H2020 – ICT-13-2018-2019

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

Grant No 824988

D3.4 Final Prototype of the MUSKETEER Platform

January 21

Imprint

Contractual Date of Delivery to the EC: **31 August 2021**

Author(s): **Mark Purcell (IBM), Ambrish Rawat (IBM)**
Participant(s): **IMP; IDSA**
Reviewer(s): **Muhammad Zaid Hameed (IMPERIAL), Antoine Garnier (IDSA)**

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

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

Contact: **markpurcell@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 (D3.4 "Final Prototype of the MUSKETEER Platform") is a document describing the demonstration of the final prototype. It is the culmination of milestone 3 and builds upon the documents D3.1/D3.2/D3.3, providing feature updates as well as highlighting how these features complete the platform requirements.

Document History

Version	Date	Status	Author	Comment
1	17 December 2020	Outline draft	Mark Purcell	First draft
2	19 January 2021	Improved draft	Mark Purcell	Section 1,2,3
3	25 January 2021	Improved draft	Ambrish Rawat	Section 4
4	27 January 2021	Review feedback	Mark Purcell	Reviewed
5	29 January 2021	Final draft	Mark Purcell	Ready
6	30 January 2021	Final review	Gal Weiss	

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
FaaS	Functions-as-a-Service
GDPR	General Data Protection Regulation
GQM	Goal/Question/Metric
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
MMLL	Musketeer Machine Learning Library
MNIST	Modified National Institute of Standards and Technology
POM	Privacy Operation Mode
SQL	Structured Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
WP	Work Package
YAML	Yet Another Markup Language

1 Introduction

1.1 Purpose

This document is the description of the fourth deliverable (D3.4) of work package 3 (WP3). It describes the final prototype 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 also supports the assessments to be carried out in WP6 and provides interfaces which allow for the development of client connectors and end-to-end demonstrations of the industrial use cases in WP7.

This document is an update to the previous deliverable documents (D3.1/2/3) for WP3. As such, if any underlying information regarding system components has not changed, these components are not discussed again. However, any enhancements or new features are discussed in this document. This is particularly relevant in relation to the features that were incomplete as of D3.2. These features are discussed in detail in this document. Similarly, the demonstration focuses on enhanced features, not re-iterating on features from previous demonstrations that are unchanged. Therefore, the scope of the document and the demonstration, is to discuss the fully-featured platform in the context of its use in a prototype that exercises the D3.4 additional features.

1.2 Related documents

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

- **D3.1 Architecture Design – Initial Version** – detailing the architecture as of M12.
- **D3.2 Architecture Design – Final Version** – detailing the final architecture as of M18.
- **D3.3 First Prototype of the MUSKETEER Platform** – the precursor to this document, detailing the first prototype as of M18.
- **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.

- **D4.3 Pre-processing, normalization, data alignment and data value estimation algorithms – Final version –** which discusses data value, important for the data economy and data aggregation procedures.

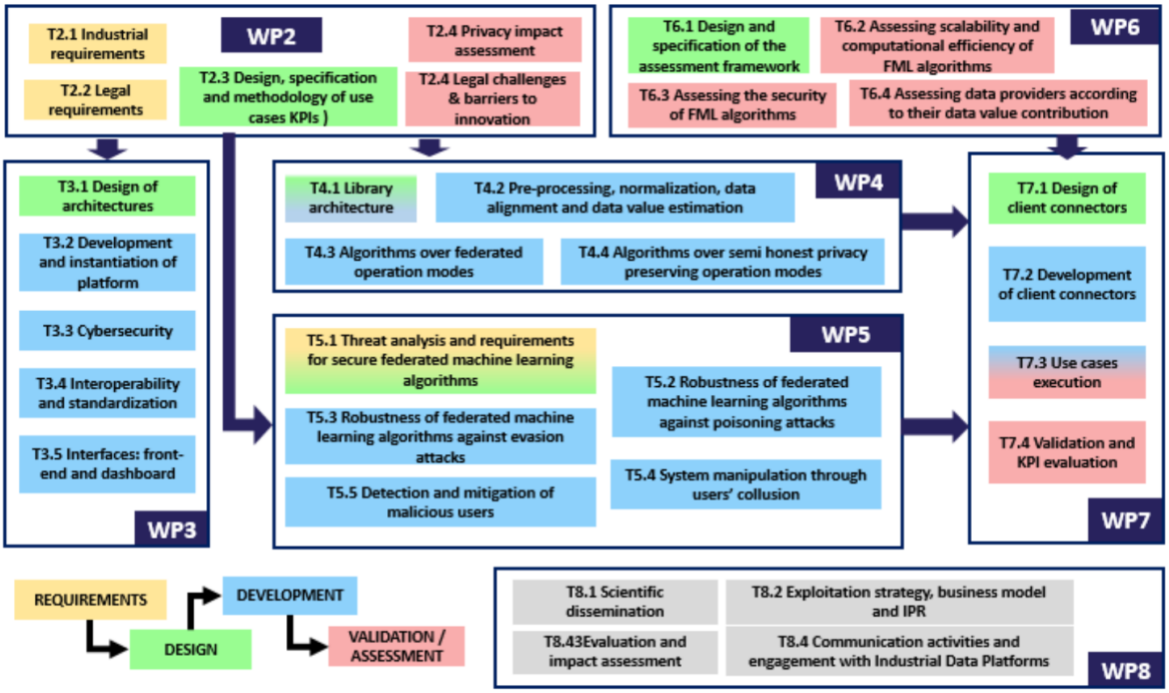

Figure 1: MUSKETEER's PERT diagram

1.3 Outline

The remainder of this document is structured as follows:

- Section 2 reviews the relevant functional and non-functional requirements related to WP3 for the final prototype.
- Section 3 describes the platform additions since D3.2 and D3.3, that support the full feature set required by the use cases.
- Section 4 provides a walkthrough of the demonstration that is based on the activities of the MUSKETEER hackathon.
- Finally, Section 5 concludes the WP3 work.

2 Requirements

As this document is the final deliverable for WP3, this chapter presents the requirements (as discussed in D2.1), highlighting the progress on each individual requirement.

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 section builds upon D3.2 and details the requirements that are now satisfied in the final prototype.

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 and D3.2, for ease of reference. Requirements already satisfied in D3.2 (highlighted in green text), are still operative unless otherwise specified. For each requirement, the ID is highlighted as green, if the final prototype, described in this document, satisfies the requirement. As of D3.4 all requirements are now satisfied by the final prototype.

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 an 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**). See section 3.1.
FR013	Ability for general users to list all the Federated ML models; view their description, definition, KPIs etc. if available (**D2.1-FR011**). See section 3.6
FR014	Ability for general users to download trained Federated ML models (**D2.1-FR013**, **D2.1-FR026**). See section 3.6.

FR015	Ability for a task creator to delete the Federated ML models trained as part of that task (**D2.1-FR021**). See section 3.6.

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. See section 3.2.
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. See section 3.2.
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. See section 3.4.
FR027	Ability for an aggregator to store information regarding the data value contributions per participants. See section 3.5.

2.2.3 Non-functional requirements

The non-functional requirements specified in D3.2 are repeated below. The features backing these requirements were monitored and validated during the Hackathon, where a number of external parties collaborated on the platform to build federated machine learning models.

Table 4: Non-functional requirements

ID	Description of the requirement
NR001	High availability (**D2.1-NR001**). See section 2.3.
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**). See section 2.3.
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**). See section 2.3.
NR006	Scalability, specifically the efficient execution of Federated ML training algorithms (**D2.1-NR006**), and efficient handling of simultaneous requests (**D2.1-NR014**). See section 2.3.
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**).

2.3 Summary

During the hackathon (described in section 4.2), the platform was shown to be always available over the course of several days of intensive use (**NR001)**. Any errors that did occur, for example, duplicate user registration entries were correctly reported back to the initiating user (**NR003)**. The platform also scaled sufficiently to support the hackathon parties (**NR006)**. More details on the hackathon can be found in section 4.2. Other features, such as **NR005**, are satisfied by recent enhancements, such as model lineage, see section 3.4.

Over the course of the remainder of the project these important requirements (scalability, availability etc.) will be monitored and KPIs provided to D7.5/6.

3 Feature Completion

Whilst the architecture (finalised in D3.2 at M18 of the project) is unchanged, several platform features were added subsequently. Some of these features were identified quite early in the project but were not required for the initial prototype (at M18), and others evolved from discussions held during the project mid-term review.

To recap, the architecture is based on the Publish / Subscribe Design Pattern [1], and interoperability between components (cloud-based and remote) is through a messaging system, backed by RabbitMQ [2]. 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. The messages are constructed within APIs inside the *Federated Machine Learning Framework* (FMLF) package [3]. The required information per-message is detailed in D3.2. A discussion about the overall platform architecture is available in D3.1/2.

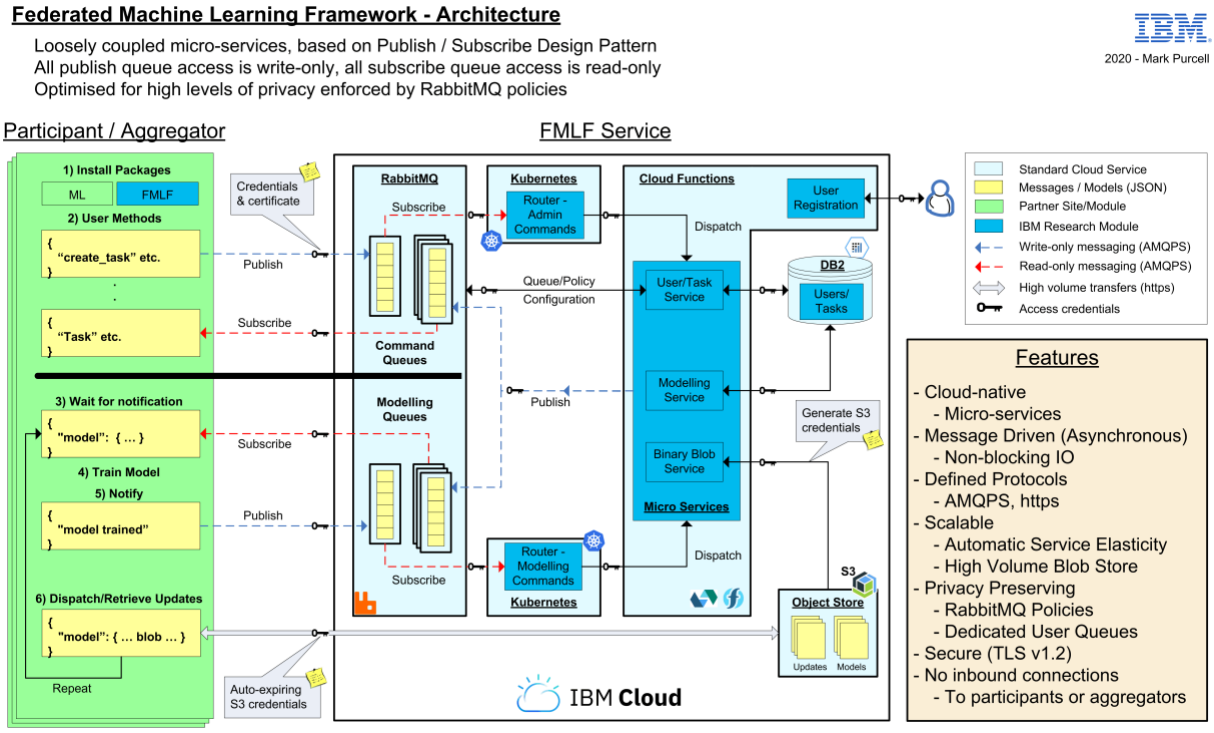

Figure 2: Architecture

What now follows is a feature-by-feature discussion of the features implemented after the initial prototype at M18. If a new or updated API is discussed, this is already implemented and available as of D3.4.

3.1 Tasks

Fulfilling the requirements in relation to task management, a modification to the existing *StopTask* feature is now available. This permits a new status to be sent to the service, "CANCEL" that aborts the current task. The API itself is unchanged.

3.2 Ring Topology

To complete the support for each aggregation procedure described in D4.3 (Section 2.2 - Data aggregation procedures), a ring or round-robin communication topology is required.

This topology exists in parallel with the star topology and can be used interchangeable during rounds of training. For example, training round one can use the star topology, round two, the ring topology and round three, back to the star topology.

The star topology is equivalent to a broadcast, whereby the aggregator dispatches a message to all participants in parallel, instructing them to start training. The *participant-training-round-complete* messages are routed back to the aggregator asynchronously.

The ring topology is similar to a round-robin, whereby the aggregator dispatches a message to a single participant, instructing that participant to start training. When that participant completes local training and responds with a *participant-training-round-complete* message, that message is routed to the next participant in the participant ring. The message from the final participant in the ring is routed to the aggregator.

The topology to use for a given round is an aggregator defined decision. This requires an update to the existing API.

3.2.1 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>", "<Topology>"]
    }
  }
}
```

Figure 3: 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)
- <Topology> - relates to POM type, e.g., “STAR” or “RING” (string)

3.3 Model Integrity

During federated learning, each participant trains a model based on local data and submits a model update to the platform. As model updates can be quite large, the platform supplies an object store and associated interface to upload/download these model updates. This object store provides a long-term storage location for models and model updates.

Note: General message flow for control plane and data plane operations are detailed in D3.2. Provided here is a zoom-in on the flow as it relates specifically to model updates:

1. Local training at the participant is complete
2. The participant user invokes the *task_update* function
    a. An object store location (key) for the model update is requested
    b. The model update is uploaded to the object store location
    c. A message is formatted with the object store key
    d. This message is published to RabbitMQ
3. The aggregator user receives a *task_update* notification
    a. Included in the notification message is the object store key
    b. The model update is downloaded from the object store location

Due to the asynchronous nature of operations on the platform, and the fact that the model update is not transferred directly to the intended recipient (aggregator), the potential could exist for the model update to be modified prior to download by the aggregator.

3.3.1 Checksum

To alleviate this and provide certainty that the model update downloaded by the aggregator is identical to that which was uploaded by the participant, a model checksum is included with the message dispatched by the participant as part of the *task_update* function. This checksum is then recalculated at the aggregator after the download and the checksums are compared.

Similarly, for models uploaded by the aggregator, participants also compare the checksum with the locally downloaded model.

The checksum employed is a cryptographic hash function based on SHA512 and the implementation of this can be found in the *pycloudmessenger/ffl* subdirectory in the open-source project [3].

3.4 Model Lineage

As aggregated models and participant model updates traverse the platform, it is possible to maintain a record of each of these activities. The full extent of these records on a task-by-task basis essentially results in a compilation of the activities that contributed to producing a complete model, i.e., the model lineage (or ancestry).

This lineage can be retrospectively queried after the federated learning task is complete, potentially providing a "replay" mechanism to the aggregator.

3.4.1 Basic Lineage

The minimal information required is to link the federated learning task (and joined participants) to the actual contributions made by each party. In essence, to provide a mapping of user (aggregator or participant) to the model object, as stored on cloud object store.

The following information is required to do this:

- Task id – the federated learning task
- User id – the user (aggregator or participant) initiating the activity
- External id – the key to the object store location
- Category – a model update, a complete model from the aggregator etc.
- Time stamp – when the activity was initiated
- Checksum – the checksum for the model (see above)

There are no API changes required to support this. It is fully encapsulated in the cloud micro-services. The checksum is handled within the client-side implementation in *pycloudmessenger*.

3.4.2 Participants Perspective

Additionally, each participant may wish to add specific information that relates to their contribution during each round of federated learning. For example, a participant could provide metadata in the form of a hashed string that reflects the data that was used during the round of training. Or it could be a string representation of a dictionary with various participant specific fields such as local optimiser state etc. This requires an update to the existing API.

3.4.2.1 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}>, "<metadata>"]
    }
  }
}
```

Figure 4: Participant Training Round Complete

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

3.4.3 Retrieving Lineage

A new API is required to retrospectively query this model lineage:

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

Figure 5: Model Lineage Command

```
{
  "service": {
    ...
    "data": [{
        "xsum": "<Checksum>",
        "metadata": "<Metadata>",
        "participant": "<ParticipantId>",
        "category": "<Category>",
        "object": "<ObjectId>",

    }]
  }
}
```

Figure 6: Model Lineage Response

- <Checksum> - the model's checksum (string)
- <Metadata> - participant specific information (string)
- <ParticipantId> - the participant (string)
- <Category> - type of model activity (update, complete) (string)
- <ObjectId> - the key to the object store location (string)

Note: the lineage returned differs depending on the invoking user. An aggregator receives a full lineage of all model activities, i.e., participant updates as well as aggregations. A participant user receives a lineage of their own specific contributions to the model.

3.5 Data Economy

One of the identified requirements for federated learning in the MUSKETEER project is to support an active data economy. In federated learning, this means providing the capability to an aggregator to assign value to a given participant's model contributions. Refer to chapter 5 in D4.3 for a detailed discussion on data value estimation.

In order for an aggregator to derive value from a participant's model contributions, and assign rewards to chosen participants, a number of additions to the model lineage mechanism are required. The model lineage feature provides a mechanism to record all model activities, updates and aggregations. Alongside these activities, additional information such as a value assessment or a reward can also be recorded. This results in a full record of all value assignments and rewards which can be reviewed by the aggregator.

The following information is added to the model lineage:

- Contribution – an aggregator defined/assigned value to a participant
- Reward – an aggregator assigned value to a participant

3.5.1 Assigning Value

A new API is required to assign contribution values and rewards:

```
{
  "service": {
    "name": "ModellingService",
    "args": {
      "cmd": "ValueAssignment",
      "params": ["<TaskName>", "<ParticipantId>", "<Contribution>", "<Reward>"]
    }
  }
}
```

Figure 7: Value Assignment

- <TaskName> - the name of the task (string)
- <ParticipantId> - the participant to assign value to (string)
- <Contribution> - the aggregator assigned value (JSON)
- <Reward> - an optional reward (JSON)

3.6 Model Access Control

The MUSKETEER platform supports running many federated learning tasks in parallel. Upon success, each of these tasks results in a complete model. However, by default, this model should not be publicly accessible, but rather, access controlled with read access granted to a discrete group of users and write (and delete) access to a smaller group of users.

Built upon the model lineage feature, access control lists are maintained automatically to grant access to models on a task-by-task basis only to participants of the given task. Each task aggregator also has write-access to the model.

3.6.1 Model Listing

A new API is required to list all models:

```
{
  "service": {
    "name": "ModellingService",
    "args": {
      "cmd": "ModelListing"
    }
  }
}
```

Figure 8: Model Listing

```
{
  "service": {
    ...
    "data": [{
        "name": "<TaskName>"
    }]
  }
}
```

Figure 9: Model Listing Response

- <TaskName> - task name for which a model is available (string)

This returns all models that are available on the platform.

3.6.2 Download Model

This API is unchanged from previously. However, internally, if access to the model requested was not granted, an access violation message is returned. Additionally, the response now also includes the checksum field, so that upon download, the model's integrity can be ascertained.

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

Figure 10: Get Model Response

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

- <Checksum> - the model's checksum (string)

3.6.3 Delete Model

Models can also be deleted. But this is access controlled, with access to the API restricted to the task aggregator only.

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

Figure 11: Value Assignment

- <TaskName> - the name of the task (string)

3.7 Towards Accountability of Federated Learning

With the addition of the model lineage enhancements discussed in section 3.4, there now exists a mechanism to review retrospectively the ancestry of a model. At a minimum, this enables a fully traceable, auditable review of how a model was trained.

Additionally, by providing the required underlying features and software for this lineage, several points in the code are now identified as locations whereby further information related to lineage or audit could be recorded. For example, future extensions to the MUSKETEER platform could now include a blockchain-style accountability framework. This could provide even more reliable accountability records, detailing all activities that occurred during the training phases.

Harnessing Federated Learning with such accountability mechanisms may become, going forward, a critical capability for the acceptance and deployment of AI models that are trained via federated learning mechanisms. For example , deployment in highly regulated and/or mission-critical contexts, certification of AI models by independent accredited bodies, and adherence to accountability as prescribed by the European General Data Protection Regulation (GDPR).

4 Demonstrations

This section describes the final demonstration [4] of the platform with an emphasis on the advanced functionalities that were introduced within this deliverable. In particular this demonstration describes how the model lineage capability can be incorporated for any federated learning systems built using the Musketeer Machine Learning Library (MMLL) [5]. Finally, it concludes with the description and outcomes of a real-world demonstration where the platform was used to enable the different aspects of a Hackathon.

4.1 Basic demo

The basic demo builds on the synthetic dataset example described in D3.3. As part of this demo a CNN classifier is trained collaboratively across multiple clients. Each participant owns a private dataset comprising of random samples from the MNIST dataset. This demo differs in its use of MMLL [5] and is inspired from MMLL-demo [6]. However, as before, the demo is driven by Python scripts which the aggregator and participants execute from their respective terminal windows.

In order to run this demo, it is required that a copy of *mnist_demonstrator.pkl* file is obtained and stored in the *input_folder*. The commands for aggregator and participants are as follows:

```
python pom1_NN_master_pycloudmessenger.py --user <user> --password
<password> --task_name <task_name>

python pom1_NN_worker_pycloudmessenger.py --user <user> --password
<password> --task_name <task_name> --id 0

python pom1_NN_worker_pycloudmessenger.py --user <user> --password
<password> --task_name <task_name> --id 1
```

The configuration and dependencies for this code base have been described in D4.4. Specific details for POM1 can be obtained from D4.3. Scripts are provided for aggregator and participants that allow them to perform the various steps required for end-to-end training within a federated learning system. This includes steps like starting a task, registering participants etc. as detailed in D3.3.

4.1.1 Model Lineage

In order to demonstrate the use of model lineage capabilities, the `Task_Manager` class of `demo_tools/task_manager_pycloudmessenger.py` of MMLL-demo was suitably adapted to include the added functionality as shown in Figure 12. Similarly, the aggregator script `pom1_NN_master_pycloudmessenger.py` was adapted to include an extra API call after the end of model training.

```
def print_lineage(self, user_name, user_password, task, display, logger, verbose=False):

    aggr_context = ffl.Factory.context('cloud', self.credentials_filename, user_name, user_password)
    aggr_user = ffl.Factory.user(aggr_context)

    with aggr_user:

        result = aggr_user.model_lineage(task)
        training_round = 0

        display(f"{'Round':5} {'Date':30} {'Origin':20} {'Id':10} {'Contribution':20} {'Reward':10}", logger,
                verbose)

        for line in result:
            if 'genre' in line:
                if line['genre'] == 'INTERIM':
                    training_round += 1
                    display(f"{training_round:^5d} {line['added']:30} {'AGGREGATOR':20} " +
                            f"{str(line['external_id'][-7:]):10}", logger, verbose)
                elif line['genre'] == 'COMPLETE':
                    display(f"Done  {line['added']:30} {'AGGREGATOR':20} " +
                            f"{str(line['external_id'][-7:]):10}", logger, verbose)
                else:
                    display(f"{training_round:^5d} {line['added']:30} {line['participant']:20} " +
                            f"{str(line['external_id'][-7:]):10} " +
                            f"{str(line['contribution']):20} {str(line['reward']):10}", logger, verbose)
            else:
                training_round += 1
                display(f"{training_round:^5d} {line['added']:30} {line['metadata']:20} " +
                        f"{str(''):10} " +
                        f"{str(line['contribution']):20} {str(line['reward']):10}", logger, verbose)
```

Figure 12: Model Lineage API call in Task_Manager

Figure 13 shoes sample model lineage obtained at the aggregator's end for a simple demonstration consisting of 2 clients which train for a total of 3 epochs.

```
2021-01-29 15:27:10 - root - INFO - Terminating all worker nodes.
2021-01-29 15:27:10 - root - INFO - POM1_CommonML_Master: Sent STOP to all workers
2021-01-29 15:27:14 - root - INFO - Round Date                          Origin              Id        Contribution        Reward
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:05.945848Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:06.683153Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:14.901632Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:15.142220Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:23.223570Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:24.572582Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:33.435890Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:36.760720Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:46.110749Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:48.253585Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:57.988672Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:26:59.614929Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:27:09.879999Z   74448896:12255232   dc61fd5   None                None
2021-01-29 15:27:14 - root - INFO -   0   2021-01-29T15:27:12.828055Z   74448896:12124160   237655a   None                None
2021-01-29 15:27:14 - root - INFO - ------------------------------------------------------------------------
2021-01-29 15:27:14 - root - INFO - ------------------------ END MMLL Procedure ------------------------
2021-01-29 15:27:14 - root - INFO - ------------------------------------------------------------------------
```

Figure 13: Example of Model lineage after two rounds of training

4.2 Hackathon

As part of the MUSKTEER program, a two-day hackathon was organised on 24-25th November 2020 (see section 7 for the full agenda). A federated machine learning environment with compromised clients was set up and numerous teams worked towards developing defence mechanisms to combat these attacks. What follows discusses the agenda, problem statement and the outcomes of this event.

4.2.1 Agenda

The event was organised as a two-day remote activity which involved a combination of talks and hacking sessions for problem solving. People from different geographies participated in the event with a total of three teams (Team A, Team B, and Team C) who registered to compete for building the most effective defence algorithm to tackle unknown attack scenarios. Over the two days the teams were presented with four scenarios with varying levels of complexity and system compromise. Each team was assigned a mentor to guide them through the hackathon process. The quality and creativity of the proposed algorithm along with the performance were used to assess the hackathon winners. The details of the setup, the attack scenarios and the evaluations are provided in the next sections.

- Day 1: Teams were introduced to the basic of federated machine learning along with detailed description of the POMs within MMLL. This was followed by a session with the mentors who helped them with the setup and installation of a federated learning environment. Finally, mentors helped them with the first round of evaluations on basic attack scenarios. With the knowledge and feedback obtained from these evaluations, the participants were in a position to devise defence algorithms. The remaining day consisted of hacking sessions where teams worked on their algorithms under the guidance of their mentors.
- Day 2: The teams were provided additional evaluations on the attack scenarios while they continued developing and improving their solutions. The teams found the engagement over the interactive sessions with the mentors to be very beneficial in hypothesising possible attack scenarios and devising solutions. Finally, the final set of evaluations were performed over the different attack solutions and the winning solution was selected based on the obtained numbers along with the quality assessment of the algorithm.

4.2.2 Setup and Problem Statement

A common federated learning setup was adopted for the different attack scenarios of the hackathon. This consisted of 10 workers which collaboratively trained a deep learning model for MNIST digit classification task for a total of 20 communication rounds. The benign workers used their private copy of randomly sampled images MNIST dataset. The Hackathon teams assumed the role of an aggregator. They were subsequently notified that their proposed methods would be evaluated against three scenarios with increasing levels of complexity.

- Scenario 0 : none of the 10 workers were malicious
- Scenario 1 : 2 of the 10 were faulty clients and supplied provided noisy updates to the aggregator
- Scenario 2 : 2 clients were faulty and 2 compromised global convergence with a label flipping attack.
- Scenario 3 : 4 clients behaved maliciously and employed an indiscriminate model poisoning attack by optimising for a sign-inverted loss function thereby maximising the loss function as opposed to minimising.

The teams' objective was to design aggregation protocols which can defend against these attacks. The specifics of the attack scenario as described above remained undisclosed to the teams.

In order to enable the learning across different geographies, MMLL with pycloudmessenger was used through the hackathon. The code provided to each team was based out of the Neural Network demo within POM1 and is publicly available [7]. This code includes files and instructions for easy injection of robust defence methods and python scripts for launching aggregators that initiate and orchestrate the overall training process. Furthermore, a set of user accounts were provided to each team to help them communicate with the cloud services. The respective mentors of each team acted as attackers and joined the tasks initiated by the teams with 10 client processes for each scenario.

4.2.3 Evaluations

The complete solution proposed by each team was evaluated across all four scenarios. The obtained accuracies on a set of benign MNIST images is reported in Table 5. In the absence of a robust method, the system employed model averaging as the aggregation scheme. The high accuracies for Scenario 0 ensure that the robust method maintains performance in the absence of malicious clients. It was noted that for Scenario 3 the accuracy fluctuated significantly across the different training rounds, and therefore an average across the last 5 communication rounds was used for comparing the performances.

Table 5: Hackathon results

Team	Scenario 0	Scenario 1	Scenario 2	Scenario 3
A	97.44%	97.42%	93.40%	Avg. 97.3%
B	97.46%	97.54%	94.48%	Avg. 77.11%
C	97.59%	96.50%	90.90%	Avg. 78.93%

The participating teams proposed a wide range of creative solutions to tackle the compromised training setups. However, Team A stood out as it achieved the best accuracy on Scenario 3 with a considerable margin and performed comparably if not better on other scenarios.

Notable, this winning solution used a combination of clustering and filtering to obtain reliable updates at the aggregator. Proposed protocols from other teams used similarity metrics and thresholding to filter outliers from the candidate updates.

4.2.4 Hackathon Conclusion

It was commented on by all, both external participants and organisers, that the event was very enjoyable and worthwhile. Perhaps especially so, given the difficult circumstances re: Covid-19 and the chance to collaborate widely, albeit within a virtual setting.

We look forward to further hackathons and experimenting on the platform with additional scenarios in the coming months.

5 Conclusions

In this deliverable, the final feature enhancements for the MUSKETEER platform were presented and their use described by means of a demonstration. This concludes the deliverable documents for WP3 and results in the completion of MS3.

All feature requirements, as initially presented in D2.1 are now implemented and available on cloud-based instances of the platform. These features have undergone significant testing, both from a traditional perspective as well as through demonstrations and intensive use during the hackathon.

The usability of the platform was shown over the course of the project, and particularly validated during the hackathon, where external parties with no prior knowledge of the system, were easily and quickly onboarded.

During some of the more recently added features, namely model lineage and data economy, it has become clear that accountability of federated learning will be very important future work. The platform has provided a strong basis to build upon, layering accountability modules on top of the existing model lineage feature. Accountability modules could leverage blockchain technology to provide fact-based provenance for federated learning. Such work would likely make a valuable contribution in light of auditing requirements and GDPR.

Finally, over the course of the remainder of the project, the platform will remain in utilised for the various use cases and will form a part of future demonstrations for other work packages. KPIs will also be monitored with a contribution made to D7.5/6.

6 References

[1] S. Tarkoma (2012). Publish/Subscribe Systems: Design and Principles, John Wiley & Sons, Ltd.
[2] https://www.rabbitmq.com/
[3] pycloudmessenger - https://github.com/IBM/pycloudmessenger
[4] D3.4 Demonstration - https://github.com/Musketeer-H2020/Demo_D3.4
[5] MMLL – https://github.com/Musketeer-H2020/MMLL
[6] MMLL-demo - https://github.com/Musketeer-H2020/MMLL-demo
[7] RobustMMLL - https://github.com/Musketeer-H2020/RobustMMLL-demo

7 Addendum

Included in this section is the hackathon information as used on 24th November 2020. Also included is the preparation session for team mentors on 20th November 2020. The workload for the hackathon was spread across multiple organisations in the consortium, reflected in the assignments below.

Hackathon Documentation and Agenda

**All times Irish time zone.

Eventbrite link: https://www.eventbrite.com/e/hackathon-shielding-federated-learning-against-attacks-tickets-126189703801

Closing date for registrations: November 19, 2020

Prep session: November 18, 2020 (zoom/webex) (**IBM**)

Prep session for mentors: November 20, 2020 (**Ambrish** will host and send an invite).

13:00-15:00 Technical prep –

Plan: share document with requirements beforehand; ensure that participants have a working environment (Python packages, GitHub repo); answer questions; facilitate test runs; onboard participants on Slack; test the new session

Outcomes: Hackathon participants have the installation and setup completed so we don't need to spend time on this on the first day. If they don't join this prep session and as a consequence don't have a working environment on the first day, it's time that they lose for the hacking phases.

Hackathon Agenda 1st day: November 24, 2020

9:00-9:15 – Hackathon welcome note [**Gal**]

9:15-10:00 – Technical Talk: Introduction to Federated Learning and MUSKETEER [**Roberto**]

Plan: Organisers provide a general introduction to Federated Learning in general (concepts, roles etc.) and the research under the MUSKETEER project in particular, along the lines e.g., of the BDVA presentation

Outcomes: general talk, intended to motivate the importance of FL, introduce general concepts / terminology, provide intuition about the working of the platform

10:00-10:45 – Hackathon rules, guidelines, general instructions, Q&A [**Ambrish**]

Plan: Rules with details on number of teams, overview of the agenda, details of comm channels, a walk though of the instructions, and logistics of evaluation, the assignment of group mentors and details on breakout rooms will be provided

Outcomes: By the end of this session, it is expected that participants should be ready to dive into the code and organisers should be ready with their breakout room setups

10:45-11:00 – Break

11:00-13:00 – [Breakout rooms] Hacking phase I [**Giulio, Zaid and Roberto**]

Plan: This session will begin in the breakout rooms where assigned mentors will work with the teams to help them run the FL setups.

Part1:

- Each group executes a vanilla FL task (**Scenario 0**).
- The participants will act as task creator/aggregator and clients and will run FL via pycloudmessenger using fresh credentials provided to them by their mentor.
- Any remaining technical issues (e.g. missing dependencies) can be resolved here (normally, if participants attended the prep session on November 18, there shouldn't be any).

Part2:

- Each group will be exposed to a malicious scenario with failure modes (**Scenario 1**).

- The participants (one of them) will act as aggregator.
- Organizers will host all the clients, some of which are malicious.
- The participants will be able to observe the degraded performance of the aggregator.

Part3:

- Participants will work on robust aggregator on their own. For local development and testing, they can either connect among themselves through pycloudmessenger, or run tests locally on their laptop via the local comms.

Outcomes:

Part 1:

- participants will have a hands-on experience with conducting FL via the platform end-to-end, and we ensure their installation works.
- They will see performance of the vanilla aggregator with no malicious participants

Part 2:

- After this, the participants will understand the failure modes; in particular, they will see the degraded accuracy of the vanilla aggregator.
- We give them a pointer why / where (in the code) this is failing (namely, where a plain average of updates is taken).
- So the participants will be motivated to defend against malicious clients and know where to get started.

13:00-13:45 Lunch break

13:45 - 16:00 Hacking phase 2 [**Giulio, Zaid and Roberto**]

Plan:

- One organizer/mentor is available on standby for each group, be available on Slack and check-in once per hour.
- Organizers communicate among themselves (how are all the groups doing, do we need to lower / raise the bar, provide hints etc.) via private Slack channel and/or dedicated call.
- An optional evaluation on **Scenario 1** can be hosted by the mentors for their respective groups at 14:15 pm.

- The required, End-of-Day-1 evaluation (still on **Scenario 1)** will be conducted by the mentors at 15:45.

Outcomes: Participants receive feedback (use the same **Scenario 1** throughout the first day, so they'll get a sense of achievement). Organisers/mentors touch base on progress.

16:00 – 16:30 Day 1 debrief

- Participants can share impressions, lessons learned
- Q&A
- Qualitative feedback
- Brief participants on what to expect on Day 2 (more challenging scenarios that would penalize overfitting to specific attacks)

Agenda 2nd day: November 25, 2020

9:00 – 9:15 – Recap from Day 1 and outlook on the day [**Ambrish**]

9:15 – 13:00 - Hacking Phase 3 (small groups). [**Giulio, Zaid and Roberto**]

Plan: Optional evaluation (on Scenario 2) at 9:15 is provided to motivate final evaluations.

13:00 - 13:45 – lunch break + final evaluations [**Giulio, Ambrish, Mark, Zaid, Luis, Roberto**]

Plan: Collect final evaluations on Scenario 0, 1, 2, 3. [Decision on how to decide the final leader boards based on the results]. Also ask the participants to walk us through their solution (explain their algorithm(s)).

14:00 – 14:30 - Technical talk on robustness of federated machine learning [**Luis**]

Plan: Technical talk on robustness of federated machine learning.

14:30 – 15:00 – Attack Scenarios used in the Hackathon [**Ambrish and Zaid**]

Plan: This talk includes details about the actual attack scenarios that were used.

15:00 – 15:30 – Assembly, winner ceremony, virtual group photo, and closing remarks [**Mark and Gal**]