H2020 – ICT-13-2018-2019

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D3.2 Architecture Design – Final Version

May 20

Imprint

Contractual Date of Delivery to the EC: **31 May 2020**

Author(s): **Mark Purcell (IBM), Mathieu Sinn (IBM), Marco Simioni (IBM), Stefano Braghin (IBM), Minh Ngoc Tran (IBM)**
Participant(s): **TREE, IMP; ENG, UC3M; IDSA**
Reviewer(s): **Joao Correia (B3D), Susanna Bonura (ENG)**

Project: **Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)**

Work package: **WP3**
Dissemination level: Public
Version: **1.0**

Contact: **mathsinn@ie.ibm.com**
Website: **www.MUSKETEER.eu**

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

This deliverable (D3.2 "Architecture Design – Final Version") is a document describing the architecture for the MUSKETEER centralized server platform. It is the culmination of task T3.1 and builds upon the initial architecture document D3.1, providing architecture/design updates as well as reporting progress in relation to the platform requirements.

Document History

Version	Date	Status	Author	Comment
1	01 April 2020	Outline draft	Mathieu Sinn	First draft
2	30 April 2020	First Draft	Mark Purcell	Section 1, 3
3	06 May 2020	Revised Draft	Mark Purcell	Section 2, 5
4	07 May 2020	Revised Draft	Marco Simioni	Section 4
5	07 May 2020	Revised Draft	Mark Purcell	Section 6
6	08 May 2020	Ready for Review	Mark Purcell	Ready for reviewers
7	21 May 2020	Incorporate review	Mark Purcell	Ready
8	22 May 2020	Final version	Mark Purcell	Ready
9	25 May 2020	Clean and submission	Gal Weiss	Final

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Abbreviation	Definition
AMQP(S)	Advanced Message Queuing Protocol (secure)
API	Application Programming Interface
COS	Cloud Object Storage
CWE	Common Weakness Enumeration
FaaS	Functions-as-a-Service
GQM	Goal/Question/Metric
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
OS	Operating System
POM	Privacy Operation Mode
RAM	Random-Access Memory
REST	Representational State Transfer
SPbD	Security and Privacy by Design
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSO	Single Sign On
TFIDF	Term Frequency – Inverse Document Frequency
TLS	Transport Layer Security
URL	Uniform Resource Locator
vCPU	Virtual Central Processing Unit
WP	Work Package
YAML	Yet Another Markup Language

1 Introduction

1.1 Purpose

The purpose of this document is to describe the MUSKETEER centralized server platform, which enables participants of the data economy to participate in Federated Machine Learning (ML) and thereby realize the value of their data assets, while preventing the leakage of information that is proprietary, confidential, personally sensitive, or that must not be shared because of other legal or regulatory requirements.

This document is the description of the second deliverable (D3.2) of work package 3 (WP3). The deliverable describes the final version of the architecture for the platform provided by WP3. Functionally, this platform provides the infrastructure and implements the services that are required to enable the federated ML algorithms developed in WP4 and WP5 in end-to-end applications. It must also support the assessments to be carried out in WP6 and provide interfaces which allow for the development of client connectors and end-to-end demonstration of the industrial use cases in WP7.

This document is an update to the first deliverable document D3.1 for WP3, which describes the initial version of the architecture. As such, if the underlying information regarding system components has not changed since D3.1, these components will not be discussed again. However, any enhancements or new features will be discussed in this document.

1.2 Related documents

This deliverable is related to the following documents (also see Figure 1):

- **D3.1 Architecture Design – Initial Version** – the precursor to this document, detailing the architecture as of M12.
- **D2.1 Industrial and technical requirements** – in so far as the platform architecture has to address functional and non-functional technical requirements described in that document.
- **D2.2 Legal requirements and implementation guidelines** – in so far as the design of the platform architecture should follow the implementation guidelines arising in the context of the applicable legal and ethical framework.
- **D2.3 Key performance indicators selection and definition** – in so far as the platform has to either provide the core capabilities that other functional components (e.g. the algorithmic library or the client connectors) require to meet their goals, or to meet specific goals itself.

- **D4.1 Investigative overview of targeted architecture and algorithms** – in so far as the platform has to provide the core capabilities to support and enable the targeted architecture and algorithms.
- **D4.2 Pre-processing, normalization, data alignment and data value estimation algorithms (initial version)** – in so far as the platform has to provide the core capabilities to support the deployment of the proposed algorithms.
- **D5.1 Threat analysis for federated machine learning algorithms** – in so far as the platform has to provide the core capabilities to support the deployment of the proposed algorithms.
- **D6.1 Assessment framework design and specification** – in so far as the platform has to provide the core capabilities to support the application of the proposed framework and meet relevant key performance indicators (KPIs).
- **D7.1. - Client connectors' architecture design (initial version)** – in so far as the platform has to provide the core capabilities to support the development and deployment of the proposed client connectors' architecture.

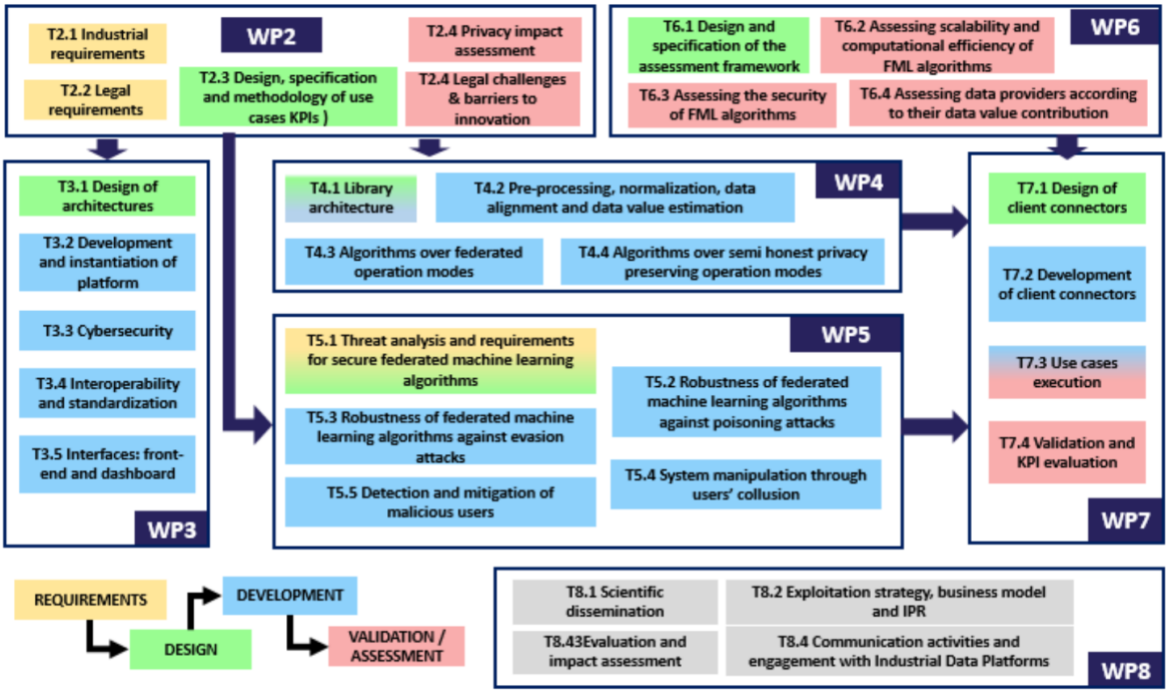

Figure 1: MUSKETEER's PERT diagram

1.3 Outline

The remainder of this document is structured as follows:

- Section 2 describes the scope of the MUSKETEER core platform (in particular vis-à-vis the algorithmic library and the client connectors software) and reviews the relevant functional and non-functional requirements outlined in the documents listed above.
- Section 3 describes the platform architecture and design. It provides detailed information on each of the platform's components as well as the underlying core technology.
- Section 4 discusses the security implications for the platform, making reference to a security by design process that is followed.
- Section 5 outlines the proposed API for utilizing the platform's services.
- Finally, Section 6 discusses possible extensions to the platform that were outside the scope of the initial version and may require further analysis in conjunction with other work packages for consideration in future versions to be developed under this project or subsequently.

2 Requirements

2.1 Scope

As discussed in D2.1, when defining the scope of the MUSKETEER platform, it is important to draw distinctions between the centralized server platform, the federated ML algorithm library, and the client connectors. This document describes the centralized server platform only. The centralized server platform neither hosts nor starts the aggregator or participant training processes. These are understood to be executed within the client software environments.

2.2 Industrial and technical requirements

D2.1 (Industrial and technical requirements) outlined all of the functional and non-functional requirements for the complete MUSKETEER platform. In this section, the centralized server platform related requirements are re-iterated, with section numbers mapping directly to the same section numbers in D3.1, for ease of reference. For each requirement, the ID is highlighted in green text if the current prototype described in D3.3 satisfies the requirement. A requirement may also be highlighted in orange text if the current prototype partially satisfies the requirement. If a requirement is not currently satisfied by the D3.3 prototype, it is not highlighted. These requirements are still subject to ongoing development.

2.2.1 User roles

There are no additional user roles beyond those identified in D3.1.

2.2.2 Functional requirements

There are no additional functional requirements beyond those specified in D3.1. What follows is a D3.3 readiness update for each requirement grouped by the type of action.

2.2.2.1 Managing platform users

Table 1: Functional requirements for managing platform users

ID	Description of the requirement
FR001	Ability for platform admin to grant username and password to new general user (**D2.1-FR034**).

FR002	Ability for platform admin to revoke username and password of existing general user (**D2.1-FR034**).
FR003	Ability for general user to avail of platform functionality through authentication with their username and password (**D2.1-FR001**).
FR004	Ability for general user to change their password (**D2.1-FR002**).

2.2.2.2 Managing Federated ML tasks

Table 2: Functional requirements for managing Federated ML tasks

ID	Description of the requirement
FR005	Ability for general users to create a new Federated ML task, including an unstructured description and all structured information that is required to define the task, such as the input data format, required mechanism for pre-processing the raw input data, the number of participants, starting/stopping criterions, etc. (**D2.1-FR016**, **D2.1-FR019**, **D2.1-FR043**).
FR006	Ability for a task creator to update the task description and information.
FR007	Ability for general users to list all the existing Federated ML tasks that have been created; view their description, definition and status; compute summary statistics, e.g., total number of tasks and participants (**D2.1-FR007**, **D2.1-FR008**, **D2.1-FR009**, **D2.1-FR010**, **D2.1-FR022**, **D2.1-FR027**, **D2.1-FR039**)
FR008	Ability for a general user to join a task that has already been created and that accepts new participants (**D2.1-FR012**).
FR009	Ability for a task member to actually participate in the training of that task's Federated ML model, either as aggregator or as participant (**D2.1-FR024**).
FR010	Ability for a task member to leave that task (**D2.1-FR029**).
FR011	Ability for a task creator to cancel that task (**D2.1-FR020**).
FR013	Ability for general users to list all the Federated ML models; view their description, definition, KPIs etc. if available (**D2.1-FR011**).
FR014	Ability for general users to download trained Federated ML models (**D2.1-FR013**, **D2.1-FR026**).

FR015	Ability for a task creator to delete the Federated ML models trained as part of that task (**D2.1-FR021**).

2.2.2.3 Executing Federated ML tasks

Table 3: Functional requirements for executing Federated ML tasks

ID	Description of the requirement
FR016	Ability for an aggregator or participant to retrieve the definition of a specific task.
FR017	Ability for an aggregator to retrieve the list of all participants of a specific task.
FR018	Ability for an aggregator to broadcast a message to all the participants.
FR019	Ability for an aggregator to send a message to a specific participant.
FR020	Ability for a participant to send a message to the aggregator.
FR021	Ability for a participant to route a message to the "next" participant (according to an underlying ring topology), without having to send it via the aggregator.
FR022	Ability for an aggregator to receive a message sent by a participant, together with an identifier of the participant who sent it.
FR023	Ability for a participant to receive a message sent by the aggregator.
FR024	Ability for a participant to receive a message routed from the "previous" participant (according to an underlying ring topology), including an identifier to distinguish from messages sent by the aggregator.
FR025	Ability for an aggregator to store task status updates.
FR026	Ability for an aggregator to store intermediate or final versions of the trained Federated ML model.
FR027	Ability for an aggregator to store information regarding the data value contributions per participants.

2.2.3 Non-functional requirements

There are no additional non-functional requirements beyond those specified in D3.1. What follows is a D3.3 readiness update for each requirement.

Table 4: Non-functional requirements

ID	Description of the requirement
NR001	High availability (**D2.1-NR001**).
NR002	Security, specifically regarding access control and adherence to industry security standards (**D2.1-NR002**).
NR003	Robustness of the overall platform with respect to software errors (**D2.1-NR016**).
NR004	Availability of appropriate logging mechanisms for all operations (**D2.1-NR010**).
NR005	Recoverability, specifically of the training of Federated ML models, from temporary system or component failures (**D2.1-NR003**, **D2.1-NR004**, **D2.1-NR005**, **D2.1-NR015**).
NR006	Scalability, specifically the efficient execution of Federated ML training algorithms (**D2.1-NR006**), and efficient handling of simultaneous requests (**D2.1-NR014**).
NR007	High usability, specifically regarding the ease of software installation for end users (**D2.1-NR009**) and the design of interfaces for interactions with the platform, including their documentation (**D2.1-NR008**).
NR008	Maintainability, specifically the availability of mechanisms to efficiently perform system or component updates with minimum downtime for the overall platform (**D2.1-NR007**, **D2.1-NR013**).

Some non-functional requirements (**NR001**, **NR003**, **NR005**) can only really be evaluated over time. As such, at this point in time, they cannot be considered complete.
Others (NR006), are deemed partially complete already, but a more thorough review over a longer period of time is also preferable.

2.3 Alignment with industrial data platform standards

The design of the MUSKETEER platform aligns with emerging standards for industrial collaborative and data sharing platforms. The MUSKETEER platform, with the associated ecosystem of external components, is converging with the International Data Space Association (IDSA) reference architecture model (RAM) [11].

In particular, with reference to the System Layer, the MUSKETEER platform is designed to operate as a broker, allowing the communication between the various participants of the training process, which are operating as Data Apps according to the reference architecture.

As noted in D3.1, the MUSKETEER platform relies on client connectors' certification to be more adherent to the IDSA-RAM, and it is beyond the scope of this deliverable. Nonetheless, the assessments that will be presented in Section 4 are in line with the required steps for the self-assessment part of the certification required by the IDSA-RAM, and therefore suggested for application to the evaluation of client connectors.

3 Platform architecture

The architecture as presented in D3.1 is largely unchanged, and as such, this chapter will primarily focus on the architecture evolutions during the prototyping phase detailed in D3.3. Figure 2 shows a diagram of the final version of the architecture for the MUSKETEER centralized server platform. The intention of this architecture is to show the inner workings of the centralized server platform and to highlight where and how remote components provided by other work packages interact with the platform.

The architecture is based on micro-services and places a significant emphasis on open standards. Many of the underlying components used are open source. The use of open standards and services avoids vendor lock-in to a significant extent, thereby enhancing the prospect of utilising alternative cloud providers or on-premise deployments in the future, if that is so desired.

It is intended that concrete instances of the architecture are deployed on the public cloud. This is a natural fit for MUSKETEER operations, as several distinct organisations need to collaborate on federated learning and a single, centrally manged, accessible and secure platform is necessary. As the public cloud is internet addressable, all collaborating organisations have access to the services (assuming no site-specific firewall restrictions).

As concrete instances of the architecture utilise existing public cloud services, these are specifically referred to in the diagram. Internally, it is a micro-services architecture [1]. The cloud infrastructure is provided by IBM, using the IBM® Cloud™ platform [2]. The platform also contains a client package for interacting with these services. Using the public IBM® Cloud™, many open source services are available in the IBM® Cloud™ catalogue. These services are quite easy to provision and secure using the cloud dashboard or cloud command line interface.

D3.3 demonstrates a simple prototype which enables end-to-end execution of federated learning via the platform and the client package; the development of full-scale client connectors lies within the scope of WP7.

Interoperability between components (cloud-based and remote) is through a messaging system, based on the Publish / Subscribe Design Pattern [3]. This is backed by RabbitMQ [4]. Messages are published to RabbitMQ and routed to subscribed parties. RabbitMQ is instantiated in the public cloud and is an internet addressable service, allowing remote clients to connect. Remote clients require appropriate credentials which are obtained through the registration process.

Using this messaging system, the initiation of all network connections is outbound only. This means that no remote component (aggregator or participant system) accepts an incoming connection with no network ports openly addressable to the internet.

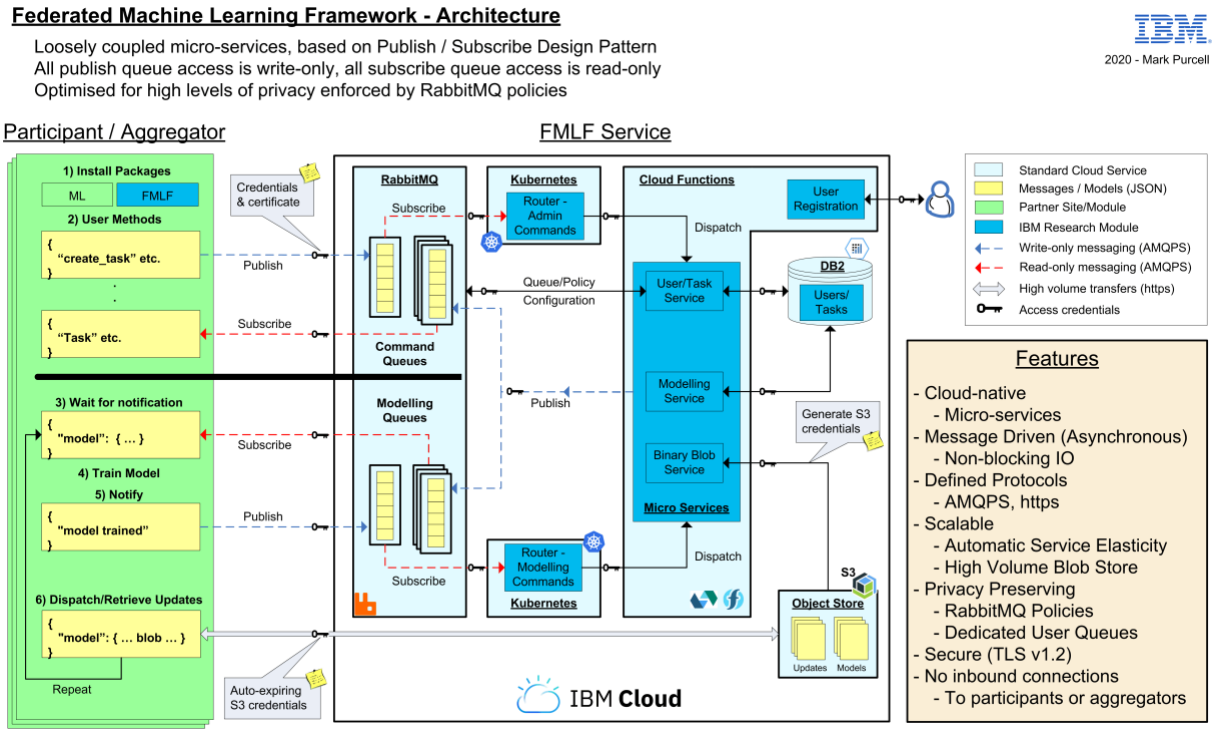

Figure 2: MUSKETEER centralized server platform architecture

All access to platform services from remote components (provided by other work packages) is through the *Federated Machine Learning Framework* (FMLF) package. This contains APIs to simplify access to the platform and is installed at remote sites.

Users of these APIs must be authenticated, but first, the *User Registration* service allows users to register with the platform. This service creates user accounts on the RabbitMQ instance. These registration details allow users to subsequently authenticate with the platform, providing access to the APIs and platform services. Access to individual APIs is also controlled through an authorisation layer (see section 5.7).

3.1 Message Flow

As previously mentioned, the platform operates in response to messages received via the *FMLF* package. Two examples of this flow are now discussed. Firstly, a control-plane example,

whereby a synchronous invocation of a platform service is detailed. And secondly, a data-plane example, which is asynchronous in nature.

3.1.1 Control-plane

An example of a synchronous command is creating a task. There is an API in the *FMLF* package called *create_task* which accepts a task name, a task topology and a task definition as inputs. This command returns a status code reflecting the success of the command. It is in effect, an atomic operation and must be synchronous. Either the creation of the task was successful, or it was not. The flow is described below and a zoom-in on the architecture highlights which components are involved.

1. The user invokes the *create_task* function
    a. A create_task message is published to RabbitMQ
    b. The function blocks, awaiting a reply
2. The command router receives the message
    a. The publishing user is validated
    b. The message is routed to the *User/Task* micro-service
3. The User/Task micro-service receives the message
    a. The database is checked for a duplicate task
    b. The task is inserted into the database (no duplicates)
4. A response is published to the command response queue
5. The *create_task* function receives the reply and returns

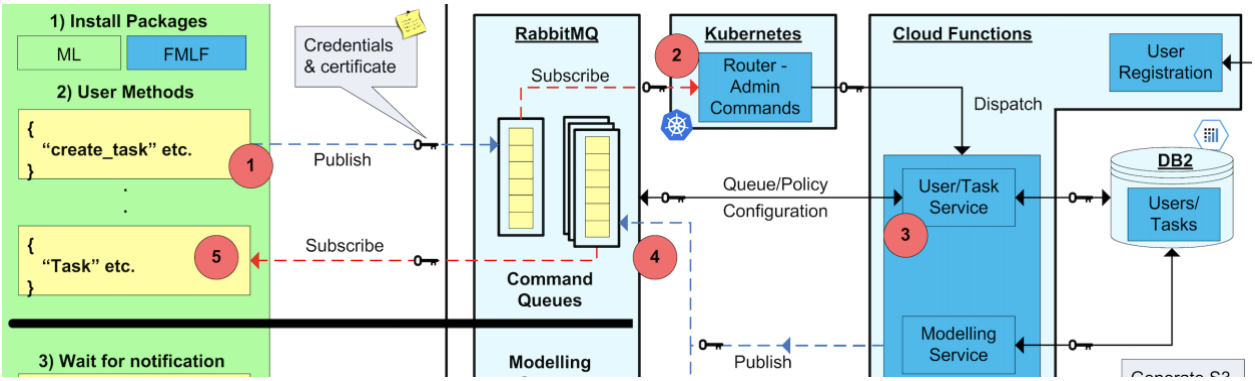

Figure 3: Control-plane flow – Zoomed

3.1.2 Data-plane

An asynchronous command example is when an aggregator starts a round of federated learning. Later, the aggregator determines that all participants have finished the training round. It would not be ideal if the aggregator blocked, awaiting completion of all participants. This process could take quite some time. Preferably, the aggregator issues the training start command, and receives notifications upon participant completion. This way, the aggregator could start to process notifications from faster participants or undertake other actions whilst waiting for the quorum of participants to complete the round of training. There is an API in the *FMLF* package called *start_task* which accepts a task name and an initial model as input. The flow is described below and a zoom-in on the architecture highlights which components are involved.

1. The aggregator user invokes the *task_start* function
    a. It is assumed that a quorum of participants is available
    b. A task_start message is published to RabbitMQ
    c. The function returns
2. The aggregator user periodically checks for notifications
3. The command router receives the message
    a. The publishing user is validated
    b. The message is routed to the *Modelling* micro-service
4. The *Modelling* micro-service receives the message
    a. The database is queried for all task participants
    b. The task_start notification is published to RabbitMQ
        i. To each participants' private queue
5. The participant users receive the notification
    a. Local training starts
    b. A task_update message is published to RabbitMQ
6. The command router receives the message
    a. The publishing user is validated
7. The message is routed to the *Modelling* micro-service
    a. The database is queried for the task details
    b. The task_update notification is published to RabbitMQ
        i. To the aggregator users' private queue

8. The aggregator user receives task_update notifications

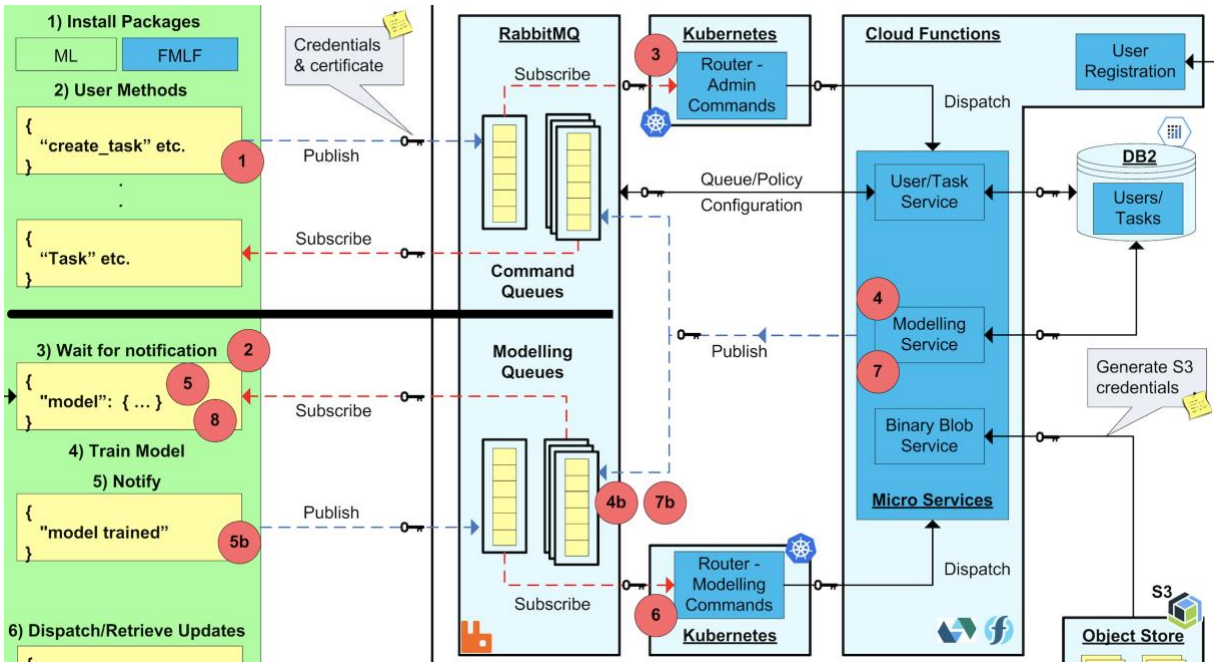

Figure 4: Data-plane flow - Zoomed

3.2 Cloud-hosted Services

The MUSKETEER architecture utilises a number of services available on the public IBM® Cloud™, each of which were described in D3.1. For the purposes of the MUSKETEER project, the public cloud data centre is located in Germany and all interactions, services and data are located and stored in this data centre.

3.2.1 Open Source/Standards

There are several open source services that are used by the platform. These are:

- IBM Cloud™ Messages for RabbitMQ
- IBM® Cloud Object Storage
- IBM Cloud™ Functions [5]
- IBM Cloud™ Kubernetes Service [6]

Additionally, a number of open standards are employed. They are:

- AMQP(S)
- HTTP(S)
- JSON
- SQL

Although the platform uses IBM® Db2® on Cloud, the underlying database schema for representing MUSKETEER tasks is fully SQL compliant. This ensures compatibility with other relational database providers.

By minimizing the effect of vendor lock-in, the use of open source components and open standards supports greater flexibility and engagement with the platform.

3.3 Initial Security/Privacy Mitigations

An in-depth discussion of security and privacy issues is covered in section 4. In this section, some of the considerations and mitigations that were put in place at a very early stage in the architecture and prototyping phase are discussed.

3.3.1 Outbound-only network connections

For any remote component, hosted by federated learning aggregators or participants, network connections are always initiated by that component. Through the *FMLF* package, these network connections are either an AMQPS connection to RabbitMQ or a HTTPS connection to Cloud Object Store. The architecture stipulates that no inbound network connection attempts to aggregators or participants are required. This reduces the risk for remote sites, due to the fact that there is no requirement for an Internet-addressable endpoint to be available at the remote site. There are no new open ports and no need for permissive inbound firewall rules.

3.3.2 Secure communications

The protection afforded by outbound-only connections is enhanced by the provision of secure communication channels. All communication between remote components and the platform (as well as between intra-platform components) use Transport Layer Security (TLS) v1.2, which is the latest available version on IBM® Cloud™. This ensures that all data transmitted between

components is encrypted. Additionally, cloud certificates are used by remote components to establish the veracity of the cloud endpoint to which outbound network connections are made.

3.3.3 Time-limited credentials

When models/updates are transferred between aggregator and participant users, IBM® Cloud Object Storage is used to store the content. Messages are then dispatched to the user *FMLF* package, detailing upload/download information for the given content. This information contains temporary, automatically expiring, once-off credentials that are used to upload or download the content. This is based on the S3 [7] standard *pre-signed* APIs. This mechanism is used due to the fact that models or updates can be quite large, and it is preferable not to have large content moving between multiple services.

3.3.4 User validation

Upon successful registration, an aggregator or participant user has their own unique credentials for the platform. These credentials are then used by the *FMLF* package to initiate a connection to RabbitMQ. Subsequently, when commands are issued by the user, the user is validated by RabbitMQ against the user who initiated the connection. By using this RabbitMQ feature, it becomes difficult for a given user to impersonate another user.

Additionally, in the command router (running in Kubernetes), an additional check is made to ensure that this RabbitMQ validation was not bypassed (somehow) by the user. The command router receives messages that originate from the aggregator or participant user. When these messages are received, RabbitMQ provides meta-data, detailing the originating user. This is checked to ensure the user is valid.

3.3.5 Queuing Policy

By applying write-only and read-only polices to RabbitMQ queues, it is not possible for users to view the contents of, or publish to, queues that they have not been given explicit access to.

For example, in the control-plane, any command messages published to the single command write-only queue, are not readable by any standard user. Read access to this queue is only granted to a single system administration user. This ensures that standard users cannot determine the services that other users are requesting.

3.3.6 Avoidance of SQL Injection attacks

Often, these attacks [13] target SQL string concatenation vulnerabilities in the underlying software. This is where the SQL code is built using techniques in languages such as Java and user input can be effectively concatenated directly into the SQL (e.g. SELECT statement).

The MUSKETEER platform micro-services that connect to the database do not use another language for building SQL statements. Instead, all SQL is contained in stored procedures which are deployed at database schema creation time. Higher-level languages invoke one of these stored procedures. This provides the benefit of presenting the underlying table structure as an API, whereby other components only interact with the database through this defined API. Internal details of how the tables are structured is not used by other components, which encapsulates the database schema design specifics behind this API. Internally, the stored procedures use standard SQL statements to interact with the database. A second benefit of this approach is that SQL injection attacks are less likely to succeed, due to the stored procedure API obfuscation of the underlying SQL statements. The SQL statements within the stored procedures are always based on *static cursors*, and string concatenation for *SELECT-WHERE* clauses are never used.

4 Security & Privacy

Since its inception, the MUSKETEER platform has been developed by leveraging elements of the IBM Security and Privacy by Design (SPbD) practice. An introduction of the SPbD concepts is given in this section, along with a detailed discussion of the Threat Modelling and Architecture Review exercise.

4.1 Security and Privacy by Design principles

Security and Privacy by Design (SPbD) is a simplified and agile set of focused security and privacy practices, including threat models, privacy assessments, security testing, and vulnerability management. As shown in Figure 5: Security and Privacy by Design, the SPbD process includes the following tasks:

- Threat Model: identifies, communicates, and understands threats and mitigations within the context of protecting something of value.
- Privacy Assessment: is the process to evaluate new projects, policies, and practices for privacy, confidentiality, or security risks associated with the collection, processing, or disclosure of personal information. It also includes developing measures that are intended to mitigate and eliminate identified risks. In particular, this assessment process must meet GDPR requirements.
- Code Scan: helps programmers locate potential flaws and determine areas of improvement within the codebase. Code scans must be performed during development and test, cover IBM developed code, and include Open Source Software.
- Security Tests: a key component of the overall test cycle which is intended to ensure that the development process resulted in secure code and that, where possible, threats identified as part of the threat model were properly addressed. Security testing helps validate that the information system in question protects data and functions as intended.
- Penetration Test (also called pen testing): an authorized simulated attack on a computer system, application, or IT environment. It can involve automated tools and must involve a form of ethical hacking. Vulnerability Management is the process of searching for software vulnerabilities in applications by using an automated security program.

- Vulnerability scanning: can be used to find vulnerabilities and remediate them before they are exploited.

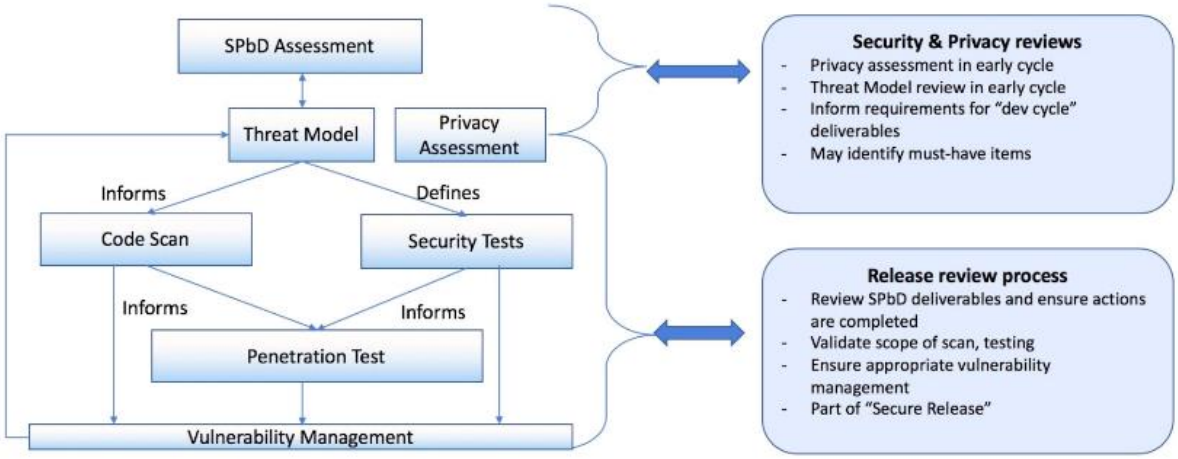

Figure 5: Security and Privacy by Design (from [8])

For more detailed information, the reader is referred to [8].

4.2 SPbD in the MUSKETEER centralized server platform

During the early stages in the Software Development Life Cycle of the MUSKETEER centralized server platform, an internal SPbD assessment was conducted. The outcome of this assessment can be summarized as follows:

1. Several security controls were already implemented since the platform inception, in order to guarantee the security of the platform and its users. However, a Threat Model and Security Architecture Review of the platform is now required. A combined Threat Model and Architecture Review exercise is sufficient in order to have a reasonable understanding of the security posture of the MUSKETEER platform.
2. If there are design changes between the initial creation and release, another review will be required, or when significant architectural changes are made to the platform, a new Threat Model and Security Architecture Review must be conducted.
3. Code Scan and Security Tests should be performed at a later stage, and a Penetration Test of the platform is also recommended. The scope of

these could include the work of other WPs (for example, the development of client connector software in WP7).

4. Security Vulnerabilities reported by the activities described in Steps 2 and 3 will be tracked in the MUSKETEER Github Repository, along with their remediation plan.

4.2.1 Threat Model and Security Architecture Review

In order to ensure that the MUSKETEER platform is designed from the ground up with security and privacy in mind, and to ensure that privacy and regulatory requirements are met by design, we have chosen to adopt a combined Security Architecture Review and targeted Threat Modelling approach that leverages common elements from both techniques, while remaining relatively light weight.

As part of the initial Threat Modelling and Architecture Security Review exercise, an Architecture overview Diagram has been produced, providing a high-level overview of the platform design including internal components, inputs, outputs and users (see Figure 2).

Following the Architecture Overview Diagram, an Inventory and Threat Model document has been produced, which in turn includes the following:

- Application Information – Background information about the platform.
- Component Inventory – Listing of components, deployment type, security logging methodology for each.
- Process Inventory – Identifies the actual application processes running on each component as part of the solution and privilege level of each.
- Datastore Inventory – List of all places where data is persisted in the solution, including type of store, data classification, tenancy model, encryption/protection method and backup type.
- Interface Inventory – Enumerates all interfaces (UI, APIs, Admin interfaces, etc) exposed by components in the solution. Highlights interface type, authentication method, access protocol and data classification.
- Credential Inventory – Identifies locations in the solutions in which credentials (keys, passwords, certificates, etc) are stored and how they are protected.
- Actors – Listing of users or systems which interface with the solution.

- Data Flows – Identifies the authentication method(s), protocols, data-classification and encryption methods for all flows which are part of the system.
- Threat Assessment – Inventory of weaknesses to assess the target system against, built from a combination of SAN 25 [9], OWASP Top 10 [10] and most frequently identified penetration test vulnerabilities. Each item needs to be assessed and mitigation and testing plans described.

Below is the Threat Assessment table, which includes comments and suggestions from reviewer(s), where applicable:

Table 5: Threat Assessment

CWE	Description	Mitigation description	Recommendation
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	Interaction with the DB is done exclusively via Stored Procedures, and these are invoked via IBM DB2 Python Package.	Inspect stored procedures and confirm there are no SQL statements being generated in an unsafe manner (e.g. via concatenation).
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	No OS commands being executed in any component.	

CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	Dependencies are not known to be vulnerable to Buffer Overflow. Messages greater than 5MB are discarded by the Router.	Investigate if it is possible to enforce the 5MB limit directly in RabbitMQ rather than discarding the messages at the Router.
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	This weakness could be in scope for the client connectors.	Handled in other work packages.
CWE-306	Missing Authentication for Critical Function	All the components in scope are subject to authentication, either via internal user database or via IBM SSO.	
CWE-862	Missing Authorization	RabbitMQ: we have policies in place to enforce write-only access or read-only access. These policies, along with the separation of the queues, guarantee a good level of isolation and authorisation. Router: no authorisation is being performed, messages are only being forwarded. Cloud Functions: authorisation is performed to limit the user to specific commands.	Some Cloud Functions have only minimal authorisation and deny certain actions. A more complete authorisation is preferable.
CWE-798	Use of Hard-coded Credentials	The only "hardcoded" credentials, if hardcoded can be considered, are for the guest account which we are circulating to the MUSKETEER consortium partners in order to enable them to register new users with the platform.	Given the current Trust Model (See Trust Model section 4.3.1 of this document), this is acceptable. However, should the Trust Model change, this needs to be reconsidered. Investigate the usage of an external source (i.e. an SSO, such as the IBM SSO via IBM Id).

			Investigate the need to create a new REST API endpoint for tokens / credentials generation.
CWE-311	Missing Encryption of Sensitive Data	RabbitMQ: Messages are stored in cleartext (but transmitted over SSL). Some messages (i.e. user registration) may contain sensitive data, and leaving this data unencrypted on the RabbitMQ queue may raise some concern. Likelihood of RabbitMQ being breached is low. IBM COS: Payloads communicated through the platform by aggregators/participants are not encrypted at this level. If required, encryption is implemented in other work packages, e.g. at the algorithmic level. IBM DB2: database encryption is left to the IBM Cloud DB2 service. Usernames are being stored for authorisation against the queues.	Investigate the feasibility of adding a component that adds an encryption layer to the COS uploads, so that no payloads are left unencrypted on COS. The likelihood of COS being breached is very low.
CWE-434	Unrestricted Upload of File with Dangerous Type	After credentials for upload have been granted to the participant, there is no validation in place for checking what the participant is uploading on COS.	Given the current Trust Model (See Trust Model section 4.3.1), this is acceptable. However, should the Trust Model change, this needs to be reconsidered.
CWE-807	Reliance on Untrusted Inputs in a Security Decision	The only security decision currently implemented relies on the username submitted along with the messages to RabbitMQ. This username is verified and validated by RabbitMQ and can be trusted.	Investigate the robustness of Routers and Cloud Functions via manual Pentest, by means of a fuzzer or other techniques.
CWE-250	Execution with Unnecessary Privileges	No sudo/administrative/high privilege commands are being executed.	
CWE-352	Cross-Site Request Forgery (CSRF)	This weakness could be in scope for the client connectors. However, some administrative UI interfaces might be in scope, but they should not be publicly accessible.	Handled in other work packages. Investigate with a manual Pentest if any administrative UI is

			vulnerable to CSRF, e.g. [12]
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	Participants receive a set of URL + credentials for downloading and uploading content from/to COS and these credentials are bound to the URL - there should be no way to perform path traversals to other buckets and upload/download content to/from other buckets. COS is not known to be vulnerable.	
CWE-494	Download of Code Without Integrity Check	Docker images are being built on a trusted Travis environment. Dependencies are being downloaded at build time only, and the likelihood of a Man-in-the-Middle attack is very low. No code/dependencies are being resolved at runtime and/or on untrusted environments.	
CWE-863	Incorrect Authorization	See CWE-862 comment above.	
CWE-829	Inclusion of Functionality from Untrusted Control Sphere	In the centralized server platform, there are no features that rely on the execution of code transmitted by the users of the platform. This weakness could be in scope for the ML library implementation.	Handled in other work packages.
CWE-732	Incorrect Permission Assignment for Critical Resource	N/A	
CWE-676	Use of Potentially Dangerous Function	A high level code review that we have performed did not highlight any obvious dangerous function or code block. However, this will be covered either with a deep source code review or a static source code analysis (i.e. appscan)	This will be confirmed by Static Source Code Analysis.

CWE-327	Use of a Broken or Risky Cryptographic Algorithm	We use TLS1.2 for SSL channels. In the future, a network scan of the interfaces might reveal if there are deprecated crypto algorithms being used. To our knowledge, the IBM Cloud services are following the best practices. In the platform code, there are no cryptographic algorithms being leveraged.	This will be confirmed by Network Vulnerability Scan.
CWE-131	Incorrect Calculation of Buffer Size	See CWE-120 comment above.	
CWE-307	Improper Restriction of Excessive Authentication Attempts	We are relying on RabbitMQ and IBM COS authentication.	Investigate the feasibility to mitigate this on RabbitMQ and IBM COS. Alternatively, investigate the feasibility to implement an additional layer that abstracts RabbitMQ and also COS, where authentication attempts could be checked for.
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')	This weakness could be in scope for the client connectors.	Handled in other work packages.
CWE-134	Uncontrolled Format String	Format strings are being used mainly for logging purposes and hardcoded. No format strings are being received by the end users of the platform.	
CWE-190	Integer Overflow or Wraparound	See CWE-120 comment above.	
CWE-759	Use of a One-Way Hash without a Salt	Passwords are only being stored in RabbitMQ. It should be confirmed whether or not RabbitMQ is using a salted hash.	Investigate and confirm RabbitMQ is storing passwords in a secure manner. Alternatively, investigate the usage of an external LDAP.

CWE-521	Use of default, weak or well-known passwords	All the management passwords we have are strong and randomly generated. We are currently not enforcing any password complexity policy in RabbitMQ.	Investigate the feasibility of implementing a password complexity logic in the Router Component, unless RabbitMQ allows a password complexity policy.
CWE-640	Uses weak or ineffective credential recovery and forgotten password processes	We currently do not have any credential recovery process in place for RabbitMQ passwords.	
CWE-308	Missing or ineffective multi-factor authentication for administrative access or access to sensitive data.	No MFA is enabled on RabbitMQ. From administration perspective, DB2 and RabbitMQ do not have any multi factor authentication.	Investigate the feasibility of using external sources (e.g. LDAP or SSO) for authenticating against RabbitMQ and DB2.
CWE-312	Clear text storage of sensitive data. (Including keys and credentials)	RabbitMQ: unauthorized access to RabbitMQ is very unlikely. The queues are only used for transient storage of commands. DB2: usernames are currently stored in cleartext. Passwords are not stored and only used by RabbitMQ.	Propose to store username via a hash/salt combination.
CWE-611	Improper restriction of XML External Entities (XXE)	N/A	
CWE-16	A6: Security Misconfiguration / Improper Hardening	We rely on IBM Cloud hardening practices.	This will be confirmed by Penetration Testing and Network Vulnerability Scan.

CWE-502	A8: Insecure Deserialization	This weakness could be in scope for the ML library implementation.	
----	A9: Using Component with unknown vulnerabilities	No components are known to be vulnerable.	This will be confirmed by Static Source Code Analysis.
CWE-778	A10: Insufficient Logging & Monitoring	Most of the components are logging locally to whatever logging interface is provided by the IBM Cloud service.	
CWE-532	Exposure of sensitive information through logs. (ie. Logging of credentials)	A high level code review showed no concerns.	Investigate all the logging points and ensure no sensitive data is being logged.
----	Failure to enforce HTTP Strict Transport Security	We rely on IBM Cloud hardening practices.	This will be confirmed by Penetration Testing and Network Vulnerability Scan.

In addition to the previous list of known weaknesses, other WPs should assess the mitigations against the following:

Threat Description	Mitigation
CWE-1039: Automated Recognition Mechanism with Inadequate Detection or Handling of Adversarial Input Perturbations	This weakness could be in scope for the ML library implementation.
ML Poisoning Attacks - ability for an attacker to poison the training data by injecting carefully designed samples to eventually compromise the whole learning process	This weakness could be in scope for the ML library implementation.
ML Extraction Attacks - ability for an attacker to extract particular information from the model	This weakness could be in scope for the ML library implementation.

4.3 Threat Assessment Conclusions

The outcome of the Threat Assessment can be summarised as follows:

- Existing mitigations designed and implemented since the platform inception are deemed effective in most cases.
- While no critical weaknesses have been identified, the implemented Authorisation mechanism may need improvements.
- Other investigations have been recommended for future improvements, and details can be found in the previous section.
- It is recommended that Static Source Code Analysis, Network Vulnerability Scanning, and Penetration Testing be conducted in order to confirm the implemented security controls.

The scope of the Threat Modelling and Security Architecture Review was limited to the MUSKETEER platform and its components as per the architecture shown in Figure 2, with the Participant / Aggregator component being the only exception: a security assessment of this component should be conducted separately within the scope of the WPs responsible for developing it.

The MUSKETEER "local platform", not depicted in the diagram but included in D3.3, was also deemed out of the scope of the Threat Modelling and Security Architecture Review. It is only a development tool that facilitates testing during the development phase. It is not meant to be used in a real-world environment.

4.3.1 Considerations on the current Trust Model

Some of the existing mitigations have been deemed as sufficiently secure based on the current Trust Model. In the current phase of the MUSKETEER project, guest credentials for accessing the platform are being shared with only a trusted set of parties who are collaborating on the project.

Device trust, user trust, transport/session trust, application trust and data trust need to be reconsidered should a Zero Trust Model approach be adopted in the future. Some of the mitigations should be reviewed as suggested in the Threat Assessment excerpt.

5 Proposed API

Section 3 described the platform and the messaging-based interoperability. In this section the messages for each individual service/API are described. This is not intended as a definitive API guide, but rather a synthesis of functionality required to build a full end-end API. The per-API parameters and results are in-line with what each API requires, but API service names etc. are instructive rather than definitive, leaving platform API developers some leeway in determining their implementation. Additionally, this API is programming language agnostic, affording considerable flexibility for future implementations.

5.1 Basic Message Structure

There is an underlying message structure that is common to all messages. It is based on JSON notation.

5.1.1 Service Request

Each message for a service request is based on a JSON structure as follows:

```
{
  "service": {
    "name": "<service name>",
    "args": {
      "cmd": "<command name>", "params": [<p1>, <p2>, … <pn>]
    }
  }
}
```

Figure 6: Service Request

- <service name> - the platform service to invoke, used by the *Router*
- <command name> - the service command
- <p1> etc. - parameters to the service command

5.1.2 Service Response

Each response from a service is based on a JSON structure as follows:

```
{
  "service": {
    "name": "<service name>",
    "method": "<command name>",
    "params": [<p1>, <p2>, … <pn>],
    "count": "<points>",
    "data": [{<data1>}, {<dataN>}]
  }
}
```

Figure 7: Service Response

- <command name> - the service command originally requested
- <p1> etc. - parameters to the service command
- <points> - the number of entries in the “data” array
- <data1> etc – row of data

5.2 Control-plane APIs

This is backed by a micro-service based on IBM Cloud™ Functions which provides machine learning task management services invoked through the control-plane. The service records task details in the database. All of the control-plane APIs are synchronous and expect to receive a response from the platform.

5.2.1 Change User Password

Allows the authenticated user to change their password.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "ChangePassword", "params": ["<Password>"]
    }
  }
}
```

Figure 8: Join Task Request

```
{
  "service": {
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 9: Join Task Response

- <Password> - the new password (string)
- <Status> - the status, e.g. “OK” (string)

5.2.2 List Tasks

Query for a list of all tasks.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "GetTasks", "params": [None]
    }
  }
}
```

Figure 10: List Tasks Command

```
{
  "service": {
    ...
    "data": [{
        "name": "<TaskName>",
        "status": "<Status>",
        "topology": "<Topology>",
        "definition": "<Definition>"
    }]
  }
}
```

Figure 11: Task Listing Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status (string)
- <Topology> - relates to POM type, e.g. “STAR” (string)
- <Definition> - parameters for the task (JSON, optional)

5.2.3 Create Task

Create a new task with the authenticated user as the designated task owner.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "CreateTask", "params": ["<TaskName>", "<Topology>", <Definition>]
    }
  }
}
```

Figure 12: Create Task Request

```
{
  "service": {
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 13: Create Task Response

- <TaskName> - the name of the task (string)
- <Topology> - relates to POM type, e.g. “STAR” (string)
- <Definition> - parameters for the task (JSON, optional)
- <Status> - the current task status, e.g. “CREATED” (string)

5.2.4 Stop Task

Stop a task previously created by the authenticated user (task creator).

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "StopTask, "params": ["<TaskName>"]
    }
  }
}
```

Figure 14: Stop Task Request

```
{
  "service": {
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 15: Stop Task Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status, e.g. “COMPLETE” (string)

5.2.5 Join Task

Join a specific task with the authenticated user details as a new participant.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "JoinTask", "params": ["<TaskName>"]
    }
  }
}
```

Figure 16: Join Task Request

```
{
  "service": {
```

```
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 17: Join Task Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status, e.g. “CREATED” (string)

5.2.6 Leave Task

Leave a task previously joined by the authenticated user.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "LeaveTask, "params": ["<TaskName>"]
    }
  }
}
```

Figure 18: Leave Task Request

```
{
  "service": {
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 19: Leave Task Response

- <TaskName> - the name of the task (string)
- <Status> - the status, e.g. “OK” (string)

5.2.7 Task Info

Query for the task information for a specific task.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "TaskInfo", "params": ["<TaskName>"]
    }
  }
}
```

Figure 20: Task Info Request

```
{
  "service": {
    ...
    "data": [{
        "name": "<TaskName>",
        "status": "<Status>",
        "topology": "<Topology>",
        "definition": "<Definition>"
    }]
  }
}
```

Figure 21: Task Info Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status, e.g. “CREATED” (string)
- <Topology> - relates to POM type, e.g. “STAR” (string)
- <Definition> - parameters for the task (JSON, optional)

5.2.8 Joined Tasks

Query for a list of all tasks joined by the authenticated user.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "JoinedTasks, "params": [None]
    }
  }
}
```

Figure 22: Joined Tasks Request

```
{
  "service": {
    ...
    "data": [{
        "name": "<TaskName>",
        "status": "<Status>",
        "topology": "<Topology>",
        "definition": "<Definition>"
    }]
  }
}
```

Figure 23: Joined Tasks Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status, e.g. “CREATED” (string)
- <Topology> - relates to POM type, e.g. “STAR” (string)

- <Definition> - parameters for the task (JSON, optional)

5.2.9 Created Tasks

Query for a list of all tasks created by the authenticated user.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "CreatedTasks, "params": [None]
    }
  }
}
```

Figure 24: Created Tasks Request

```
{
  "service": {
    ...
    "data": [{
        "name": "<TaskName>",
        "status": "<Status>",
        "topology": "<Topology>",
        "definition": "<Definition>"
    }]
  }
}
```

Figure 25: Created Tasks Response

- <TaskName> - the name of the task (string)
- <Status> - the current task status, e.g. “CREATED” (string)
- <Topology> - relates to POM type, e.g. “STAR” (string)
- <Definition> - parameters for the task (JSON, optional)

5.2.10 Download Model

Query for the model for a specific task. The authenticated user must be the task aggregator or a participant in the task.

```
{
  "service": {
    "name": "ModellingService",
    "args": {
      "cmd": "GetModel", "params": ["<TaskName>"]
    }
  }
```

```
}
```

Figure 26: Task Info Request

```
{
  "service": {
    ...
    "data": [{
      "name": "<TaskName>",
      "model": {
        "url": "<ModelURL>",
        "model": {<Model>}
      }
    }]
  }
}
```

Figure 27: Task Info Response

- <TaskName> - the name of the task (string)
- <Model> - an initial model (JSON, optional)
- <ModelURL> - a URL to an initial model (string, optional)

5.3 Data-plane APIs

This is backed by a micro-service based on IBM Cloud™ Functions which provides machine learning modelling services invoked through the data-plane. All of the data-plane APIs are asynchronous and do not expect to receive a response from the platform immediately. Rather, a series of notifications are expected at a later time.

5.3.1 Aggregator Start Training Round

As the task creator (the authenticated user), start a round of federated learning.

```
{
  "service": {
    "name": "ModellingService",
    "args": {
      "cmd": "StartTraining",
      "params": ["<TaskName>", {<Model>}, "<ParticipantId>"]
    }
  }
}
```

Figure 28: Aggregator Start Training Round Request

- <TaskName> - the name of the task to start training (string)
- <Model> - an initial model (JSON, optional)
- <ParticipantId> - the id (obfuscated) of a participant (string, optional)

5.3.2 Aggregator Notification

The platform issues this notification to the aggregator in response to participant actions.

```
{
  "notification": {
    "type": "<NotificationType>",
    "participant": "<ParticipantId>",
    "status": "<Status>"
  }
  "params": {
    "model": {
      "url": "<ModelURL>",
      "model": {<Model>}
  }
}
```

Figure 29: Aggregator Received Notification

- <NotificationType> - “joined”, “updated”, “left” (string)
- <ParticipantId> - the id (obfuscated) of a specific participant (string)
- <Model> - an initial model (JSON, optional)
- <ModelURL> - a URL to an initial model (string, optional)

5.3.3 Participant Notification

The platform issues this notification to participants in response to aggregator actions.

```
{
  "notification": {
    "type": "<NotificationType>",
  }
  "params": {
    "model": {
      "url": "<ModelURL>",
      "model": {<Model>}
  }
}
```

Figure 30: Participant Received Notification

- <NotificationType> - “started”, “stopped” (string)
- <Model> - an initial model (JSON, optional)
- <ModelURL> - a URL to an initial model (string, optional)

5.3.4 Participant Training Round Complete

As a task participant (the authenticated user), inform the platform that local training is complete.

```
{
  "service": {
    "name": "ModellingService",
    "args": {
      "cmd": "TrainingComplete",
      "params": ["<TaskName>", <{Model}>]
    }
  }
}
```

Figure 31: Participant Training Round Complete

- <TaskName> - the name of the task (string)
- <Model> - a trained model (JSON, optional)

5.4 Registration APIs

In addition to the control-plane and data-plane APIs, a registration service also exists, allowing users to both register with the platform and to reset their password if necessary.

As this is a rarely used service, a manual process to perform the action by a system administrator could well be sufficient. Alternatively, it could be implemented similarly to the control-plane service or as a dedicated stand-alone service linked to a user interface.

In any event, the service supports the following interactions and is described as if it were implemented as a control-plane style service. Note: if implemented in the same manner as the control-plane services, a guest user account is needed to allow minimal access to the platform.

5.4.1 Register User

Allows a new user to register with the platform.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "AddUser", "params": ["<Name>", "<Password>"]
    }
  }
}
```

Figure 32: Register User - Request

```
{
  "service": {
    ...
    "data": [{"status": "<Status>"}]
  }
}
```

Figure 33: Register User - Response

- <Name> - the new username (string)
- <Password> - the user password (string)
- <Status> - the status, e.g. “CREATED” (string)

5.4.2 Reset User Password

Allows a previously registered user to reset their password.

```
{
  "service": {
    "name": "UserTaskService",
    "args": {
      "cmd": "ResetPassword", "params": ["<Name>"]
    }
  }
}
```

Figure 34: Reset User Password - Request

```
{
  "service": {
    ...
    "data": [{"password": "<Password"}]
  }
}
```

Figure 35: Reset User Password - Response

- <Name> - the username (string)
- <Password> - the newly reset password (string)

5.5 Administration APIs

It is envisaged that a certain number of administration APIs will be required. For example, an administrator may wish to: list all users, remove a user, view all participants by task, expire an inactive task. These administration APIs, if implemented, must be protected by an elevated level of authorisation for specific users.

5.6 Federated Machine Learning Framework (FMLF) Package

This is a Python package that is installed at the aggregator and participant user sites. It provides a high-level API that wraps the messaging control and data plane functions as described in the previous sections. It is described in more details in D3.3.

From a platform user perspective, classes are provided to cover three modes of operation.

5.6.1 Basic User

This mode of operation only uses control-plane features. It is how authenticated users create, join or list tasks. Ideal for use in a user interface, where, for example, tasks could be joined with a click.

5.6.2 Aggregator User

This user starts (5.3.1), stops (5.2.4), and manages rounds of federated learning. It receives notifications from participants (5.3.2) as rounds of training are progressing.

It is intended that this user primarily uses data-plane features, when modelling is underway.

5.6.3 Participant User

This user awaits notifications from the aggregator (5.3.3) before commencing a round of federated learning. Upon completion of a round of training, a message (5.3.4) is issued to this user.

It is intended that this user primarily uses data-plane features, when modelling is underway.

5.6.4 Open Source

As this package is the primary mechanism to interact with the platform, the intention is to make it available as an open source contribution.

5.7 Authorisation

For every API available in the FMLF package, an authorisation on a per-user basis is enforced. This is applied based on the user's role and there are three user roles in the platform: *guest, standard, administrator*. Upon invocation of any micro-service, this authorisation is applied

based on the authenticated user. An error is returned if the required level of authorisation is not present. The table below specifies which API is available to which user role:

Table 6: Authorisation by Operation

Operation	Authorisation	Comments
5.2.1 Change User Password	standard	The authenticated user
5.2.2 List Tasks	standard	Any user can list tasks
5.2.3 Create Task	standard	Any user can create a task
5.2.4 Stop Task	standard	Task aggregator user only
5.2.5 Join Task	standard	Any user can join a task
5.2.6 Leave Task	standard	Success if previously joined
5.2.7 Task Info	standard	Any user can view task info
5.2.8 Joined Tasks	standard	All previously joined tasks
5.2.9 Created Tasks	standard	All previously created tasks
5.2.10 Download Model	standard	If previously joined task
5.3.1 Aggregator Start Training Round	standard	Task aggregator user only
5.3.2 Aggregator Notification	standard	Task aggregator user only
5.3.3 Participant Notification	standard	Task participant user only
5.3.4 Participant Training Round Complete	standard	Task participant user only
5.4.1 Register User	guest	Minimal permission
5.4.2 Reset User Password	guest	Minimal permission
List Users	administrator	May not be needed
Remove User	administrator	Obsolete user accounts
List Task Participants	administrator	All participants of all tasks
Expire Task	administrator	Prune abandoned tasks

6 Conclusions and possible future extensions

To conclude this document, an outlook is given on possible future extensions to the platform. This was already covered in D3.1, and that content is still relevant. This section provides an update to the D3.1 content and details additional extensions over and above what was described in D3.1. This is in addition to the requirements not currently satisfied by the D3.3. prototype. These new extensions may be implemented in due course or in the future.

Explore synergies and possible integration points with the AI4EU platform

Discussed in D3.1 and still applicable.

Organizing platform user access permissions in groups

Discussed in D3.1 and still applicable. This has progressed and is covered in sections 5.6 and 5.7.

Permissions for downloading models

Discussed in D3.1 and still applicable. This has progressed and is covered in section 5.2.10.

Model serialization

Discussed in D3.1 and still applicable. For the D3.3 prototype, to ensure a wide range of compatibility, a base64 encoding scheme is used for model serialization. The evaluation of D3.3 in conjunction with other work packages will guide future development.

Task lifecycles

Discussed in D3.1 and still applicable. The evaluation of D3.3 in conjunction with other work packages will guide future development.

User roles

Discussed in D3.1 and resolved. The task creator is deemed to also be the designated aggregator.

Data value estimation

Discussed in D3.1 and still applicable. The evaluation of D3.3 in conjunction with other work packages will guide future development.

Encryption / key management

Discussed in D3.1 and resolved. This is handled outside of the platform, within other work packages.

Security Review Recommendations

After the security review is complete (see section 4), a number of possible enhancements may come to light. These will be considered for future development.

Audit Trail

Somewhat related to task lifecycles, it could be interesting to have a full audit log of each task, detailing training rounds, participants, aggregator insights, participant effectiveness etc.

Such a feature would effectively mean that tasks are never fully removed from the platform upon completion, but rather are archived to a task history record.

External Authentication Services

If the MUSKETEER effort were to move towards a supported product, it would be desirable to link the user authentication mechanism to an external (enterprise) authentication service. RabbitMQ can use LDAP to perform authentication by deferring to an external LDAP provided service. It could then be possible to use an enterprise single sign-on service to also provide access to the Federated Learning platform. Alternatively, for inter-enterprise workloads, a cloud-based authentication service could be used.

7 References

[1] S. Newman (2015). Building Microservices – Designing Fined-Grained Systems, O' Reilly.
[2] https://cloud.ibm.com/
[3] S. Tarkoma (2012). Publish/Subscribe Systems: Design and Principles, John Wiley & Sons, Ltd.
[4] https://www.rabbitmq.com/
[5] https://openwhisk.apache.org/
[6] https://kubernetes.io/
[7] https://docs.aws.amazon.com/s3/index.html
[8] W. Grunbok, M. Cole, "Security in Development - The IBM Secure Engineering Framework", https://www.redbooks.ibm.com/redpapers/pdfs/redp4641.pdf
[9] CWE/SANS TOP 25 Most Dangerous Software Errors, https://www.sans.org/top25-software-errors
[10] OWASP Top Ten, https://owasp.org/www-project-top-ten/
[11] International Data Spaces Association, "Reference Architecture Model – Version 3", April 2019, https://www.internationaldataspaces.org/wp-content/uploads/2019/03/IDS-Reference-Architecture-Model-3.0.pdf
[12] https://packetstormsecurity.com/files/148229/RabbitMQ-Web-Management-Cross-Site-Request-Forgery.html
[13] https://owasp.org/www-community/attacks/SQL_Injection