

DARE UK

TRE-FX

Technical Documentation - DataSHIELD Implementation

- **Title:** TRE-FX Technical Documentation - DataSHIELD Implementation
- **Date:** 2024-01-11
- **Authors:** Stuart Wheeler, Thomas Giles, Philip Quinlan, Carole Goble
- **Cite as:** <https://doi.org/10.5281/zenodo.10375984>
- **Abstract:** DataSHIELD (<https://datashield.org>) provides a technological solution that can address many of the disclosure challenges in facilitating analysis by researchers and other health care professionals of individual level data. Although initially developed for work in the biomedical and social sciences, DataSHIELD can be used in any setting where microdata (data on individual subjects) must be analysed but cannot physically be shared with the research users. This report documents a TRE-FX implementation, which explored and integrated new analysis techniques (including DataSHIELD, which traditionally hasn't been deployed in TREs) and interaction protocols without using the common components of the other two implementations, and instead implemented alternative components. This demonstrates the use of RO-Crates with different implementation choices and that the framework implementation can be modified and reconfigured.

DataSHIELD Implementation

The DataSHIELD implementation for testing is a lightweight testing framework that replaces the submission layer and TRE agent. The source code for this implementation is available from GitHub (<https://github.com/trefx/tre-interconnect-prototype>).

The purpose of this implementation of a Federated Analysis support Framework, is to provide a framework in which architecture and implementation decisions can be prototyped and evaluated. It is anticipated that individual TREs will have different processes and expectations which can be examined using this framework. The base design and implementation uses a modern enterprise application construction frameworks. Such frameworks aim to support development, deployment and maintenance of microservices or serverless based applications. The software infrastructure chosen for this approach were Quarkus (<https://quarkus.io/>) and Kubernetes (<https://kubernetes.io/>). Both Quarkus and Kubernetes are open source and based on open standards, and possess commercial supported versions, if required.

The implementation is constructed of four virtual network regions which are isolated from each other, such that there is either little or no network communications between network regions. The four network regions are:

- “Home TRE” (demo-lab): is a network region containing a set of microservices and queues which support authoring and submission of requests for analysis and receiving and inspecting responses from the analysis.
- “Analysis TRE’s DMZ” (demo-dmz): is a network region containing a set of microservices and queues which connect to the internet and isolate the “Analysis TRE’s” from unwanted interactions.
- “Analysis TRE’s Invocation Management” (demo-sde): is a network region containing a set of microservices and queues which check requests for and responses from analysis, including potentially manual checks. The invocation of services which facilitate the right analysis if performed.
- “Analysis TRE’s Analysis” (demo-analysis-datashield): is a network region containing a set of microservices and queues which combine to ensure reliable execution of the analysis tasks.

In this investigation DataSHIELD forms an example of a type of analysis which could be requested using the system developed. DataSHIELD has some unusual characteristics for use in a TRE environment, DataSHIELD is traditionally used interactively and contains mechanisms to enforce that only non-disclosed results are provided. This means that in the right circumstances, manual inspection of results may not be required.

As a framework for prototyping and evaluation of design alternatives, the implementation requires to be easily modified and reconfigured. This is achieved using the support of microservices, and the ability for such microservices and groups of microservices to be shutdown, replaced and the replacements started cleanly. The microservice automatically rebinding permits continuation of the service. This approach is supported by 'podman', an open-source tool supported by Red Hat.

Deployment Process

The main system dependencies required for deployment of the "TRE Interconnect Prototype" are 'git' to obtain the system source, and 'podman' to manage the microservices which make up the system. These software dependencies can be deployed using the following instructions:

- git 2.41.0 (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>)
- podman 4.7.0 (<https://podman.io/docs/installation>)

To obtain the "TRE Interconnect Prototype" source issue the command
`git clone git@github.com:trefx/tre-interconnect-prototype.git`

To deploy the "TRE Interconnect Prototype" microservices change directory to the directory `tre-interconnect-prototype/kube` edit file `demo-config.yml` to set "lab_amqpbroker_host", "dmz_amqpbroker_host", "sde_amqpbroker_host" and "analysis_ds_amqpbroker_host" to the primary IP address of your host machine(s).

To setup the data volume for DataSHIELD analysis demo (analysis-data directory), issue the command:

```
tar xf analysis-data_2023-12-20.tar.gz
```

To start the deployments (Kubernetes pods) issue the commands:

```
podman kube play --configmap demo-config.yml demo-lab.yml
podman kube play --configmap demo-config.yml demo-dmz.yml
podman kube play --configmap demo-config.yml demo-sde.yml
podman kube play --configmap demo-config.yml demo-analysis-datashield.yml
```

The deployment can be monitored using the microservices (pods) logging output, this can be obtained using the commands:

```
podman pod logs -f demo-lab
podman pod logs -f demo-dmz
podman pod logs -f demo-sde
podman pod logs -f demo-analysis-datashield
```

and to stop and remove the deployments use the commands:

```
podman kube down demo-lab.yml
podman kube down demo-dmz.yml
podman kube down demo-sde.yml
podman kube down demo-analysis-datashield.yml
```

Deployment Structure

Host TRE network zone: this network zone contains microservices which supports the analyst initiating invocations and inspecting the invocation's (multiple) results. Access to the microservices within the Host TRE network zone is via a web interface for the Analysis (via the Internet) and by AMQP Queues for access to the microservices within other TREs' DMZ's (via the Internet).

Admin network zone: this network zone is assumed to exist in some form in all TREs, a secure location from which administrative action can be performed on other parts of this TRE. In this demo the Admin network zone is assumed to be accessible from the Internet.

DMZ network zone: this network zone contains microservices which isolate the other network zones within the TRE from the internet. The DMZ network zone does not initiate interactions with other network zones within the TRE.

Invocation management network zone: this network zone contains the microservices which are responsible for checking invocations (pulled from the DMZ network zone), both request and response. It is also responsible for initiating (and responding to the completion of) the right analysis subsystem within the Analysis network zone.

Analysis network zone: this network zone contains a number of sub-zones (Kubernetes Pods) containing microservices which manage particular forms of Analysis. The invocation requests are placed in a queue by a microservice in the Invocation management network zone, and invocation responses are placed in an additional queue to be collected by a microservice in the Invocation management network zone.

Data network zone: this network zone is the highly secure network zone in which the sensitive data is stored.

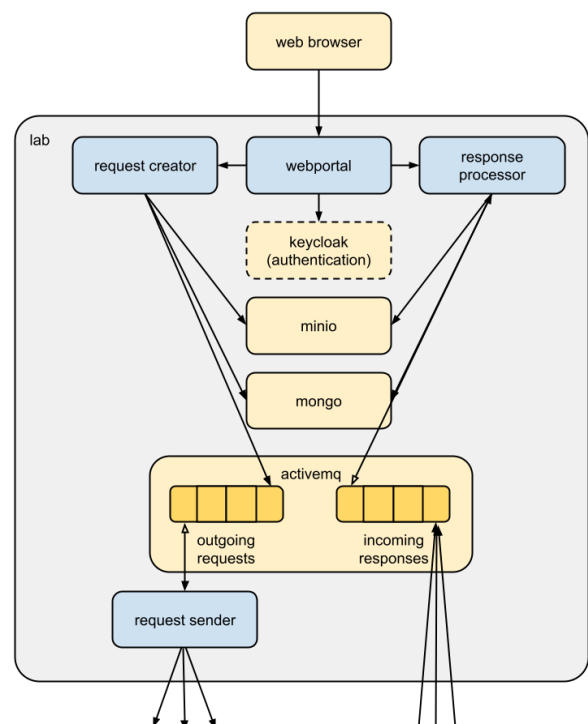
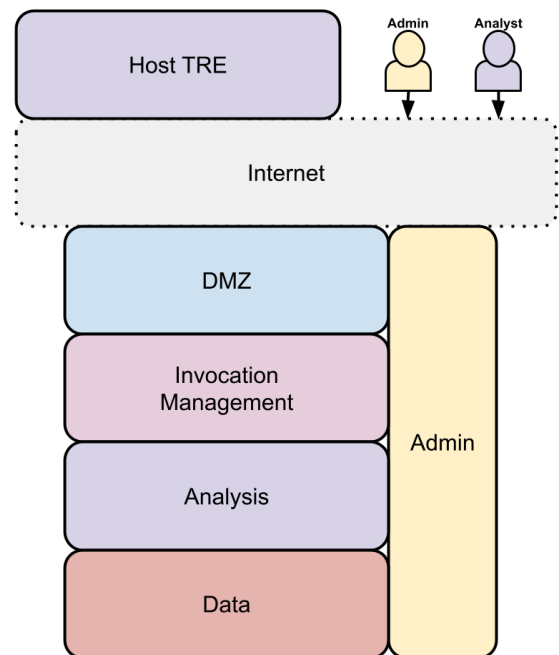
Admin person: the administrator is responsible for performing a number of configuration actions on the TRE's internal network zones, generally from the Admin network zone.

Analyst person: the analyst is interested in creating analysis requests and having them being sent to multiple TREs for evaluation, and monitoring the results.

Details of Host TRE network zone/region

request_creator: the purpose of the request creator microservice is to provide an API through which analysis requests can be created, in the form of an RO-Crate which represents the request for analysis to be performed. The created request RO-Crates, are placed in the 'outgoing requests' queue. Currently the request_creator supports three forms of requests: A hardwired test RO-Crates, templated RO-Crates which can be replaced by user values, and DataSHIELD specific request RO-Crate.

response_processor: the purpose of this microservice is to process RO-Crates which represent responses. This includes APIs for making the responses available to the analyst.



request_sender: the purpose of this microservice is to take request RO-Crates form the outgoing requests queue and forwards to the appropriate TRE.

devel: the purpose of this microservice is to provide APIs to services which assist development and demos. This includes initialization of databases and resetting database contents. This microservice wouldn't be deployed on a production deployment.

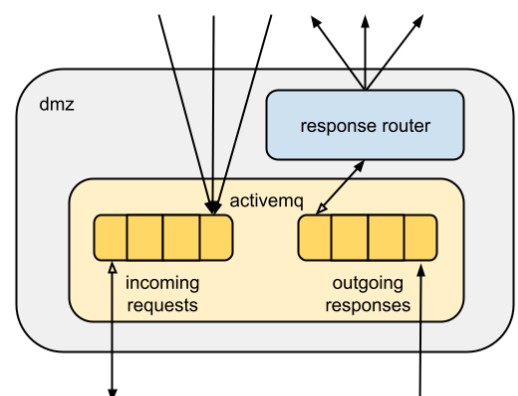
webportal: the purpose of this microservices is to respond to http(/s) requests for both the content of the webportal and the REST based interactions. The web portal is provided by delivering the static content generated by Angular 17.0 and Material 17.0. The webportal microservice also acts as a reverse proxy to the service provided by the other microservices within the Pod.

micro-services	intra-pod endpoints	external accessible endpoints
request_creator	7000 (http)	:8080/service/request_creator/... (http)
response_processor	7010 (http)	:8080/service/reponse_processor/... (http)
request_sender	7020 (http)	-
devel	7090 (http)	-
webportal	80 (http)	:8080/... (http)
minio	9000, 9001	-
mongo	27017	-
activemq	5672 (amqp)	5772 (amqp)

queues	server
outgoing_requests	activemq
incoming_responses	activemq

Details of DMZ network zone/region

response_router: the purpose of this microservice is to take outgoing response RO-Crates form the outgoing response queue and forwards them to the appropriate TRE, based on the content of the response RO-Crate.



micro-services	intra-pod endpoints	external accessible endpoints
response_router	7100 (http)	-

queues	server
incoming_requests	activemq

activemq

5672 (amqp)

5872 (amqp)

outgoing_responses

activemq

Details of Invocation Management network zone/region

request_receiver: the purpose of this microservice is to obtain RO-Crates from the DMZ's incoming_requests queue and place them in the unchecked_requests queue. This microservice can perform structural and syntactic checks on RO-Crates which all potentially valid requests should pass.

request_checker: the purpose of this microservice is to perform the automatic checks on the RO-Crates required, obtained from the unchecked_requests queue, to support the Five Safes. The checks are performed by a series of objects which can augment the RO-Crate with additional provenance information. The checking methods can also indicate if the request requires manual checking. Manual checking will be performed if any checker indicates a manual check should be performed, or if no checker expresses an opinion. If the request RO-Crate is to be manually checked to be stored in the appropriate place with the Minio database, alternatively if no manual check is required the request RO-Crate is placed in the checked_requests queue.

analysis_dispatcher: the purpose of this microservice is to obtain a checked request RO-Crate (from the checked requests queue) and inspect the contains to determine which analysis module needs to pass the request and where the response will be placed. This response will then be placed in the unchecked response queue.

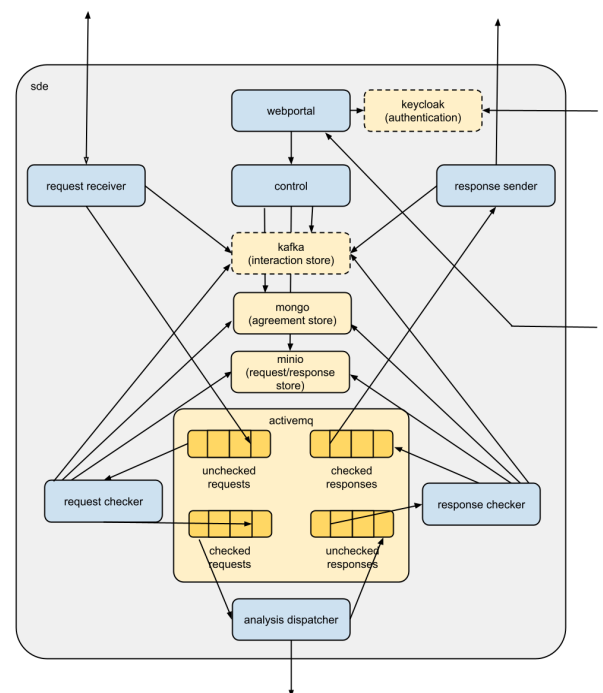
response_checker: the purpose of this microservice is to perform the automatic checks on the RO-Crates response, obtained from the unchecked_responses queue, to support the Five Safes. The checks are performed by a series of objects which can augment the RO-Crate with additional provenance information. The checking methods can also indicate if the response requires manual checking. Manual checking will be performed if any checker indicates a manual check should be performed, or if no checker expresses an opinion. If the response RO-Crate is to be manually checked to be stored in the appropriate place with the Minio database, alternatively if no manual check is required the response RO-Crate is placed in the checked_response queue.

response_sender: the purpose of this microservice is to transfer checked responses to the TRE's DMZ outgoing_responses queue.

control: one purpose of this microservice is to permit the processing of requests and responses which require manual checks.

devel: the purpose of this microservice is to provide service which assist development and demos. This includes initialization of databases and resetting database contents. This microservice wouldn't be deployed on a production deployment.

webportal: the purpose of this microservices is to respond to http(s) requests for both the content of the webportal and the REST based interactions. The web portal is provided by delivering the static content generated



by Angular 17.0 and Material 17.0. The webportal microservice also acts as a reverse proxy to the service provided by the other microservices within the Pod.

micro-services	intra-pod endpoints	external accessible endpoints
request_receiver	7000 (http)	-
request_checker	7010 (http)	-
analysis_dispatcher	7020 (http)	-
response_checker	7030 (http)	-
response_sender	7040 (http)	-
control	7050 (http)	-
<i>devel</i>	<i>7090 (http)</i>	-
webportal	80 (http)	:8090/... (http)
minio	9000, 9001	-
mongo	27017	-
activemq	5672 (amqp)	-

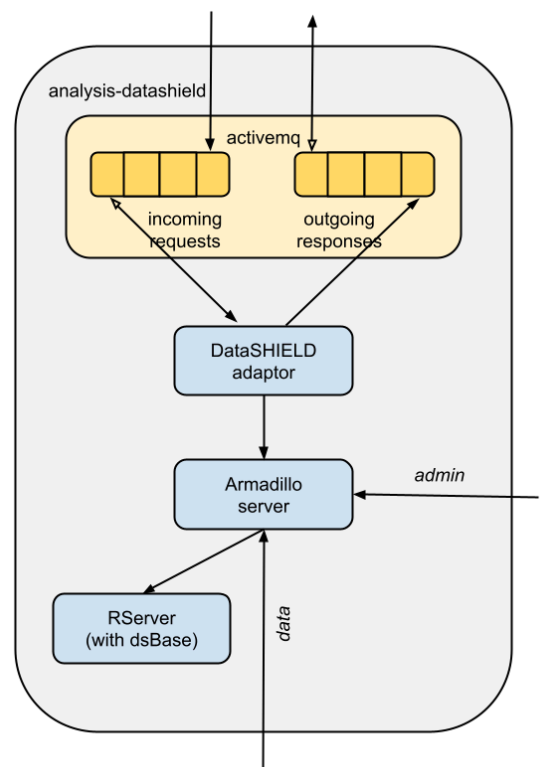
queues	server
unchecked_requests	activemq
checked_requests	activemq
unchecked_responses	activemq
checked_responses	activemq

Details of Analysis network zone/region

analysis-datashield-adaptor: the purpose of this microservices is to check requests which appear in the incoming_requests queue. These checks include checking that the required information has been provided to perform a DataSHIELD analysis. If these checks are passed, the microservice will perform a series of REST calls on the Armadillo server to cause the performance of a DataSHIELD operation. The results(/outcome) of the DataSHIELD analysis will be placed in the RO-Crate which is added to the outgoing_response queue.

armadillo: this microservice is an Armadillo server implemented by the Molgenis team in University Medical Center Groningen, Netherlands, as part of the EUCAN-Connect and ELIXIR projects. The APIs provided by the Armadillo server can be used to define the context in which the DataSHIELD analysis is performed. A WebPortal is also provided by the Armadillo server so Admin can perform configurations from the Admin network zone, if required.

trefx-rserver: this microservice is an R server (Rock) within which the DataSHIELD methods are performed. The R server can be



preloaded with “profiles” which contain a set of analysis packages, for example, Omics, Survival, Mediation, Mice, ... analysis packages. The RO-Create which represents a DataSHIELD analysis request contains the context and details of the analysis required. This information includes the type of DataSHIELD server to be used, Armadillo or Opal, which profile should be used, what data should be available to the server and the R methods to perform the analysis. All the details in the RO-Create about the request can be checked by the Invocation Management sub-system, and the Analysis sub-system.

micro-services	intra-pod endpoints	external accessible endpoints	queues	server
analysis-datashield-adaptor	7000 (http)	-	incoming_requests	activemq
armadillo	8080 (http)	-	outgoing_responses	activemq
trefx-rserver	6311	-		
activemq	5672 (amqp)	6072 (amqp)		

RO-Crate LifeCycle

One of the main purposes of this implementation was to provide a framework for evaluating different approaches to RO-Crate processing, in particular prototyping approaches to enforcing the Five Safe framework’s principles, in an adaptable manner. The experimental nature of this implementation means its processing model for RO-Crates doesn’t match that of the Primary Implementation.

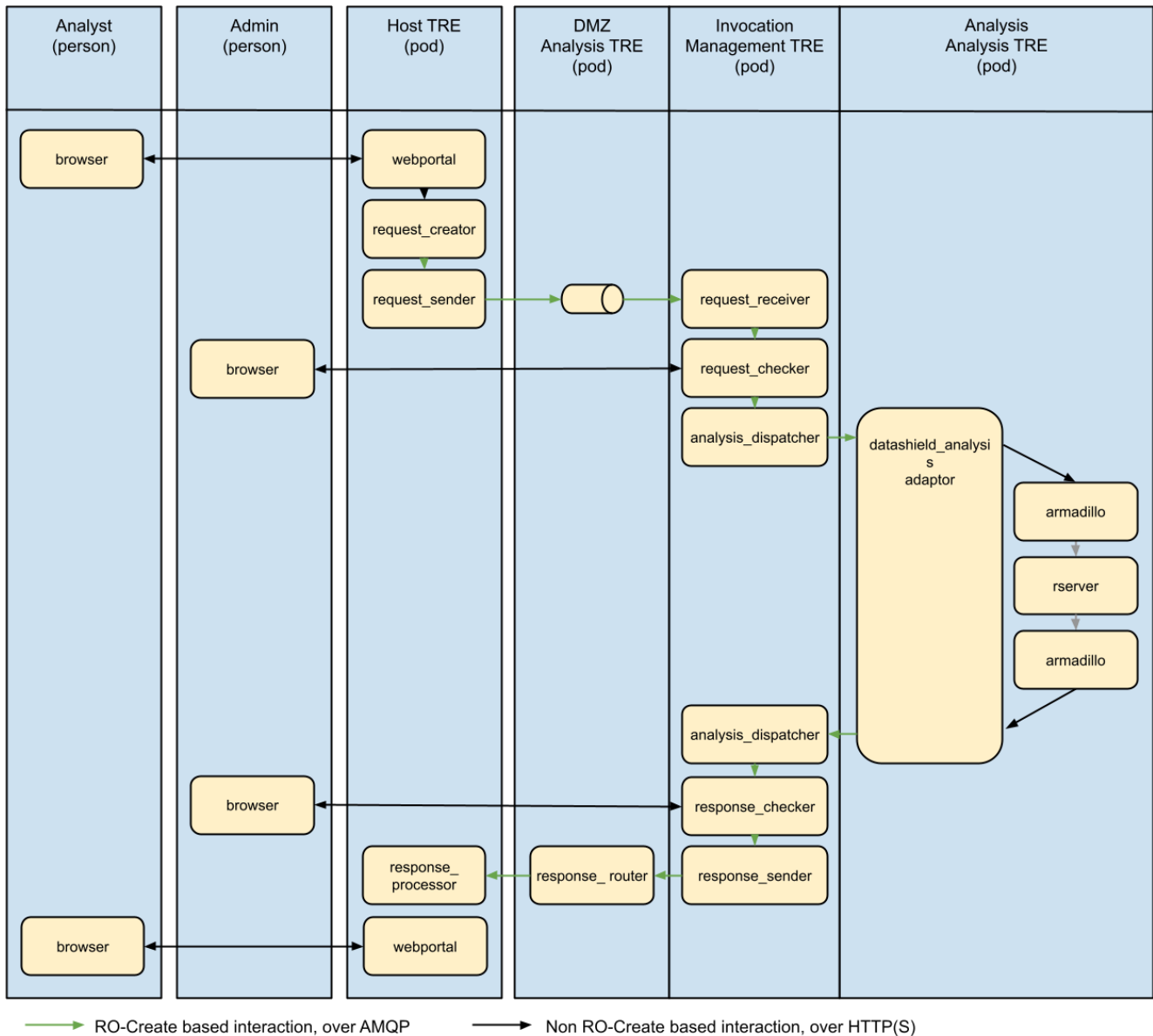
The general architecture of this implementation’s framework is that RO-Crate processing is based on executing a pipeline of services, which perform the required checks and augmentations of the RO-Crates representing analysis requests and responses. The RO-Crate contains all the state and history of the request/responses and its position in the pipeline the progress made on fulfilling the request. The checking which can be performed by a service on the RO-Crate will be constrained by the information available to the service, and requires careful consideration in a TREs environment. For example, checking if a request’s creator is permitted access to certain data can only be checked by a service which has access to the data access agreements entered into by the TRE holding the sensitive data. This pipeline will span multiple TREs, and will need to be able to tolerate delays and failures. The pipeline is implemented using durable queues (or sometimes shared databases, forming a Set rather than a Queue). The individual service in the pipeline will be governed by the TRE which provides that part of the pipeline, so permitting the TRE to choose appropriate implementation of services which match to their policies. It also permits extra services to be added to the pipeline to match any unusual policies of a TRE.

This implementation structure was chosen to reflect the autonomy and subsidiarity which TREs wish to maintain, but still permitting mutual collaboration when desired. TRE need to agree on the structure and meaning of the contents of a RO-Crate, but each TRE can specify what they regard as a request which they are willing to process.

The services which perform checks on the RO-Crate contents are at liberty to augment the RO-Crate with reasons why the checks were passed or failed, so building an provenance chain which can be examined manually or automatically if required. The provenance chain can be passed to the originating analyst, in part or in full, which could be invaluable to an analyst to determine the cause of an analysis request failing.

The overall structure of the pipeline consists of services from the TREs which the request/response passes through, grouped by network zone within the TRE. An analysis request will be created in the analyst’s home TRE, it is then passed to the DMZ network zones of TREs which have the sensitive data required for the analysis. Then the

analysis TRE's invocation management network zone, for further more detailed checking on permissibility of the request. If the request is found permissible, it is passed to the analysis network zone for performing the analysis. The resulting RO-Crate from the analysis, referred to as the response, this then checked by the invocation management network zone, against the to ensure features such as non-disclosiveness. The response is then passed via the TRE's DMZ to the analyst's home TRE, where it can be examined by the analyst.



In the current prototype the services which make up the pipeline have been implemented to perform particular tasks to enforce the Five Safe framework's principles. If any of the checks (including manual checks) on the request are not successful, a flag will be added to the request RO-Crate to prevent the analysis being performed. Also if any of the checks on the response RO-Crate are not successful, any results within the response RO-Crate will be removed.

The current planned service checks are:

- Invocation Management network zone
 - *request_receiver service*: intended to check generic request RO_Crate structure and content.

- *request_checker service*: intended to check request RO_Crate's contents against the data access agreements held within the invocation management, and if required, manual checking.
- *analysis_dispatcher service*: check requested analysis is available, and can be performed
- *response_checker service*: intended to check response RO_Crate's contents against the data access agreements held within the invocation management, and if required, manual checking
- *request_sender service*: intended to check generic response RO_Crate structure and content.
- Analysis network zone
 - *analysis-datashield-adaptor service*: ensure requested DataSHIELD analysis is available, and has been completed successfully.
 - *armadillo service*: ensure analysis is completely specified.
 - *trefx-rserver service*: utilise DataSHIELD checks to ensure analysis results in response are non-disclosive.

The *request_checker* and *response_checker* services are implemented using Contexts and Dependency Injection (CDI) to obtain a list of objects which represent "checker objects". This results in a checker interface which can be implemented and easily packaged into a plugin which can be deployed to the services. The SDE webportal permits the enabling/disabling of individual "checker objects", so allowing the TRE's administrator to customise the checks, at runtime, which their TRE performs.

Evaluation of DataSHIELD implementation

The architecture approach of having the analysis invocation pipe-line/path being based on RO-Crates be passed between microservices using reliable synchronous messaging proved to be flexible and robust. This architecture was able to show that it was possible to:

- Support the replacement (or addition) of microservice to cause the system to conform to the requirements of individual TREs.
- That restriction on initiation of network interactions could be enforced, increasing the security of the operations of the TREs.
- To provide tests on RO-Crates contents so as to confirm the requested analysis conformed to the Five Safes.
- A pattern to support the addition of new analysis techniques.
- Alternative interaction protocols could be provided between TREs, if required.
- Support for automated test of request and response content.
- Integration of manual checking (both request and response) into the analysis invocation pipe-line/path
 - Indication of automated checked performed (contained in RO-Crate)
 - Reports about of previous analysis requests and responses (contained in Apache Kafka)
 - Access to access agreements for manual checkers
- That the highly unpredictable duration of analysis operation and manual checking was not a problem.
- The ease by which microservices could be developed and modified.

It was observed, but not investigated in detail:

- Integration with SACRO
- Light-weight configuration to support nano-TREs, micro-TREs and mini-TREs could be developed.
- Supporting Dynamic Data Sharing Agreements between organisations.
- Dynamic, on demand, deployment of analysis techniques.
- Nodes in the tree being TREs, and edges representing sub-invocations between TREs, which honour the mutual governance constraints between TREs involved. This will potentially facilitate the Root TREs having access to analysis via an intermediate TRE.