



**Advanced Cyber-Threat Intelligence, Detection, and Mitigation
Platform for a Trusted Internet of Things
Grant Agreement: 786698**

D7.2 CYBER-TRUST distributed ledger architecture

**Work Package 7: Distributed ledger technology
for enhanced accountability
Document Dissemination Level**

P	Public	<input checked="" type="checkbox"/>
CO	Confidential, only for members of the Consortium (including the Commission Services)	<input type="checkbox"/>

Document Due Date: 30/04/2019
Document Submission Date: 28/05/2019



Co-funded by the Horizon 2020 Framework Programme of the European Union



Document Information

Deliverable number:	D7.2
Deliverable title:	Cyber-Trust distributed ledger architecture
Deliverable version:	1.0
Work Package number:	WP7
Work Package title:	Cyber-Trust Distributed Ledger Technology
Due Date of delivery:	30/04/2019
Actual date of delivery:	XX/05/2019
Dissemination level:	Public
Editor(s):	Pavué Clément (Scorechain) Griffiths Romain (Neofacto) Marchal Grégoire (Scorechain) Fo kossi Dagbegnikin (Neofacto)
Contributor(s):	Nikolas Kolokotronis (UOP) Konstantinos Limniotis (UOP) Sotirios Brotsis (UOP)
Reviewer(s):	Emanuele Bellini (Mathema) Evangelos Sfakianakis (OTE)
Project name:	Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things
Project Acronym	Cyber-Trust
Project starting date:	1/5/2018
Project duration:	36 months
Rights:	Cyber-Trust Consortium

Version History

Version	Date	Beneficiary	Description
0.1	12/04/2019	SCORECHAIN	1 st draft circulated
0.2	18/04/2019	SCORECHAIN	Input requested
0.3	22/04/2019	MATHEMA, OTE, KEMEA	Review deliverable and provide feedback
0.4	22/05/2019	UOP	Input in Section 1.5, 2.2
0.5	23/05/2019	SCORECHAIN	Final version for review
0.6	27/05/2019	OTE, MATHEMA	Review
0.7	28/05/2019	SCORECHAIN	Final version for quality assurance review
1.0	28/05/2019	KEMEA	Quality assurance check and submission

Acronyms

ACRONYM	EXPLANATION
ISP	Internet Service Provider
LEA	Law Enforcement Agency
DLT	Distributed Ledger Technology
CA	Certificate Authority
SDK	Software Development Toolkit
API	Application Program Interface
IoT	Internet of things
GKE	Google Kubernetes Engine
ReST	Representational State Transfer
CRUD	Create Read Update Delete
UC	Use case
TLS	Transport Layer Security
OSN	Ordering service node
HLF	Hyperledger Fabric
ORM	Object relational mapping

Table of Contents

Introduction.....	7
1. Technology Update.....	8
1.1 Why we choose Hyperledger Fabric.....	8
1.2 Current State of Hyperledger Fabric.....	8
1.3 Confidentiality model	8
1.3.1 Definitions.....	9
1.3.2 Transaction confidentiality	9
1.3.2.1 Public to the members	9
1.3.2.2 N-Peer Channel.....	9
1.3.2.3 Private Data	9
1.3.2.4 Off-Chain with Hash On-Chain.....	9
1.4 Why we choose 1.4.0?.....	10
1.5 Security model in Hyperledger	10
2. Application Architecture	12
2.1 Hyperledger Architecture	12
2.1.1 Hyperledger Fabric Components.....	12
2.1.1.1 Fabric Peer	12
2.1.1.2 Fabric Orderer	12
2.1.1.3 Fabric Chaincode	13
2.1.1.4 Fabric CA (Certificate Authority).....	13
2.1.2 Transaction flow	13
2.2 Components Decomposition and relation	14
3. Interfaces	16
3.1 Relation to other platform Components.....	16
3.2 Communication patterns.....	16
3.2.1.1 REST Call	16
3.2.2 Blockchain Bus	17
3.3 Use case Mapping (Table)	17
3.3.1 Ownership Management.....	17
3.3.2 Authority Management	18
3.3.3 Trusted File Storage.....	19
3.3.4 Mitigation Management.....	19
3.3.5 Forensic Evidence Storage.....	19
4. Data Architecture	21
4.1 Class Diagram	21
5. Technology stack	24
5.1 Tools/technologies used/ Components to develop	24

5.1.1	Hyperledger Fabric 1.4	24
5.1.2	Go for Chaincode	24
5.1.3	Loopback 4.....	24
5.1.4	Docker, Kubernetes & Helm	24
6.	Physical Architecture	26
6.1	Deployment View	26
6.2	Physical VM Requirements	26
6.3	Security measures	27
6.4	Deployment Process.....	27
6.5	Application Security domains	28
6.6	Infrastructure Security domains	28
7.	Conclusion.....	30
8.	References	31

Table of Figures

Figure 2.1: Hyperledger Fabric Native Architecture	12
Figure 2.2: DLT subcomponents	14
Figure 3.1: Communication with other components	16
Figure 4.1: Class Diagram of the data stored on chain.....	22
Figure 6.1:Deployment of DLT-Admin & DLT-Service.....	26
Figure 6.2:Infrastructure View.....	27
Figure 6.3: How the DLT will be deployed.....	28
Figure 6.4: Security for the DLT	28
Figure 6.5: Dockerization of the DLT	29

Introduction

The Cyber Trust DLT will have many purposes such as reliable source for evidence and file storage and authority management. Devices will be able to fetch the URL of the last patch to update themselves. Evidences metadata stored in the DLT won't be alterable ensuring the trust for officials to take decisions if needed. No one, will be able to modify the data on the DLT, making it harder to cover its tracking while perpetuating an attack on the network.

The DLT will also allow Cyber Trust partners to communicate together in a trusted way.

In this document, we will explain how we plan to implement the DLT to answer end user requirements previously established in D2.3. After a brief introduction, we will begin by explaining the Application Architecture. In this section, we will depict the components we are developing for Cyber Trust needs and how they are related to the Use cases previously written in D2.3. Then we will explain our Data Architecture. This section will show the data we plan to store in the DLT as well as the methods we will provide, so the other members of the consortium will be able to use this document to understand how to interact with the DLT we provide for the project. To continue helping the other members of the consortium, we provide in the next section, the protocol our DLT will use to interact with the other component built of Cyber Trust. Finally, we will explain in the last section how we plan to develop our components on Cyber Trust backend and our requirement in terms of hardware.

1. Technology Update

1.1 Why we choose Hyperledger Fabric

The technology choice is the conclusion of D7.1 based on the project needs. In terms of privacy, CYBER TRUST must use a private DLT. This is due to the confidentiality required by the type of data stored inside the DLT such as forensic evidence for instance. Moreover, the DLT needs to be highly scalable which is a requirement of the IoT ecosystem, this leads us to the use of a private DLT, which is more scalable because only the node creating the transaction needs to validate the new transaction via a smart contract call. The propagation of the transactions will be made by the central nodes of the DLT, most likely run inside of the Cyber Trust backend. Regarding this architecture, we decided to use Hyperledger Fabric¹ as it is the most advanced and most maintained framework of blockchain.

1.2 Current State of Hyperledger Fabric

Hyperledger is an active open-source project, with new features created frequently. To be sure to exploit the technology as much as possible, we went through the latest release, to understand the latest major improvements and their potential interest for the project.

Here is a brief summary extracted from the release page of Hyperledger's GitHub²:

v1.4.1 - April 11, 2019

- FAB-6135 - Raft Consensus: Allow using Raft and not relying on Zookeeper / Kafka for ordering

v1.4.0 - January 9, 2019

- FAB-5093 - **Private data reconciliation**: Allows peers for organizations that are added to private data collections to retrieve the private data for prior transactions to which they now are entitled. This feature is only supported on peers that have joined a channel since v1.4.
- FAB-11409 - **Private data client access control**: Ability to automatically enforce access control within chaincode based on the client organization collection membership without having to write specific chaincode logic. We will configure this feature by using the collection configuration property.

v1.3.0 - October 10, 2018

- FAB-10120 - Identity Mixer for anonymous transactions Keep client identities anonymous and unlinkable using zero-knowledge proofs.

v1.2.0 - July 3, 2018

- FAB-8718 - **Channel Private Data**: Keep chaincode data confidential among a subset of channel members.
- FAB-8727 - **Access control for peer functions**: Configure which client identities can interact with peer functions, per channel.

As we can see a lot of work have been made recently on data privacy and user confidentiality, with a new way to store data in a more confidential manner, the private data.

1.3 Confidentiality model

There are different capabilities of storing data with Hyperledger. In this section, we will explain each of them and explain their pros and cons.

¹ <https://www.hyperledger.org/projects/fabric>

² <https://github.com/hyperledger/fabric/releases>

1.3.1 Definitions

Channel: A channel is a private blockchain overlay which allows for data isolation and confidentiality. Channels are defined by a Configuration-Block, which contains configuration data that specifies members and policies.

Private Data Collections: Used to manage confidential data that two or more organizations on a single channel want to keep private from other organizations on that channel. The collection definition describes a subset of organizations on a channel entitled to store a set of private data, which by extension implies that only these organizations can transact with the private data.

Private Data: Confidential data that is stored in a private database on each authorized peer, logically separate from the channel ledger data. Access to this data is restricted to one or more organizations on a channel via a private data collection definition. Unauthorized organizations will have a hash of the private data on the channel ledger as evidence of the transaction data. Also, for further privacy, hashes of the private data go through the Ordering-Service, not the private data itself, so this keeps private data confidential from the Orderer.

1.3.2 Transaction confidentiality

Each of this model have a different confidentiality level that might discard them for this project. The objective of this section is to list the different confidentiality level and select the one that will be used for the project.

1.3.2.1 Public to the members

A channel open to all members having a peer can be created.

It can be used to broadcast information with the Fabric properties (integrity, traceability, etc).

Once set no data can't be deleted unless resetting all the systems.

1.3.2.2 N-Peer Channel

A channel open to select members only.

It can be used to make a reduced broadcast with the Fabric properties.

Once set no data can't be deleted unless resetting all the systems.

1.3.2.3 Private Data

Private data can be used to store data only on selected peers and choose a delete period when the data should not be available anymore.

This can be used for data needed in chaincode but that should not be visible for all peers and/or data covered by European legal framework.

1.3.2.4 Off-Chain with Hash On-Chain

Suitable for large data or data that are not needed/used? as chaincode input.

Integrity of the data is still insured by the on-chain hash.

Regarding the list above of confidentiality level we decided to choose to use the private data, as we most likely need to delete data stored inside of the DLT due to legal aspects. To reduce the amount of data stored on chain and thus improve the scalability of the DLT, we will also use Off-Chain data with hash On-Chain. The data that won't be stored on chain will be centralized. The reason to only store them Off-Chain is that they are not needed by other partners like LEA, but only by the partner that generate these data.

1.4 Why we choose 1.4.0?

In the oldest versions of Hyperledger Fabric Channel were originally thought to give confidentiality to the transactions. But there are several drawbacks:

- It is impossible to delete data once put in a channel
- We need to create a new channel for each SP and deploy a new chaincode on it
- It needs a lot of DevOps to manage channels

The major improvement of the version 1.4.0 is that it brings new solutions to those problems above. Private data enable:

- Private collections for each ISP that are not accessible to another ISP
- Definable data retention period
- Only Hash of data is present in the public Channel so every organization can see the state of the ledger but can't access the log or any other data stored in the private data without the level of accreditation necessary for that.

1.5 Security model in Hyperledger

Permissioned blockchains, are comprised of a set of identified and known participants and provide a way to secure interactions that take place among a group of entities. These entities may have a common goal but do not fully trust each other. As blockchains can execute programmable transaction logic, smart contracts may function as trusted distributed applications and obtain their security from the underlying consensus and blockchain protocol. This way resembles the approach creating applications with state-machine-replication (SMR)³.

Many smart-contract blockchains mechanisms follow the design of SMR and implement the active replication phase⁴. The consensus protocol first orders the transactions, propagates them to in the network and sequentially, each peer executes these transactions. This process is named as order-execute architecture and requires all the peers to run the transactions and each transaction has to be deterministic. This makes permissioned blockchains to suffer from several limitations, in particular [01]:

- The consensus protocol is hard-coded within the network and the validation of new transactions may not be adapted to smart contracts.
- Smart contracts have to be written in a domain-specific language (Go, Java and Node.js)⁵ and might lead to programming errors.
- The sequential execution of all the transactions by all peers in the network may result to limitations of the performance and the complex measures that are needed to thwart denial-of-service attacks originated from untrusted contracts or peers.
- It is difficult to ensure programmatically that all transactions have to be deterministic.
- Each smart contract runs on all the nodes in the system and by this way prohibits the broadcast of contract code to a subset of nodes.

Hyperledger Fabric [01] overcomes such limitations and introduces a new blockchain mechanism that has as primary goal the scalability, confidentiality, flexibility and resiliency of its system [02]. Designed as an extensible permissioned blockchain supports distributed applications written in programming languages and allows the execution of contracts across many peers. By this way Hyperledger Fabric is one of the firsts distributed operating systems designed for permissioned blockchains that follows an execute-order-validate process to execute untrusted code in an untrusted or even in an adversarial environment. The execute-order paradigm of Hyperledger Fabric separates the transaction flow [02],[03], (that might be executed by different nodes of the system) into the next steps. Firstly, in the execution phase, the execution of the transactions is taking place by checking its accuracy by a subset of the network, called endorsing peers or simply endorsers. These peers are specified by an endorsement

³ <https://infoscience.epfl.ch/record/256238?ln=en>

⁴ <https://medium.com/s/story/lets-take-a-crack-at-understanding-distributed-consensus-dad23d0dc95>

⁵ <https://hyperledger-fabric.readthedocs.io/en/release-1.4/fabric-sdks.html>

policy that vouches for the execution of given smart contract. Then, in the ordering phase, the ordering service nodes (OSN) order all the transactions including the signatures of the endorsing peers and finally, in the validation phase the transactions are validated to prevent race conditions events [01].

This hybrid design mixes passive and active replication to tolerate Byzantine faults, make Hyperledger Fabric a scalable ledger for permissioned distributed ledgers, support adaptable trust assumptions and resolve the issues mentioned above. To implement such an architecture Hyperledger Fabric is comprised of modular blocks for the following components of the system:

- The ordering service that disseminates state updates and organizes the order of the transactions.
- The Membership Service Provider who associates the peers of the network with cryptographic identities in order to maintain the permissioned nature of Hyperledger fabric.
- A peer-to-peer gossip protocol to broadcast the new blocks created by the OSN.
- Smart contracts, which are executed in an isolated environment, written in programming languages and have no access to the state of the ledger.
- Peers who maintain the blockchain and a snapshot of a current state.

The greatest new advantage of Hyperledger is that it will help us to provide the proof that every type of data stored inside the DLT was not altered or corrupted since they are stored. For example, it assures that the forensic evidence was not altered since it was collected to when it comes in a court of law. Hyperledger will help us to assure:

- **Integrity:** No entity has corrupted or altered the evidence during the transferring.
- **Authentication:** The authorized entities that interact with the evidence must provide proof of their identities. For example, only an authorized LEA officer can have access to a specific log he asks access for.
- **Verifiability:** Each entity that owns for a particular time the evidence must verify all the processes.
- **Traceability:** Each authorized entity must be able to trace the evidence, from the moment of its creation until the moment of its elimination.

2. Application Architecture

2.1 Hyperledger Architecture

2.1.1 Hyperledger Fabric Components

Hyperledger Fabric (HLF) goal is to provide a complete consortium Blockchain framework with minimal centralization and complete trust management.

HLF provides all the components needed to manage trust between 2 or more organizations and a central platform managing consensus.

The end users can be internal back-end of the company or external client signed by any participating and authorized Organization.

Organization can restrict the confidentiality of their interactions using a channel and the transaction validity are checked against business rules inside a chaincode (smart contracts).

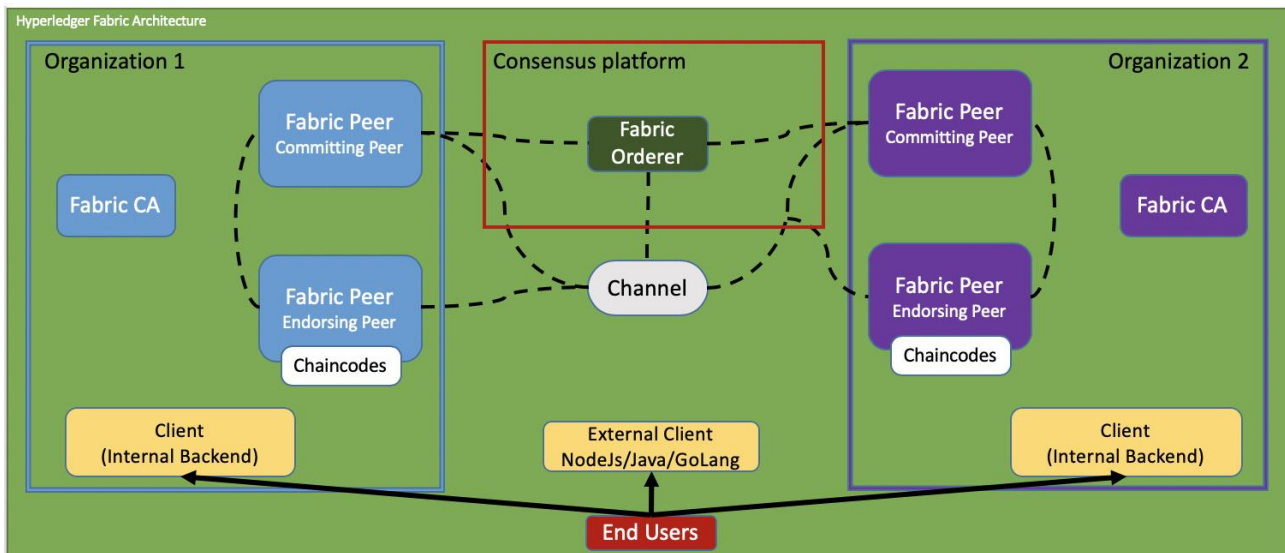


Figure 2.1: Hyperledger Fabric Native Architecture

2.1.1.1 Fabric Peer

The peers are a set of nodes who form the blockchain network. Due to the permissioned nature of Fabric all the nodes that participate in the protocol have an identity given by the Membership Service Provider (MSP). The peers maintain the state of the network and a copy of the ledger; execute transaction proposals and validate new transactions in a form of a hash chain.

We have two different types of peers: endorsing and committing peers. All the peers will commit blocks to the distributed ledger.

- **Endorsers:** simulate and endorse transactions as stated by the policy of the chaincode according to which the transactions pertain.
- **Committers** verify endorsements and validate transaction results, prior to committing transactions to the blockchain.

2.1.1.2 Fabric Orderer

As a permissioned blockchain, Hyperledger needs a concept of a whitelisted peer to validate a new block into the blockchain. The Ordering Service Nodes (OSN) is a subset of the network that forms the ordering service

that basically establishes in Fabric the total order of all transactions. Each transaction encloses dependencies and state updates computed during the execution phase and cryptographic signatures of the endorsers. The orderers do not participate in the execution and the validation phase.

2.1.1.3 *Fabric Chaincode*

As a reminder, smart contracts are computer programs that contain logic to execute transactions and modify the state of the assets stored within the ledger (world state). In Hyperledger Fabric the smart contracts are called chaincode, which are computer code that runs during the execution phase and implements the business logic for Hyperledger Fabric. The chaincode is a key part of a distributed application in Hyperledger Fabric that directs how to manipulate assets within the network and can support plugins to add new programming languages such as Go, Java and Node.js.

2.1.1.4 *Fabric CA (Certificate Authority)*

His role is to manage the certificates of the different infrastructures. The certificates will rely on the Public Key Infrastructure where the Certificate Authority would sign the certificate meaning that a particular public key is of a particular user. Then the user can use his signed private key to transact.

2.1.2 Transaction flow

Two organizations want to interact securely in a blockchain application based on Hyperledger Fabric.

Whenever an end-user needs to securely transact in the blockchain network, the client application interacts with the blockchain using the Hyperledger Fabric SDK.

Each Fabric Client submit Cryptographic Certificates to their organization **Fabric CA** (Certificate Authority). The Certificates can then be used throughout the platform.

The **client** proposes the transaction to **endorsing peers** who in turn capture the read-write sets and validate with the smart contract available. This endorsement is called execution.

Once the execution is done at the **endorsing peers** the response would be sent back to the **client** with endorsement signatures.

The further process of the transaction depends on the endorsement policy which is decided during the initial setup of the network. The transaction would then be forwarded to the **orderer**.

The **orderer** would order all transactions received in a time frame in a block and then **deliver** this block to all the **peers** in the network.

Finally, the transactions sent from the ordering service are validated at each **peer** for endorsements and consistency of each transaction. Once the validation is done each of the **peers** emit events.

The ordering service in the Hyperledger fabric is also pluggable and the organizations can also use their own ordering service they desire. Some of the ordering services used in the fabric are **SOLO** or **Kafka**⁶ for High Availability.

⁶ <https://kafka.apache.org/>

2.2 Components Decomposition and relation

Inbound communication to the blockchain will pass exclusively through the ReST server. All Blockchain Events will be forwarded to the ActiveMQ⁷ message bus, other components would then access the DLT through ReST if message data is not enough.

The component to develop will be the ReST server and the chaincodes. The other components (Peer, CA, Orderer) have to be configured and need important devOps work to be industrially integrated.

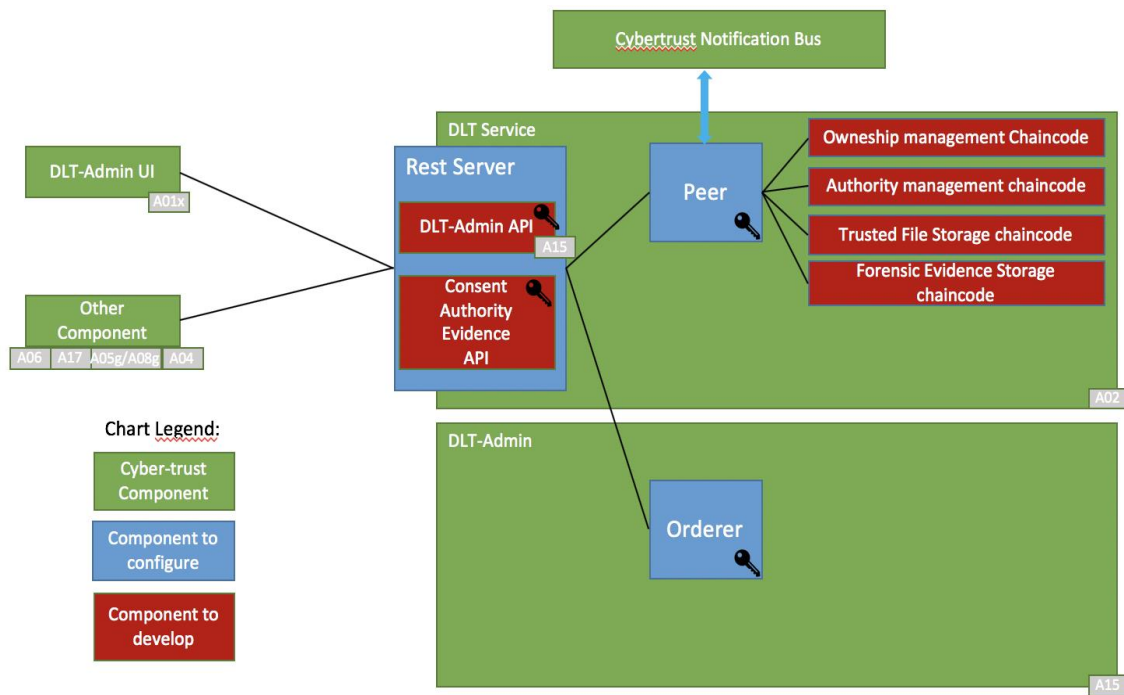


Figure 2.2: DLT subcomponents

DLT Admin: it endorses the Orderer system role for the transactions on the DLT. It is the central part of the system and it will be running on Generic Cyber-Trust platform only.

DLT-Admin UI: This is the user interface of the DLT manager (Admin)

DLT Service: It regroups and depicts all the functionalities offered by our DLT, manages the certificate authority, peers and their chain code for our DLT platform. It will be running inside of every platform (Generic platform, ISP, LEA)

REST Server: it is our architectural style that provides requests functionalities for mutation, creation and deletion interactions between other components and our DLT. That also permits to the DLT-Admin to access the DLT platform through his UI portal.

Ownership management Chaincode: it will define how we manage, devices and their class, organizations and their worker's registration/ deregistration on the platform. That also manages the data sharing level chosen by the device owner and the risk attack level that the device is exposed to.

⁷ <https://activemq.apache.org/>

Authority management Chaincode: it depicts how the platform manages the patch database available for every device.

Trusted File Storage Chaincode: it will manage how the logs will be stored on the DLT, define who has the right access to explore or/and export the logs. It also manages the internal process of evidence block validation and the setup of the devices.

Forensic Evidence Storage Chaincode: it will manage how the evidence will be stored on the DLT, define who has the right access to visualize or/and export the evidence.

3. Interfaces

3.1 Relation to other platform Components

We regroup end user requirements with the same purpose together. These are the blue boxes on the chart below. Then we associate these UCs with components developed by other members of the Cyber Trust consortium. These components will communicate together via Rest calls with the DLT to push or retrieve data from the DLT.

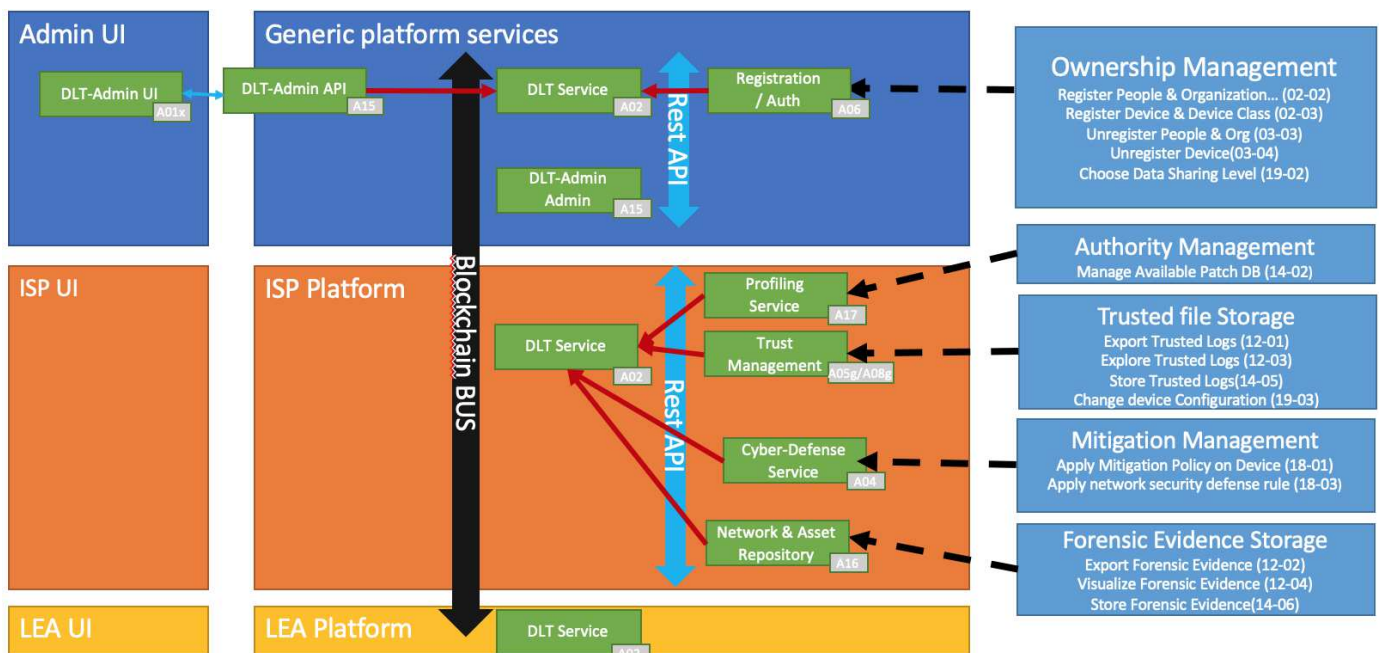
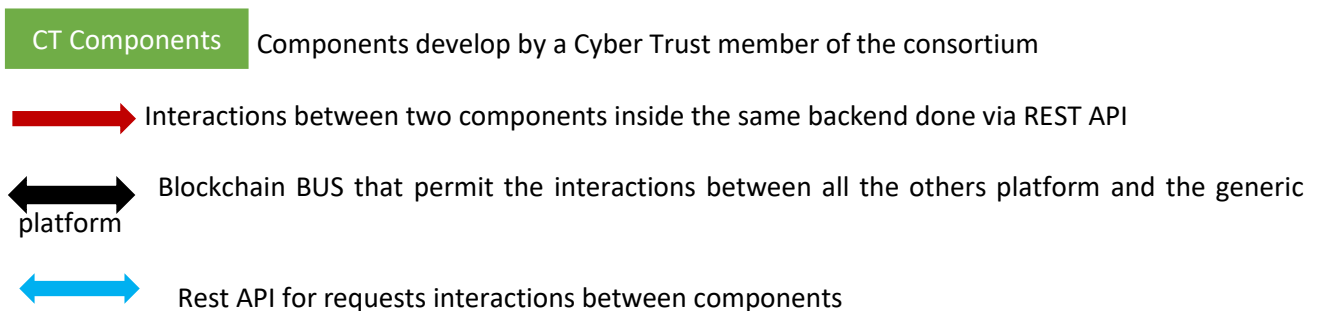


Figure 3.1: Communication with other components

Legends:



One of the advantages of this architecture is that the end user won't interact directly with the DLT. It will ease the development and the future changes that will be made in the interfacing of the DLT. It will be easier for end user to interact with the platform by abstracting this communication.

3.2 Communication patterns

3.2.1.1 REST Call

As explained in the above figure the components running on the same backend as one of our [A02] DLT Service will be able to interact with it via REST API calls. These components won't be able to interact with any other nodes of the DLT expect this one. Here is an example of interaction with the DLT via the REST protocol.

UC-19-02 Choose Data Sharing Level

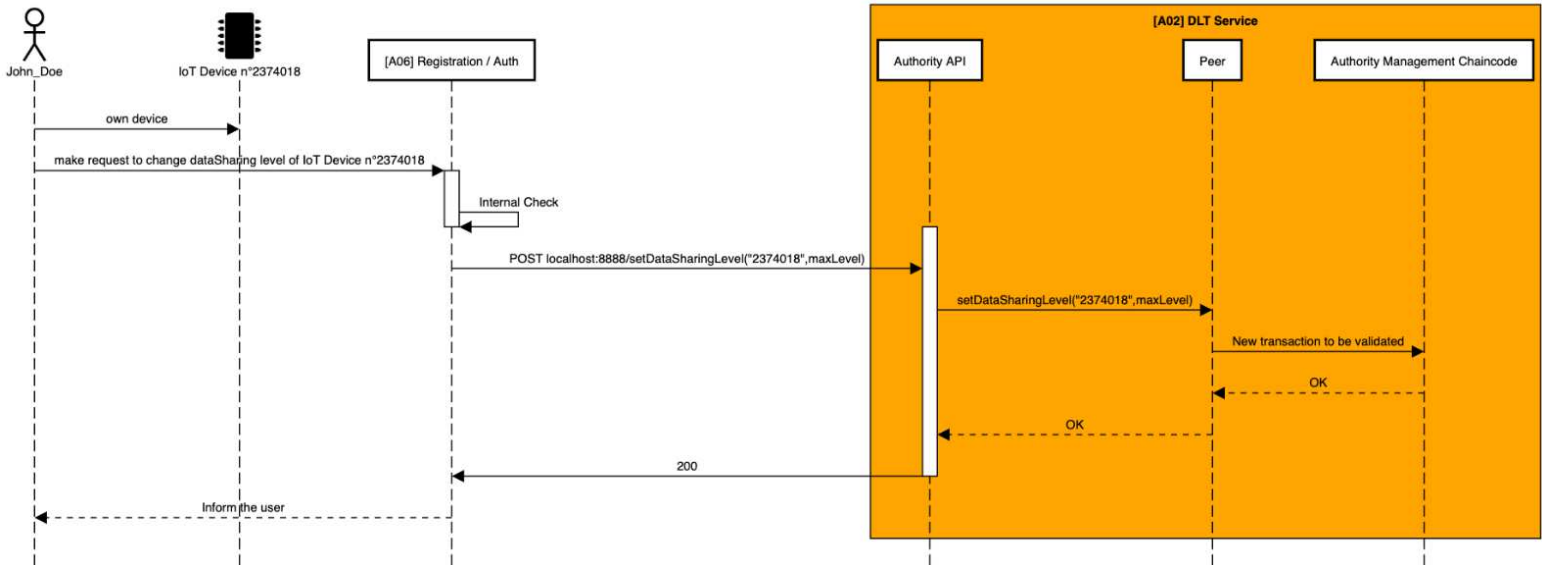


Figure 3: Sequence Diagram of UC 19-02

In this example, a user wants to change the data sharing level of a device he owns. He makes the request via the UI. A06 makes internal checks. Then make a ReST request to the DLT service [A02] via the Authority API. After validation of the new transactions the API returns 200 to the call maker.

3.2.2 Blockchain Bus

When designing the communication patterns of the DLT, we started from the postulate that different Cyber Trust partners won't trust each other. For instance, we can't expect two ISPs from the same country to trust each other while communicating sensitive information. So, we decided to provide them a way to communicate in a trusted way, the DLT bus. The nodes of the DLT that will exchange information published by the different partners, will only be able to interact together via the blockchain bus. The information will only circulate if the two nodes have sufficient permission to interact together. This will harden the data privacy and traceability.

3.3 Use case Mapping (Table)

Here we will answer to the specification of the UC. SDK / API to explain how the others components will interact with the DLT. Chaincode method will be depicted here with the event they will potentially generate.

For each use cases, we listed in figure 3.1, we create an API endpoint following the implementation of the 4 basic functions of CRUD.

3.3.1 Ownership Management

DLT will be use to handle ownership management. This include storing who owns which device, produced by which company as well as its internet provider. From D2.3 we found the UC related to Ownership management.

UC ID	UC Name	Initiating Actor	API Endpoint
02-02	Register Administrators & organizations into the Cyber-Trust Platform	[O3] LEA	/Organization /Administrators
02-03	Register Device & Device Class into the Cyber-Trust Platform	[O3] Police Officer	/Device /DeviceClass
03-03	Unregister administrators & Organization	[O3] Police Officer	/Organization /Administrators
03-04	Unregister device & Device class	[O3] Police Officer	/Device /DeviceClass
19-02	Choose Data Sharing Level	[P1, P2] user	/DeviceRights

3.3.2 Authority Management

Authority managements is here to ensure that devices receive an authentic and unmodified copy of a vendor's software when updating.

UC ID	UC Name	Initiating Actor	API Endpoint
14-02	Manage Available Patch DB	System, [O6] Smart Device Manufacturer	/DeviceClass /Patch
19-03	Change device Configuration	[O3] LEA	/Configuration

3.3.3 Trusted File Storage

Trusted file storage is about keeping tracks of activity of devices activity on the network and restituting it if the call maker has the sufficient permission.

UC ID	UC Name	Initiating Actor	API Endpoint
12-01	Export Trusted Logs	[O3] LEA	/DeviceLogs
12-03	Explore Trusted Logs	[O3] LEA	/DeviceLogs
14-05	Store Trusted Logs	System	/DeviceLogs

3.3.4 Mitigation Management

Mitigation management purpose is to ensure that the person that want to operate a change of a device he does not own has the sufficient permission on it.

UC ID	UC Name	Initiating Actor	API Endpoint
18-01	Apply Mitigation Policy on Device	System, [P2] A Smart Device Owner	/Admin
18-03	Apply Network Security Defense rule	System, Cyber-Defense Service [A04]	/Admin

3.3.5 Forensic Evidence Storage

Forensic evidence storage is here to keep tracks of forensic evidence by storing metadata related to the attack and retribute it if the call maker has the sufficient permission.

UC ID	UC Name	Initiating Actor	API Endpoint
12-02	Export Forensic evidence	[O3] Police Officer	/Evidence
12-04	Visualize forensic evidence	[O3] Police Officer	/Evidence
14-06	Store forensic evidence	A07 Forensic eVDB	/Evidence

4. Data Architecture

4.1 Class Diagram

To match the end user requirements as defined in D2.3 and the initial architecture of D4.1 the following data will be stored on-chain.

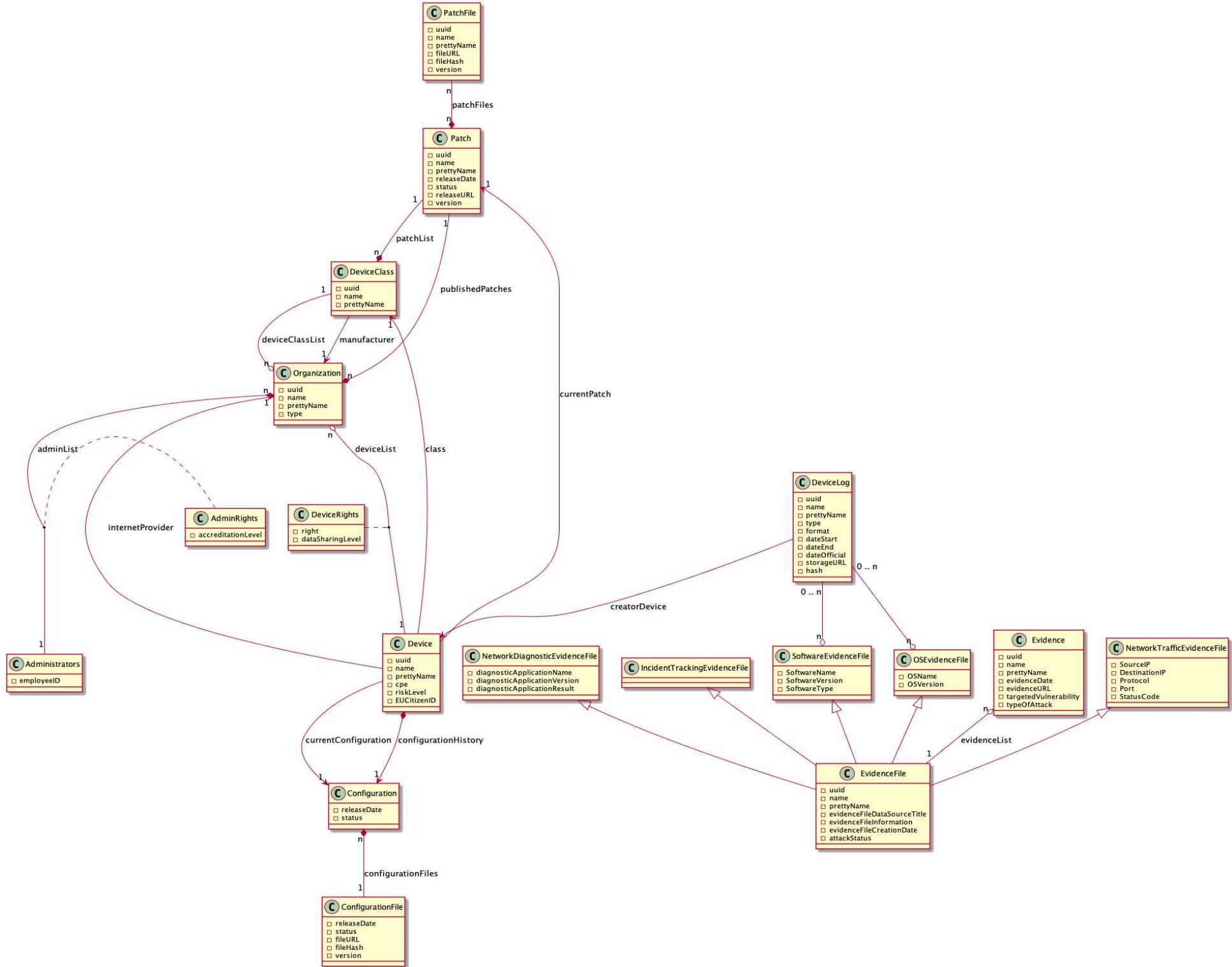


Figure 4.1: Class Diagram of the data stored on chain

Organization: Metadata of an organization register in Cyber Trust

- Administrators: Information of an employee of an organization
- AdminRights: The accreditation level of an Administrators. Will be used for the mitigation management.
- Device: Metadata of a device owned by a EU citizen after he/she register it on the platform.
- DeviceClass: Metadata of a type of device register its manufacturer.
- DeviceRights: The dataSharing level of a device given by its owner;
- DeviceLog: Metadata of a log file produced by a device with the sufficient permission to do it.
- Patch: Metadata of a patch register on the platform.
- PatchFile: A file belonging to a patch.
- Configuration: Metadata of a configuration register on the platform.
- ConfigurationFile: A file belonging to a configuration.
- Evidence: Metadata of an evidence.
- EvidenceFile: The interface of a file belonging to an evidence.
- NetworkDiagnosticEvidenceFile: Metadata produced by a traffic analysis software.
- NetworktrackingEvidenceFile: Metadata of the action of the device on the network.
- IncidentTrackingEvidenceFile: Metadata of a file produced by incident management tracking software.
- SoftwareEvidenceFile: Metadata of a file produced by a software on a device.
- OSEvidenceFile: Metadata of the operating system of a device.

5. Technology stack

5.1 Tools/technologies used/ Components to develop

5.1.1 Hyperledger Fabric 1.4

Hyperledger Fabric is the leading consortium Blockchain platform.

The version 1.4 is stable and usable but it is still under active development delivering new needed features on confidentiality or consensus. Please refer to D7.1 for further explanations.

5.1.2 Go for Chaincode

Chaincodes can be developed in JavaScript, Java and Go, and also with a proprietary framework named Composer (JavaScript Framework).

Composer is reaching its end-of-life because HLF developers wants to re-integrate part of its functionalities into Fabric Code. But today the future of this framework is still uncertain.

We decided to code the chaincode in Go to use the same language as the main development of Fabric. We think that the new features will be first integrated in the Go SDK and will be of better quality.

5.1.3 Loopback 4⁸

The HLF JavaScript SDK is the most complete and well tested, so we only considered JavaScript framework.

Express has a lot of middleware and contributions, but it does not enforce any coding practice or convention. We really wanted to have dependency injection to enable easy testing of components.

Koa⁹, Strapi ¹⁰Hapi¹¹, Sails and Nest provides nice frameworks but each with its own problem: small community, too much unneeded features, lots of boilerplate, focus on microservice or lack of ORM if we need it.

Our choice, Loopback 4 is a typescript framework aimed to replace express.

It features a dependency injection framework, a configurable request processing pipeline and ReST API easy configuration and documentation support through Swagger, simple ORM and model. It is also maintained by IBM that has a huge implication in HLF.

5.1.4 Docker¹², Kubernetes ¹³& Helm¹⁴

Manual following a document to install a software is not recommended for Hyperledger Fabric as the number of components and interactions are very high.

Deployment and automation tools

We use Docker to create the software Images, Kubernetes to provide infrastructure and deployment instructions and Helm for Kubernetes configuration packaging and to configure different environments.

⁸ <https://loopback.io/doc/en/lb4/>

⁹ <https://koajs.com/>

¹⁰ <https://strapi.io/>

¹¹ <https://hapijs.com/>

¹² <https://www.docker.com/>

¹³ <https://kubernetes.io/>

¹⁴ <https://helm.sh/>

This software suite enables single-command deployments of containers and has no real competitor on the market.

We could have used vagrant, packer, and bare docker, but we feel that Kubernetes gives a complete and coherent solution.

6. Physical Architecture

6.1 Deployment View

To deploy our components, we plan to dockerize them and use Kubernetes for automatic deployment. This will ease the deployment of the backend for new Cyber Trust partners. The container has 3 open ports for a partner's peer and 4 for a central peer of the DLT. The figure depicts the deployment Architecture of the DLT.

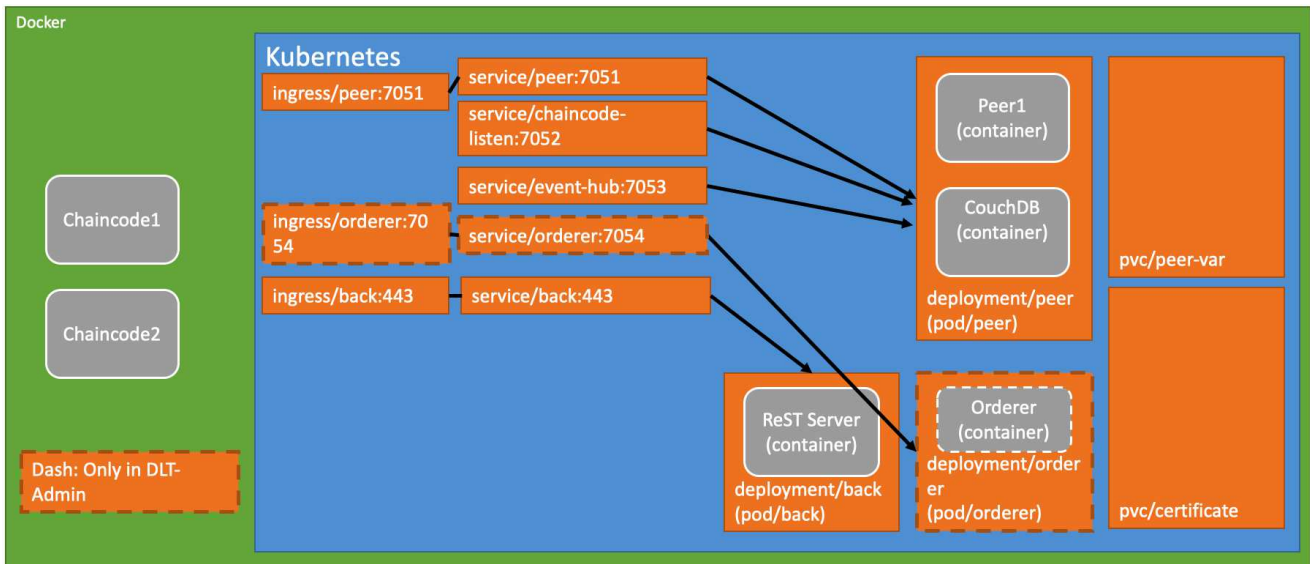


Figure 6.1:Deployment of DLT-Admin & DLT-Service

The chaincodes started by the peers on the nodes' Docker engine.

Ingress/*: enable access external internet access to services

Service/*: expose pod ports inside Kubernetes network

Deployment/*: Control start/restart/scale of Pod

Pod/*: Group containers in a single logical host

Pvc/*: Persistent Volume Claim, Provision durable disk storage

6.2 Physical VM Requirements

To run the service, we take as basis a GCloud template ¹⁵with, for each platform:

- 1 Google Kubernetes Engine Master (auto-provisioned by GKE)
- 2 Google Kubernetes Engine Workers (1vCPU 3.75Gb RAM N1-Standard-1)
- 2 persistent volumes of 1GB each

The whole service can also be deployed on a Personal Computer with Docker/Kubernetes and at least 8Go and a Core I5 2,3Ghz.

¹⁵ <https://cloud.google.com/compute/docs/instance-templates/>

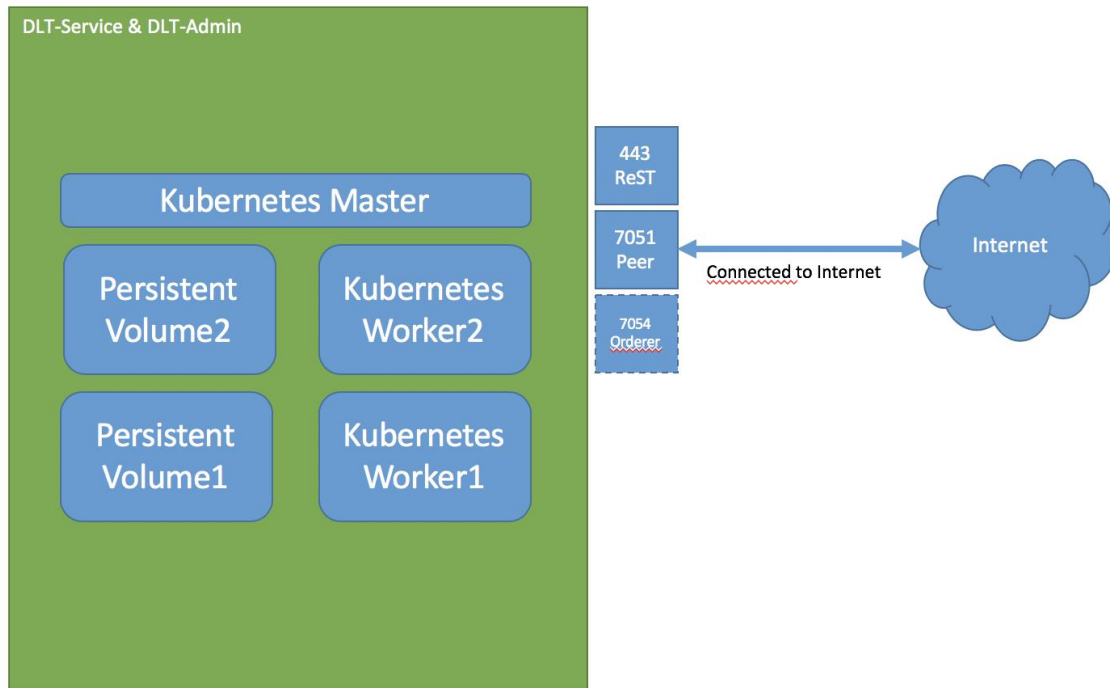


Figure 6.2:Infrastructure View

We choose not to include a Kafka queue to lighten the architecture but it can be added later on, if during the test the DLT is not scalable enough.

The general public platform will run one node of the DLT, same for partners of Cyber-Trust (LEA, ISP, ...).

6.3 Security measures

Direct access to the Kubernetes nodes will be forbidden.

Only the 443 (ReST), 7050 and 7054 ports will be open to Internet. Ports 7051 and 7054 will be protected by TLS authentication at the application level. Port 443 will be protected by TLS authentication with the certmanager Kubernetes tool

6.4 Deployment Process

To deploy a new instance of the DLT on a backend, the code will be pull from the GitLab, build, and store into the GitLab registry. Then the Kubernetes Worker will be able to pull image and start it. This process will allow auto deploy as well when needed. The figure below depicts the flow of information between the different actors of the deployment process.

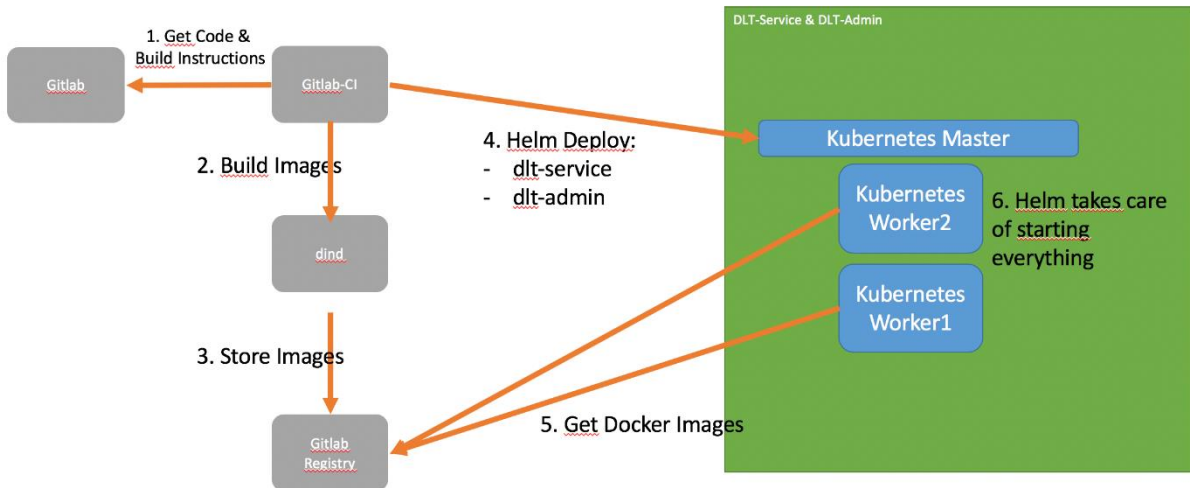


Figure 6.3: How the DLT will be deployed

6.5 Application Security domains

To ensure the security of each component, their domains should be segregated otherwise the system security cannot be ensured. The figure below depicts the separation between the different components.

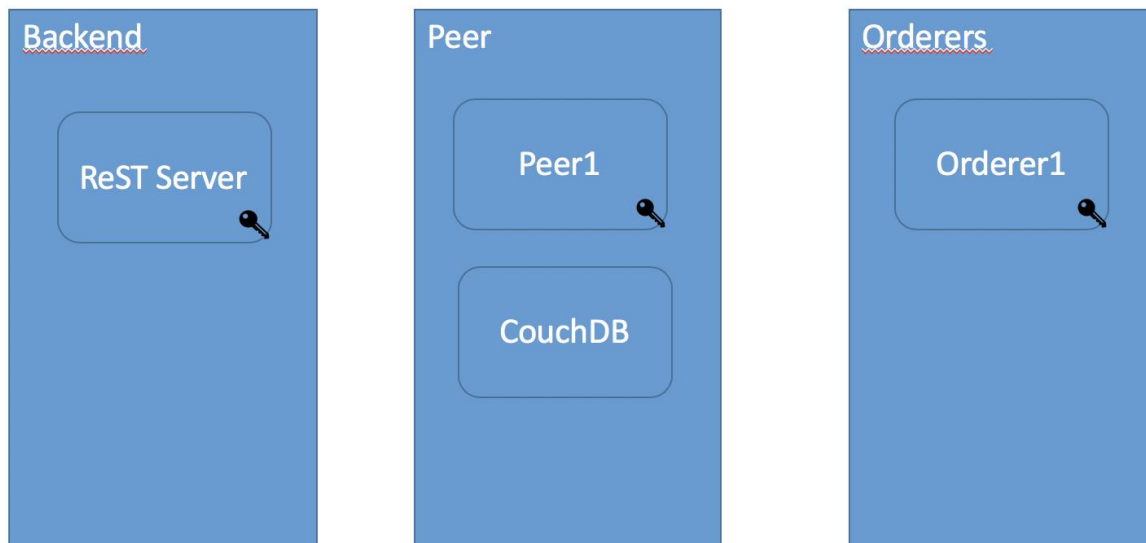


Figure 6.4: Security for the DLT

The keys in the figure is here to represent that the communication is encrypted between components. The CouchDB is only accessible via request to the Peer1, no direct communication is allowed as the communication won't be encrypted.

6.6 Infrastructure Security domains

To guaranty the infrastructure security, we will encapsulate components. Each rectangle has critical control under inner rectangle, as well as the Kubernetes Master has critical control on the workers. Each rectangle has no control on its parent or grandparents. The picture below depicts its architecture.

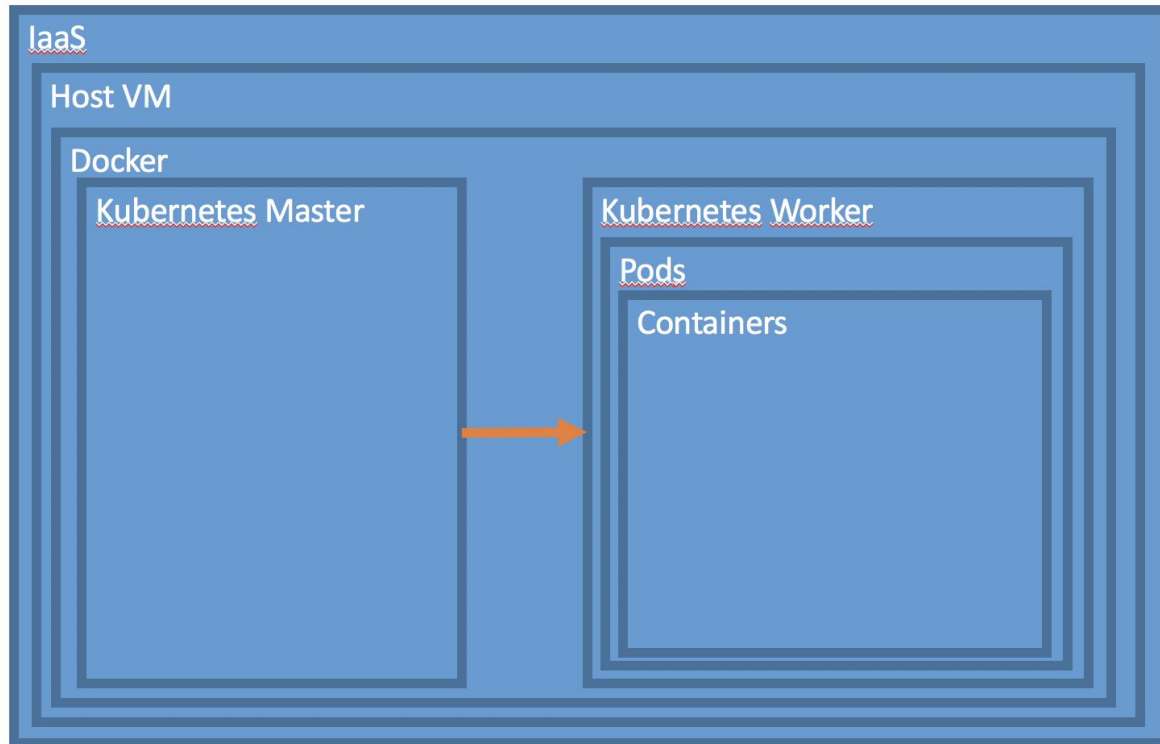


Figure 6.5: Dockerization of the DLT

7. Conclusion

In this document, we refined our initial choice of technology and transformed it into an architecture with regards of the use case requirements and technical specificities of Cyber-Trust. We took care of the data that will be stored into the DLT in term of usage for the end user and end user privacy as well. We also designed and depicted our API and communications pattern to ease the work of the other members of the Cyber-Trust consortium.

8. References

- [01] Androulaki, Elli, et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains." *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018.
- [02] Sousa, Joao, Alysson Bessani, and Marko Vukolic. "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform." *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2018.
- [03] Thakkar, Parth, Senthil Nathan, and Balaji Viswanathan. "Performance benchmarking and optimizing Hyperledger fabric blockchain platform." *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018.