BROKERAGE AND MARKET PLATFORM
FOR PERSONAL DATA

D5.5 KRAKEN Marketplace First Release

www.krakenh2020.eu

This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 871473

D5.5 KRAKEN Marketplace First Release

Grant agreement	871473
Work Package Leader	Lynkeus
Author(s)	Rob Holmes (TX)
Contributors	Donato Pelegrino (TX), Davide Zaccagnini (Lynkeus), Tilen Marc (XLAB), Javier Presa Cordero (Atos)
Reviewer(s)	Sebastian Ramacher (AIT), Davide Zaccagnini (Lynkeus)
Version	Final
Due Date	31/08/2021
Submission Date	28/09/2021
Dissemination Level	Public

Release History

Version	Date	Description	Released by
v0.1	18/06/2021	Initial version (ToC and initial marketplace text)	Rob Holmes
v0.2	24/06/2021	Section MPC node	Tilen Marc
v0.3	01/07/2021	Atos updates to Marketplace SSI Agent section	Javier Presa Cordero
v0.4	12/07/2021	TEX integration of all partner updates and writing of intro, exec summary and conclusion read for review	Rob Holmes
v0.5	22/07/2021	Review from LYN	Davide Zaccagnini
v0.6	22/07/2021	Review from AIT	Sebastian Ramacher
v0.7	30/07/2021	Incorporation of feedback from reviewers. Final version ready for quality manager review	Rob Holmes
v0.8	23/09/2021	Final version	Rob Holmes
v1.0	28/09/2021	Submitted version	Atos

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Description
API	Application Programming Interface
REST	Representational State Transfer
SSI	Self-Sovereign Identity
SMPC	Secure Multi Party Computation
ID	Identifier
WASM	Web Assembly
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
DB	Database
GUI	Graphical User Interface
ERC20	Ethereum Request for Comments 20
DID	Decentralised ID
TPS	Transactions Per Second
VM	Virtual Machine
CA	Certificate Authority
TLS	Transport Layer Security
HLF	Hyperledger Fabric

Executive Summary

This document describes the current software of the initial KRAKEN marketplace implementation, covering all modules, programs and tools used for the marketplace first release. It is the first deliverable of Task 5.4 - Backend/Frontend Development within Work Package 5 - Reference platform implementation, pilot's integration and validation.

The general objective of Work Package 5 is to design and develop a fully functional pilot marketplace for the KRAKEN general infrastructure for biomedical and wellbeing data and for educational data. This initial marketplace release consists of an integrated set of infrastructures that together facilitate encrypted batch data transfer to eligible data consumers through directory sharing. Batch data can be defined as a static record or collection of records, as opposed to real-time data streams which provide a sequence of data points in time.

This Deliverable 5.5 provides a brief description of each component of the initial Marketplace release, describing the interfaces, deployment, source codes and tool usage.

A short description of the components described in this document is provided below:

- Marketplace API (Section 3)
  - Handles any requests sent by the Marketplace Frontend, the Marketplace xDai Watcher and the Ledger uSelf Broker of the SSI Agent.
  - Sends requests to the SMPC node, the Ledger uSelf Broker of the SSI Agent, the Marketplace Catalogue Database, and the marketplace Consortium Blockchain Node.
- Marketplace Catalogue Database (Section 4)
  - Persistently stores metadata associated with Data Products' and marketplace users.
  - Providing the users with the ability to browse and filter Data Products.
- Marketplace Frontend (Section 5)
  - Software component within the marketplace system that is equipped with a graphical User Interface (GUI) allowing users to perform all marketplace operations.
- Marketplace Smart Contract (Section 6)
  - Responsible for the publication and purchase of Data Products on the xDai blockchain.
- Marketplace xDai Watcher (Section 7)
  - An intermediary between the Marketplace Smart Contract and Marketplace API, updating the API about any new Data Product status on the xDai blockchain.
- Consortium Blockchain Node (Section 8)
  - Stores associated Data Product policies and eligible Data Product transactions on the ledger and controls which users have legitimate, compliant access to which Data Products.
- MPC Node (Section 9)
  - Used to securely evaluate various analytics on datasets in a privacy-preserving fashion as well as a mechanism for the key distribution process for access to batch Data Products.
- Marketplace SSI Agent Ledger uSelf Broker (Section 10)
  - Facilitates the integration between the marketplace and SSI, easing the complex interactions with the SSI solution for user registration and authentication such as exchanging DIDs, issuing of verifiable credentials and the presentation of proofs.

Please note that this deliverable represents the status of the marketplace software components for the first release. At the time of this release the project is in August 2021 (month 21) and further development will continue beyond this initial release with the final software components released in July 2022 (month 32). Any changes to the marketplace software will be reflected in the report accompanying Deliverable 5.6 - KRAKEN Marketplace Final Release in July 2022.

1 Introduction

1.1 Purpose of the document

The KRAKEN marketplace consists of an integrated set of infrastructures that together have been designed to facilitate the secure GDPR compliant exchange of biomedical and educational data. The aim of this deliverable 5.5 (D5.5) document is to provide a brief description of each of the Marketplace software components that form part of the first release, including their purpose, interfaces, how they are deployed, where to find the source code and which baseline technologies or tools have been used to build them.

1.2 Structure of the document

This report commences with a brief overview of the initial prototype release as part of D5.5 (Section 2). It has then been split into dedicated sections for each of the modules, programs and tools developed and to be deployed as part of the Marketplace first release. These include the marketplace API (Section 3), marketplace catalogue database (Section 4), marketplace front end (Section 5), marketplace smart contract (Section 6) and xDai watcher (Section 7) developed by TEX, the consortium blockchain node (Section 8) from Lynkeus, the MPC node (Section 9) from XLAB and finally the SSI agent (Section 10) from Atos. Each section begins with a brief description of the component, followed by sub-sections related to interfaces, source codes, deployment instructions and baseline technologies and tools.

1.3 Glossary adopted in this document

Term	Definition
Blockchain	Network provided with a decentralized consensus algorithm and an immutable ledger.
Wallet	A data store for the DIDs and related information (including private keys and more).
Fork	Division of the development flow in two different branches
Smart Contract	Computer program executed by a blockchain.
xDai	Public blockchain
Catalogue	Collection of Data Products metadata
Dataset Key	Cryptographic key used to encrypt a Data Product Dataset
Key Shares	Shares generated from the Dataset key for the SMPC computation
Key Computation	SMPC computation of the Dataset key
Web Assembly	binary-instruction format for virtual machines
DID-Connection	SSI connection between two agents provided with two unique DIDs
Metadata	Descriptive set of information related to a Data Product (Description, tags, image, etc...)
Verifiable Credential	SSI Certificate that can be verified by SSI agents
Metamask wallet	Browser extension to manage crypto assets
DataCoin	ERC20 token adopted by the Streamr network
Batch	Group of data points collected within a given time period
Streams	Data that is continuously generated by different sources

Table 1: Glossary

2 Initial Marketplace Release Overview

The data marketplace is one of the key pillars of the KRAKEN project for enabling the sharing and trading (monetisation) of personal data. This initial marketplace release is the result of intensive design and development efforts to integrate multiple technology components at varying levels of technological maturity. The high-level technological components that have been integrated within this release are:

1) A forked and adapted instance of the Streamr marketplace [1];
2) The Lynkeus MHMD [2] Hyperledger Fabric Blockchain;
3) The Self-Sovereign Identity (SSI) Agent uSelf Broker; and
4) The SMPC Network.

For a full description of the initial marketplace integrated architecture and an explanation of the inner workings of the platform please refer to D5.3 - Initial KRAKEN marketplace integrated architecture document. However, briefly, the high-level integrated technological components described above are mapped across to the three areas, or layers, of the marketplace architecture introduced in Section 2 of D5.3 as follows:

1) Permission management layer: Lynkeus MHMD Hyperledger Fabric Blockchain and SSI Agent
2) Data protection layer: SMPC Network supported by the marketplace for standard encryption
3) Data transaction management layer: Forked and adapted instance of Streamr marketplace

It is planned that the final KRAKEN marketplace will facilitate three types of data access modalities:

1) Batch data transfer through directory sharing;
2) Transfer of encrypted streaming data; and
3) Secure Multiparty Computation.

Within this initial marketplace release the first data access modality for encrypted batch data transfer through directory sharing has been realised for individual users of the marketplace and organizations. This modality was first described in D2.6 - Marketplace Technical Specification, Section 2.3.2. The transfer of encrypted streaming data and Secure Multiparty Computation (analytics) modalities will be included at a later release.

For ease of reference Figure 1 below provides an overview of how the batch data transfer through directory sharing modality works. In this data access modality data consumers are given temporary access to process personal data in line with legal parameters (GDPR and national regulations) and individual policies such as pre-defined purposes of use that have been pre-approved by the data providers. All legal parameters and policies are managed by the Lynkeus Consortium permissioned blockchain. For paid Data Products, temporary access is also dependent upon successful payment using the ERC-20 DATA token [3], which is managed by the marketplace smart contract on xDai public blockchain [4]. For a full explanation of the functioning of this modality please refer to D5.3.

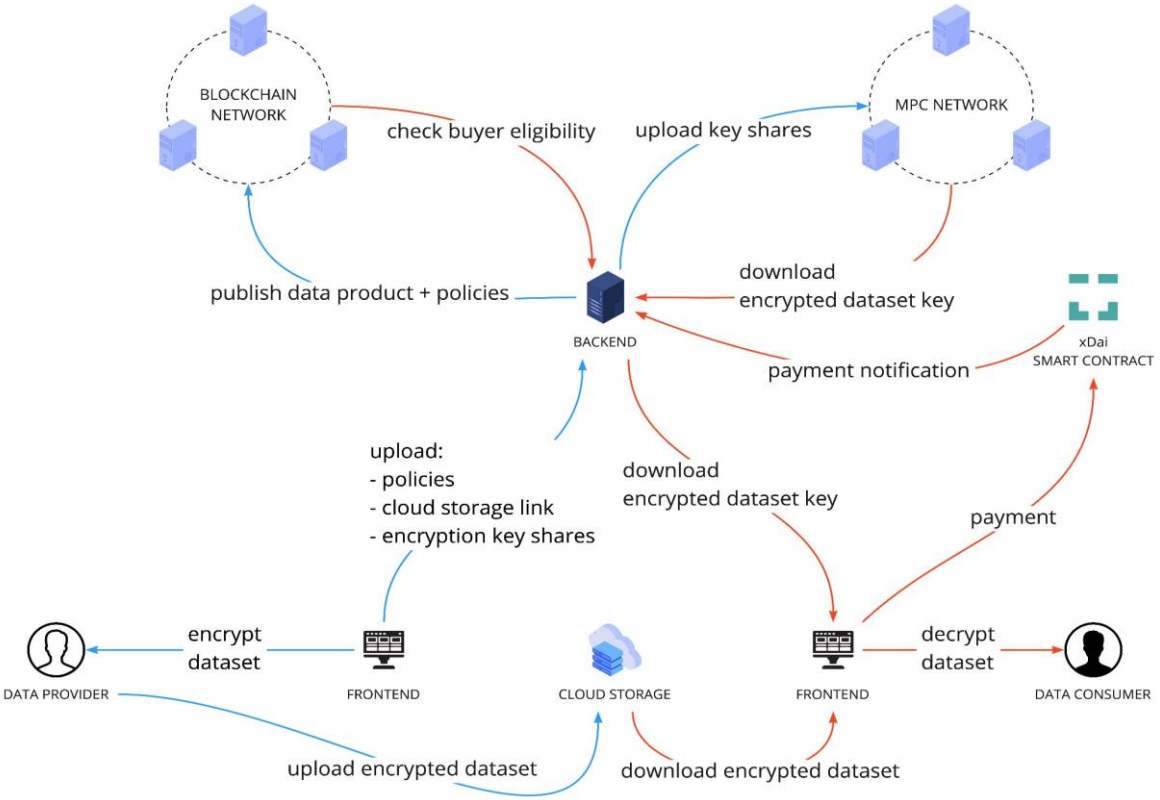


Figure 1: How batch data transfer works in KRAKEN

For this first release, the SMPC Network is used as a mechanism for key distribution in the batch data transfer through directory sharing modality. In a future release the SMPC network will also be used for the third data access modality (Secure Multiparty Computation), which will allow privacy-preserving computation / analytics to be performed on local data assets. Again, both applications of SMPC are fully described in D5.3.

It should also be noted that whilst an initial integration with the SSI has been realised through the integration with the SSI Agent's uSelf broker, in order to finalise the processes of registration and login to the marketplace the present proof functionality and the possibility to specify a credential ID must be integrated. It is expected that this shall be completed in advance of the first round of user evaluations.

A full list of the lower-level components developed and deployed as part of the first marketplace release, and described in the following sections of this document, are listed below. Some of the individual modules will be deployed as a single instance hosted by a single KRAKEN partner organization, whilst other modules will be deployed as multiple instances hosted by multiple KRAKEN partner organisations. This hosting is also indicated below.

- TEX
  - Marketplace Smart Contract
  - Marketplace API
  - Marketplace xDai watcher
  - Marketplace frontend
  - Marketplace Catalogue Database
  - Consortium Blockchain Node
  - SSI Agent
  - SSI Agent uSelf Broker

- Lynkeus
  - Consortium Blockchain Node
- XLab
  - Three instances of the SMPC Node

Each of the integrated technological components identified in the above list are also shown visually in Figure 2 below.

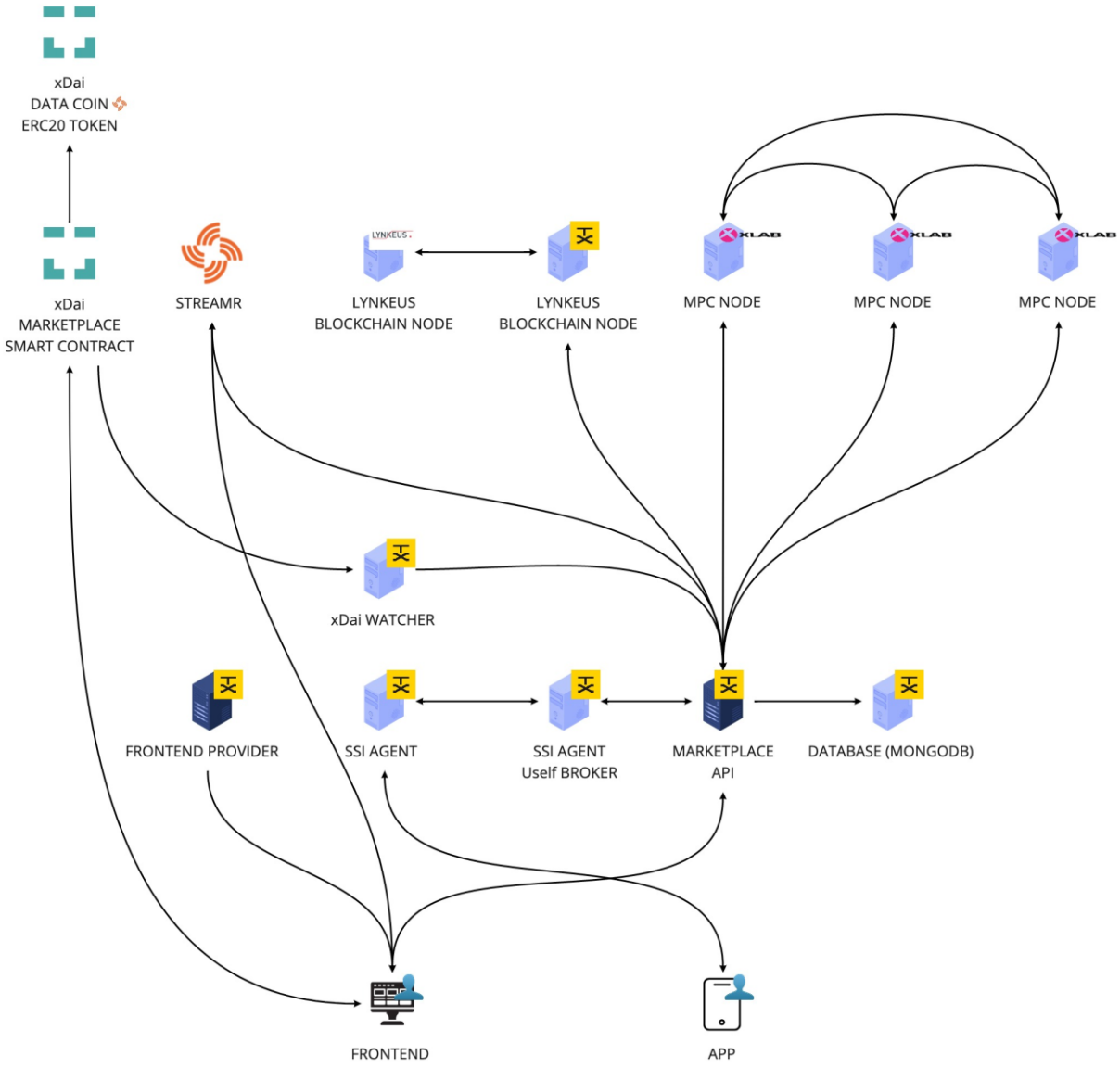


Figure 2: KRAKEN marketplace components diagram

3 MARKETPLACE API

This section describes the Marketplace API component. This component is used by the marketplace system to receive any type of client request related to the publication of Data Products and the purchase of access to Data Products.

3.1 Description

The Marketplace API is a REST API that handles any request sent by the Marketplace Frontend, the Marketplace xDai Watcher, the KRAKEN mobile app, and the Ledger uSelf Broker of the SSI Agent. These requests enable the following set of operations: Data Product catalogue download and filtering, Data Product publication and modification, Data Product publication on xDai, Data Product purchase, buyer eligibility check, buyer payment and SMPC Dataset key request.

This component also sends requests to the SMPC node, the Ledger uSelf Broker of the SSI Agent, the Marketplace Catalogue Database, and the Consortium Blockchain node. These requests enable the following set of operations: Data Product metadata storage, Data Product publication on the Consortium Blockchain, buyer's eligibility check, Dataset key shares storage, user registration and authentication.

3.2 Interfaces

A list of interfaces for the Marketplace API are provided below:

- GET/did-connection
  Download did-connection invitation information to perform a did-connection with the marketplace SSI agent.
- GET/products
  Download the list of public Data Products available on the marketplace.
- POST/products
  Upload a new Data Product on the marketplace.
- GET/products/:id
  Download metadata of a Data Product with a specific ID.
- PUT/products/:id
  Modify metadata of a Data Product with a specific ID.
- GET/products/:id/streams
  Get list of Data Streams of a Data Product with a specific ID.
- POST/products/:id/keyRequest
  Request the key computation for a Data Product with a specific ID.
- POST/products/:id/deployFree
  Publish a Data Product with a specific ID on the marketplace for free.
- POST/products/:id/setDeploying
  Inform the marketplace that a Data Product with a specific ID is being published on the payment blockchain.
- POST/products/:id/setDeployed
  Inform the marketplace that a Data Product with a specific ID has been published on the payment blockchain.
- POST/products/:id/stateEligibleBuyer
  Request Consortium blockchain eligibility to purchase access to a Data Product with a specific ID for the user sending the request.
- GET/subscriptions
  Download list of subscriptions of the user sending the request.
- POST/subscriptions
  Add a new subscription to a specific Data Product for the user sending the request.

- GET/products/:id/permissions/me
  Download list of permissions of the user that is sending the request.
- GET/users/me/products
  Download list of data products owned by the user that is sending the transaction.
- GET/split.wasm
  Download the web assembly code needed to perform the cryptographic operations for the processing of encryption keys and analytics datasets on the user frontend.

Figure 3 below shows the above listed interfaces in a visual form, including the components in the marketplace system that use them.

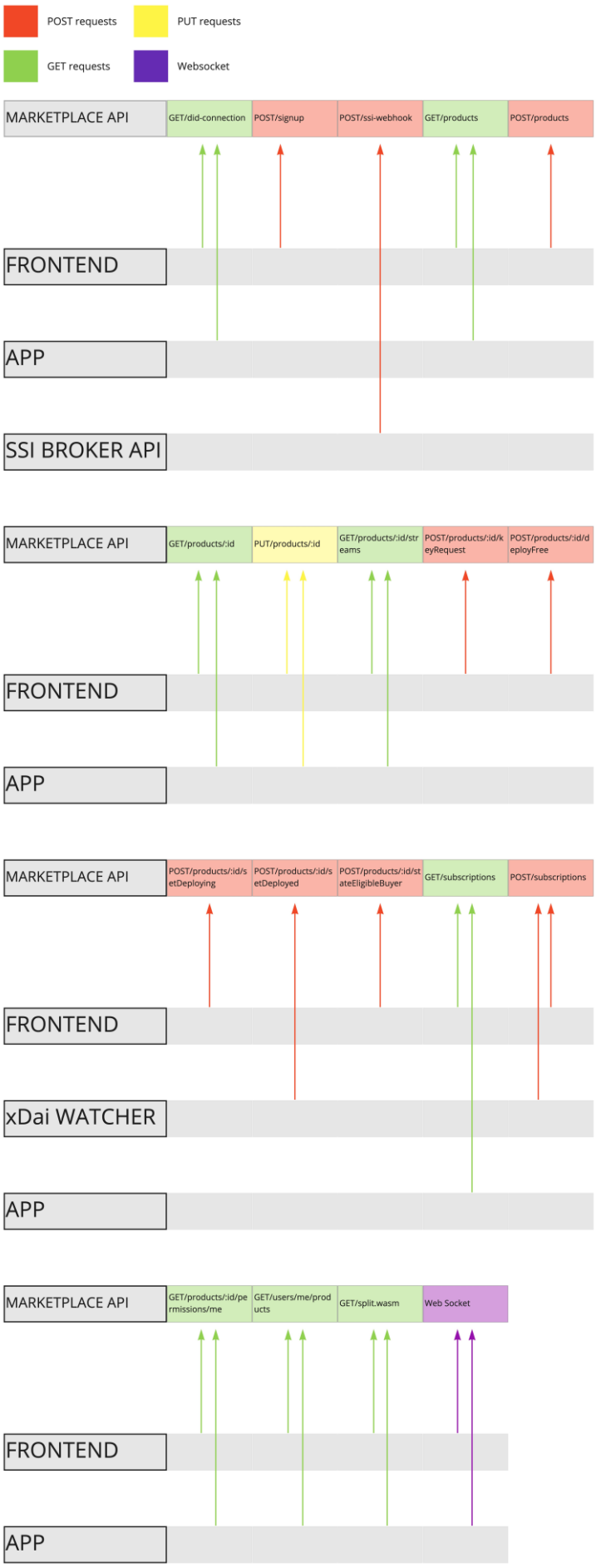


Figure 3: Marketplace API interfaces

3.3 Deployment

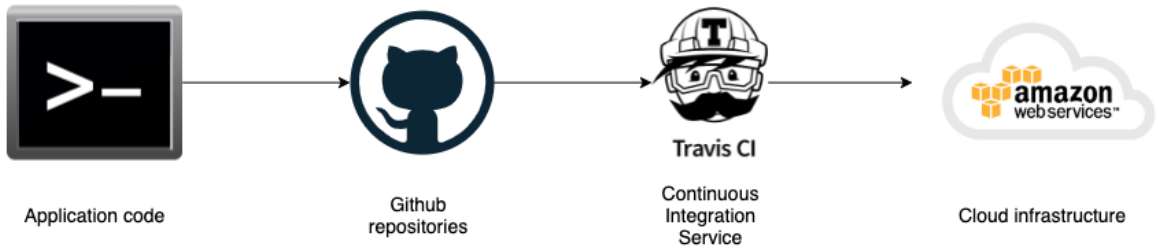


Figure 4: Automated code deployment

Automated code deployment tools are utilized in the deployment of the project code base. As new versions of the component code are committed to a preconfigured deployment branch on GitHub [5] they are compiled on Travis CI [6] and upon successful compilation they are deployed to a cloud infrastructure hosted by amazon web services [7].

The code compilation, deployment and cloud infrastructures environment settings are managed with a specific configuration file appended to the code base. Capacity provisioning and health monitoring of cloud infrastructure is handled by AWS Elastic Beanstalk services [8].

Travis.yml file guides Travis CI on how the code should be compiled and deployed, e.g. which commands need to be executed and if there are any tests or code linting, how they should be run and where should they be deployed given that previous stages of code compilation are successful.

3.4 Source Code

The code is available on a private repository on GitHub

3.5 Baseline Technologies and Tools

The Marketplace API is a REST API developed using JavaScript. The libraries the code depends on are express, fabric-network, ws for the web socket communication with the SMPC network, mongoose for the database communication and the Lynkeus libraries for the communication with the Consortium Blockchain.

4 MARKETPLACE CATALOGUE DATABASE

This section describes the Marketplace Catalogue Database. This component of the marketplace system is used by the Marketplace API to persistently store data products descriptions in the form of metadata associated with every Data Product and data on users' credentials, specifically stored on the Data Products Catalogue.

4.1 Description

The Marketplace Catalogue Database is a MongoDB [9] database instance that is used by the Marketplace API to persistently store all information related to:

- A Data Product's associated metadata;
- The information about registered users' verifiable credentials; and
- A log of the DID connections established with the users.

The stored metadata serves the purpose of providing users with the ability to browse and filter Data Products on the marketplace catalogue and describe and define specific details about the Data Product when publishing data descriptions.

4.2 Interfaces

A list of Data Models is provided below:

DATA PRODUCT

- id: String
  Hexadecimal string of 64 characters representing the product ID. This ID represents the same data product on the xDai blockchain, on the backend and on the Lynkeus blockchain. It's generated for the first time on the Lynkeus blockchain.
- ownerID: String
  Registration Verifiable Credential of the owner of the product
- name: string
  User-provided product name
- description: string
  User-provided product description
- shortDescription: string
  User-provided product short description
- educationTags
  Set of tags to categorise a product belonging to the education pilot
  - university: string
    Data product university name
  - studyProgram: string
    Data product study program name
  - course: string
    Data product course name
- heathTags: [String]
  List of tags to categorise a data product belonging to the health pilot. These tags are selected between the ones available in the MeSH ontology. The integration with the ontology is specified in D3.3
- fileStructureAndFormat
  set of parameters describing the dataset
  - filename: String
    Name of the dataset file

    - format: String
      Format of the dataset file
    - filesize: Number
      Size of the dataset file
- owner: String
  Data Product owner name/pseudonym
- imageUrl: String
  URL of the Data product image
- state: String
  Deployment state of the Data product. Available options: undeployed, deploying, deployed
- created: Date
  Creartion date of the Data product
- updated: Date
  Ltest update date of the Data product
- minimumSubscriptionInSeconds: Number
  Minimum amount of seconds a subscription can be purchased or extended
- ownerAddress: Address
  xDai address authorised to apply changes to the product
- beneficiaryAddress: Address
  Destination address for subscription tokens
- pricePerSecond: NumberString
  Data product price per second
- priceCurrency: ContractCurrency
  User-selected data product price currency. At the moment the only available one is DATA COIN
- timeUnit: TimeUnit
  Data product time unit chosen by the user between: hour, day, week, month
- price: NumberString
  Data product price per time unit
- isFree: boolean
  Data product payment requirement binary indicator
- type: ProductType
  Data product type chosen by the user between: Batch, Analytics and Real time stream. At the moment only the batch data product can be selected
- sector: string
  Market sector of the data product chosen between health and education
- anonymizeDataset: boolean
  Data produt anonymisation binary indicator
- requiresWhitelist: boolean
  Binary parameter indicating if a Data product is provided with a whitelist
- policies
  Access control parameter selected by the user
    - personalDataFromOtherPeople: Boolean
      Binary parameter corresponding to the answer the user gave to the question: “Does the data you're publishing contain personal data from persons other than yourself?”
    - consentFromIncludedSubjects: Boolean
      Binary parameter corresponding to the answer the user gave to the question: “Has informed, explicit and free consent been obtained from all data subjects whose data is included?”

    - purposes: [String]
      List of purposes specified by subjects other than the data product publisher for sharing their data as stated in their consent.
    - dataAcces: [String]
      List of users categories that can access the data.
    - dataAccessCountries: [String]
      List of countries where the data product can be purchased
- keyShares: [[Number]]
  Shares of the dataset encryption key encrypted for every for the SMPC nodes.
- datasetUrl: string
  URL for the download of the encrypted dataset
- streams
  List of streams IDs

ACCESS ELIGIBILITY

- userID: String
  Registration Verifiable Credential of the user requesting access to the data product.
- transactionID: String
  ID of the blockchain transaction stating the eligibility of the user to buy the data product
- product: ProductSchema
  Data product for which the user requested access to

SUBSCRIPTION

- userID: String
  Registration Verifiable Credential of the user subscribing to the data product.
- transactionID: String
  ID of the blockchain transaction stating the eligibility of the user to buy the data product
- endsAt: Date
  Expiry date of the subscription
- dateCreated: Date
  Creation date of the subscription
- lastUpdated: Date
  Latest extension (if any) of the subscription
- product: ProductSchema
  Purchased data product

USER CREDENTIAL

- state: Number
  Deployment state of the User's verifiable credential.
- registrationInfo
  Verifiable credential content
    - ID: String
      Unique ID of the credential
    - firstName: String
      User's first name
    - secondName: String
      User's second name
    - email: String
      User's email

    - residenceCountry: String
      Country of residence
- didConnection
    - state: Number
    - id: String
    - invitationID

4.3 Deployment

For the first release, the MongoDB instance exploited in the KRAKEN marketplace will be a cluster deployed and provided by MongoDB's Atlas cloud clusters system.

4.4 Source Code

The code including schema definitions and the functions used to update them are included in the Marketplace API repository.

4.5 Baseline Technologies and Tools

The database itself is a MongoDB instance. The library used to integrate the database with the Marketplace API is mongoose [10].

5 MARKETPLACE FRONTEND

This section describes the Marketplace frontend. This component of the marketplace system is the tool that is used by the users to interact with the KRAKEN marketplace and perform all available operations.

5.1 Description

The Marketplace frontend is the software component within the marketplace system that is equipped with a Graphical User Interface (GUI) to allow users of the marketplace to perform the following operations: Registration and login on the platform, Data Product browsing, Data Product publication, Data Product purchase, Data Product dataset download (after purchase).

The frontend is integrated with an SSI wallet which enables the operations of SSI Verifiable Credentials issuing and presentation, which is used during the user registration, login and demonstration of a user's institutional affiliation in the marketplace. The frontend is integrated also with a Metamask wallet [11] for the operations of publishing and crypto payment to access Data Products within the marketplace, which is facilitated by a solidity smart contract [12] published on the xDai blockchain.

This component also sends requests to the Marketplace API and the Marketplace Smart Contract for any kind of operation related to Data Products publication, purchase and browsing.

5.2 Interfaces

A list of interfaces for the Marketplace frontend are provided below:

- Marketplace home page
  Browse Data Products that are listed based on the search criteria and the market sector.
- Sign In page
  Sign in on the marketplace using the SSI wallet app.
- Signup page
  Signup on the marketplace using the SSI wallet app.
- Connect wallet page
  Connect the metamask wallet to the platform to perform payment and publication on the marketplace smart contract.
- Data Product page
  Show all metadata about a specific Data Product and the policies set by the Data Provider, function to buy the data product and consume it.
- User products page
  Browse user's Data Products added to the platform.
- Edit product page
  Edit a Data Product's metadata, policies, price, publication.
- Subscription's page
  List user subscriptions.

5.3 Deployment

The deployment of this component follows the same process as the one described for the Marketplace API. Please refer to Section 3.3 for further details.

In addition to the configuration files specified in Section 3.3, this repository includes also an .ebextensions file that manages the deployment environment specific settings and configurations, e.g. number of file handlers on linux operating system.

5.4 Source Code

The code is available on a private repository on GitHub.

5.5 Baseline Technologies and Tools

The Marketplace frontend is a fork of the Streamr core-frontend repository publicly available on GitHub:

https://github.com/streamr-dev/core-frontend

The software is written in JavaScript using the React framework [13]. The relevant libraries already present on the streamr-core-frontend repository and added for the purpose of KRAKEN are web3, js-nacl and the web assembly library to perform user operations for SMPC.

6 MARKETPLACE SMART CONTRACT

This section describes the Marketplace Smart Contract. This component is used by the marketplace system to perform the publication of a Data Product on the xDai blockchain. It also allows users of the marketplace that are eligible to access a Data Product based on the data provider's predefined policies to make payments with an ERC20 token before accessing a data set.

6.1 Description

The Marketplace Smart Contract is the component responsible for the publication and purchase of Data Products on the xDai blockchain. The Data Product publication and purchase requests sent to this component come only from the Marketplace Frontend. This component is interfaced with the DataCoin ERC20 smart contract deployed on xDai, allowing users of the marketplace to exploit Streamr's DataCoin token for monetary transactions between data providers and data consumers within the marketplace.

6.2 Interfaces

A list of interfaces is provided below:

- getProduct(bytes32 id)
  Fetch the info of a product with a a specific id.
- createProduct(bytes32 id, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Creates a new product on the smart contract specifying all the parameters.
- createProductWithWhitelist(bytes32 id, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Creates a new product on the smart contract specifying all the parameters and enables it to be whitelisted.
- deleteProduct(bytes32 productId)
  Undeploys a product with a specific id.
- redeployProduct(bytes32 productId)
  Deploys an existing undeployed product with a specific id.
- updateProduct(bytes32 productId, string memory name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds, bool redeploy)
  Updates the product info.
- offerProductOwnership(bytes32 productId, address newOwnerCandidate)
  Offers the product ownership to another address.
- claimProductOwnership(bytes32 productId)
  Claims the product ownership if the ownership was previously offered by the previous owner.
- setRequiresWhitelist(bytes32 productId, bool _requiresWhitelist)
  Sets up a product to require whitelisting or not.
- whitelistApprove(bytes32 productId, address subscriber)
  Approve an address to be able to purchase a product.
- whitelistReject(bytes32 productId, address subscriber)
  Prevent an address to be able to purchase a product.
- whitelistRequest(bytes32 productId)
  Request to be whitelisted for a product with specific id.
- getWhitelistState(bytes32 productId, address subscriber)
  Fetch the state of the whitelist on a certain address.
- getSubscription(bytes32 productId, address subscriber)
  Fetch the subscription info of an address and a product.

- getSubscriptionTo(bytes32 productId)
  Fetch the subscription info of the transaction sender address and a product.
- hasValidSubscription(bytes32 productId, address subscriber)
  Checks if the specified address has a valid subscription on the specified product.
- grantSubscription(bytes32 productId, uint subscriptionSeconds, address recipient)
  Give free access to a data product to a specific address. This function can only be called by the product owner.
- buyFor(bytes32 productId, uint subscriptionSeconds, address recipient)
  Buy the product for a specific address.
- buy(bytes32 productId, uint subscriptionSeconds)
  Buy the product for the transaction sender address.
- updateExchangeRates(uint timestamp, uint dataUsd)
  Update the exchange rates of the token associated with the smart contract.
- getPriceInData(uint subscriptionSeconds, uint price, Currency unit)
  Get the current price of a product in the form of the token associated with the smart contract.
- halt()
  Prevent the execution of any function that could provoke modification to the internal state of the smart contract. This function can only be called by the smart contract owner.
- resume()
  Revert the actions of “halt()”
- reInitialize(address datacoinAddress, address currencyUpdateAgentAddress, address prev_marketplace_address)
  Override the current addresses of the token associated with the smart contract, the currency updater address and the previous marketplace address (if it exists).
- setTxFee(uint256 newTxFee)
  Sets a transactions fee that is sent to the smart contract owner on every paid purchase on the smart contract.

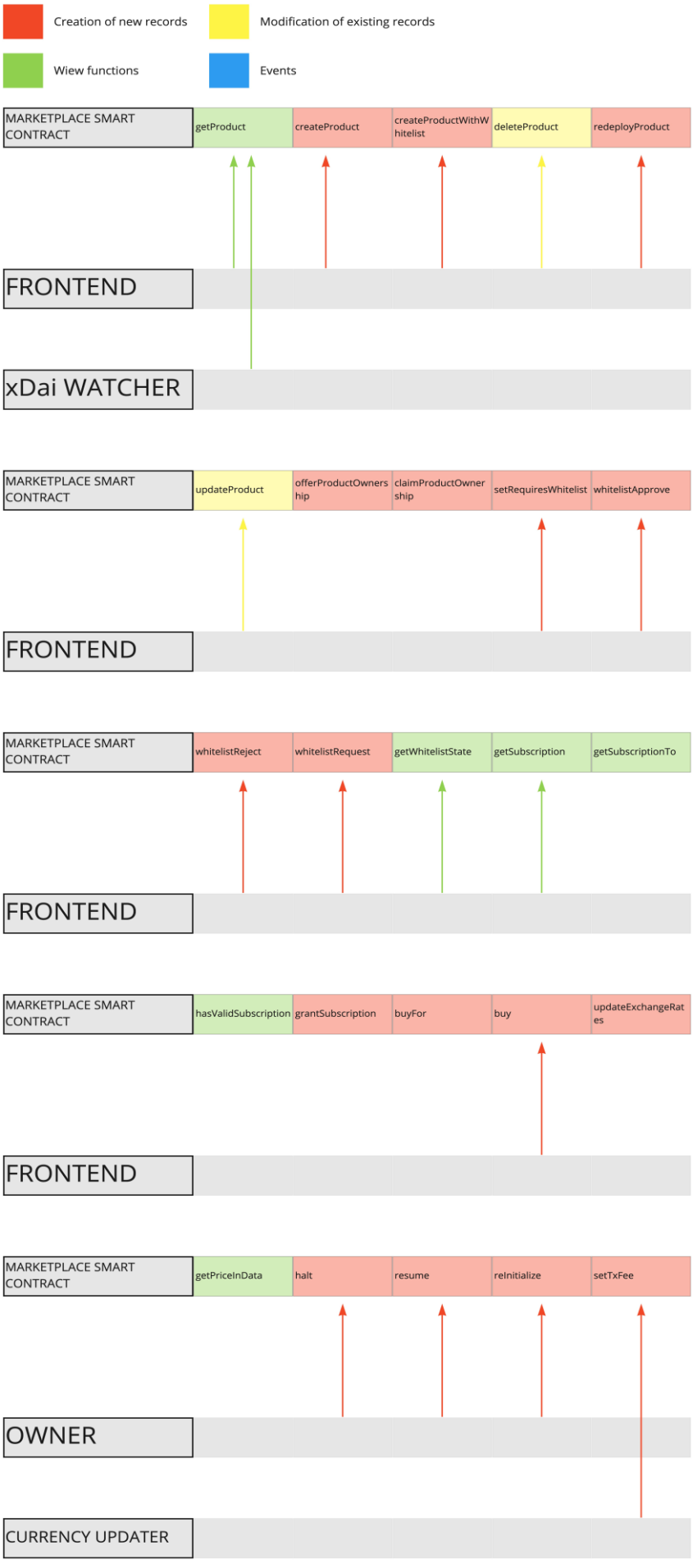
A list of events is provided below:

- ProductCreated(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Notifies about the creation of a new product.
- ProductUpdated(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Notifies about the update of a product providing all the info.
- ProductDeleted(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Notifies about the deletion of a product providing all the info.
- ProductImported(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  In the eventuality of the presence of a version 1 smart contract, it notifies about the import of a product providing all the info.
- ProductRedeployed(address indexed owner, bytes32 indexed id, string name, address beneficiary, uint pricePerSecond, Currency currency, uint minimumSubscriptionSeconds)
  Notifies about the redeployment of a product providing all the info.
- ProductOwnershipOffered(address indexed owner, bytes32 indexed id, address indexed to)
  Notifies about the product’s ownership offer to a new owner providing the info about current owner, product id and potential new owner.
- ProductOwnershipChanged(address indexed newOwner, bytes32 indexed id, address indexed oldOwner)

Notifies about the product's ownership change to a new owner providing the info about new owner, product id and old owner.

- Subscribed (bytes32 indexed productId, address indexed subscriber, uint endTimestamp)
  Notifies about a new subscription or an extension of an existing subscription providing the product id, the subscriber address and the end of the subscription.
- NewSubscription(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)
  Notifies about a new subscription.
- SubscriptionExtended(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)
  Notifies about the extension of an existing, not expired subscription.
- SubscriptionImported(bytes32 indexed productId, address indexed subscriber, uint endTimestamp)
  In the eventuality of the presence of a version 1 smart contract, it notifies about the import of a subscription providing all the info.
  Notifies about the import of a subscription from a
- ExchangeRatesUpdated(uint timestamp, uint dataInUsd)
  Notifies about the change of the exchange rates used by the smart contract to convert DATA COIN in dollars or euros and viceversa.
- WhitelistRequested(bytes32 indexed productId, address indexed subscriber)
  Notifies about the request of being whitelisted to purchase a product.
- WhitelistApproved(bytes32 indexed productId, address indexed subscriber)
  Notifies about the approval of whitelisting request
- WhitelistRejected(bytes32 indexed productId, address indexed subscriber)
  Notifies about the rejection of whitelisting request
- WhitelistEnabled(bytes32 indexed productId)
  Notifies about the enabling of the whitelisting functionality on a product.
- WhitelistDisabled(bytes32 indexed productId)
  Notifies about the disabling of the whitelisting functionality on a product.
- TxFeeChanged(uint256 indexed newTxFee)
  Notifies about the change of the marketplace's owner transactions fee.

Figure 5 below shows the above listed interfaces and events in a visual form, including the components in the marketplace system that use them (if any).


Creation of new records
Modification of existing records
Wiew functions
Events
MARKETPLACE SMART CONTRACT
getProduct
createProduct
createProductWithWhitelist
deleteProduct
redeployProduct
FRONTEND
xDai WATCHER
MARKETPLACE SMART CONTRACT
updateProduct
offerProductOwnership
claimProductOwnership
setRequiresWhitelist
whitelistApprove
FRONTEND
MARKETPLACE SMART CONTRACT
whitelistReject
whitelistRequest
getWhitelistState
getSubscription
getSubscriptionTo
FRONTEND
MARKETPLACE SMART CONTRACT
hasValidSubscription
grantSubscription
buyFor
buy
updateExchangeRates
FRONTEND
MARKETPLACE SMART CONTRACT
getPriceInData
halt
resume
reInitialize
setTxFee
OWNER
CURRENCY UPDATER

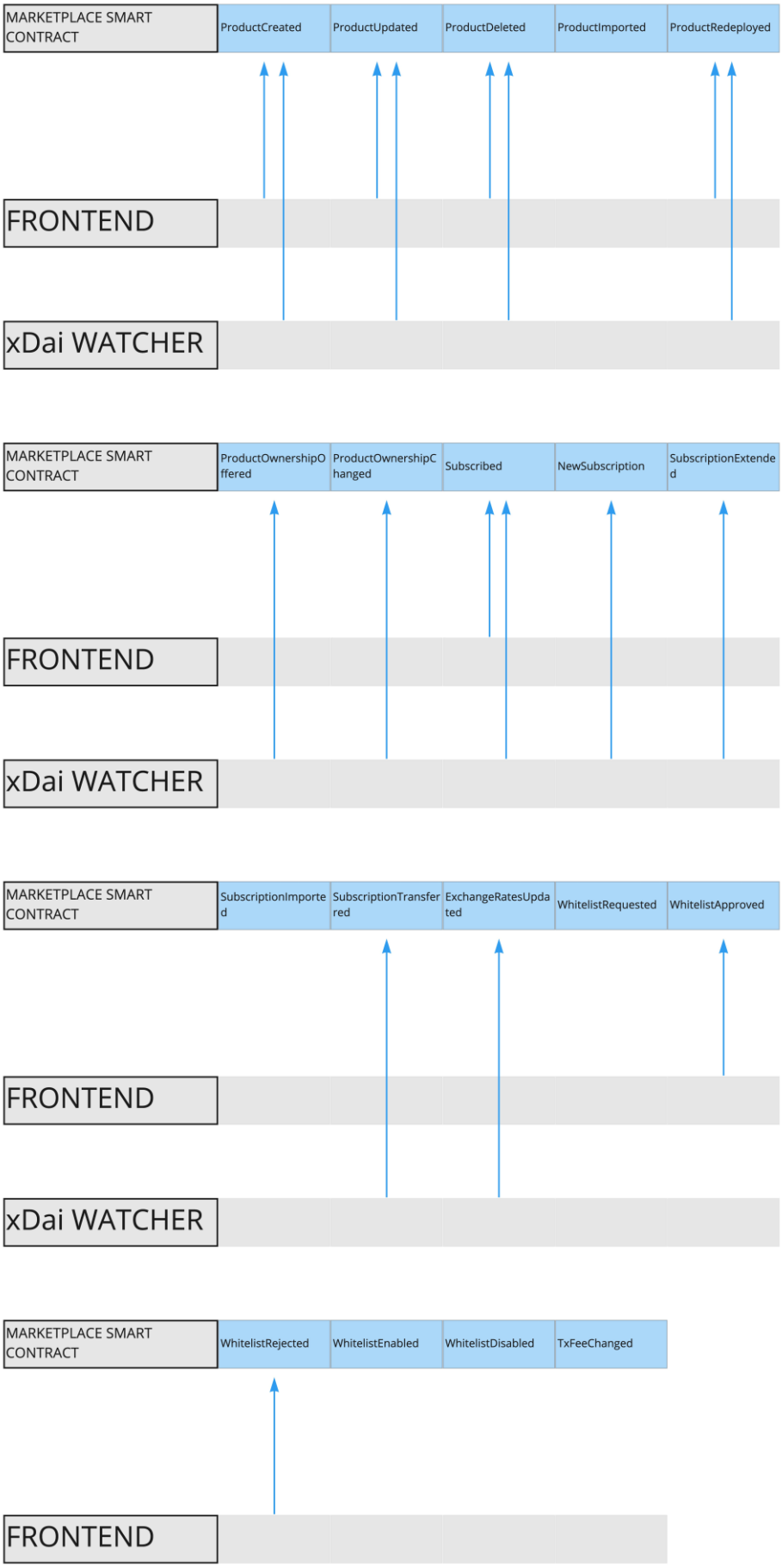


Figure 5: Marketplace Smart Contract Interfaces

6.3 Deployment

The deployment of the Marketplace Smart Contract is performed on the xDai blockchain. Currently the Smart Contract is deployed and has the following address:

0x4B36680C67EFa16AED5a693726c20dB59428859C

The deployment has been performed using the Truffle library migration functionality [14]. The smart contract can be also monitored on blockscout at the following address:
https://blockscout.com/xdai/mainnet/address/0x4B36680C67EFa16AED5a693726c20dB59428859C/transactions

6.4 Source Code

The source code is available on a private repository on GitHub at the following link:

https://github.com/technology-exploration/kraken-marketplace-contracts

6.5 Baseline Technologies and Tools

The Marketplace Smart Contract is a fork of the Streamr-marketplace-contracts repository publicly available on GitHub:
https://github.com/streamr-dev/marketplace-contracts

The modifications performed for KRAKEN include the setup of the deployment of the Marketplace Smart Contract to occur on the xDai blockchain instead of Ethereum and other modifications related to the Smart Contract that have contributed also to the Streamr-marketplace-contracts repository. Specifically, the above-mentioned modifications have been applied to make the smart contract independent from an already deployed version 1 of the Streamr marketplace smart contract. The modifications are publicly visible at the following link:

https://github.com/streamr-dev/marketplace-contracts/pull/44/

The software is written in Solidity using the React framework. The relevant libraries already present on the Streamr marketplace contracts repository are Openzeppelin, Truffle, web3, mocha.

7 MARKETPLACE XDAI WATCHER

This section describes the Marketplace xDai Watcher, this component is used by the Marketplace API to get updates on events happening on the Marketplace Smart Contract and perform consequent actions.

7.1 Description

The Marketplace xDai Watcher is an intermediary between the Marketplace Smart Contract and the Marketplace API. Its role is to update the Marketplace API about any new events registered on the Marketplace Smart Contract, including Data Product publication, modification, deployment, subscription and deletion.

7.2 Interfaces

The Marketplace xDai Watcher does not offer any interface. However, it subscribes to the following list of events on the Marketplace Smart Contract:

- ProductCreated
- ProductRedeployed
- ProductDeleted
- ProductUpdated
- ProductOwnershipChanged
- Subscribed

When any of these events are triggered, the Marketplace xDai Watcher updates the Marketplace API on the following interfaces:

- POST/products/:id/setDeployed
- POST/products/:id/setUndeployed
- POST/products/:id/setPricing
- POST/subscriptions

7.3 Deployment

The deployment of this component follows the same process as the one described for the Marketplace API. Please refer to Section 3.3 for further details.

7.4 Source Code

The source code for the Marketplace xDai Watcher is available on a private repository on GitHub.

7.5 Baseline Technologies and Tools

The Marketplace xDai Watcher is a fork of the streamr-marketplace-contracts repository, which is publicly available on GitHub:

https://github.com/streamr-dev/streamr-ethereum-watcher

The modifications performed for KRAKEN include the setup of the Watcher to listen to a Marketplace Smart Contract deployed on the xDai blockchain and to send updates on the Marketplace API endpoints.

8 CONSORTIUM BLOCKCHAIN NODE

This section describes the Consortium Blockchain Network, a set of organization's hosting nodes that run the smart contracts which handle the marketplace catalogue, user related data and data access control policies.

8.1 Description

The Consortium Blockchain Network (Built on Hyperledger Fabric [15] at the First Release consists of two Organizations: Lynkeus and Tex. Each organization hosts two peer Nodes. Additionally, there is a third logical Organization handling the Ordering Service which is called Orderer Organization and is under the management of Lynkeus and Tex.

The Peer Organizations are joined in a channel in which a Chaincode (Package containing Smart Contracts) is deployed. This Chaincode handles the user preferences and other related data regarding the marketplace, data catalogue operations, as well as enforcing access control policies on the data between a buyer and seller creating agreements.

Marketplace users interact with the Blockchain Network in the following ways:

- They register/enroll to an Organization's Certificate Authority to obtain crypto material (certificate) that allows them to perform chaincode operations.
- They can call chaincode functions via the marketplace UI such as creating the account, creating a product or buying data.

For more information on the Blockchain Network and Hyperledger Fabric refer to Deliverable 5.3.

8.2 Interfaces

8.2.1 Application (Node JS) API

CA Services

- registerAppUser. Register a user on the selected Certificate Authority (CA)
- enrollAppUser. The user enrolls using a Certificate Signing Request and obtains a signed certificate.
- updateUser. Update user data on the CA
- deleteUser. Delete a user from the CA
- isAdmin. Check if a certificate belongs to an admin
- reenrollAppUser. Reenroll a user to obtain a new certificate
- getExpirationDate. Get the max expiration date among all the certificates of a user

Cache Queries

- queryUsers. Query all users
- queryUser. Query user by username
- queryProducts. Query all products
- queryProduct. Query product by id
- queryCatalogue. Query the available products (All products with non-expired certificates)
- queryFilteredProducts. Query and filter the catalogue to match the browsing user's preferences

Block Listening

- createBlockListener. Create a listener that fetches each newly appended block
- removeBlockListener. Remove a listener
- handleTransactionData. Extract all the Events and EventData from the fetched blocks and forward them to the database

Util

- connectGateway. Connect to the Fabric Blockchain Network (all nodes) using a connection profile.

Signing Offline

- sendTransaction. Sign a transaction manually on the user side and send it to the network.

Documentation is generated using jsdoc and will be published at a later stage.

8.2.2 Chaincode API

User Credentials

- CreateUser. Create a user account on the ledger
- UpdateUser. Update a user account
- ReadUser. Read user data
- DeleteUser. Delete a user account

Data Catalogue

- CreateProduct. Create a product
- UpdateProduct. Update a product
- ReadProduct. Read product data
- DeleteProduct. Delete a product
- BuyProduct. Calls Agreements Contract to validate the eligibility of the buyer to access this product

Agreements

- NewAgreement: Store a new agreement if the validation is successful and the transaction status
- UpdateAgreement: Update an agreement's status

Documentation is generated using godoc and will be published at a later stage.

8.2.3 Installation APIs (CLI)

Regarding the setting up of the network and the organizations, we have created APIs that simplify the most essential processes of a peer and a CA client. These APIs can be used by a network operator to set up an organization, join a peer to channel, install chaincodes, manage the CAs, and so on.

CA Client API Tasks
Register user to TLS CA / CA
Enroll user
Re-Enroll user
View list of Identities in CA
Setup and launch CA Server
Setup Organization MSP

Table 2: CA Client API Tasks

Peer API Tasks	
Creates a channel transaction from profile config	Install chaincode to peer
Submit channel transaction to orderer	Query installed chaincodes on peer
Joins a peer to channel	Approve chaincode as Org
Create and submit anchor peer update transaction	Query approved chaincodes on channel
List channels a peer has joined	Check commit readiness of chaincode
Package a chaincode	Commit chaincode definition to channel
Query committed chaincodes on channel	Fetch configuration of channel
Create an update transaction to add an organization	Sign configuration transaction as organization
Start a node, peer/orderer	

Table 3: Peer API Tasks

8.3 Deployment

Each of the components mentioned above represents a different physical or virtual node. In our implementation, each node is deployed as a docker container inside a Virtual Machine (VM) hosted in cloud services. The TLS CA is only used for the enrollment of nodes, so after the initial enrollment, it will be down if another node will not be joined to the network. Thus, we have used one VM for both CAs for better resource management.

The individual network nodes are depicted in the following figure.

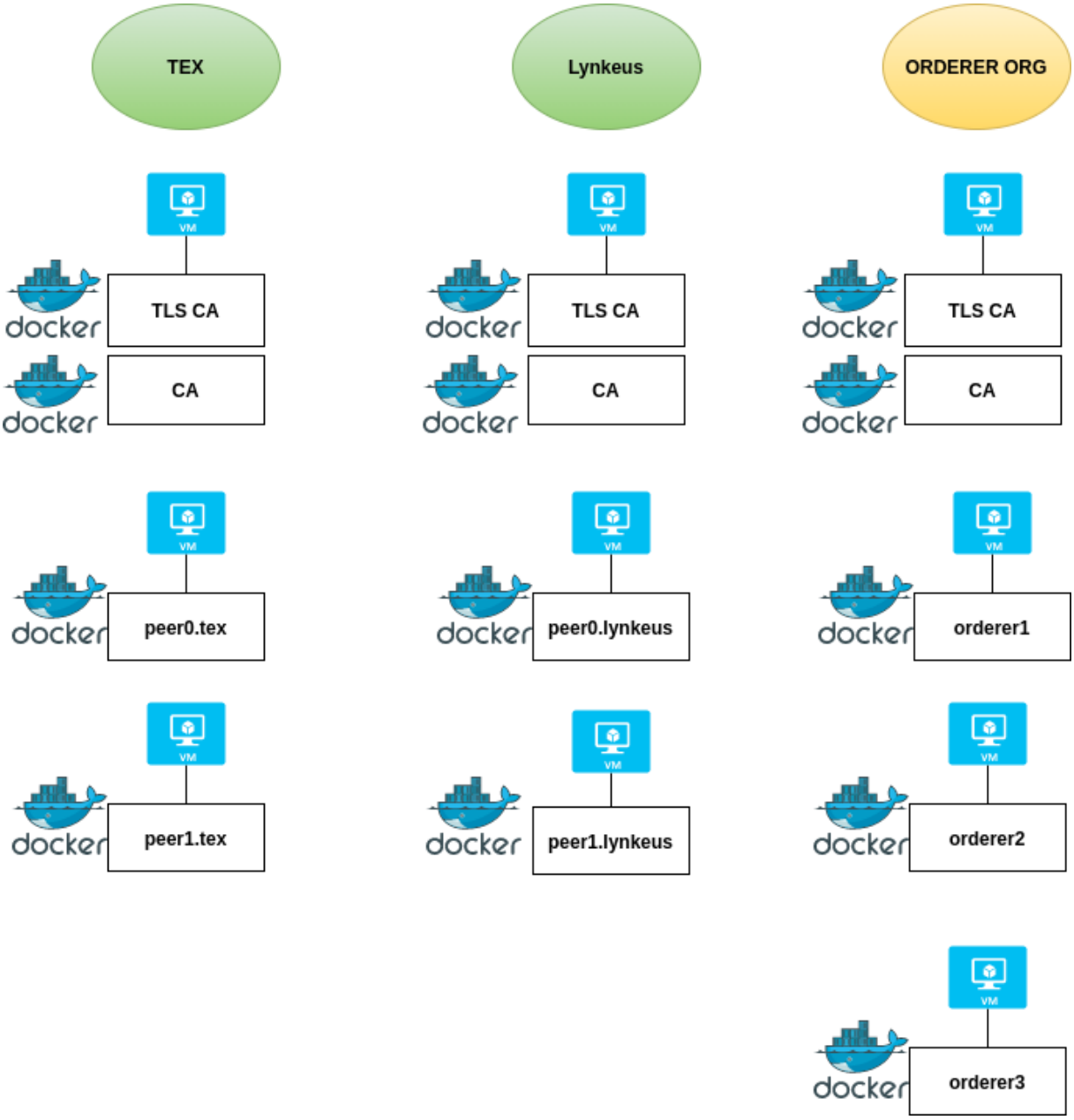


Figure 6: Individual network nodes

The deployment of the network is a collaborative effort between the organizations and requires several steps to be fully set up. The steps are well defined in a file called “DEPLOY.md” in the repository.

8.4 Source Code

The Hyperledger Fabric network code is currently stored on a private repository of the developer and is shared with members of other teams.

8.5 Baseline Technologies and Tools

8.5.1 Private Network

Hyperledger is the framework leveraged in Kraken building on the MHMD work on this front, expanded for the specific requirements at hand. Specifically, the Fabric instance can achieve two times more transactions per second (TPS) with an appealing implementation and code structure from a development standpoint.

8.5.2 Cache Database

This database is a MongoDB instance.

8.5.3 Smart Contracts

HLF smart contracts are written in GO, which is strict in terms of syntax and logic and is thus of great use when writing smart contracts. These contain data access parameters that are computed by the filtering algorithm running on the Blockchain to compute data access permissions.

8.5.4 Application

The backend application is written in Node.js and is built mainly using HLF's Node SDK which utilizes functionalities to provide connection and interaction with the blockchain network. The application is the middleware between the network and the user and has been optimized to achieve high throughput, security, and privacy.

8.5.5 Documentation

The smart contracts code is documented using godoc and its standard code documentation practices. In respect of the Node.js application, we used the tool JSDoc.

9 MPC NODE

This section describes the MPC nodes component. This component is used by the KRAKEN platform as a mechanism for key distribution and will also be used to perform secure computation / analytics on published datasets. We briefly summarize the component here, however for further details please refer to Deliverable 4.3.

9.1 Description

Secure Multi-Party Computation allows users to evaluate various programs including analytics on encrypted data, without revealing the data itself. This is achieved by splitting the queries among MPC nodes that can jointly and securely perform the desired computation. The privacy of the dataset is guaranteed by the decentralization.

The KRAKEN platform will use MPC as a mechanism for the key distribution and to evaluate various analytics on the datasets, see D2.4 and D5.3 for further details on MPC and a detailed explanation of these two applications.

9.2 Interfaces

A MPC node is a service interfacing in the following way:

- Accepts requests from the Marketplace API component.
- Returns results of the computations to the Marketplace API component.
- Communicates with other MPC nodes to securely evaluate functions.

All the above communication is done through sockets. See D4.3 for a detailed explanation of the structure of the requests and responses. However please note that while the MPC nodes directly communicate only with the Marketplace API, this component serves only as a connector between data sellers, data buyers and the MPC nodes in the KRAKEN platform. No data can be revealed to the Marketplace API.

9.3 Deployment

MPC nodes can be deployed as Docker containers with specified addresses of the other MPC nodes. In the first release we will deploy 3 nodes on XLAB servers, while in the following releases the nodes will be distributed among the partners, presumably XLAB, ATOS and TEX.

9.4 Source Code

All of the source code is available to all the partners in the KRAKEN consortium in a private repository. We plan to make this repository public in the future.

9.5 Baseline Technologies and Tools

The MPC node component is implemented in Go programming language managing the communication and scheduling of tasks. It uses a fork of SCALE-MAMBA [16] https://github.com/KULeuven-COSIC/SCALE-MAMBA, for building and evaluating secure multi-party computations. It uses a homomorphic proxy encryption protocol for decentralized key management, please see D5.3 for further information.

10 MARKETPLACE SSI AGENT

To deploy an SSI solution that ensures a smooth interaction between the user and the marketplace for registering, authenticating and sharing data within the marketplace, different components of the SSI must be integrated in the architecture. There is a specific component called Ledger uSelf Broker that will be responsible for facilitating the integration with the marketplace.

10.1 Description

The Ledger uSelf Broker will be located between the Marketplace services and the SSI Server Agent. This way, it will act as a facilitator between the marketplace and the user (via mobile app), easing the complex interactions in the key processes of an SSI solution like the exchanging of the DIDs, when the verifiable credentials are issued or when the proof shall be presented. For that purpose, only three requests are needed, one for each of these processes. This comes from the Hyperledger Aries protocol implementation used in the SSI approach.

More information related to the mentioned SSI Server Agent and the other SSI components, i.e. the SSI Mediator Agent and the SSI Mobile Agent can be found in the deliverable D3.1 Self-Sovereign Identity Solution First Release, where it is explained that those agents will use the underlying framework of Hyperledger Aries.

10.2 Interfaces

It is possible to make a distinction between the entry points of the Ledger uSelf Broker, having a few for the usage and the rest for configuring and managing the server.

The following are dedicated for the usage:

- POST/connections/generate-invitation
  For generating an invitation to be translated into a QR code
- POST/issue-credential/issue
  For issuing a credential.
- POST/present-proof/request-proof
  For requesting a proof
- GET/kms/init
  Initialization method

More information regarding the rest of the interfaces can be found in D3.1.

10.3 Deployment

Regarding the deployment, this component must be deployed as a docker container following the diagram in Figure 7 and the instructions together with the source code (see Section 10.4).

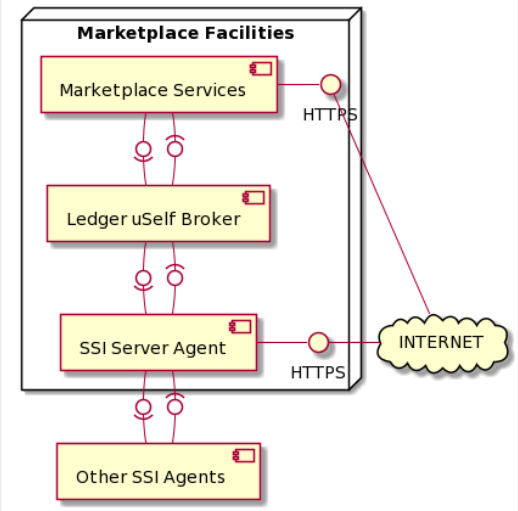


Figure 7: Ledger uSelf Broker deployment

10.4 Source Code

The source code for the Ledger uSelf Broker is in a private repository.

There it is possible to obtain not only the source code but all the needed information for the deployment, configuration and some instructions on the usage.

10.5 Baseline Technologies and Tools

The implementation of the Ledger uSelf Broker has been made in Kotlin language [17], based on the Go language implementation provided by the Hyperledger Aries framework [18].

11 Conclusion

This D5.5 document describes the current software of the initial KRAKEN marketplace implementation. As part of this initial release a first marketplace implementation has been established for both the education and biomedical pilot domains which allows data provider and data consumers to exchange data using the encrypted batch data transfer through directory sharing modality.

This implementation has resulted from intensive design and development efforts to link multiple technology components at varying levels of technological maturity. The high-level technological components include a forked and heavily adapted instance of the Streamr marketplace codebase, the Lynkeus MHMD consent management / data access layer (Hyperledger Fabric Blockchain), the Self Sovereign Identity system and the SMPC Network, which in this first release is used as a mechanism for key distribution in the batch data sharing use case and in a future release will be used to perform secure privacy preserving computations / analytics on data.

In the following months of the project until the final marketplace release in July 2022 (month 32), development efforts will continue to further mature and polish the marketplace user workflows and GUI for navigating the marketplace, and publishing, consuming and managing batch Data Products.

As discussed earlier in this paper, an initial integration with the SSI has been realised, integrating with the Agent's uSelf broker. However, the present proof functionality and the possibility to specify a credential ID must be integrated to finalise the processes of registration and login using SSI. It is expected that this shall be completed in advance of the first round of user evaluations. The marketplace mobile application is also being developed on React Native and React.js javascript frameworks to seamlessly interact with the rest of the components in the platform. The application will run on Android phones.

Further future efforts will also be focused into the development of the remaining two data sharing modalities which include Secure Multiparty Computation, where the marketplace will further exploit the SMPC network for privacy preserving analytics, and the transfer of encrypted streaming data, which will exploit the Streamr Network for real-time data transportation.

All updates and changes to the marketplace software between now and July 2022 will be reflected in the final report on the marketplace software accompanying Deliverable 5.6 - KRAKEN Marketplace Final Release.

12 References

[1] https://streamr.network/discover/marketplace

[2] http://www.myhealthmydata.eu/

[3] https://streamr.network/docs/data-token

[4] https://www.xdaichain.com/

[5] https://github.com/

[6] https://travis-ci.org/

[7] https://aws.amazon.com/

[8] https://aws.amazon.com/elasticbeanstalk/

[9] https://www.mongodb.com/

[10] https://mongoosejs.com/

[11] https://metamask.io/

[12] https://docs.soliditylang.org/en/v0.4.24/index.html

[13] https://reactjs.org/

[14] https://www.trufflesuite.com/docs/truffle/getting-started/running-migrations

[15] https://www.hyperledger.org/use/fabric

[16] https://wiki.mpcalliance.org/Scale_Mamba.html

[17] https://kotlinlang.org/

[18]https://github.com/hyperledger/aries-framework-go

Atos

FONDAZIONE BRUNO KESSLER

AIT AUSTRIAN INSTITUTE OF TECHNOLOGY

sic

LYNKEUS
STRATEGY CONSULTING | BLOCKCHAIN & SMART CONTRACTS | DATA ANALYTICS

XLAB

TX

KU LEUVEN CiTiP
CENTRE FOR IT & IP LAW

IAIK TU Graz

InfoCert
TINEXTA GROUP

@KrakenH2020

Kraken H2020

www.krakenh2020.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871473