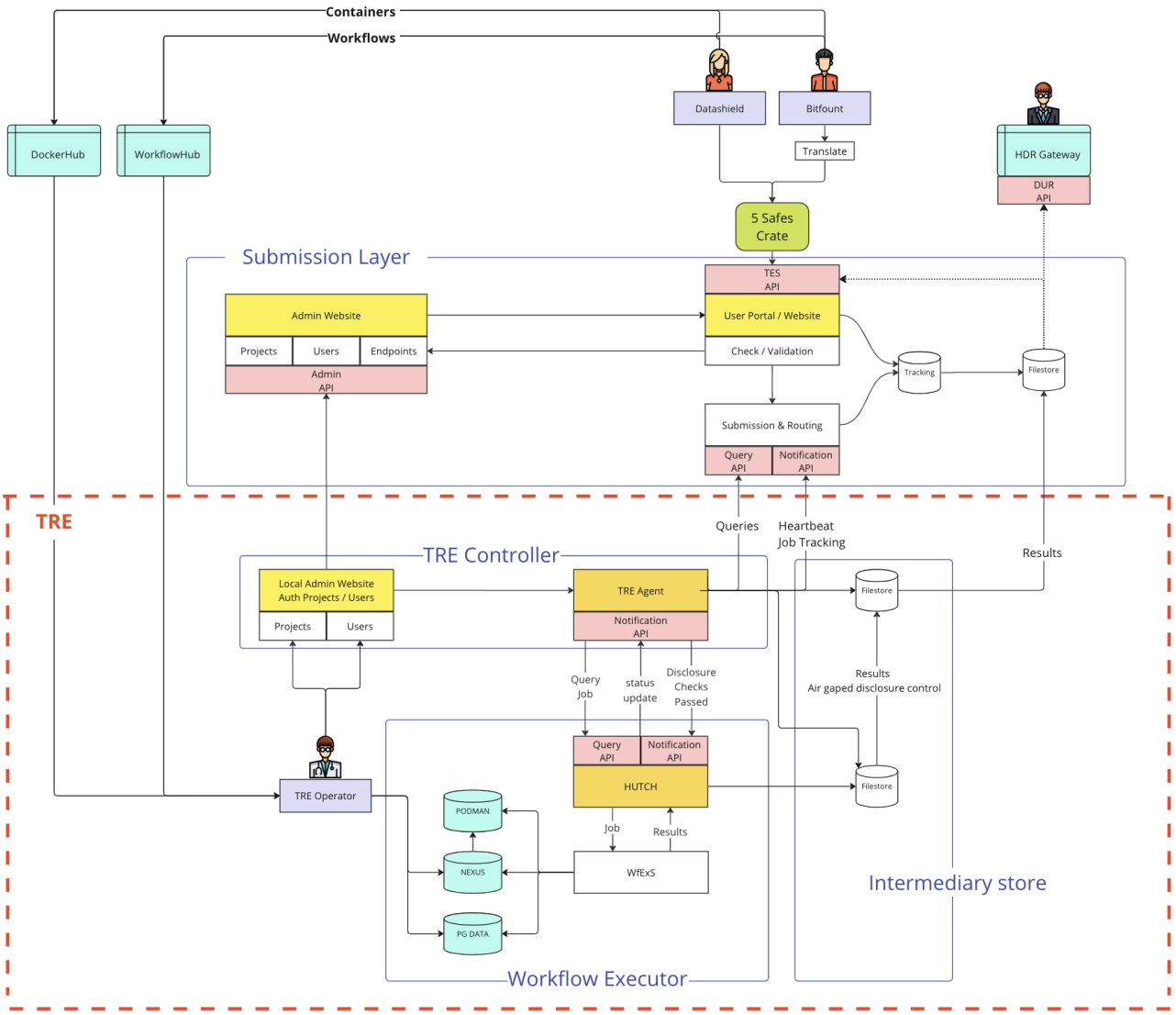

Figure 1: ***Diagram illustrating the Primary TRE-FX implementation. The Submission Layer, TRE controller and Workflow Executor are all VMs running sets of microservices, additional requirements include Keycloak and the Intermediary MINIO store.***

Components

Each of these components can either be contained within a single virtual machine per component or deployed as a number of separate sub services within a larger architecture model. The expectation is that in a productionised service other TRE services will be integrated with these components to ensure that appropriate governance conditions are met.

Submission Layer

The Submission Layer is external to the Trusted Research Environments (TREs). It serves as the initial point of contact for incoming analytical tasks. This layer is responsible for receiving user-submitted tasks, performing preliminary validations, and queuing them for further processing. By positioning the Submission Layer outside the TREs, the system ensures a level of abstraction that enhances security and task management.

In our primary implementation both power users and federated software vendors interact with the Submission layer via a User portal / Website and a GA4GH TES API. Before a user / vendor can submit a query to the GA4GH TES API they must first register for an authentication token in the GUI, this is then used to authenticate following transactions. All user credentials are stored in Keycloak for cross validation across the platform. An admin API is provided to allow the TREs to pre-validate whether a query should be ingressed based on user and project credentials.

When a TES message is submitted, with an embedded link to WorkflowHub (workflowhub.eu) to the Five Safes RO-Crate, and is stored in a MINIO instance and held for validation. Assuming the checks pass, the TES message is then staged in a queue ready for collection by the TREs.

A query API is provided to return the results of analyses from TREs. Upon receipt, the results packet (wrapped in a Five Safes RO-Crate) is stored in the MINIO instance and held for validation. Assuming the output checks pass, the results packet is made available for collection via the GA4GH TES API and the output Five Safes RO-Crate can be posted to the HDR Data Use Register for transparency.

The User portal / Website also provides users with a graphical representation of the jobs running across the federated architecture and displays job status updates. These are reported back to the User portal / Website by the TRE Controller via a notifications API.

TRE Controller

The TRE-Controller layer acts as an internal administrative and messaging service layer. Situated within the TREs (and composed of a *TRE-Agent* and *Local Admin Website* in the SAIL/SERP implementation), this controller handles task scheduling, data coordination, and messaging between the different layers and services. It provides a crucial link between the Submission Layer and the Workflow Execution Layer, ensuring that tasks are executed in the correct environment and that results are appropriately managed.

In our primary implementation the TRE-Controller employs various dockerised software modules. The use of Docker containers is mainly to ensure consistent deployment including dependencies, the containers can be ported to other engines like Podman or Singularity.

The TRE-UI is the user interface, offering an integrated view of ongoing tasks, logs, and system metrics. It also has a basic disclosure control egress service that can act as an air gap preventing the disclosure of unsafe analyses. The TRE-API orchestrates between the different layers and services, managing API calls and messaging between the submission layer, TRE Admin UI and the Workflow Executor (via RabbitMQ). NGINX serves as the gateway for both the Admin UI and API interfaces, offering a secure access point for TRE controllers and messaging .

PostgreSQL 14 is used for storing essential data like which users have been authorised to run analyses against which projects and task statuses and for both debugging and logging, SEQ is utilised.

A MINIO server is also provided as an intermediary storage layer for holding the Five Safes RO-Crates during ingress and egress.

To facilitate secure and authenticated interactions with the external Submission Layer, the TRE-Controller employs tokens from a Keycloak server for credential verification. These tokens validate not only the tasks but also the data transferred between layers.

Workflow Executor

The Workflow Execution Layer also resides within the TREs and is close to the data. This layer is dedicated to the actual computational work, executing algorithms and analytics tasks on the data. It fetches tasks from the queue managed by the TRE-Controller and applies the specified analytics workflows and enriches the Five Safes RO-Crate with the information necessary for it to be used for governance and disclosure assessment. Post-execution, it returns results to the TRE-Controller, which then manages the return of outputs accordingly. It must also be capable of posting status updates.

In our primary implementation Hutch (https://github.com/HDRUK/hutch) serves as an open-source "Workflow Executor". It is designed to interact with a Trusted Research Environment (TRE) Controller over HTTPS via APIs.

TRE Controllers can either post job requests to a Query API endpoint in Hutch or hutch can listen to an endpoint to receive jobs. For workflow retrieval, Hutch offers flexibility. It can source workflows from publicly accessible repositories such as WorkflowHub, or, in air-gapped (isolated) environments, can fetch approved workflows from a local HTTP source. Sonatype Nexus is also integrated into the Hutch system, serving dual roles as a local workflow store and a container registry. This enables efficient management and storage of both workflow files and container images, essential for seamless execution. Workflows are expected to be in the Workflow Profile RO-Crate format (a specific schema designed for workflow representation compliant with WorkflowHub).

The execution of workflows is managed by WfExS (https://github.com/inab/WfExS-backend), a workflow execution orchestrator. WfExS supports multiple container engines, such as Docker, Podman, and Singularity but in the case of air-gapped environments Podman is used to ensure compatibility and security.

Hutch communicates with the TRE-Controller using its REST API for various notifications. These include status updates during workflow execution and a notification when the data analysis is completed (RabbitMQ manages message queues). Post-analysis, Hutch retains all documents, results and Crates in a holding state until it receives an explicit approval for disclosure control. Once approved, Hutch moves the data to a designated directory suitable for data egress.

For intermediary storage needs, Hutch is flexible. It supports either the AWS S3 API (in this implementation a Minio server is provided) or a traditional mounted filesystem path. This allows it to adapt to different infrastructure setups. specific workflows may also require local databases for storing the data for analysis, in this implementation PostgreSQL is used. To facilitate secure and authenticated interactions with the TRE-Controller, Hutch also employs tokens from a Keycloak server for credential verification. These tokens validate not only the tasks but also the data transferred between layers.

Transparency layer

The Transparency Layer is an external entity to the Trusted Research Environments (TREs) and functions as a public interface for research oversight and auditability. Situated outside the secure boundaries of the TREs, this layer provides researchers and other stakeholders with visibility into executed analytical tasks and their corresponding

metadata. By offering this transparency, the system not only enhances accountability but also fosters confidence in the research ecosystem.

In this implementation an API on the HDR gateway has been developed to ingest the output Five Safes RO-Crates posted by the submission layer, store and display them.

RO-Crate Usage

The components detailed above use the Five Safes RO-Crate profile as outlined by TRE-FX: as a transport format to bundle metadata for the submission along with required input parameters and everything the implementation needs to know in order to perform the requested analysis. The components, per the Five Safes RO-Crate Profile, continually augment the crate data and metadata throughout the process, as means of documenting the submission's journey through the process as well as fulfilling the submission's request by providing analysis output. This includes: any checkpoints, automatic and manual, that have been passed; metadata surrounding users and tools that handle the crate; incorporating referenced files, such as the workflow definition; analysis outputs that have passed disclosure checks for egress; and publishing information such as the origin, licence and publish date of the outputs. The Five Safes RO-Crate profile helps describe this journey by specifying a series of *phases* (https://trefx.uk/5s-crate/0.4/#review-process), referenced throughout the below.

The RO-Crate flows through the system in the following order:

- Submission Layer (initial submission)
    - The RO-Crate is submitted to the Submission Layer (referenced in a TES payload).
    - The Submission is like a partial Five Safes RO-Crate, as that profile outlines several phases and, at the point of submission, not all of those phases have passed (by design), and therefore not all the final metadata is present (e.g. when compared to a Results form of the crate).
    - The Submission Layer performs initial validation and authorisation checks, before queuing the submission for target TRE endpoints.
    - In future, the checks that are done may be recorded in the crate's metadata - particularly those expected by the **Check Phase** (https://trefx.uk/5s-crate/0.4/#check-phase) - this is not currently implemented due to time constraints.
- TRE-Controller Layer (TRE ingress):
    - The submission is fetched by the TRE-Agent, which interacts with Keycloak and the Local Admin Website for authorisation checks internal to the TRE, and is then sent to the Workflow Executor Layer via the Intermediary Store.
    - In future, the checks that are done may be recorded in the crate's metadata - particularly those expected by the **Validation Phase** (https://trefx.uk/5s-crate/0.4/#validation-phase) and the **Sign-off Phase** (https://trefx.uk/5s-crate/0.4/#sign-off-phase) - this is not currently implemented due to time constraints and is planned to be recorded as part of the API interaction.
- Workflow Executor Layer (execution):
    - The RO-Crate arrives at Hutch, which validates that it has reached the expected phase (the **Sign-off Phase** and required prior phases should be completed)
    - Hutch internally re-validates the crate's checksums and structure, then fetches the referenced workflow per the **Workflow Retrieval Phase** (https://trefx.uk/5s-crate/0.4/#workflow-retrieval-phase)
        - note that per the spec this phase can occur immediately *before* or immediately *after* the **Sign-off Phase**; this implementation does it immediately *after*.
    - Hutch then triggers the underlying Workflow Execution Service (WfExS) to execute the retrieved workflow with the provided inputs.

    - WfExS executes the workflow and outputs the results locally in its execution environment
    - Upon completion of the local workflow execution, Hutch updates the RO-Crate locally per the **Workflow Execution Phase** (https://trefx.uk/5s-crate/0.4/#workflow-execution-phase)
    - Assuming successful execution, Hutch stores the outputs in the Intermediary Store for egress checks and sends information for egress checking back to the TRE-Agent
    - Note that while the RO-Crate is updated locally by Hutch as it goes along, this updated crate doesn't leave the Workflow Executor Layer environment at this time – The "raw" workflow outputs are what is shared for egress checking, not the in-progress crate
- TRE-Controller Layer (egress checks)
    - The workflow outputs are sent to the TRE-Agent, which in turn interacts with the Egress checking system
    - The results of egress checks are returned to the TRE-Agent, which in turn sends them on to Hutch
    - The RO-Crate is not currently updated at this point – this will be logged by the Disclosure phase.
- Workflow Executor Layer (bundling):
    - Hutch now updates the RO-Crate with all additional information that has come through the process, completing the **Disclosure Phase** (https://trefx.uk/5s-crate/0.4/#disclosure-phase) and the **Publishing Phase** (https://trefx.uk/5s-crate/0.4/#publishing-phase)
    - At this point, the RO-Crate is an augmented version of the original Submission crate that will now meet the profile's requirements for all the "phases".
    - Hutch uploads this "Results" crate to the Intermediary Store and returns to the TRE-Controller Layer
- TRE-Controller Layer (response)
    - The TRE-Controller fetches the Results form of the crate from the Intermediary Store.
    - The final RO-Crate is then returned to the user through the Submission Layer
    - The user is then able to perform the any of the steps outlined in the **Receiving Phase** (https://trefx.uk/5s-crate/0.4/#receiving-phase) as desirable.

Quickstart Guide for TREs

Within the TREs, both the TRE Controller and the Workflow Executor can be deployed as single virtual machines.

Base specification

1. The TRE Controller: Ubuntu 22.04 x64, 4 vCPU, 16 GB RAM, 30GB disk
2. Workflow Executor: Ubuntu 22.04 x64, 4 vCPU, 16GB RAM, 128GB disk.

Higher specification may be required for productionisation. No commercially licensed software is used, the entire stack is open source and available on github:

Submission layer and TRE Controller: https://github.com/SwanseaUniversityMedical/DARE-TREFX-Environment1

Workflow Executor: https://github.com/HDRUK/hutch

Deployment

The TRE controller can be Deployed entirely through Docker-compose using versioned built images. The Workflow Executor relies on tools that can not easily be deployed in a container and thus an ansible script is provided to facilitate rapid deployment, this configures ASP.NET Core 7 runtime, WfExS, Docker and HostFile Configuration for

Proxying Workflow Retrieval. The remaining Workflow Executor components are deployed via Docker-compose using versioned built images. To run the Hutch based Workflow Executor application stack from source, there are multiple components that need installing. They all communicate via TCP, so a data partner can put any combination of them on the same machine or different machines, provided they are able to talk to one another on the right ports over a network.

Networking

Only the TRE controller communicates with the outside world. Connectivity is outbound using REST API to a single submission endpoint running on port 443, web proxy (no auth) can be used to detach the resources.

The TRE controller vm provides a user admin web portal and API server and an Egress web portal. The expectation is that these resources will only accept local inbound connections, and that calls to these services are routed through a single NGNIX endpoint, configured for SSL (assuming appropriate certificates are provided). The default port for this API service is 8072.

The Workflow Executor vm the only component with inbound access is HUTCH. This tool listens via a single HTTPS NGNIX endpoint that can be configured to bind any unprivileged port above 1024 (By default Hutch is configured for API communication on HTTP port 5209 and HTTPS port 7239) full details of the HUTCH API can be found on the swagger https://hdruk.github.io/hutch/swagger.

Both the TRE controller and Workflow Executor host MINIO instances for data storage. These are accessed on ports 9000/9001.

Authentication Mechanisms and Configuration

Internally, the system employs OpenID Connect for authentication. A dedicated Keycloak server can be either deployed on-premises or an existing OpenID authentication server can be configured for use. Note that configuration of specific claim attributes must be properly documented. In the current deployment, an external test Keycloak server is in use. External authentication to the REST API of the submission layer is facilitated via OpenID Connect. An account for the TRE is established on the submission layer, and its credentials are registered within the TRE Agent. These credentials are subsequently utilised to request authentication tokens from an external authentication server, permitting API interactions.

Database credentials for each authorised project are temporarily housed within the TRE Agent. These credentials are dynamically conveyed as part of the message to HUTCH upon acceptance of a submission. Currently, these credentials are stored in encrypted form within the TRE Agent's database. Future iterations (version 2) are planned to integrate Hashicorp Vault as an alternative, more secure storage solution.

Processes model (from the perspective of a TRE)

The TRE Controller, Submission Layer, The Workflow Executor, MINIO and Keycloak all communicate through REST API calls, Job progression is driven by status. On each status update TRE Controller calls the Submission layer to pass status updates back to the query submitter. During data processing and egress, the Workflow Executor calls to the TRE Controller with the status updates which are then routed back to the Submission layer. This is illustrated in Figure 2 and detailed in Table 1:

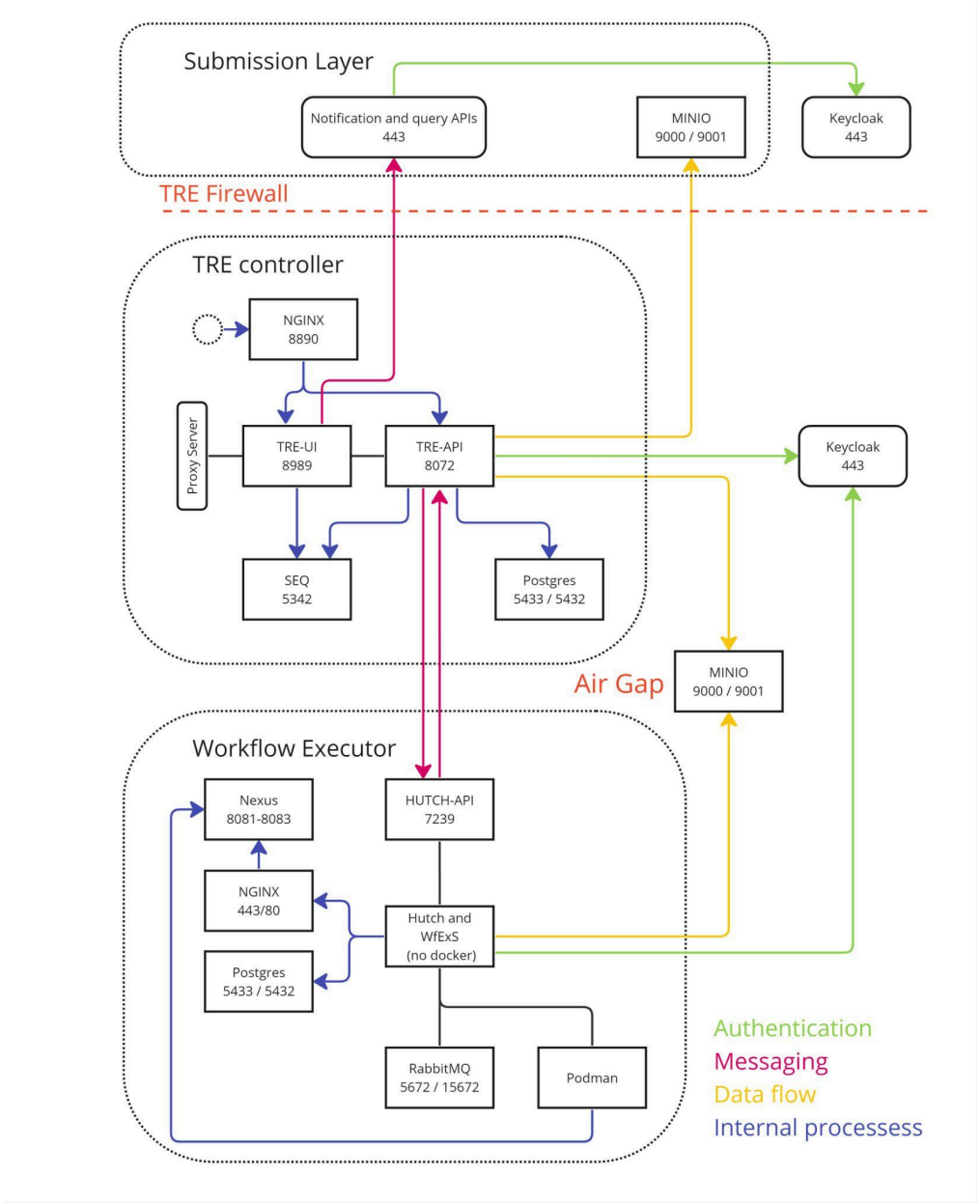

Figure 2: *Diagram illustrating the port transaction involved in TRE-FX implementation. All connections across the TRE boundary are "outbound only" to a single IP address; these API communications are managed by Keycloak authentication which can be either inside the TRE or as a separate entity.*

Table 1: Process flow of all API interaction that occur whilst processing a query:

From	To	Ports	Authentication	REST Type	Process
TRE Controller	Keycloak	443		GET	Retrieve a token for other API calls
TRE Controller	Submission Layer	443		GET	Retrieve a list of all users and projects
TRE Controller	Submission Layer	443		GET	Retrieve a list of Jobs for the TRE
TRE Controller	Submission Layer	443		GET	Pull down the input 5 Safes RO-CRATE
TRE Controller	Intermediary MINIO	9000/ 9001		PUT	Store the input 5 Safes RO-CRATE
TRE Controller	Workflow Executor	7239		PUT	Tell workflow executor that a Job ready to be processed
Workflow Executor	Intermediary MINIO	9000/ 9001		GET	Get the input 5 Safes RO-Crate
Workflow Executor	Intermediary MINIO	9000/ 9001		PUT	Store the results of the analysis and the output 5 Safes RO-Crate
Workflow Executor	TRE Controller	8072		PUT	Request output 5 Safes RO-Crate Egress checking
TRE Controller	Intermediary MINIO	9000/ 9001		GET	Check the output 5 Safes RO-Crate
TRE Controller	Workflow Executor	7239		PUT	Tell the workflow executor that the output 5 Safes RO-Crate has been checked
Workflow Executor	Intermediary MINIO	9000/ 9001		PUT	Move the output 5 Safes RO-Crate to per-Egress bucket
Workflow Executor	TRE Controller	8072		PUT	Tell the TRE controller that the output 5 Safes RO-Crate is ready for Egress
TRE Controller	Intermediary MINIO	9000/ 9001		PUT	Move the output 5 Safes RO-Crate to the Submission layer

Dependencies

TRE controller Software Components Deployed via Docker Compose:

- NGINX: Serves the Admin UI and API interface.
- TRE-API: Handles API calls and coordinates between components.
- TRE-UI: Provides the user interface.
- SEQ: For logging and debugging.
- PostgreSQL 14: Used for storing user and project information.
- Optional MINIO: Acts as an intermediary storage bucket.
- RabbitMQ: Responsible for message queuing to ensure asynchronous communication between workflow components.

Workflow Executor (Hutch+WfExS) Native Software Components:

These components are installed directly onto the VM and are initialised through the Ansible automation scripts.

- Hutch: Implemented on an ASP.NET Core 7 runtime, Hutch serves as the workflow agent passing jobs to WfExS and managing interactions with external components in the TRE-FX stack via APIs.
- WfExS: This Python 3.10-based component facilitates workflow execution.
- Docker: Utilised for containerization, Docker aids in isolating workflow tasks and ensures a consistent execution environment, as well as providing the “docker load” functionality for pre-caching approved container images in the environment
- HostFile Configuration for Proxying Workflow Retrieval: This configuration is imperative for redirecting and handling workflow retrieval requests, ensuring they are correctly sourced.
- Git
- graphviz

Workflow Executor (Hutch+WfExS) Software Components Deployed via Docker Compose:

These auxiliary components are containerized and deployed using Docker Compose, supplementing the native elements in specialised functionalities.

- RabbitMQ: Responsible for message queuing to ensure asynchronous communication between workflow components.
- Sonatype Nexus (Optional): This component can optionally act as a local repository for Workflow Crates and Container Images. An alternative source can be configured if necessary.
- Nginx (Optional but Required for Air Gapped Workflow Retrieval): Utilised to intercept and redirect remote workflow calls (e.g., from WorkflowHub) towards the local Nexus filestore or alternative locations, bypassing the need for public internet access.
- PostgreSQL 14 with OMOP CDM 5.3 Tables (Optional): This relational database, containing tables formatted according to the Observational Medical Outcomes Partnership Common Data Model (OMOP CDM) version 5.3, can provide a dataset that is accessible by workflows for data analytics or other operations.
- Optionally local air gapping services
- Optionally Adminer
- Optional MINIO: Acts as an intermediary storage bucket.