Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

D4.12 – Blockchain Tokenization and Smart Contracts - III

Revision Number	3.0
Task Reference	T4.4
Lead Beneficiary	IBM
Responsible	Fabiana Fournier
Partners	FBK, BOUN, ENG, GFT, HPE, and IBM
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2022-03-31
Delivered Date	2022-03-21
Internal Reviewers	GLA and UPRC
Quality Assurance	INNOV
Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Beatrice Plazzotta
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	**Role**[1]	**Author(s)**[2]
IBM	Lead Beneficiary	Fabiana Fournier, Inna Skarbovsky
FBK	Contributor	Bruno Lepri and Gabriele Snatin
GLA	Internal Reviewer	Richard McCreadie
UPRC	Internal Reviewer	Dimosthenis Kyriazis
INNOV	Quality Assurance	John Soldatos

Revision History

Version	Date	Partner(s)	Description
0.1	2022-01-15	IBM, FBK	First draft of ToC
0.1	2022-01-20	IBM	Revised ToC
0.2	2022-02-10	IBM	Initial contributions of Sections 2, 3, and 4
0.3	2022-02-18	IBM	Revision of Sections 2, 3, and 4
0.4	2022-02-20	IBM	Contributions to Sections 1
0.5	2022-02-22	IBM	Contributions to Sections 5 and 6
0.6	2022-02-28	IBM	Finalisation of Sections 1, 2, 3, 4, 5, and 6
0.7	2022-03-01	IBM, FBK	Executive summary
1.0	2022-03-06	IBM	First Version for Internal Review
2.0	2022-03-15	GLA, UPRC, GFT	Version for Peer review and Quality Assurance
3.0	2022-03-31	GFT	Version for Submission

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

[2] Can be left void

Executive Summary

In today's financial landscape, we are witnessing a growing need by financial companies for sharing insights and information on clients in a privacy-preserving and value-preserving way. However, this is a complex task as financial companies have only partial information on their customers, data are siloed within organizations, and privacy is not adequately protected.

Recent advances and the rise of new paradigms in computing science enable addressing these challenges by:

- Assuring that raw data never leave data repositories and are processed locally (no centralized storage of data)
- Moving the algorithms to the data and trade insights and not raw data
- Applying vetted algorithms (algorithms are checked, e.g., for data provenance, privacy violations, and biases)
- Leveraging decentralized architectures, such as blockchain

Federated Learning is an emerging approach that aims at establishing cooperation between a group of agents for the solution of a machine learning task, with the aim of improving each agent's performance without disclosing any data. Blockchain technology provides all network members with an immutable ledger of the assets managed in the blockchain assuring full provenance, single source of truth, and non-repudiation. Therefore, it is the natural platform to support privacy, trust, as well as a secure and verifiable environment for the execution of federated learning algorithms, and a platform for digital trading of the algorithm outcomes. Our Blockchain-based Federated Learning environment and Data Marketplace framework enables securely accessing, managing, and sharing/trading insights across financial and insurance institutions based on three pillars:

- A Random Forest-based federated and cooperative learning algorithm
- A secure execution framework, built on top of the Hyperledger Fabric blockchain, that provides a verifiable privacy-preserving computation environment for federated learning scenarios
- A Data Marketplace based on the Hyperledger Fabric blockchain that supports the managing and trading of the learning algorithm outcomes with digital tokens

This report provides in-depth technical details of the implementation of a Minimum Viable Product (MVP) that realises the *Blockchain based federated learning environment and data marketplace* introduced in D4.14 - Encrypted Data Querying and Personal Data Market – II. This is a collaboration between partners FBK and IBM that leverages the work carried out in the scope of the INFINITECH project in federated learning in the context of a fraud detection scenario, blockchain as an enabler for the secure execution environment of the learning algorithm process, and tokenization as a means for trading the algorithm results in a data marketplace. This report covers the blockchain aspects of the MVP, while D4.15 - Encrypted Data Querying and Personal Data Market – III, submitted at the same time, complements the technical details from the machine learning side.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Abbreviation	Definition
API	Application Programming Interface
BC	Blockchain
CA	Certificate Authority
CRUD	Create, Read, Update, Delete
D	Deliverable
DoA	Description of Action
ERC	Ethereum Request for Comments
IEEE	Institute (of) Electrical (and) Electronic Engineers
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
M	Month
ML	Machine Learning
MSP	Membership Service Provider
MVP	Minimal Viable Product
RESTful	Representational State Transfer
SDK	Software Development Kit
T	Task
WP	Work Package

1 Introduction

Towards the end of the first year of the project two of the partners working together on T4.4 and T4.5, FBK and IBM, became aware of the possibility of combining two promising technologies: Machine Learning (ML) and Blockchain (BC) towards what became a *Blockchain-based federated learning environment and data marketplace framework*. This collaboration gave birth to innovative research in the interlock between Federated ML (promoted by FBK) and BC (promoted by IBM) and leveraged the knowledge and previous work on digital tokens. The goal and scope of the deliverable in hand is taking the envisioned novel framework reported previously in D4.14 – Encrypted Data Querying and Personal Data Market – II to the next level – its realisation into a Minimal Viable Product (MVP) in the domain of fraud detection.

D4.14 gives the motivation that drove this joint work and presents the design of a generic framework in which BC and ML complement each other in the following manner:

- Federated Learning is an emerging approach that aims at establishing cooperation between a group of agents for the solution of a machine learning task, with the aim of improving each agent's performance without disclosing any data. This requires an organization to trust another organization with the local execution of the machine learning algorithm on the data residing in the premises of the latter. Since the computation is defined by one party (the algorithm's owner) and the data is owned by another party that also runs the computation, there might exist an inherent trust problem – how to ensure the correct computation is performed, and that the reported results are the corresponding outcome of that computation. The underlying assumption is that organizations participating in the federated learning process earn from the cooperation and sharing of insights and results while not revealing their own organizational data.
- A blockchain (BC) is essentially a digital ledger of transactions that is duplicated and distributed across the participants in the blockchain network. Transactions are recorded in a final and immutable manner by the blockchain, providing all network members with an identical and trustworthy real-time view of the state. Due to its inherent characteristics, blockchain is the natural platform to support privacy and trust as well as a secure execution environment. Our proposed BC solution ensures a secure, auditable, and verifiable framework for the execution of federated learning algorithms. The idea is that each learning node in the BC network publishes intermediate results (a.k.a. insights or estimators in the case of fraud detection) at the end of each iteration. These results can be consumed by other learning nodes to improve the accuracy of their next computations. The outcomes of the learning algorithm can be then stored in the BC, managed, and exchanged using digital tokens in a data marketplace. Note that these intermediate results comprised of estimators and scores are relatively of small size and therefore can be efficiently stored on BC.

While the design of the framework is fully documented in D4.14, in order to meet the structure and commitments of the Description of Action (DoA) the MVP realisation is presented in two separate but interconnected and complementary deliverables:

- D4.12 - Blockchain Tokenization and Smart Contracts – III led by IBM, documents the technical details of the MVP from the BC perspective
- D4.15 - Encrypted Data Querying and Personal Data Market – III led by FBK, documents the technical details of the MVP from the ML perspective

This work is a joint collaboration between FBK and IBM within the INFINITECH project. FBK has developed a fraud detection federated learning algorithm, run by a consortium of organizations in their own nodes and on their own data while sharing the intermediate learning results with each other to achieve higher precision. IBM provides a blockchain-based secure execution framework for the distributed fraud algorithm execution, recording all the meta-information regarding the execution, and recording the shared intermediate results on a transparent, immutable, and verifiable ledger. At the end of the execution, a completed model can be shared as a tradeable asset on a blockchain-based data marketplace also developed by IBM. The blockchain-

based data marketplace provides a tradeable assets catalogue, where consumer organizations can search for available assets and gain access by paying for those assets using tokens.

We have applied Hyperledger Fabric (simply Fabric) as the underlying blockchain technology for our proposed framework. Fabric is the BC technology chosen and applied in the INFINITECH project and the one leveraged towards tokenization mechanisms. We assume the reader is familiar with the concepts provided in the corresponding previous deliverables and specifically technical blockchain related concepts, with emphasis on Fabric. Therefore, description of concepts such as peers, organizations, orderers, certificate authority (CA), channels, and chaincodes (i.e., smart contracts in the Hyperledger Fabric jargon) are out of the scope of this document. For more details on Hyperledger Fabric, the reader can refer to [1] and [2].

Deliverable D.12 (D4.12) – Blockchain Tokenization and Smart Contracts – III constitutes the third and final iteration of the work performed under Task 4.4 (T4.4) - Tokenization and Smart Contracts Finance and Insurance Services.

1.1 Objective of the Deliverable

In our Blockchain-based federated learning environment and data marketplace framework, blockchain technology plays a dual role:

- Ensuring the secure, auditable, and verifiable environment for the execution of federated learning algorithms. BC ensures that the "correct" computation is performed and that the reported results are the corresponding outcome of that computation.
- Enabling the trading of the learning algorithm outcomes by leveraging the usage of digital tokens in a data marketplace.

Our realisation of the proposed framework in a fraud detection use case MVP demonstrates the fulfilment of the above requirements.

The overarching goal of this deliverable is to document the technical aspects, both design and implementation, of the BC aspects of the MVP. It's worth noting that this document relates only to the BC side of the framework, and we don't address the correctness and validity of the federated learning algorithm (including the algorithm to run) nor to the precision of the model created as outcome of the execution of the algorithm. Those concerns are addressed in D4.14.

1.2 Insights from other Tasks and Deliverables

Deliverable D4.12 is released in the scope of Work Package 4 (WP4) - Interoperable Data Exchange and Semantic Interoperability activities and documents the outcomes of the work performed within the context of T4.4 - Tokenization and Smart Contracts Finance and Insurance Services. Specifically, we focus on the implementation details of the blockchain aspects of the framework for the federated learning environment and data marketplace introduced in the scope of D4.14 - Encrypted Data Querying and Personal Data Market – II, therefore heavily relying on this deliverable.

The MVP developed and described in this document leverages previous work on tokenization performed in T4.4 and therefore is closely related to the two previous deliverables on tokens:

- D4.10 – Blockchain Tokenization and Smart Contracts – II – ERC20
- D4.11 – Blockchain Tokenization and Smart Contracts – II – ERC1155

1.3 Structure

This document is structured as follows:

- Section 1: introduces the document, describing the context of the outcomes of the work performed within the task and highlighting its relation to other tasks and deliverables of the project.
- Section 2: provides the overall architecture design of the MVP.
- Section 3: details the implementation of the MVP.
- Section 4: details the provenance aspects of our solution.
- Section 5: summarizes the main contributions of the work and addresses potential extensions; and
- Section 6: concludes the document.

2 MVP architecture

Section 2 presents the overall detailed architecture of the MVP implementing the blockchain based federated learning environment and data marketplace introduced in D4.14. We depict the building blocks and the technological stack of our solution; the execution flows characterized by the different phases and managed assets of the federated learning execution and trading process; the API service; and the chaincodes.

2.1 Blockchain-based architecture for federated learning

In our previous deliverable D4.14, we have addressed the different building blocks and the flows comprising the secure execution framework for federated learning and data marketplace. Specifically, we have outlined the various logical components used in the framework: (i) the c*ontainer registry*, where the images of the ML algorithms are stored, (ii) the *python tasks* that support the creation of computational tasks from the ML algorithm image on the learning nodes, and (iii) the blockchain network along with the appropriate chaincodes that manage the specific artifacts (addressed in next sections): *image record*, *learning process record*, *execution task record*, and *model record*. We have also defined the concept of data marketplace and suggested how to implement it, allowing access to the artifacts stored in the *model* chaincode by trading them for tokens applying ERC20 or ERC1155 *token* chaincodes. Important to note that the tokens chaincodes have been designed and developed in the scope of D.10 (ERC20) and D.11 (ERC1155) and therefore not specified in this report. Our proposed framework leverages these token chaincodes as part of the process of purchasing assets from the data marketplace as described in the subsequent section.

However, when working on the MVP design for the model chaincode and its interaction with the tokens chaincode, we have decided to extend the initial design. Instead of the *model* chaincode we introduce the *data marketplace* chaincode. The Data marketplace chaincode is responsible for two main functionalities:

- Storage of tradeable assets including models and other organizational data assets for trading with external organizations. This functionality was the original role designed for the model chaincode.
- Managing access for tradeable assets. The data marketplace chaincode is responsible for checking whether the requesting user (i.e., an external organization) has granted access to a particular asset. If so, it creates an access request in *granted state* after checking that: (a) there are enough tokens in the user's account to pay for the asset and (b) that the required amount of tokens is transferred to the owners' account. The chaincode is also responsible for marking access requests as *rejected* in case such transfer cannot be completed, or per owner's request. In addition, this chaincode also provides provenance history of access requests for a particular asset or assets for a particular organization.

The updated solution architecture for the BC-based secure federated learning environment and data marketplace is shown in Figure 1 (refer to D4.14 for further information on the architecture).

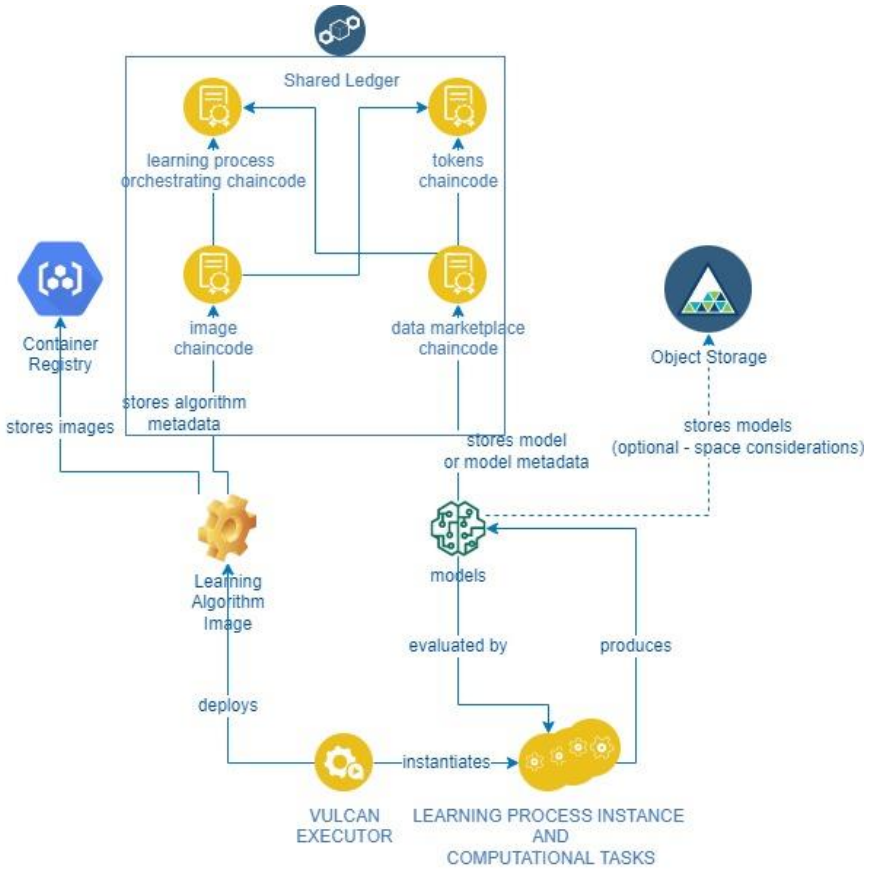

Figure 1 - BC-based secure execution environment and data marketplace architecture

2.2 Scenario and execution flows

The proposed BC-based framework supports the three phases of the federated learning execution and trading process:

- Creation of the learning algorithm and its publication in blockchain
- Execution of the chosen learning algorithm by a consortium of collaborating organizations with the goal of running a common learning process on their own private data without sharing the data but sharing only the computation results.
- Assets' trading in the data marketplace that stores the catalogue of data assets, creates access requests, and grants permissions in return for digital tokens.

Note that these three phases verification of image, execution of image, and publishing of results are initiated and executed by the same entity, therefore preventing fraudulent activities such as executing incorrect image by one of the participating organizations.

The execution scenario for the MVP is depicted in Figure 2 and described henceforth. It is also animated and shown step by step in the demo movie in the project's YouTube channel and accessible in the project marketplace at:

- https://www.youtube.com/watch?v=J7ekCHSoWrg&list=PL9suUK-Ys8V3Dkzm7qmZnIb-VIc1eSMwp&index=22.

In the figure below, we can see the three main building blocks of our solution: the blockchain network, the docker image repository, and the Application Programming Interface (API) service, which is presented in detail in section 2.4. We also show the nodes of the three different organizations that participate in the federated learning process supported by the blockchain.

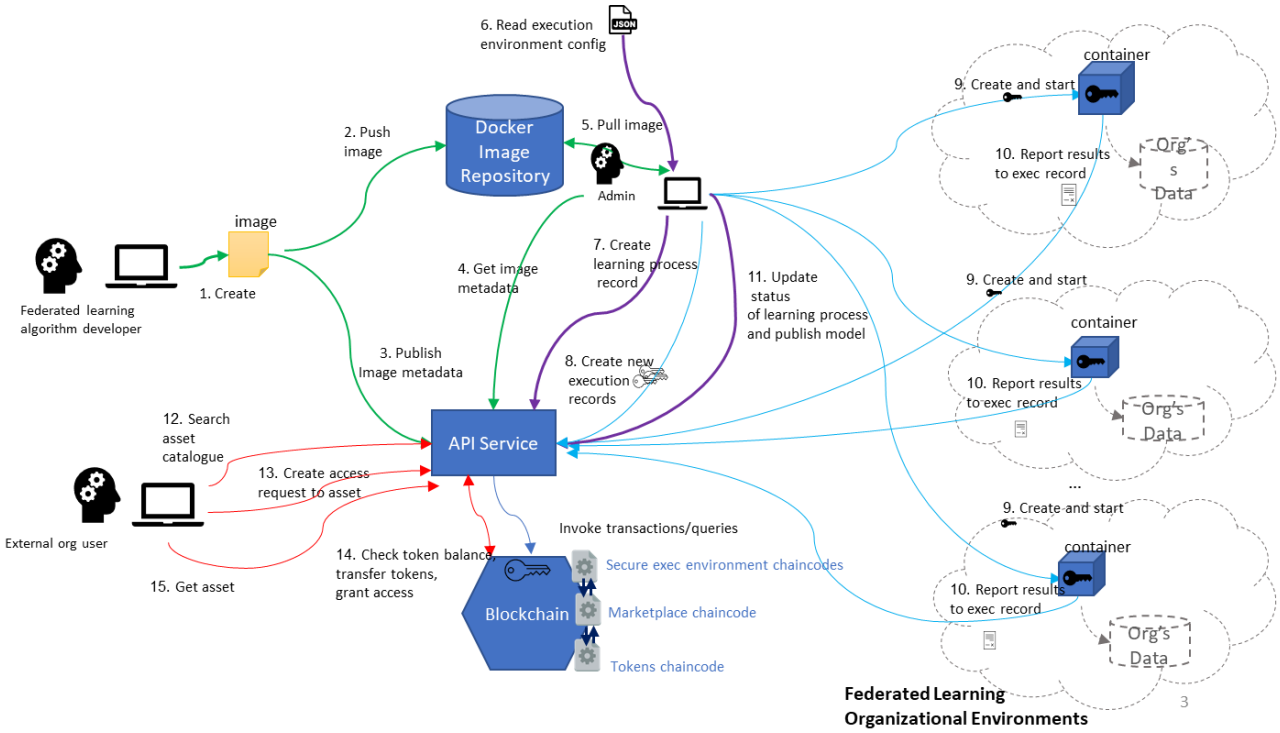

Figure 2 - MVP end to end execution scenario (fraud detection ML algorithm)

The overall flow demonstrates the federated learning and model trading scenario from beginning to end: the starting point is the federated learning algorithm development; converting it to docker image; publishing this image to image repository and the image metadata to BC ledger; instantiating of the learning process execution based on this image and storing and reading each learning iterations' results from each node in a secure and verifiable manner; publishing the outcome of the federated learning (the model) to the data marketplace; and finally the trading of the model with an external organization in exchange for tokens. The colour coded arrows in Figure 2 denotes each of these steps and are described below.

Green arrows relate to *image* flow:

- Docker container image creation from fraud detection python algorithm (step 1).
- Pushing of the image to docker images repository (step 2).
- Publishing of the image metadata to BC (step 3).
- Reading the image metadata from BC (step 4).
- Pulling the image from the image repository (step 5). Note that step 5 is based on the output from step 4. We compare the hash stored on the BC with the hash of the image in the image registry to validate they are the same.

Purple arrows indicate the *learning process flow* while the blue ones indicate *execution tasks records* flows:

- As described in the last step of the image flow, the image hash of the image pulled from the docker image repository is validated against the corresponding image hash stored in the image metadata record to make sure the image is valid. Next, the learning process configuration information is read from the configuration file (step 6) and the learning process record is created on BC (step 7) containing metadata, such as the participating organizations and their nodes, and the algorithm image being executed.
- For each of the nodes participating in the federated learning, the learning process orchestration script creates an execution record containing relevant metadata of the instance of the computational task which is going to be instantiated on the node, such as, task id, image id, name of the node, and iteration number (step 8). Then, this computational task is instantiated and run on a node (step 9) reading the results of previous node's iterations from BC, reading the source data (local or any kind of other data source, e.g., cloud object), performing the computation, and publishing the result of

the current iteration to BC (step 10). This process is repeated per node and iteration until all nodes report completion of computation on their end. Once this happens, the whole learning process is completed, its status is updated in the corresponding learning process record on the blockchain, and the completed outcome of the federated learning (i.e., the final model), is published to the data marketplace with accompanying metadata (step 11).

Red arrows indicate the *data marketplace and tokenization flows*

- At the end of the learning process execution, a completed model is published on the BC, along with provenance metadata of the model, such as the organizations which are the owners of the model (organizations that participated in the learning process), the algorithm image this model was an outcome of, and the requested price in tokens for model purchase (see step 11 before).
- The Data marketplace catalogue provides functionality to search for data assets based on their metadata, such as: algorithm image the assets (in this case model) were based on, the owning organizations, and all data assets which are the outcomes of a particular learning process (step 12).
- The information returned in such queries returns the asset's metadata but not the "content" of the asset – which is the actual data asset to be traded (this could be a model as in our example, a file, or a binary).
- If an external organization wishes to acquire access to a specific data asset, it will create an access request. This access request is stored on BC (step 13).
- The data marketplace chaincode validates that (step 14):
    - The requesting organization is not part of the asset owners' organizations (otherwise no access request is required).
    - The requesting organization has enough tokens on its account to satisfy the requested price for the asset (this is done by invoking the tokens chaincode).

  In case both these conditions hold, the *tokens* chaincode is invoked to transfer the requested number of tokens from the buying organization's account to the owner's organization account and an access request in the state *granted* is created for the requested asset (model).
- When an asset is being read (step 15), the chaincode ensures that the accessing organization is either an owner or has a granted access request for this asset. In case this validation fails, the chaincode will respond with an error otherwise, it will return the full record of the data asset, both the metadata and the "content", that is, the asset or model itself in our case.

Note that there is another unnumbered arrow called "invoke transactions/queries" connecting the API service with the BC. As will be described in Section 2.4, all the interactions with BC are performed via the RESTful APIs of the API service that acts as the Fabric client submitting all transactions to BC. Transactions are an integral part of any invocation with BC. However, these interactions are not an "active" part of any flow but rather they act "passively" as background actions. Therefore, are shown in the diagram without any specific number.

2.3 Component architecture and technological stack

Figure 3 describes the MVP component architecture comprised of the following modules:

- Local docker image repository
- The python script representing the fraud learning algorithm, and dockerfile necessary for building the algorithm docker image
- The python scripts initialize the learning process based on the algorithm image
- API service enveloping BC client and exposing Representational State Transfer (RESTful) APIS (in the demo invoked by POSTMAN, cli and python APIs), provides an interface for interacting with the blockchain network
    - Javascript Express framework
    - Fabric Client Javascript Software Development Kit (SDK) v1.4

- Dockerized blockchain network Fabric 1.4 version
  - Solo ordering service docker
  - 2 Organization peer network, each organization represented by 2 peer nodes and 1 CA node.
  - Javascript Fabric chaincodes based on Fabric SDK v1.4 and deployed on the Fabric network
    - *Secure execution environment* chaincodes comprised from *image* chaincode, l*earning process record* chaincode and *execution record* chaincode
    - *Data marketplace* chaincode comprised from *tradeable_asset* and *access_request* chaincodes
    - *Tokens* chaincode

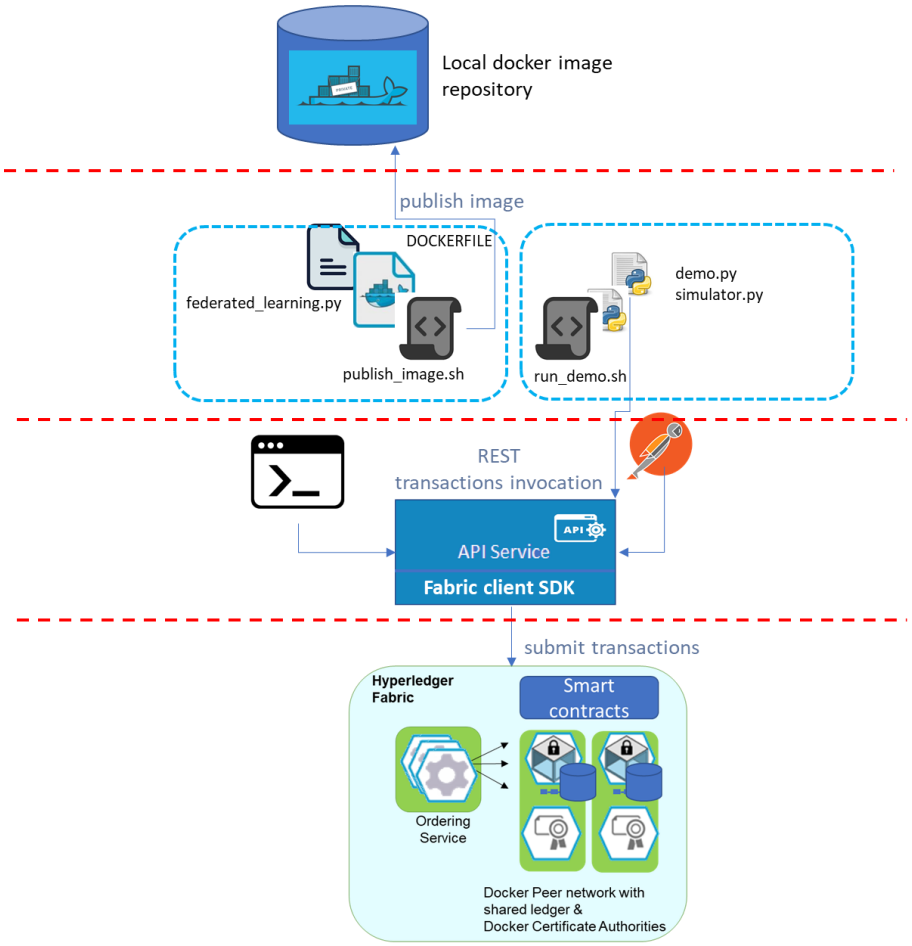

Figure 3 - MVP components

All the secure execution environments and data marketplace chaincodes are separate logical entities but packaged into one Fabric contract artifact which is deployed on the blockchain channel. The tokens chaincode is packaged into another such artifact, which is also deployed to the same blockchain channel. Such deployment allows inter-chaincode invocations and querying, and updating of the world state of one chaincode by another chaincode (for example, updating account balance when transferring tokens invoked by data marketplace when granting access to an asset to an external organization).

2.4 API service design

The API service is a NodeJS Express application, exposing RESTful interfaces for invocation of various chaincode transactions. The API service is a wrapper around Fabric NodeJS SDK client providing connection to the Fabric network. The API service architecture is shown in Figure 4.

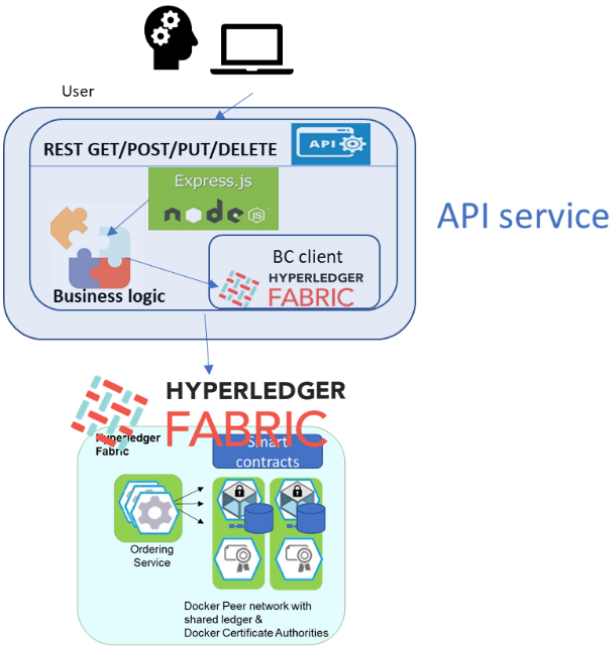

Figure 4 - API service design

The RESTful APIs exposed by the API service are shown in Figure 5.

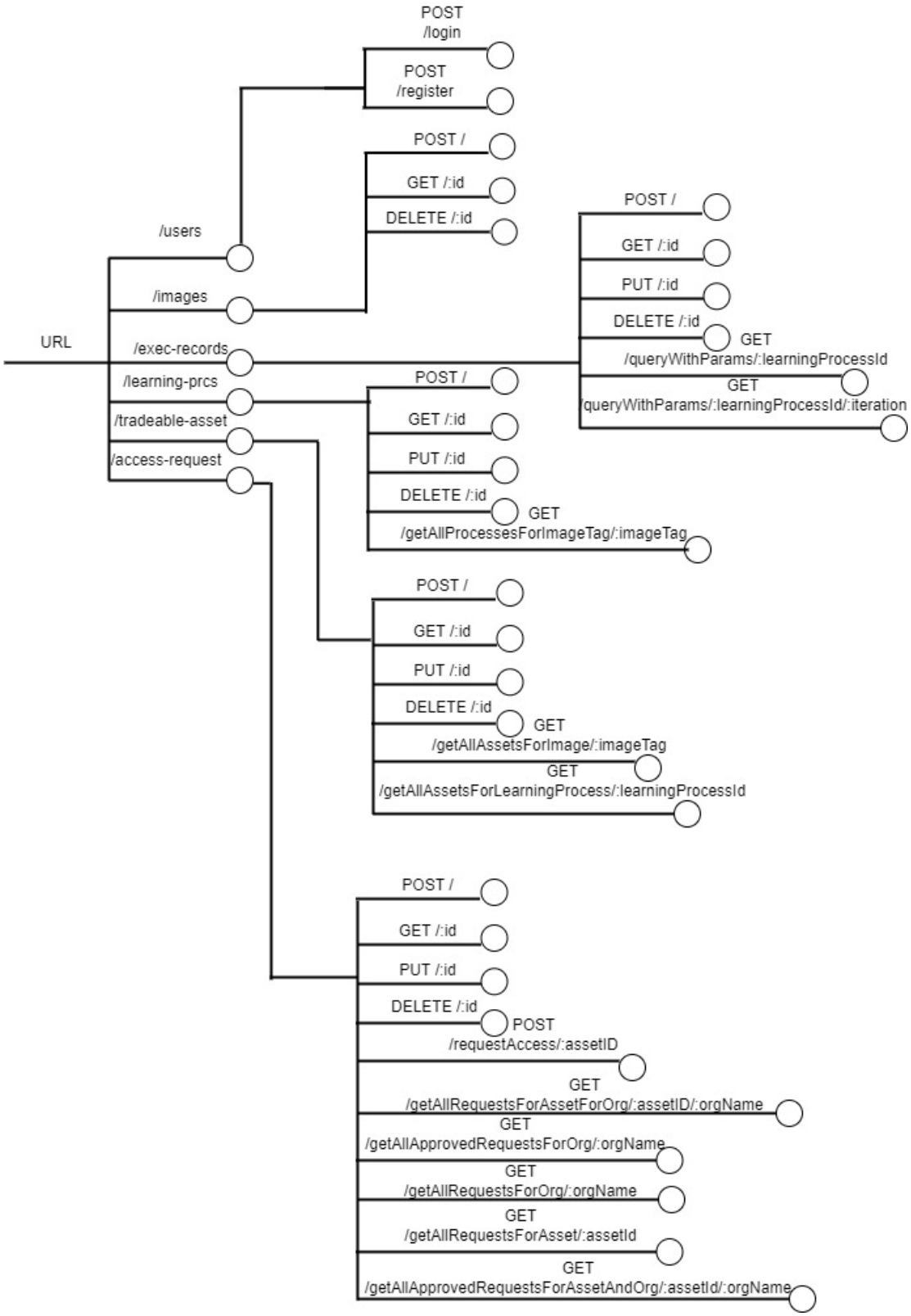

Figure 5 - RESTful APIs exposed by API service

The */users* route manages the authorization and authentication of the APIs users. In order to invoke the chaincodes, a client needs to be registered and enrolled to the blockchain network using a proper Certificate Authority (CA) of this organization. When a user is registered and enrolled, the CA issues an X509 certificate, public and private keys which are stored in an applicational wallet and in the blockchain Membership Service Provider (MSP). When a user logs in to the API service, existence of such a user certificate in the application

wallet is verified. If such a certificate exists, the application issues an AUTH token for the user which the user continues to use throughout the session.

The */users/login* API path takes care of such logging-in process for an already registered and enrolled Fabric user.

The */users/register* path is used by an authorized user to enrol and register other users as a Fabric user, who can later-on login and perform operations using the API service. Note that only an authorized user (i.e., a user that belongs to a BC organization and has a special permission to register other users) is allowed to perform enrolment and registration.

The other routes will be explained in detail in Section 3 when dealing with the relevant artifacts.

2.5 Chaincodes design

As aforementioned, we devised the following chaincodes comprising the framework: image chaincode, learning process orchestration chaincode (consisting of learning process and execution record chaincodes), and data marketplace chaincode (comprised from tradeable asset and access request chaincodes) and tokens chaincode.

The structure for all the chaincodes except tokens chaincode is depicted in Figure 6. As pointed out before, tokens chaincode has been devised at the beginning of the project and has been addressed extensively in D4.10 and D.11. We apply it in our framework for the trading of the outcomes of the federated as described in Section 2.2.

The chaincodes structure is based on the concept of *AssetStore* (implemented on BC ledger) which stores managed assets of a certain *assetType*. The basic functions of AssetStore are Create, Read, Update, and Delete (CRUD) for the managed asset, which is represented by a unique key, which is a combination of assetType + unique asset ID. This structure also allows querying the AssetStore for a particular asset based on query parameters and on partial key. The managed asset also has fields indicating its provenance such as creation and update timestamps of the asset record in the BC ledger.

Each chaincode is represented by a particular managed asset type: execution record (*ExecutionRecord* asset), learning process (*LearningProcess* asset), learning algorithm image (*DockerImage* asset), tradeable asset (*TradeableAsset* asset), and access request (*AccessRequest* asset). Each has its unique structure (information kept on the ledger), and also a set of unique queries per this managed asset type (see Figure 6).

The *FederatedLearningContext* is an extension of Fabric's shim `Context`[3] class and represents the unique context for interacting with the ledger (executions of CRUD operations on the ledger). It is extended to hold reference to the BC based asset store described earlier. The *FederatedLearningContract*, which is an extension of Fabric's shim `Contract`[4] class is the high-level entry point to the execution of all the described chaincodes and offers APIs to invoke all chaincode operations.

[3] https://hyperledger.github.io/fabric-chaincode-java/release-1.4/api/org/hyperledger/fabric/contract/Context.html

[4] https://hyperledger.github.io/fabric-chaincode-node/release-1.4/api/fabric-contract-api.Contract.html

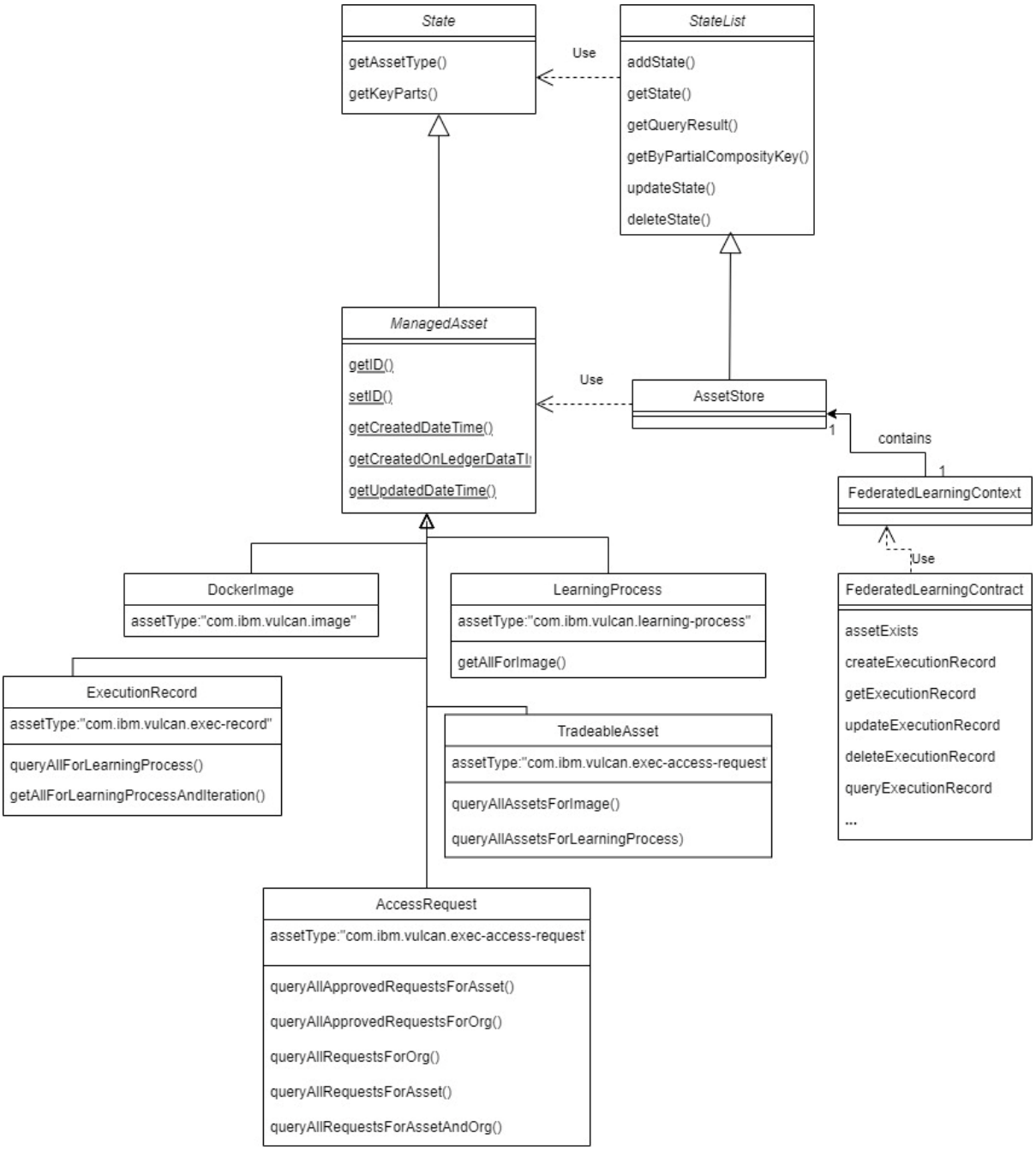

Figure 6 - Chaincodes structure

3 MVP detailed design and implementation

Section 2 portrays an overall view of the MVP architecture including the execution flows, chaincodes, and API service. In Section 3 we drill down into the specific artifacts created and managed through the different execution flows. We detail the design and implementation of RESTful APIs of the API service relevant for creating, reading, updating and deleting a managed artifact in the blockchain, the data models of these artifacts, and the different developed chaincode functions that operate on these artifacts.

3.1 Image flow and artifacts

3.1.1 Artifacts and execution flow

The fine-grained execution flow of the preparation of the algorithm image representing the fraud detection algorithm and publishing this image in the docker image repository and in the BC is described in Figure 7.

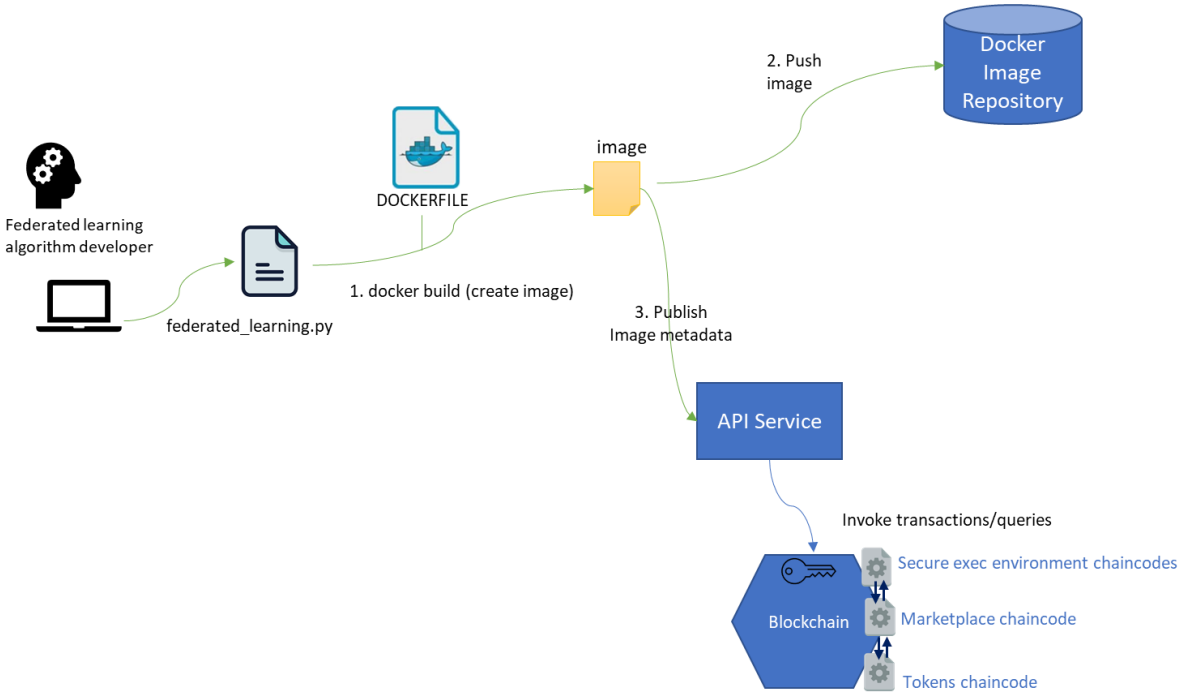

Figure 7 - Image flow

The following artifacts have been created/used in the MVP and have been orchestrated into the image flow:

- Python code of the fraud learning algorithm
- *Dockerfile* for the creation of the docker image of the python fraud learning algorithm implementation
- Build image script which runs *docker build* command to build the docker image from dockerfile
- Image repository for storing the built image for the fraud detection algorithm
- Chaincode implementing the image metadata publishing to/reading from/querying from BC
- API service routes and controllers related to image chaincode operations

The *federated_learning.py* is the fraud federated learning algorithm provided by FBK (more information on the specific algorithm can be found in D4.15). We used this particular algorithm in the MVP, but from point of view of the framework it is a black box implementing a python `main()` function, the framework can use any other algorithm as long as it has a runnable main method. The *Dockerfile* describes the necessary steps

for building a container image out of the algorithm, such as the base image, installing python prerequisite libraries and the entry point of python code to run. The image is built with the *docker build* command, after which the image is pushed to the docker image repository. Finally, the image metadata is being stored in the blockchain ledger by using the API service RESTful invocation of the appropriate chaincode function for image record creation.

3.1.2 Implementation

3.1.2.1 Image record implementation

Figure 8 shows the RESTful APIs of the API service relevant for creating, reading, updating and deleting an image record on the blockchain.

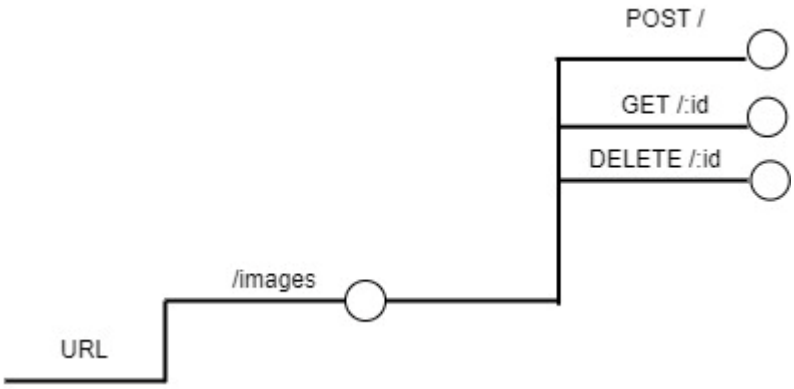

Figure 8 - Image record endpoints

The following RESTful APIs for image record are provided by the API service:

- POST <URL>/images/ - creates an image record by providing a JavaScript Object Notation (JSON) body element with the following information: ID representing the imageTag, and dockerImageID which is the hash of the docker image.

```
{
  "ID": "<imageTag>",
  "dockerImageID": "<dockerImageId>"
}
```

- GET <URL>/images/:id – reads the image record defined by the provided *id* parameter value from the BC.
- DELETE <URL>/images/:id – deletes the image record defined by the provided *id* parameter value from the BC. Only the current image record from the world state would be deleted, the provenance of this record is stored in BC transaction log and can be read and examined.

Table 1 presents the image record data model in the image chaincode.

Table 1 – Image record data model

Image Represent a learning algorithm image		
Field	**Type**	**Description**
`ID`	String	Image tag

`dockerImageId`	String	Image Identifier – a digest, which contains an SHA256 hash of the image's JSON configuration object
`createdDateTime`	Date	Image creation timestamp
`createdOnLedgerDateTime`	Date	Timestamp of create transaction invocation

We have three chaincode functions: `CreateIDockermage`, `DeleteDockerImage`, and `GetDockerImage`. Note that no update image function is provided as a new version of the algorithm would represent a new image (with different algorithm metadata).

- `CreateDockerImage (dockerImageJSON JSON)`: Performs the necessary actions in order to create a new image metadata entry on the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the image information.
- `GetDockerImage (imageId string)`: Reads the required image metadata entry from world state and returns the information in JSON format to the invoking client.
- `DeleteDockerImage (imageId string)`: Deletes the specified image

3.2 Learning process and execution records flows and artifacts

3.2.1 Artifacts and execution flow

Figure 9 demonstrates the instantiation and execution of the federated learning process.

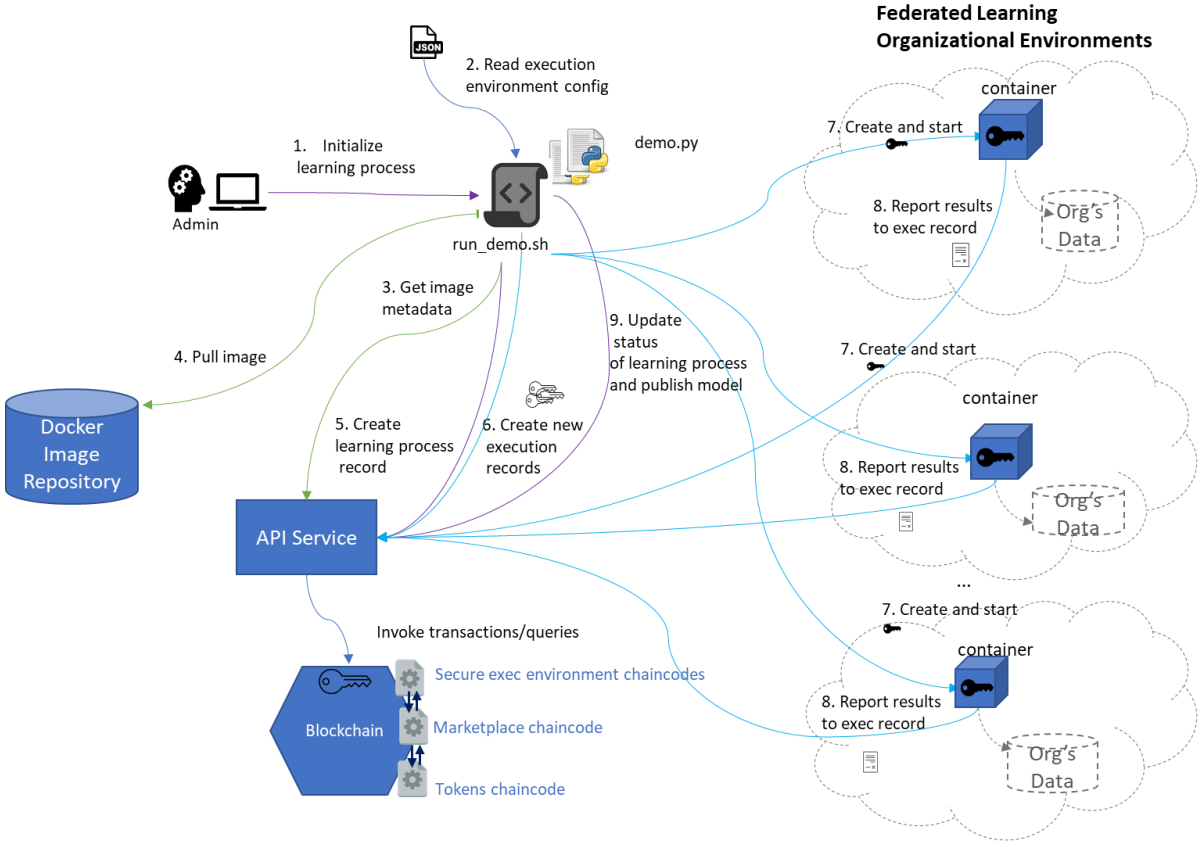

Figure 9 - Learning process and execution records flows

After the algorithm image is pushed to the image repository and its metadata is stored in the BC (see Section 2.2), the administrator or any other authorized user can initiate the federated learning process based on this algorithm. This user runs the script initializing the learning process. The following steps take place:

1. The Learning process execution script is run.
2. The information on the execution environment, such as the number of computation nodes, the owner organizations of computation nodes and information pertaining to the training configuration is read from the configuration file.
3. The BC is queried to fetch the required image metadata record.
4. Based on the image metadata (particularly the image tag), the relevant image artifacts is loaded from the docker repository, and its authenticity is verified when the image is pulled from the docker image repository comparing the image hash to the fingerprint from the blockchain record
5. Learning process record is created in blockchain. The Learning process record contains information relevant to the execution instance of the algorithm image – a unique ID of the learning process, the image this process executes, the executing organizations participating in the learning processes, the current status of the process (new/running/completed).
6. For each learning node and each learning iteration (in case the learning process is iterative), the following is performed:
   6.1. A key pair for the execution is generated – a signing key (private) and a verification key (public).
   6.2. A new execution record is created in blockchain with the following fields:
      - Execution identifier
      - Image identifier
      - Public Verification key
7. The docker containers instantiated from the algorithm docker image are deployed on the computation environment (in the MVP we instantiate a docker container per computation node, but those could also be instantiated on the cloud, for example as Kubernetes pods). The private signing key is incorporated in the container(s) and is used to sign the execution results.
8. The containers read their own organizational data from premises (in the MVP we used a local file system but in cloud environment, any organizational data source, such as cloud object store can be used), read previous iteration's results (estimators of other nodes) from BC, execute the current iteration's computation and publish the current computation results to BC, signed with the private signing key which was incorporated into the container at its instantiation. The chaincode writing the execution results on chain does so only after validating that the execution result reported by a specific container is signed by the correct private key corresponding to the public key saved in the execution record representing this container.
9. Steps 6,7,8 are repeated until all the iterations of the learning process are completed. The learning process status is updated to reflect this. The final outcome of the learning process, a completed model, is published on the chain as a tradeable asset. The owner organizations written in the model record in BC are the organizations which participated in the learning processes and, therefore, contributed to the model creation.

The following artifacts have been created/used in the MVP:

- The configuration file describing the organizations, nodes participating in the learning process, and paths to data sources
- `run_demo.sh` - Admin script for the instantiation of the learning process which is a wrapper around *demo.py* python program. It pulls the specified image from the image repository, accessing the BC through the API service RESTful APIs to store learning process metadata. In addition, it instantiates the docker containers for each node and orchestrates the execution of the containers until the federated learning completes.
- Image repository from which the specified built image of the fraud detection algorithm is pulled in order to instantiate the docker containers.
- Chaincode implementing the learning process and execution records metadata publishing/reading/querying BC

- API service routes and controllers related to the learning process and execution record chaincode operations.

3.2.2 Implementation

3.2.2.1 Learning-process record implementation

The relevant RESTful APIs of the API service for creating, reading, updating, and deleting a learning process record in the blockchain is shown in Figure 10.

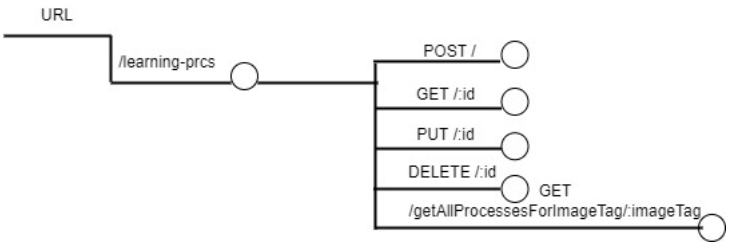

Figure 10 - Learning process record endpoints

The following RESTful APIs for the learning process record are provided by the API service:

- POST <URL>/learning-prcs/ - creates a learning process record by providing a JSON body element with the following information: ID representing the learning process unique ID, imageTag representing the unique ID (tag) of the image this learning process is going to execute, consortiumDef listing the consortium of organizations participating in the learning process.

```
{
  "ID": "<learning-process-ID",
  "imageTag": "<imageID>",
  "consortiumDef":["<Org1>","<Org2>","<Org3>"]
}
```

- GET <URL>/learning-prcs/:id – reads the learning process record defined by the provided *id* parameter value from the BC.
- PUT <URL>/learning-prcs/:id – updates the existing learning process record with the new record provided by JSON body
- DELETE <URL>/ learning-prcs /:id – deletes the learning process record defined by the provided *id* parameter value from the BC.
- GET <URL>/learning-prcs/getAllProcessesForImageTag/:imageTag – queries all learning process records and return the ones where the `imageTag` property is equal to the value provided in the parameter.

Table 2 presents the learning process record data model in the learning process chaincode.

Table 2 - Learning process record data model

Learning process Represent a learning process record.		
Field	**Type**	**Description**
`ID`	String	Identifier of the learning process instance.
`imageTag`	String	The identifier of the algorithm learning image used by this learning process.

`consortiumDef`	Array of String	A list of nodes' identifiers (orgs) participating in the federated learning.
`status`	Enum	An enumeration value identifying the current status of the learning process (COMPLETED, IN_PROGRESS, FAILURE).
`iteration`	Number	For iterative processes – current iteration

We have five chaincode functions: `CreateLearningTask`, `UpdateLearningTask`, `GetLearningTask`, `DeleteLearningTask`, and `GetLearningProcessesForImageTag`

- `CreateLearningTask (processData JSON)`: Performs the necessary actions in order to create a new learning process' metadata entry on the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the learning process information.
- `UpdateLearningTask (updatedProcessesData JSON)`: Updates existing process record in the ledger with new information. For example, it updates the iteration of the learning execution or the state of the process.
- `GetLearningTask (learningTaskID string)`: It retrieves information about the learning process designated by the ID
- `DeleteLearningTask (learningTaskID string)`: Deletes the specified learning process record designated by the ID.
- `GetLearningProcessesForImageTag (imageId string)`: Retrieves all learning processes for a particular algorithm image.

3.2.2.2 Execution record implementation

The relevant RESTful APIs of the API service for creating, reading, updating, and deleting an execution record in the blockchain are shown in Figure 11.

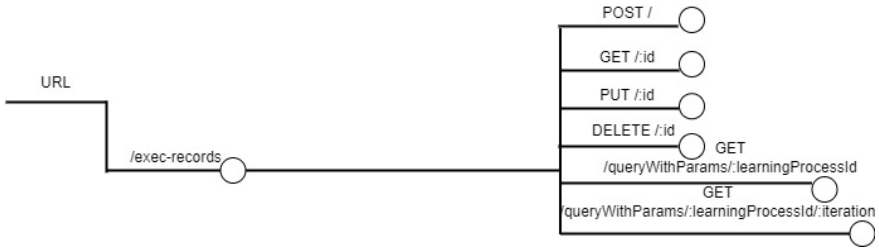

Figure 11 - Execution record endpoints

The following RESTful APIs for execution record are provided by the API service:

- POST <URL>/exec-records/ - creates an execution record by providing a JSON body element with the following information: ID representing the execution record unique ID, imageTag representing the unique ID (tag) of the image the execution task represented by this execution record is going to execute, and containerPubKey which is the public key of the generated key pair. As described before, the private key is passed to the executing container and is used to sign results produced by the computation in this container. When writing the results to BC, the chaincode verifies the signer with this public key. The ID is generated to represent an execution belonging to a particular learning process and algorithm image in the following format `task-<learningProcessID>-<iterationNumber>-<learningNode>-<timestamp>`

```
{
  "ID": "<executionRecordID>",
  "imageTag": "<imageID>",
  "containerPubKey": "<public key of the generated key pair>"
```

```
}
```

- GET <URL>/exec-records/:id – reads the execution record defined by the provided *id* parameter value from the BC.
- PUT <URL>/exec-records /:id – updates the existing execution process record with the new record provided by JSON body (for example to update the results of the execution by the container representing this execution record, or to update the status of the execution)
- DELETE <URL>/exec-records/:id – deletes the execution record defined by the provided *id* parameter value from the BC.
- GET <URL>/queryWithParams/:learningProcessID – queries all execution records and return the ones which belong to the specified learning process.
- GET <URL>/queryWithParams/:learningProcessID./:iteration– queries all execution records and return the ones which belong to the specified learning process and its particular iteration.

Table 3 presents the execution record data model in the execution record chaincode.

Table 3 - Execution record chaincode data model

Execution Record Represent an execution record		
Field	**Type**	**Description**
`ID`	String	Identifier of the learning process instance. Generated in the format `task-<learningProcessID>-<iterationNumber>-<learningNode>-<timestamp>`
`containerPubKey`	String	Public key of the generated key pair
`status`	Enum	An enumeration value identifying the current status of the learning process (NEW, ITERATION_COMPLETED, COMPLETED, FAILURE). NEW is the status assigned to the record just created. ITERATION_COMPLETED is the status assign if the execution record represents one of the process iterations when it completes. COMPLETED is the status given to the last iteration execution record in the learning process when done.
`result`	Object	The result of the computation of the execution task.
`result_signature`	String	The signed result of the execution (signed by the private key of the executing container).

We have six chaincode functions: `CreateExecutionRecord`, `GetExecutionRecord`, `SetExecutionResult`, `DeleteExecutionRecord`, `GetAllExecutionRecordsForLearningProcess`(), and `GetAllExecutionRecordsForLearningProcessAndIteration`.

- `CreateExecutionRecord (executionRecordData JSON)` **:** Performs the necessary actions in order to create a new execution record entry on the ledger by updating the world state with a new

record and appending new blocks at the end of the ledger. The function receives the data containing the execution record information.

- `SetExecutionResult (updatedProcessesData JSON)`: Updates existing execution records in the ledger with new information. For example, the updated JSON will contain the result of the computation as well as the signed result for verification that the signed result is indeed reported by the designated execution container (an execution container that was instantiated with the private key whose matching public key is saved in the execution record).
- `GetExecutionRecord (executionRecordId string)`: Retrieves information about of the execution record designated by the ID
- `DeleteExecutionRecord (executionRecordId string)`: Deletes the execution record designated by the ID
- `GetAllExecutionRecordsForLearningProcess (learningTaskID string)`: Retrieves information about all the execution records belonging to the specified learning process ID
- `GetAllExecutionRecordsForLearningProcessAndIteration(learningTaskID string, iteration number)`: Retrieves all the execution records belonging to the specified learning task ID and its particular iteration.

3.3 Data marketplace and tokenization flows and artifacts

As explained above, the Data marketplace chaincode is comprised of two chaincodes: the tradeable asset chaincode and the access request chaincode. Each chaincode is represented by a particular managed asset type. In our case, these are the tradeable asset (for tradeable asset chaincode) and the access request (for access request chaincode).

3.3.1 Artifacts and execution flow

Figure 12 shows the Data marketplace execution flow.

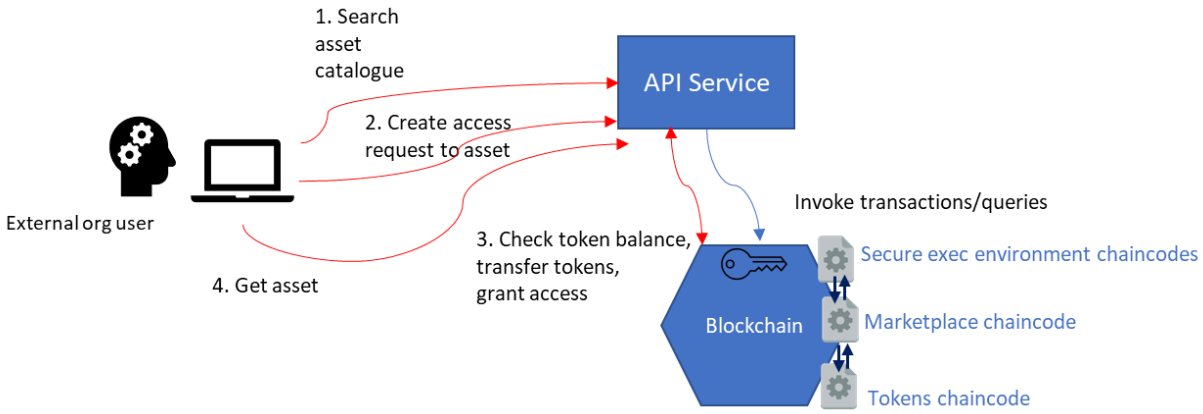

Figure 12 - Data marketplace execution flow

Once the learning process is completed, the final outcome, that is, a model suitable for consumption by external organizations is published as a tradeable asset. In the case of a fraud detection use case, the model can serve for an external organization to detect fraudulent activities. An external organization interested in purchasing an asset, can search the tradeable assets catalogue based on different parameters (for example, all models generated as a result of execution of a particular fraud algorithm). In order to gain access to a particular asset once found, the external organization needs to submit an access request to the asset and for this access request to be approved. This access request is not required in case the organization is one of the asset's owners, meaning, it was part of the asset generation. In our case, participated in the fraud detection algorithm learning process. The sequence of operations for handling access request by the inter-operation of chaincodes are shown in Figure 13.

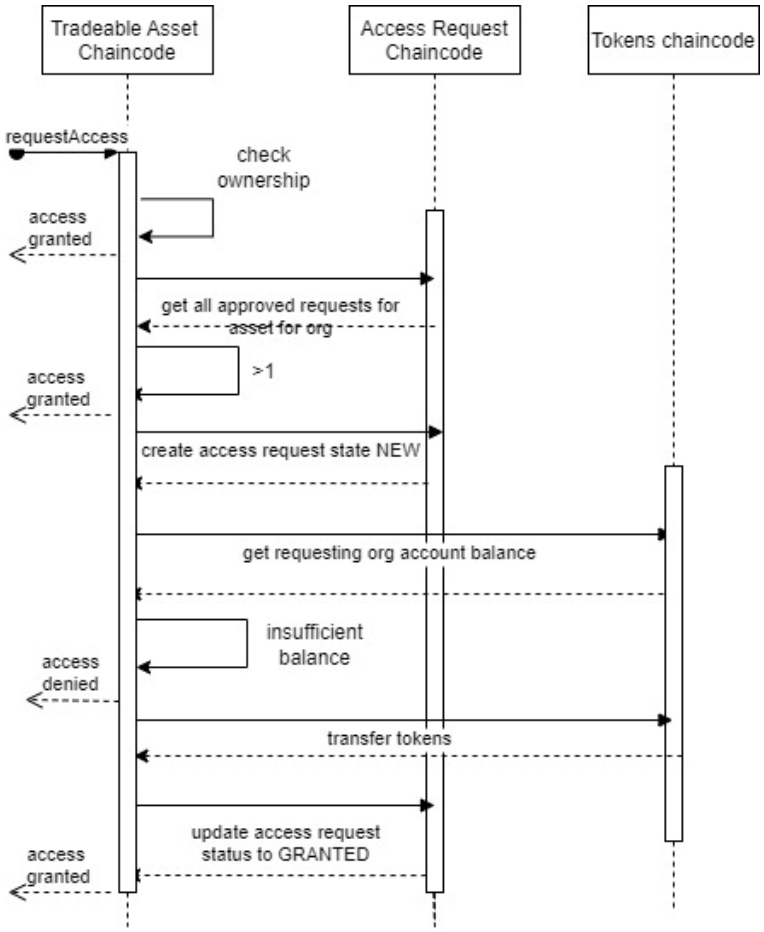

Figure 13 - Request access to a tradeable asset sequence diagram

As aforementioned, if the requesting organization is one of the asset's owner organizations, access will be granted immediately without creating an access request. Otherwise, if an approved access request already exists for the requesting organization for the specified asset, the organization will be allowed to read the asset. Otherwise, an access request will be generated for this organization and this asset, initially in state NEW. Account balance (in tokens) of this organization's account will be verified, and if the account has enough tokens to satisfy the price of the asset (which is designated in the tradeable asset record) that amount will be transferred to the owner(s) organization's account, the access request status will be updated to GRANTED, and the requesting organization will be allowed to access the asset. Note that in the scope of the MVP, we transferred the tokens to the first owner in the list of owners. However, more sophisticated payment options can be taken into consideration (refer to different payment models as one potential extension in Section 5).

The following artifacts related to this flow have been created/used in the MVP:

- Chaincodes implementing the tradeable asset and access request records publishing to/reading/querying the BC
- Tokens' chaincode implemented in D4.10. We note that specifically in the MVP we have applied the ERC20 implementation described in D4.10, but other ERC implementations such as ERC1155 detailed in D4.11 can be applied as well, since they comply with standard APIs tokens.
- API service routes and controllers related to tradeable asset and access request chaincode operations.

The interaction between the tradeable asset and access request chaincodes and the tokens chaincode is depicted in Figure 13. The functions invoked from the tokens chaincode are **`balanceOf()`** and **`transfer()`** functions (refer to D4.10). The invocation of one chaincode from another chaincode, as long as they are

running on the same blockchain channel, is straightforward and is described in Hyperledger Fabric documentation[5].

3.3.2 Implementation

3.3.2.1 Tradeable asset record implementation

The relevant RESTful APIs of the API service for creating, reading, updating, and deleting a tradeable asset in the blockchain are shown in Figure 14.

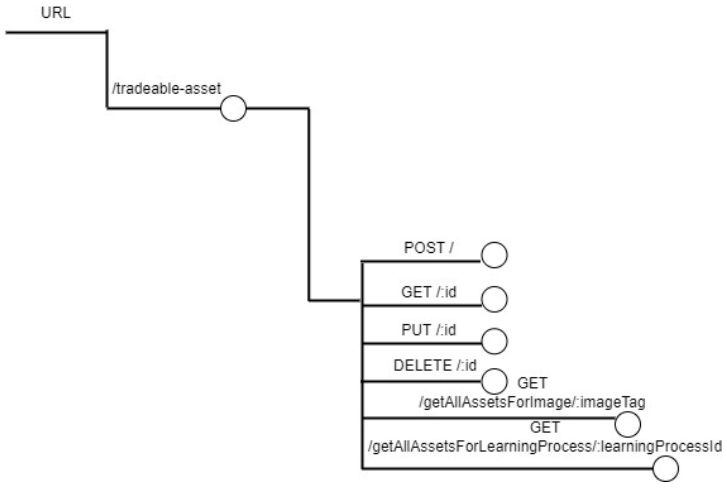

Figure 14 - Tradeable asset endpoints

The following RESTful APIs for learning process records are provided by the API service:

- POST <URL>/tradeable-asset / - creates a tradeable asset record by providing a JSON body element with the following information: ID and name representing the asset, the object itself (can be any JSON element type), provenance information of the asset, such as owning organizations, `imageTag` representing the unique ID (tag) of the image, which was executed to generate the asset, learning process the execution of which generated the asset, and the asset price.

```
{
  "ID": "<tradeableAsetId>",
  "name" : "<assetName>",
  "object": {"value":"<assetObject>"},
  "ownerOrgs":["<org1>","<org2>"],
  "imageTag": "<imageTag>",
  "learningProcessId":"<learningProcessID>",
  "priceInTokens": <price>
}
```

- GET <URL>/tradeable-asset/:id – reads the tradeable asset record defined by the provided *id* parameter value from the BC. This method returns a value only if the invoking user belongs either to the asset owner's organization or to an organization which has a valid approved asset request for the requested asset. Otherwise, the method will return an error stating lack of access authorization.
- PUT <URL>/tradeable-asset/:id – updates the existing learning process record with the new record provided by the JSON body. Allowed only to an owning organization.
- DELETE <URL>/ tradeable-asset /:id – deletes the tradeable asset record given by the specified asset id. The operation allowed only to an owning organization.

[5] https://hyperledger-fabric.readthedocs.io/en/latest/developapps/chaincodenamespace.html#considerations

- GET <URL>/tradeable-asset/getAllAssetsForImage/:imageTag – queries all tradeable asset records and returns the ones where the `imageTag` property is equal to the value provided in the parameter. Only the asset metadata will be returned in the query, not the "object" property representing the asset itself.
- GET <URL>/tradeable-asset/getAllAssetsForLearningProcess/:learningProcess – queries all tradeable asset records and returns the ones where the `learningProcessID` property is equal to the value provided in the parameter. Only the asset metadata will be returned in the query, not the "object" property representing the asset itself.

Table 4 presents the tradeable asset record data model in the tradeable asset chaincode.

Table 4 - Tradeable asset data model

TradeableAsset Represent a tradeable asset record.		
Field	**Type**	**Description**
`ID`	String	Identifier of the tradeable asset instance.
`name`	String	The name of the asset
`imageTag`	String	The identifier of the docker image which was executed to create this asset.
`object`	Any	The object representing the asset. In case of large assets, they can be stored offchain (i.e., in a cloud object store) and this field can hold a URL and credentials for accessing the asset in the offchain repository, and a hash which can be used to verify the validity of the asset.
`ownerOrgs`	Array of Strings	The owning organizations.
`learningProcess`	String	The identifier of the learning process which generated this asset.
`price`	Number	The price in tokens amount to pay for access to this asset.

We have implemented six chaincode functions: `CreateTradeableAsset`, (`UpdateTradeableAsset`, `GetTradeableAsset`, (`DeleteTradeableAsset`, `GetAllAssetsForImageTag`, and `GetAllAssetsForLearningProcess`.

- `CreateTradeableAsset (tradeableAsset JSON)`**:** Performs the necessary actions in order to create a new tradeable asset metadata entry in the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the tradeable asset information.
- `UpdateTradeableAsset (updatedAssetData JSON)`**:** Updates an existing tradeable asset in the ledger with new information assuming the invoking user belongs to an owner asset organization.
- `GetTradeableAsset (assetID string)`**:** Retrieves information about the tradeable asset designated by the provided `assetID`. As explained in the REST API section, this method will only return the full asset information if the invoker is the owner organization or has a valid access request for this asset in GRANTED state. Otherwise, the method will return an error.
- `DeleteTradeableAsset (assetID string)`**:** Deletes the tradeable asset in case the requesting user belongs to the asset owner organization.

- `GetAllAssetsForImageTag (imageTag string)`: Queries the ledger and returns the assets generated as a result of the algorithm image specified by the provided tag. It returns a list of relevant asset metadata records, without the actual asset content (i.e., without the object property value)
- `GetAllAssetsForLearningProcess (learningProcessID string)`: Queries the ledger and returns the assets generated as a result of the learning process specified by the provided ID. It returns a list of relevant asset metadata records, without the actual asset content (i.e., without the object property value)

3.3.2.2 Access request record implementation

The relevant RESTful APIs of the API service for creating, reading, updating, and deleting an access request record in the blockchain are shown in Figure 15.

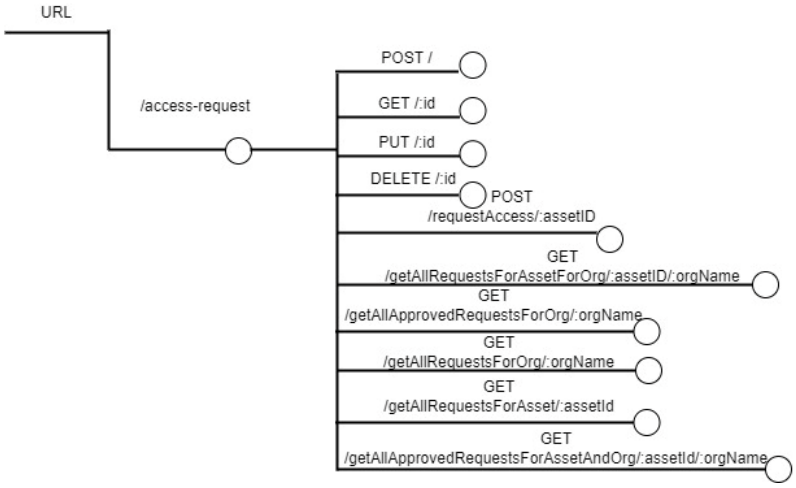

Figure 15 - Access request endpoints

The following RESTful APIs for access request record are provided by the API service:

- POST <URL>/access-request / - creates an access request record by providing a JSON body element with the following information: ID for the access request, the id and the name of the requested asset, the requesting organization, and the request date. The status of the newly created request is always NEW. This method, in combination with the `updateAccessRequest()` call setting the status of the request to GRANTED, is usually called by an organization (an asset owner) on behalf of another organization (the procuring organization) in order to grant access to the asset while bypassing the regular access request and payment process (where the procuring organization invokes `requestAsset()` operation to initiate credit validation, payment and granting of access process). Only the asset owner organization is allowed to create access requests in this way, other organizations need to invoke `requestAccess()` operation.

```
{
  "ID": "<access request id> ",
   "assetID" : "<requested asset id>",
   "assetName":"<requested asset name>",
   "requestingOrgName":"<requesting org name>",
   "requestDate":"<request data>"
}
```

- GET <URL>/access-request/:id – reads the access request record defined by the provided *id* parameter value from the BC.
- PUT <URL>/access-request /:id – updates the existing access request status. Allowed only to the owner organization of the asset.

- DELETE <URL>/ access-request /:id – deletes the access request record identified by the specified asset id. The operation is allowed only to the requesting organization or the asset owner organization.
- POST <URL>/access-request/requestAccess/:assetID/ – initializes the process for creating access request and granting access for the specified asset.
- GET <URL>/ access-request /getAllRequestsForAssetForOrg/:assetID/:orgName – queries all access request records and returns the ones where the `assetID` and `orgName` properties are equal to the values provided in the parameters.
- GET <URL>/ access-request /getAllApprovedRequestsForOrg/:orgName – queries all access request records and returns the ones where the `orgName` property is equal to the value provided in the parameter and the status of the request is GRANTED.
- GET <URL>/ access-request /getAllRequestsForOrg/:orgName – queries all access request records and returns the ones where the `orgName` property is equal to the value provided in the parameter.
- GET <URL>/ access-request /getAllRequestsForAsset/:assetID – queries all access request records and returns the ones where the `assetID` property is equal to the value provided in the parameter.
- GET <URL>/ access-request /getAllApprovedRequestsForAssetAndOrg/:assetID/:orgName – queries all access request records and returns the ones where the `assetID` and `orgName` properties are equal to the values provided in the parameters and the status of the request is GRANTED.

Table 5 presents the request record data model in the access request chaincode.

Table 5 - Access request record data model

Access Request Represent an access request record.		
Field	**Type**	**Description**
`ID`	String	Identifier of the tradeable asset instance.
`assetID`	String	The ID of the asset.
`assetName`	String	The name of the asset.
`requestingOrgName`	String	The name of the organization for which the asset request is created.
`ownerOrgs`	Array of String	The owning organizations.
`requestDate`	Timestamp	The creation date of the access request.
`status`	Enum	The status of the access request (NEW, DECLINED, GRANTED).

We have implemented ten chaincode functions: `CreateAccessRequest`, `UpdateAccessRequestStatus`, `GetAccessRequest`, `DeleteAccessRequest`, `getAllRequestsForAssetAndOrg`, `getAllRequestsForAsset`, `getAllRequestsForOrg`, `getAllApprovedRequestsForOrg`, `getAllApprovedRequestsForAssetAndOrg`, and `requestAccess`

- `CreateAccessRequest (accessRequest JSON)`**:** Performs the necessary actions in order to create a new access request entry on the ledger by updating the world state with a new record and appending new blocks at the end of the ledger. The function receives the data containing the access request information. Only the asset owner organization is allowed to create access requests this way (others will get error notification upon attempt). Non-owner organizations that intent to request access to a certain asset need to invoke the `requestAccess()` function.

- `UpdateAccessRequestStatus (updatedAccessRequest JSON)`**:** Updates an existing asset request, specifically its status assuming the invoking user belongs to the organization which is the asset owner.
- `GetAccessRequest (accessRequestID string)`**:** Retrieves information about the access request designated by the provided ID.
- `DeleteAccessRequest (accessRequestID string)`**:** Deletes the access request record.
- `getAllRequestsForAssetAndOrg(assetID string, orgName string)`**:** Queries the ledger and returns all the access requests for a particular asset by a particular organization.
- `getAllRequestsForAsset(assetID string)`**:** Queries the ledger and returns all the access requests existing for a particular asset.
- `getAllRequestsForOrg(orgName string)`**:** Queries the ledger and returns all access requests by a particular organization.
- `getAllApprovedRequestsForOrg(orgName string)`**:** Queries the ledger and returns all the approved access requests by a particular organization.
- `getAllApprovedRequestsForAssetAndOrg(assetID string, orgName string)`**:** Queries the ledger and returns all approved access requests for a particular asset by a particular organization.
- `RequestAccess (assetID string)`**:** Request access to a particular asset.

4 Artifact provenance

One of the core properties of a blockchain platform is the immutable chain of transaction blocks which ensures provenance of all the artifacts stored on its ledger. Based on this, organizations working with BC as their world state can establish verifiable and transparent history of updates by any organization in the BC network and linkage between different artifacts stored on the ledger.

Accordingly, one of the functionalities envisioned in our secure execution framework and data marketplace is to allow provenance of the whole chain of execution of the federated learning process on one hand, and of trading processes taking place in the data marketplace, on the other.

For the secure execution framework, we would like to ensure:

- The secure execution validation and verification of artifacts used in the learning process by establishing the validity of the algorithm image the learning process executes and verifying that the result of each execution task is written to the ledger by the expected execution container and not an "imposter".
- The provenance of the learning process and algorithm image.
- The provenance of the computation execution task.

For the data marketplace we would like to provide the following functionalities:

- Provenance of the tradeable assets. For example, for the ML models we would like to know their origin (e.g., which learning algorithm was applied, which learning algorithm image they are built from, and what are the organizations which participated in the model creation).
- Provenance of access to tradeable assets. For example, being able to answer which organizations requested access, which requests were approved, and which were denied.

These functionalities are useful for audit trails and data governance, quality assurance, and regulatory compliance along many other use cases. Understanding the provenance of deployed models is important when we would like to know, for example, which models were derived from a particular algorithm. Proven, verifiable, and immutable audit trail of execution tasks producing such models can help establishing without doubt that the models are derived from the desired ML algorithm, the desired version of that algorithm, and no mistakes were made in the process.

In the previous sections we have described the functionalities and the artifacts implementing the secure execution framework and the data marketplace. In this section, we emphasise the particular functionalities related to provenance and auditing.

Relevant audit functionalities per artifact type:

- Learning image
  - Provenance of the docker image stored in the docker image repository. The validity of the image is established by comparing the image hash saved in the image metadata on BC with the docker image hash of the image stored in the docker image repository.
- Learning process
  - Audit of particular image usage in various learning processes: `getLearningProcessesForImageTag()` query for retrieving all the learning processes run on a particular algorithm image.
- Execution record
  - Verification of the results of the execution task and the identity of executing entity by using public-private key mechanism for instantiating execution tasks and signing the execution results. Verification in chaincode that the signing entity holds the private key matching the appropriate public key stored in execution record representing this execution entity (see Section 3.2 for details).

    - Execution record provenance: `getAllExecutionRecordsForLearningProcess()` and `getAllExecutionRecordsForLearningProcessAndIteration()` queries allowing to retrieve all execution tasks of a particular learning process, or all execution tasks of a given learning process and a given iteration.
- Tradeable asset
    - Provenance of the tradeable asset: `getAllAssetsForImageTag()` retrieves all assets which were created as a result of particular algorithm execution while `getAllAssetsForLearningProcess()` retrieves all assets which were created as outcome of a particular learning process.
- Access request
    - Provenance of access to particular assets: `getAllRequestsForAsset()` retrieves all access requests for a tradeable asset; `getAllRequestsForOrg()` retrieves all access requests issued by particular organization; `getAllApprovedRequestsForOrg()` retrieves only GRANTED requests for a particular organization; `getAllRequestsForAssetAndOrg()` retrieves all access request made by particular organization for a particular asset; and `getAllApprovedRequestsForAssetAndOrg()` retrieves all approved requests made by a particular organization for a particular asset.

These queries are by no means an exhaustive list. More queries can be developed to support different use cases according to the required functionalities of their business needs (see possible extensions in next section).

5 Summary and possible extensions

The deliverable in hand provides a deep dive into the technical details of the implementation of a blockchain based federated learning environment and data marketplace MVP that instantiates the framework presented in D4.14 in the domain of fraud detection.

This document provides a comprehensive description of the architecture of the MVP, the flows of the use case scenario along with the implemented chaincodes, APIs, functions, and the assets (the learning models in our case) stored in the blockchain ledger. In addition, we provide a blockchain based data market that enables the secure trading of the learning process outcomes by leveraging tokenization mechanisms developed by IBM during earlier phases of the project.

It is important to note that the devised framework is generic and suitable for a wide variety of federating learning algorithms. We have realized the framework for the particular case of fraud detection, but the same MVP can support other financial and insurance use cases as well, including:

- The execution of different algorithms, as long as they can be represented and packaged as a computational task (e.g., a docker image).
- Different tokenization mechanisms - We have demonstrated cross-chaincode invocation and usage of previously developed ERC-based token smart contracts inside the data marketplace. We are currently applying ERC20 implementation for assets trading, but other token implementations, such as ERC1155, could be deployed instead and used in the same manner.

Possible extensions to our MVP include:

- Learning process orchestration script
  - The orchestration of the computational tasks is based on the synchronized execution of multiple iterations of single-image based computation tasks on different nodes. This scenario is suitable for fraud detection. Other types of orchestration including single computational tasks, multiple unsynchronized computational tasks, and multi-step computations, could be added according to the type of the learning algorithm.
- Data marketplace
  - Support for more data assets with different metadata than the one applied for fraud detection. For example, we might include assets in the form of files and not models.
  - Support for secondary functionalities of a data marketplace including:
    - Usage of different pluggable pricing and payments models.
    - Auctions and price negotiations - Bids issuance and support for a negotiation process to reach an agreement on the asset price.
- Artifacts' usage audit and provenance
  - More sophisticated queries can be added to support different aspects of the assets' provenance.

A demo showing step-by-step the execution flows of the MVP can be accessed through the project marketplace at: https://www.youtube.com/watch?v=J7ekCHSoWrg&list=PL9suUK-Ys8V3Dkzm7qmZnIb-VIc1eSMwp&index=22. A short movie showing the key features of the MVP can also be accessed through the project data marketplace at https://www.youtube.com/watch?v=H8M8PMIA8YU&list=PL9suUK-Ys8V3Dkzm7qmZnIb-VIc1eSMwp&index=21.

6 Conclusions

Financial drivers are pushing financial organizations to find innovative ways to share data and information on clients in a privacy-preserving and value-preserving way. To this end, partners FBK and IBM joined forces. During the course of the project, the teams of FBK and IBM realized that there is enormous potential in leveraging blockchain technology for the sake of federated machine learning algorithms. Specifically, applying blockchain as a secure environment for the execution of the learning process and a platform for trading the algorithm outcomes. The outcome of this fruitful collaboration yielded a novel framework for a blockchain based federated learning environment and data marketplace. The vision, motivation, and design of this framework were provided in D4.14 - Encrypted Data Querying and Personal Data Market – II.

The purpose of the deliverable herewith, entitled D4.12 - Blockchain Tokenization and Smart Contracts - III was to provide the technical deep dive into the design and development of an MVP blockchain application in the domain of fraud detection that materializes the framework for the blockchain based federated learning environment and data marketplace presented in D4.14.

Main achievements of our work include:

- Implementation of a blockchain based marketplace for assets managing and their trading using digital tokens.
- Implementation of a blockchain based environment for the secure execution of federated machine learning algorithms materialised for the case of fraud detection.
- A movie (presented during the *2nd Workshop on Blockchain Applications for Digital Finance* held on March 2, 2022) emphasising the highlights of the work (https://www.youtube.com/watch?v=H8M8PMIA8YU&list=PL9suUK-Ys8V3Dkzm7qmZnIb-VIc1eSMwp&index=21)
- A deep dive demo (step by step) of the MVP BC technical aspects available in the INFINITECH project marketplace (https://www.youtube.com/watch?v=J7ekCHSoWrg&list=PL9suUK-Ys8V3Dkzm7qmZnIb-VIc1eSMwp&index=22)
- Submission of a joint paper with FBK titled *A Framework for Verifiable and Auditable Federated Fraud Detection* to the IEEE Transactions on Big Data journal special issue on "Trustable, verifiable, and auditable federated learning".

The MVP described in this document is based on the federated learning algorithm for fraud detection detailed in D.14. However, it is worth noting that the BC environment is agnostic to the algorithm which acts as a black box, that is, any kind of runnable computation algorithm, which can be represented as a container image suitable for execution, can be converted to a docker image and be executed in the developed solution. This report should be read in conjunction with D4.15 - Encrypted Data Querying and Personal Data Market – III which complements the technical details of the MVP with the machine learning parts to get a comprehensive understanding of the complete framework implementation.

The current deliverable constitutes the final report of Task 4.4 and concludes the activities of the specific task.

Table 6 – Conclusions (TASK Objectives with Deliverable achievements)

Objectives	**Comment**
Enhance the permissioned blockchain with cryptographic tokenization features, as a means of enabling assets trading	Two most popular standards (ERC20 and ERC1155) have been implemented on top of Fabric in the scope of the previous deliverables and leveraged in the proposed framework and MVP as enablers for assets trading in a data marketplace.

Specify and implement Smart Contracts for the secure exchange of data	Required chaincodes have been designed and developed to enable the secure framework proven in a fraud detection scenario minimal viable product.
Provide the means for trading access to data and information through the permissioned blockchain.	The BC network developed enables the governance of access to the assets stored in the ledger and the trading of these assets via digital tokens.

Table 7 – (map TASK KPI with Deliverable achievements)

KPI	**Comment**
Realize a blockchain based federated learning environment	*Target Value = 1* Our MVP realizes the devised framework for the secure execution of federated machine learning algorithms exploiting BC technology in a fraud detection use case.
Realize a blockchain based data marketplace	*Target Value = 1* Our MVP provides the mechanisms for controlling and managing the access rights of the assets stored in the BC (fraud detection federated algorithm outcomes) and for trading these assets via digital tokens.

Appendix A: Literature

[1] Hyperledger Fabric – Hyperledger," 2020. [Online]. Available: https://www.hyperledger.org/use/fabric. [Accessed 15-March-2022].

[2] Gaur, N., Desrosiers, L., Ramakrishna, V., Novotny, P., Baset, S.A., and O'Dowd, A. (2018). Hands-on Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer. Packt Publishing.

[3] D4.10 – D4.11 as background.