TRE-FX

Technical Documentation- Bitfount Implementation

UK Research and Innovation

HDRUK Health Data Research UK

ADRUK Data-driven change

- **Title**: TRE-FX Technical Documentation - Bitfount Implementation
- **Date**: 2024-01-11
- **Authors**: Blaise Thomson, Naaman Tammuz, Thomas Giles, Philip Quinlan, Carole Goble
- **Cite as**: https://doi.org/10.5281/zenodo.10376572
- **Abstract**: Bitfount is a platform for data collaboration with privacy-preservation features such as avoiding direct data sharing. This report describes how the Bitfount submission layer was modified to dispatch Five Safes RO-Crates, and Bitfount Pod, an open-source component of their stack, was modified to serve as the TRE-Controller and made capable of interacting with Hutch. Although an alternative implementation, Bitfount maintains interoperability with the primary TRE-FX submission layer.

Bitfount implementation

This project implements the TRE-FX architecture from the viewpoint of Bitfount.

We aimed to illustrate that each of the components in this Microservice architecture is interchangeable with equivalent components. Figure 1 below shows the communication diagram in the Bitfount integration. It is clear to see that almost all components at this level, as well as the structure of the components' communication are equivalent to those in the primary implementation.

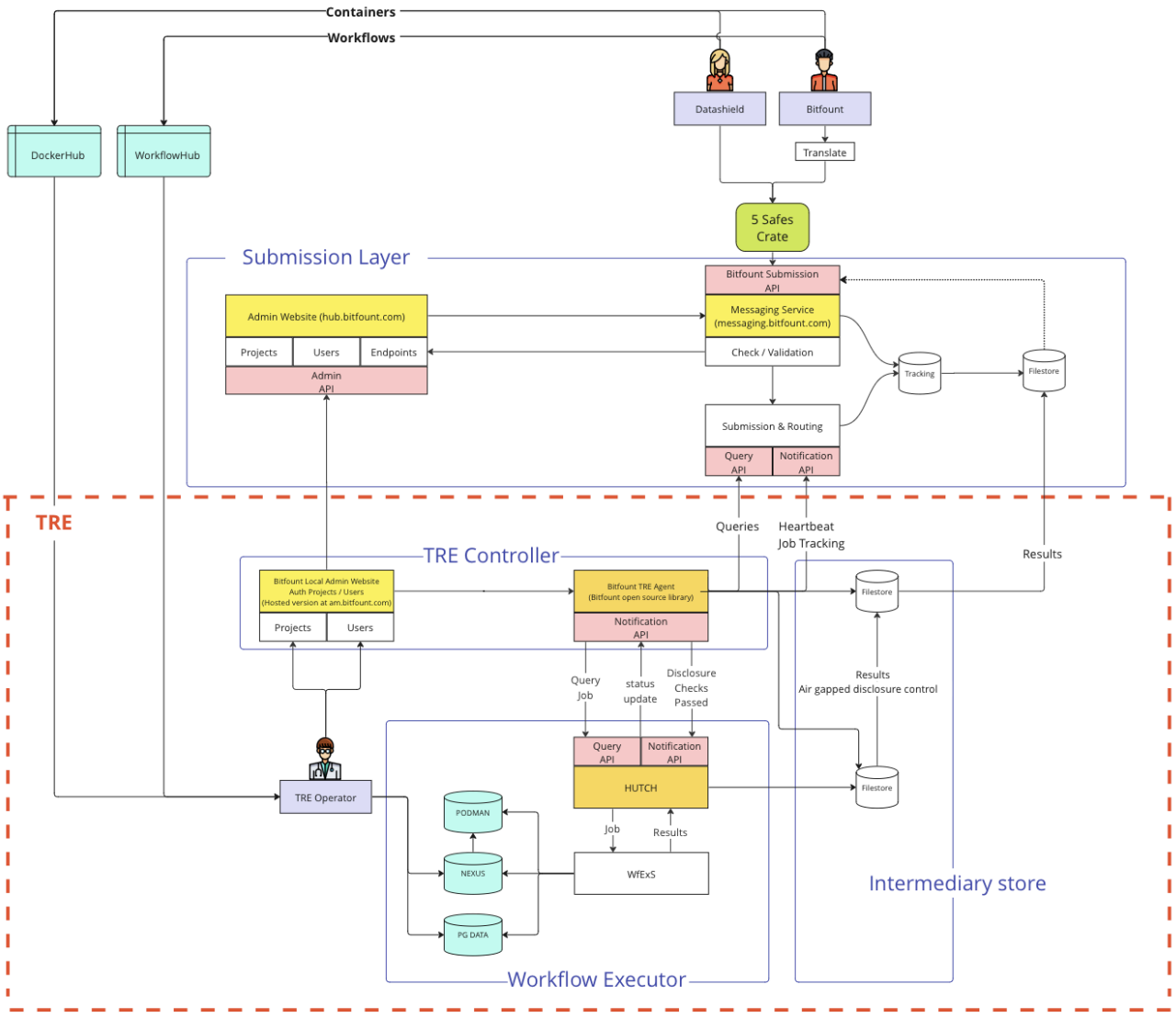

Figure 1:

Diagram illustrating the Bitfount implementation. The Submission Layer is a Bitfount managed service, with the Admin Website hosted at hub.bitfount.com and the job submission service hosted at messaging.bitfount.com. The TRE agent is equivalent to Bitfount's "Pod" concept and can run as a Docker image, Windows/Mac Desktop application or through python (pip install bitfount). The local admin website is Bitfount's "Access Manager" service, which can run within the TRE or is available as a managed service at am.bitfount.com. By default, authentication is managed by Auth0. Additional requirements include the Intermediary MINIO store.

Components

Submission Layer

In the Bitfount implementation, the Submission Layer consists of the managed services that Bitfount operates: hub.bitfount.com and messaging.bitfount.com. 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.

Interaction with the Submission layer is via the Bitfount Hub Website and Bitfount's GRPC-based Task submission API, for which there is also an open source python client. Before a user / vendor can submit a query they must first register for an account with the Submission layer. Authentication is managed by Auth0 and can proceed via any of:

- OpenID Connect (OIDC) Device Authorisation Flow (Default Authentication Method)
- OpenID Connect (OIDC) Authorisation Code Flow
- Security Assertion Markup Language (SAML)
- Private Key

When a Five Safes RO-Crate query is submitted to the task submission API it is validated within the service. Assuming the checks pass, it is then presented on a task queue ready for collection by the TREs.

Unlike in the primary implementation, on task acceptance, the Bitfount submission layer sets up communication channels for ongoing communication between TREs and the task submitter through the lifetime of a task. The primary implementation is currently restricted to a single message for task submission and then a single message for retrieving results. This ongoing communication enables the Bitfount platform to support use cases such as Secure Aggregation, Private Set Intersection and Federated Learning, which have all already been built into the platform.

For RO-Crate based tasks, the results packet (wrapped in a Five Safes RO-Crate) is stored in a MINIO instance and held for validation. Assuming the output checks pass, the results packet containing the Five Safes RO-Crate is returned to the task submitter.

The Submission Layer (Bitfount Hub) also provides users with a graphical representation of the tasks they have running or have run across the federated architecture and displays job status updates. An audit history of all historical tasks as well as access grants is also provided through the website.

TRE Controller

As in the primary implementation, the Bitfount implementation of the TRE Controller consists of two main services. The access manager service provides the access controlling system that decides whether tasks should be accepted. Bitfount's "Pod" service handles task scheduling, data coordination, and messaging between the different layers and services.

The Pod service polls the Submission Layer for new tasks to execute. Any new task request is sent to the Access Manager for two of the five safes checks:

- Does the requesting user have the appropriate permissions? (Safe User)
- Has the TRE joined to this project? (Safe Project)

Once approved, the task is forwarded to the Workflow Executor, which is run in a separate environment to ensure two more of the five safes are attained:

- The environment has very reduced permissions to minimize risks of data leakage (Safe Settings)
- Only appropriate data is made available within the environment (Safe Data)

After execution, the resulting RO-CRATE is then sent to an egress service for the final check on whether the data is approved for release, ensuring the final principle of the five safes:

- Only approved outputs are released from the TRE (Safe Outputs)

Workflow Executor

The Workflow Execution Layer in the Bitfount implementation is the same as the one used in the primary implementation. We use Hutch (https://github.com/HDRUK/hutch) and WfExS as an open-source "Workflow Executor", where Hutch runs as an HTTPS server, receives requests from the Pod via APIs, and forwards them on to WfExS for workflow execution.

RO-CRATE usage and data flow

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

- TRE-Controller Layer (initial submission):
  - The RO-Crate is submitted to the Bitfount Submission Layer.
  - The submission is like a partial 5 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 crate).
  - The submission is transferred to Bitfount's TRE-Agent, is sent to the Bitfount "Local Admin Website" for initial authorisation checks, and is then sent to the Workflow Executor Layer
  - In future, the checks that are done may be recorded in the crate's metadata, but this is not currently done.
- Workflow Executor Layer (execution):
  - The RO-CRATE arrives at "Hutch", which then sends it to the WfExS execution system.
  - The WfExS system runs the workflows and outputs the results locally in its execution environment
  - Hutch stores those outputs in Minio for egress checks and sends information for Egress checking back to the TRE-Controller Layer
  - The RO-Crate is updated locally by Hutch as it goes along but 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 original RO-CRATE and results information is returned to the Bitfount TRE-Agent, sent to the Egress checking system, returned to the Bitfount TRE-Agent and then sent back to Hutch for bundling
  - The RO-CRATE is not updated at this point
- Workflow Executor Layer (bundling):
  - The RO-CRATE is now updated with all additional information that has come through the process
  - This 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 the final crate to Minio and returns to the TRE-Controller Layer
- TRE-Controller Layer (response)
  - The final RO-CRATE is then returned to the user through the Submission Layer

Quickstart Guide for TREs

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

Base specification

- The TRE Controller = Ubuntu 22.04 x64, 4 cores / 8gb / 16gb disk
- Workflow Executor = Ubuntu 22.04 x64, 4 vCPU, 16GB RAM, 128GB disk.

Higher specification may be required for productionisation. The Bitfount Pod, Hutch and WfExS workflow system are all open source and available on github.

Deployment

The Bitfount Pod can be deployed using versioned built images, or run as a python service by first installing the open source `bitfount` library and using the `run_pod` command. The built images are docker images, so for this route docker must be installed in the TRE environment, and Docker-compose can be used to run the images. If utilising the `bitfount` library directly, then a suitable python environment must be configured. This can be done simply by creating a virtual python environment, and then using `pip` to install `bitfount`. For both the docker and `run_pod` approaches, the Bitfount Pod can be configured via a YAML file, which specifies all the necessary details of the Bitfount pod. This method ensures that Bitfount pods can be scaled up easily, and more Bitfount Pods can easily be added based on similar configurations.

Networking

Only the TRE controller communicates with the outside world. Connectivity is outbound using REST and GRPC APIs to hub.bitfount.com and messaging.bitfount.com, running on port 443. The Bitfount TRE agent (Pod) within the TRE controller operates on a zero-trust principle such that every request is authenticated, and requests are only received through polling by the agent in an outbound fashion.

Within 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.

The Workflow Executor hosts a MINIO instance for data storage, which is accessed on port 9001.

Authentication Mechanisms and Configuration

The Bitfount system supports authentication via OpenID Connect, SAML or public key. The Submission layer uses Auth0 for authentication and can be integrated with on-premises or OpenID authentication servers within each TRE. Every user or TRE has an account established on the submission layer, and the local admin server (Bitfount access manager) maintains its own checks on users' permissions.

Processes model (from the perspective of a TRE)

Unlike in the primary implementation, the TRE Agent and Messaging service communicate over GRPC (in order to benefit from protobuf structured messages). All other communication (between Submission layer, TRE Controller, the Workflow Executor, MINIO and Authentication services) are all through REST API calls.

Dependencies

TRE controller Software Components

- Bitfount TRE agent: 'Pod' service used for task scheduling, data coordination (with workflow executor) and messaging.
- Bitfount optional local access manager: implements access control checks for task acceptance. If not installed locally, a hosted version at am.bitfount.com can be used

Workflow Executor (Hutch+Wfexs) Native Software Components:

These components are as in the primary implementation, and 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.
- Podman: Utilised for containerization, Podman aids in isolating workflow tasks and ensures a consistent execution environment.
- HostFile Configuration for Proxying Workflow Retrieval: This configuration is imperative for redirecting and handling workflow retrieval requests, ensuring they are correctly sourced.
- Git2
- 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.
- Git 2
- Optionally local air gapping services
- Optionally Adminer
- Optional MINIO: Acts as an intermediary storage bucket.