5

Distributed Data Storage System Considerations

5.1 Objectives

Every federated information system requires means to store and share data securely. The i3-MARKET network is not an exception; hence, a well-thought solution that is secure, reliable, and usable by all entities in the i3-MARKET network is needed. The aim of data storage is to store common data in a federated network of data marketplaces. The common data shared between participating data marketplace instances may include identity information, shared semantic models, meta-information about datasets and offerings, semantic queries, sample data, smart contract templates and instances, crypto tokens, and payments. No single party should fully control the data storage system and there shall be no single point of failure. In order to fulfil the needs of the aforementioned data types, two separate storage solutions are used: the decentralized and the distributed one.

The former supports the management of distributed identities and smart contracts. However, the latter has an important role in data synchronization between different i3-MARKET nodes and, optionally, storage of datasets on sale. Moreover, the distributed storage supports non-repudiation service and auditable accounting features of i3-MARKET.

The design of the distributed storage has been an iterative process.

Data storage system takes full advantage of available base technologies and builds on top of these in order to satisfy i3-MARKET needs and requirements, with a focus on federated system architecture. The underlying technologies chosen for decentralized and distributed storage means are Hyperledger BESU and CockroachDB, respectively.

The federated query engine index (SEED Index) management solution is available and integrated into the i3-MARKET network, deployed as a smart contract on Hyperledger BESU. Moreover, a solution called verifiable data

integrity has been implemented on top of auditable accounting to further increase the reliability of data. And, finally, access management solution governing the data access has been designed and implemented, depending on reliable and secure key management solution.

The common data shared between participating data marketplace instances may include identity information, shared semantic models, meta-information about datasets and offerings, semantic queries, sample data, smart contract templates and instances, crypto tokens, and payments. No single party should fully control the data storage system and there shall be no single point of failure.

The high-level capabilities that the data storage aims to provide are:

1. Decentralized storage
2. Distributed storage

The decentralized storage shall provide the highest available security guarantees in a federated network. The decentralized storage subsystem is built on a secure Byzantine fault-tolerant consensus-based distributed ledger. Due to high security requirements, the performance and storage space of such a system may be relatively limited compared to conventional databases.

The distributed storage shall provide a database-like subsystem that is scalable, deployed on all i3-MARKET nodes, has a rich query interface (SQL), and can handle large amounts of data, while the i3-MARKET shall rely on the API of the decentralized storage provided out-of-box.

5.2 Solution Design/Blocks

The storage system consists of two main subsystems for implementing the decentralized storage and distributed storage features, respectively. The subsystems are relatively independent of other systems and also with each other.

The diagram of a decentralized storage subsystem is shown in Figure 5.1. The decentralized storage subsystem is implemented as a blockchain-based distributed ledger network. The software implementation is Hyperledger BESU in a permissioned setup using IBFT 2.0 consensus. Hyperledger BESU uses internally an embedded RocksDB instance for storing linked blocks (the journal of transaction) and world state (the ledger). Hyperledger BESU can instantiate and execute smart contracts for supporting the use cases of i3-MARKET framework (Figure 5.1).

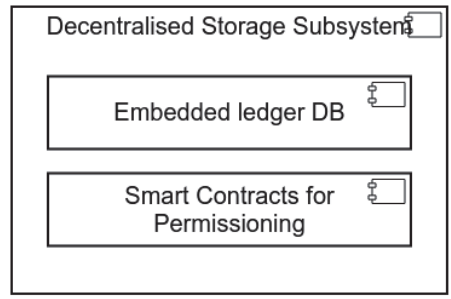

Figure 5.1 Decentralized storage subsystem.

The components depending on the decentralized storage subsystem uses Hyperledger BESU's native JSON-RPC-based interface. A separate interface layer for accessing (or limiting access to) decentralized storage is not planned, as the nodes of the decentralized storage will already validate all transactions submitted to the ledger. The diagram of a distributed storage subsystem is shown in Figure 5.2. The subsystem consists of database nodes. The database provides an SQL interface to other i3-MARKET framework components. The software implementation database is CockroachDB that can be accessed via PostgreSQL-compatible wire protocol for which a large number of client libraries exist in different languages and platforms. Only secure access (TLS with mutual authentication) to the database will be enabled; hence, all clients need to use private keys and valid certificates to access the database.

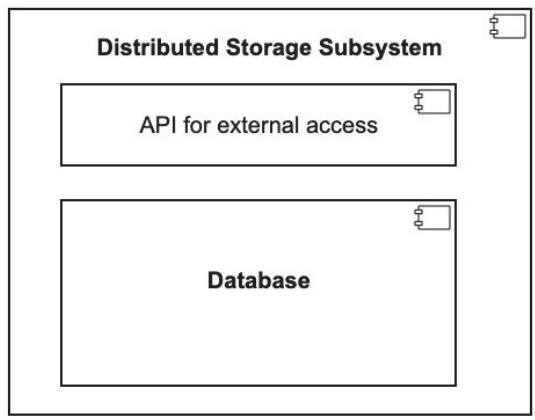

Figure 5.2 Distributed storage subsystem.

The distributed storage component is an internal component with no external access. That is to say that it will have connections only with other trusted services within the i3-MARKET Backplane. Even though this simplifies the necessary measures in terms of authentication and authorization, it is still needed to secure machine-to-machine connections between the i3-MARKET services since they can be deployed on shared infrastructure.

The authentication and authorization solution relies on providing the distributed storage behind a TLS server endpoint and requiring TLS client certificates for the different connecting services. The setup guarantees end-to-end security between the distributed storage service and any of its client services.

The governance of the certificates has followed up to now the *keep it simple* approach. The distributed storage system is in charge of issuing the servers' and clients' certificates, unless the instance has its own certificate authority (CA), in which case the CA is responsible for issuing server certificates to the distributed storage server component and client certificates to the clients.

5.2.1 Service availability

The storage subsystem is a critical component of the i3-MARKET network contributing to the proper functioning of the platform. Hence, appropriate measures in the form of design, choice of technologies, and deployment have to be applied. Fortunately, the two main subsystems used in the storage solution already have strong built-in availability features that are summarized below.

5.2.1.1 Distributed storage

The distributed storage solution is based on a CockroachDB server. Initially, the database was deployed as a global cluster of database nodes; however, after the initial testing of the entire network, a couple of issues were discovered. First, the deployment of a CockroachDB instance and connecting it to a cluster is not an automated process, but rather manual as configuration steps must be tightly coordinated between the nodes. This contradicts with the overall concept of i3-MARKET, which should be operable without any central administration.

The second and far greater problem, which was eventually acknowledged, is that each node in a CockroachDB cluster has equal rights with full administrative privileges over the cluster. This is a problem because any node can

alter data and there is no consensus mechanism to agree on the changes. Furthermore, in case of the rise of a rogue node could potentially lead to full erasure or silent corruption of the entire database.

Therefore, a decision was made to replace the global cluster with independent clusters deployed at each i3-MARKET instance. In this deployment mode, each instance is responsible for its own operation and a configuration mistake in one instance, or a malicious act cannot affect the stored data at other instances. There was only one implication to this change – SEED Index would not work in such a setup anymore. As a result, the index was migrated from the distributed storage to the decentralized storage.

5.2.1.2 Decentralized storage

The decentralized storage used in the platform is a Hyperledger BESU network, which uses the IBFT 2.0 (proof of authority) consensus protocol. In this network, there are four validator nodes based on the genesis configuration stored in the corporative Nexus. In this configuration, there are three accounts to be used by the i3-MARKET federation.

The federated search engine index service uses the Hyperledger BESU blockchain as its storage backend. For this purpose, a smart contract storing the endpoints of all SEED instances along with the associated data categories has been deployed on the blockchain.

In this scenario, different components like auditable accounting, SEED Index, etc., are capable to deploy and manage smart contracts and transactions over those accounts.

5.2.2 Verifiable database integrity

The purpose of the VDI is to provide an infrastructure for data to be stored in a way that its presence, or lack thereof, can be cryptographically proven. It takes advantage of the blockchain technology to determine the integrity of the data it contains.

The VDI component consists of a library that implements a Compact Sparse Merkle Tree (CSMT)[1] and exposes an API that allows for data to be inserted, retrieved, updated, and removed. The API consumer can later obtain proofs of membership/non-membership and verify those proofs against the existing Merkle tree.

[1] Compact Sparse Merkle Trees: https://eprint.iacr.org/2018/955.pdf

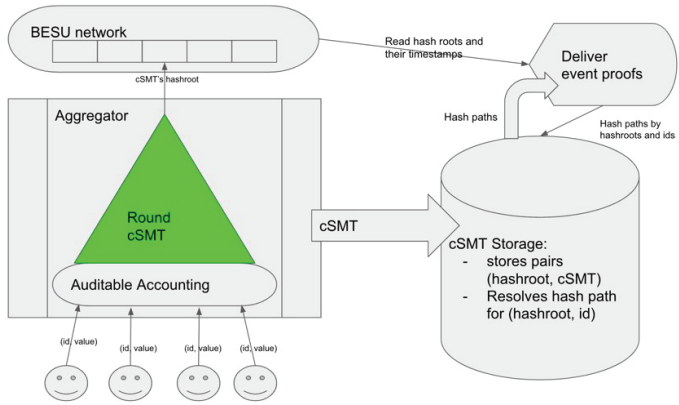

Figure 5.3 VDI integration with auditable accounting.

This data structure works on the same principles of verifiable maps[2], periodically generating a root hash that, after been made public, can guarantee the integrity of the data at that point in time. Membership or non-membership of any given key can be cryptographically proven against the Merkle tree.

- **Integration:**

The VDI is not a separate application on its own but is integrated into the auditable accounting component, as can be seen in Figure 5.3. The MerkleTreeService class exposes methods to create a CSMT tree from the array of hashes obtained from registries. The individual proofs are then stored along with their corresponding root hash into the registries repository in the database. The unregistered blockchain record is also stored in the database along with the serialized Merkle tree data itself.

The library[3] is implemented in TypeScript and distributed as a node package. The main data structure of the VDI is a class called CSMT. This class keeps a map of all the nodes that are stored in the tree as well as the tree's root hash. On an empty tree, the root hash is initialized as a zero node. Data (in key-value format) can then be inserted into the tree, producing new nodes and updating the root hash.

What follows is a summary of the functions that are available for the consumers of the CSMT class.

[2] Verifiable data structures, p. 2: https://continusec.com/static/VerifiableDataStructures.pdf
[3] https://github.com/i3-MARKET-V3-Public-Repository/SP4-VerifiableDatabaseIntegrity

- **Add:**

This method adds a given key-value pair to the tree. The key has to be in byte array format. The value can be any arbitrary string and is optional. If the provided key already exists in the tree, an error is returned, and no duplicate keys are allowed. The data is combined in a new array that contains the key in a hexadecimal format, the hash of the value and an entry mark that flags this as a leaf node. This data is then inserted into the nodes map, where the hash of the data is, in itself, the key for this record in the map. Finally, the tree's root hash is recalculated.

- **Insert:**

This is a convenient method to insert data in bulk to the tree. This method just validates the data and calls the method *add* above on each individual element.

- **Get:**

This method looks up for a given key in the nodes map. It returns the hash of the corresponding value for the key, or *undefined* if the key is not present.

- **Delete:**

This method looks up for a given key in the nodes map and removes it. The tree's root hash is then recalculated based on the remaining nodes. If no key is found, an error is returned.

- **Create proof:**

This method looks up for a given key in the nodes map and creates a new proof object. The proof object contains the data itself (if present), the chain of additional nodes along the tree traversal, the root hash of the tree, and a membership flag (true if the given key is present in the tree, false otherwise).

- **Verify proof:**

When the consumer has a proof object, it can verify whether that proof matches against the existing tree by calling this method. It verifies whether the root hashes and node chain (in case of membership) matches with what the CSMT class has stored internally. It returns true if the proof matches. If it returns false, it means that either the proof does not belong to this tree or that the proof was tampered with.

5.2.3 Federated query engine index management

The decentralized storage sub-component of data storage provides functionality to manage an index used for semantic data discovery. The federated query engine index supports federated queries, a concept implemented by the semantic engine. The distributed storage plays a vital role in supporting the verifiable data integrity, non-repudiation service, and auditable accounting.

Decentralized storage implements two main use cases – managing the index and querying the index – in order to provide the required functionality to the semantic engine for accessing the content of the index.

The index is a collection of data categories together with the endpoint location addresses of the corresponding semantic engines. One semantic engine is not limited to storing offerings belonging to one category but to several of them. Hence, the index contains one to many relationships, linking a specific semantic engine to a set of data categories.

Each semantic engine instance has a private key, while the corresponding public key serves as an identifier that is associated with a set of SEED Index records. The private key is needed to update corresponding index records. Moreover, in order to pay for update transaction, the SEED account must have enough resources. The owner of the SeedsIndexStorage smart contract can assign administrator roles to other keys that can update records stored under any public key.

Every new marketplace joining the i3-MARKET network will connect to the decentralized storage through a semantic engine. If the marketplace has been around for a while, the marketplace has most probably stored offerings metadata. This metadata should also be stored in the SEED to participate in federated queries. Therefore, such a marketplace would have to populate the index by inserting category information to the decentralized storage.

- **Manage:**

In order to provide the most recent and accurate information to the semantic engines in the i3-MARKET network, the index must be kept up to date at all times. Therefore, functions – insert, update, and delete – to maintain the index are required. All these activities are limited to registered i3-MARKET nodes only and the authentication uses self-signed certificates.

- **Insert:**

Before an i3-MARKET instance receives any data offering registrations, the semantic engine has no reason to insert any content into the index. Although it is possible to insert an empty entry containing the endpoint address and an empty category list to the index, it is recommended to keep the

index clean of unnecessary information. After receiving the first data offering, the semantic engine inserts the first entry to the index, revealing to other i3-MARKET instances the category of offerings stored in that specific semantic engine.

- **Update:**

Over the course of the market lifecycle, data offerings of different data categories are stored in a single marketplace. Upon the registration of a data offering belonging to a category that is not yet present in the semantic engine, the semantic engine updates the index with relevant information (data category, endpoint address, etc.) by inserting a new entry to the database.

- **Delete:**

The final management activity of the index lifecycle allows the removal of entries from the index. It is the responsibility of the semantic engine to keep the index up to date; therefore, redundant and outdated information is removed from the index. In the event of closing down of an i3-MARKET marketplace instance, either temporarily for maintenance or indefinitely, the semantic engine has to remove unavailable content from the index. Moreover, this function should be accessible by a system administrator to remove relevant entries from the index, in case of a sudden shut down of a marketplace/i3-MARKET node.

- **Query:**

In case a semantic engine needs to perform a federated query among all other instances in the i3-MARKET network, the index shall provide input to the federated query. The semantic engine firstly queries the index with relevant parameters (data category, description, etc.) and the distributed storage shall return information from the index indicating which i3-MARKET instance contains the data that the SEED is looking for.

5.3 Diagrams

Federated query engine index management:

The sequence diagram in Figure 5.4 shows the interaction of the decentralized storage on the SEED regarding the federated query engine index management. Each function – insert, update, delete, and query – has been depicted on a single sequence diagram, as there is no relevant complexity to be shown for each interaction. Index record identifiers (*uuid* in the figure) are derived from node public keys via cryptographic hashing and all requests must be signed with an authorized key (e.g., corresponding private key).

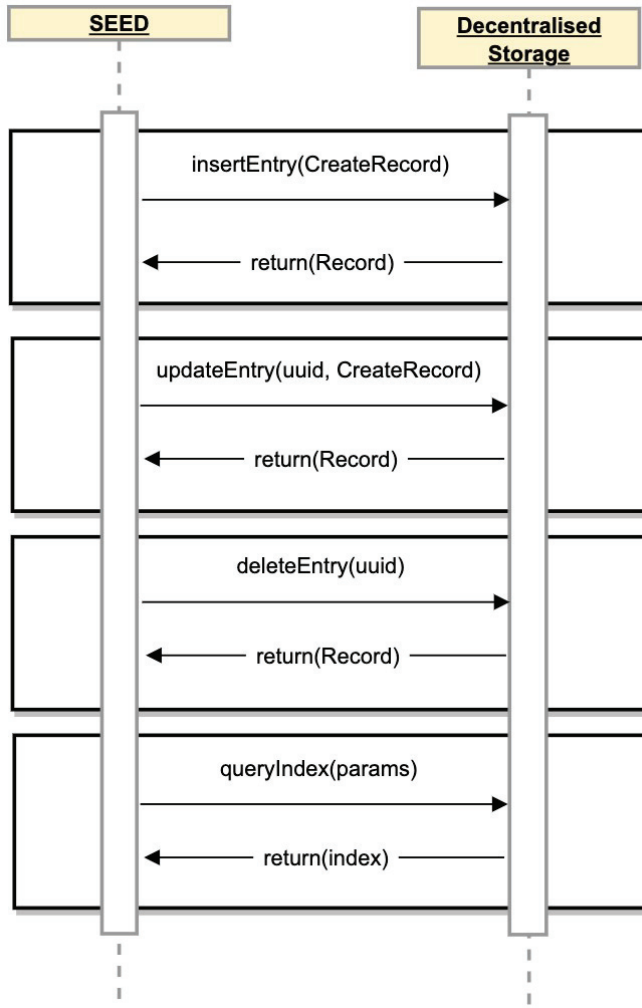

Figure 5.4 Federated query engine index management.

Verifiable data integrity:

The sequence diagram in Figure 5.5 demonstrates the integration of verifiable data integrity with the auditable accounting subsystem. All features are displayed on a single diagram, as there is no specific complexity within the functions.

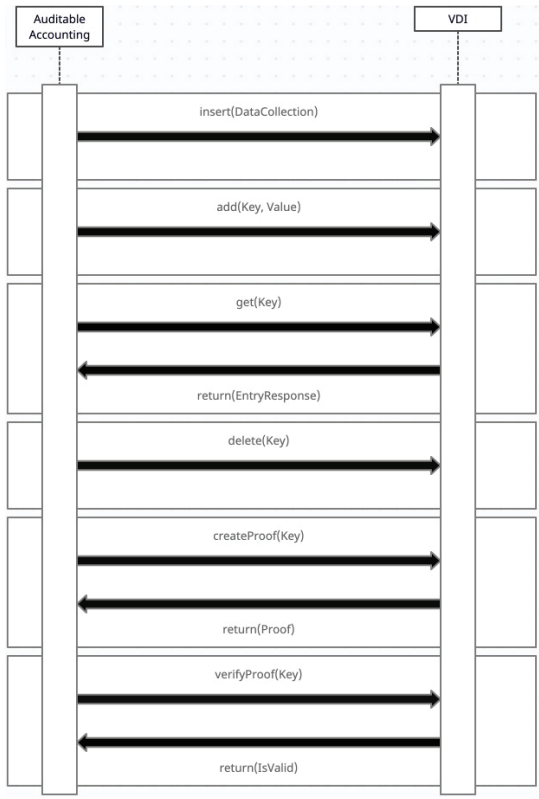

Figure 5.5 Verifiable data integrity.

5.4 Interfaces

The distributed storage subsystem does not expose a bespoke API for internal or external services. Each system within the Backplane uses the storage component's out-of-box means for connectivity.

Likewise, the API provided by the decentralized storage comes out-of-box with the solution. The service can be accessed via JSON-RPC protocol offered by Hyperledger BESU. Please refer to *BESU Documentation*[4] for the details of the API provided by Hyperledger BESU and client libraries.

The SEEDS Index solution consists of a Java library, called SeedsIndex, which provides wrappers for the smart contract and utility functions for

[4] Hyperledger BESU documentation, https://besu.hyperledger.org/en/stable/

convenience. The SeedsIndex library uses Web3j[5] library for accessing the BESU network. The library interface is documented by extensive Javadoc comments and a complete usage example included in the library.

5.5 Background Technologies

The decisions for the choice of selected technologies in order to satisfy the high-level capabilities are:

- Hyperledger BESU to satisfy decentralized storage.
- CockroachDB[6] to satisfy storage requirements.

Hyperledger BESU is an open-source Ethereum client. The decision to select Hyperledger BESU to satisfy the needs for decentralized storage has been made based on the following assumptions:

- Self-sovereign identity and access management has decided to base the reference implementation on Veramo, which specifically requires Ethereum-based blockchains.
- Auditable accounting and data monetization require smart contract whose functionalities are easily satisfied on Ethereum-based blockchains.

CockroachDB is a relational database management system. CockroachDB has been chosen as the storage solution due to previous experience of the technology by partners. Moreover, it is highly scalable, designed to deliver fast access and resilient to network outages. The only shortcoming of the chosen technology is the lack of features guaranteeing data integrity in case of the presence of malicious (e.g., because of honest mistakes or sophisticated external attacks) users. As all nodes of a CockroachDB cluster that is a part of an i3-MARKET instance are under the control of that instance, this shortcoming is not relevant in the i3-MARKET architecture. For client authentication in CockroachDB, mutual TLS authentication is used.

The semantic search engine index is implemented as a smart contract (written in Solidity programming language), which is deployed on BESU blockchain for distributed access. Updates to the index are authorized with digital signatures.

[5] https://www.web3labs.com/web3j-sdk

[6] CockroachDB, https://www.cockroachlabs.com/product/