Technical design

Windesheim

Spark! Living Lab Conditioned Goods

Version 0.8



**Authors:**

|  |  |  |
| --- | --- | --- |
| Name | Student number | Email |
| Tim Imming | S1132262 | S1132262@student.windesheim.nl |
| Thijs Mansveld | S1127391 | S1127391@student.windesheim.nl |
| Sjoerd van de Kerkhof | S1129492 | S1129492@student.windesheim.nl |
| Gerard Wesseling | S1131366 | S1131366@student.windesheim.nl |
| Berat Guzel | S1127994 | S1127994@student.windesheim.nl |

**Version control**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | What |
| V0.1 | 08/04/21 | Tim Imming, Gerard Wesseling, Berat Guzel | Introduction and scope first version, chaincode |
| V0.2 | 9/04/21 | Thijs Mansveld, Tim Imming, Sjoerd van de Kerkhof | Overview Services, Infrastructureservices |
| V0.2 | 13/04/21 | Thijs Mansveld, Tim Imming, Sjoerd van de Kerkhof | Layer 2 & 3 drawings, naming convention |
| V0.2 | 15/04/21 | Tim Imming, Sjoerd van de Kerkhof, Berat Guzel | Layer 2 & 3 drawings |
| V0.2 | 16/04/21 | Tim Imming, Thijs Mansveld, Sjoerd van de Kerkhof, Gerard Wesseling | Security matrix, layer 2 and 3 drawings, chaincode |
| V0.4 | 21/04/21 | Tim Imming, Thijs Mansveld, Sjoerd van de Kerkhof | Introduction and scope corrected, layer 2 & 3 drawing captions, Infrastructure services |
| V0.4 | 23/04/21 | Gerard Wesseling, Thijs Mansveld, Sjoerd van de Kerkhof, Tim Imming | API, Infrastructure services |
| V0.4 | 27/04/21 | Thijs Mansveld, Sjoerd van de Kerkhof, Tim Imming | Infrastructure services |
| V0.4 | 28/04/21 | Thijs Mansveld, Sjoerd van de Kerkhof, Tim Imming | Infrastructure services |
| V0.4 | 29/04/21 | Thijs Mansveld, Sjoerd van de Kerkhof, Tim Imming | Infrastructure services |
| V0.4 | 7/05/21 | Sjoerd van de Kerkhof | API Validation |
| V0.6 | 21/05/21 | Tim Imming, Thijs Mansveld, Sjoerd van de Kerkhof | Infrastructure services |
| V0.6 | 25/05/21 | Tim Imming, Sjoerd van de Kerkhof | Infrastructure services |
| V0.6 | 26/05/21 | Tim Imming, Sjoerd van de Kerkhof | Infrastructure services, buffers |
| V0.6 | 27/05/21 | Tim Imming, Sjoerd van de Kerkhof, Gerard Wesseling | Infrastructure services, indexes, UML-diagrams, charts |
| V0.8 | 28/05/21 | Tim Imming, Sjoerd van de Kerkhof, Gerard Wesseling, Berat Guzel | Infrastructure services, pagination, API, Dashboard |
| V0.8 | 1/6/21 | Tim Imming, Berat Guzel | Final infrastructure services completed/final touches |

**Distribution**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | When | Addressee | Reason |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Approval**

|  |  |  |
| --- | --- | --- |
| Signature | Date | Name |
|  | 4/06/2021 | Maxime Bouillon |

Index

[1 Introduction 4](#_Toc73438572)

[2 Scope 5](#_Toc73438573)

[3 Overview Services 6](#_Toc73438574)

[3.1 Layer 3 drawing 7](#_Toc73438575)

[3.2 Layer 2 drawing 8](#_Toc73438576)

[3.3 Subnet summary 9](#_Toc73438577)

[3.4 Naming conventions 10](#_Toc73438578)

[3.5 Securitymatrix 12](#_Toc73438579)

[4 Infrastructure services 13](#_Toc73438580)

[4.1 Technical components and specifications for Datazone Protection.Firewall 13](#_Toc73438581)

[4.2 Technical components and specifications for Data Transport.Router 15](#_Toc73438582)

[4.3 Technical components and specifications for Identity and Permissions Management.Hyperledger 19](#_Toc73438583)

[4.4 Technical components and specifications for Relational Data Management.Datastore 24](#_Toc73438584)

[4.5 Technical components and specifications for Application Hosting.Webinterface 30](#_Toc73438585)

[*4.6* Technical components and specifications for *Message* Handling*.Orderer* 35](#_Toc73438586)

[4.7 Technical components and specifications for Facilities Deployment.Docker 41](#_Toc73438587)

[5 Application services 46](#_Toc73438588)

[5.1 Chaincode 46](#_Toc73438589)

[5.2 API 49](#_Toc73438590)

[5.3 Dashboard 52](#_Toc73438591)

[6 References 55](#_Toc73438592)

# Introduction

Together with Spark! Living Lab, the research and development team are working on a solution to make track and trace in a supply chain more transparent and integer. This project focusses on a conditioned goods supply chain, which means the transported goods must be cooled according to predetermined guidelines. These predetermined guidelines will be recorded in a Service Level Agreement (SLA). To ensure these shipments meet the Service Level Agreements data gathering and sharing is needed, which can be achieved by using blockchain. This technical design will focus on the blockchain framework Hyperledger Fabric.

In this technical design document, we will specify how requirements within the project scope of the Conditioned goods use case will be realized from a technical perspective. We will go in dept about the required infrastructure, naming convention and security matrix. Furthermore, we will describe the technical aspects of the chaincode and other used techniques and software.

# Scope

The scope for this technical design will be limited to the design of the infrastructure and blockchain within the Conditioned goods use case. This includes:

* Overview of services
* Infrastructure services
* Application services

Based on the technical design a proof of concept (PoC) will be realized including all the requirements labeled as must have in the requirement traceability matrix (RTM) [1] using the MoSCoW method [2]. The layer 2 and 3 drawings and services overview are for the full production network whereas the infrastructure services and application services are only described for the PoC. These services are designed to be able to be implemented in the production network.

# Overview Services

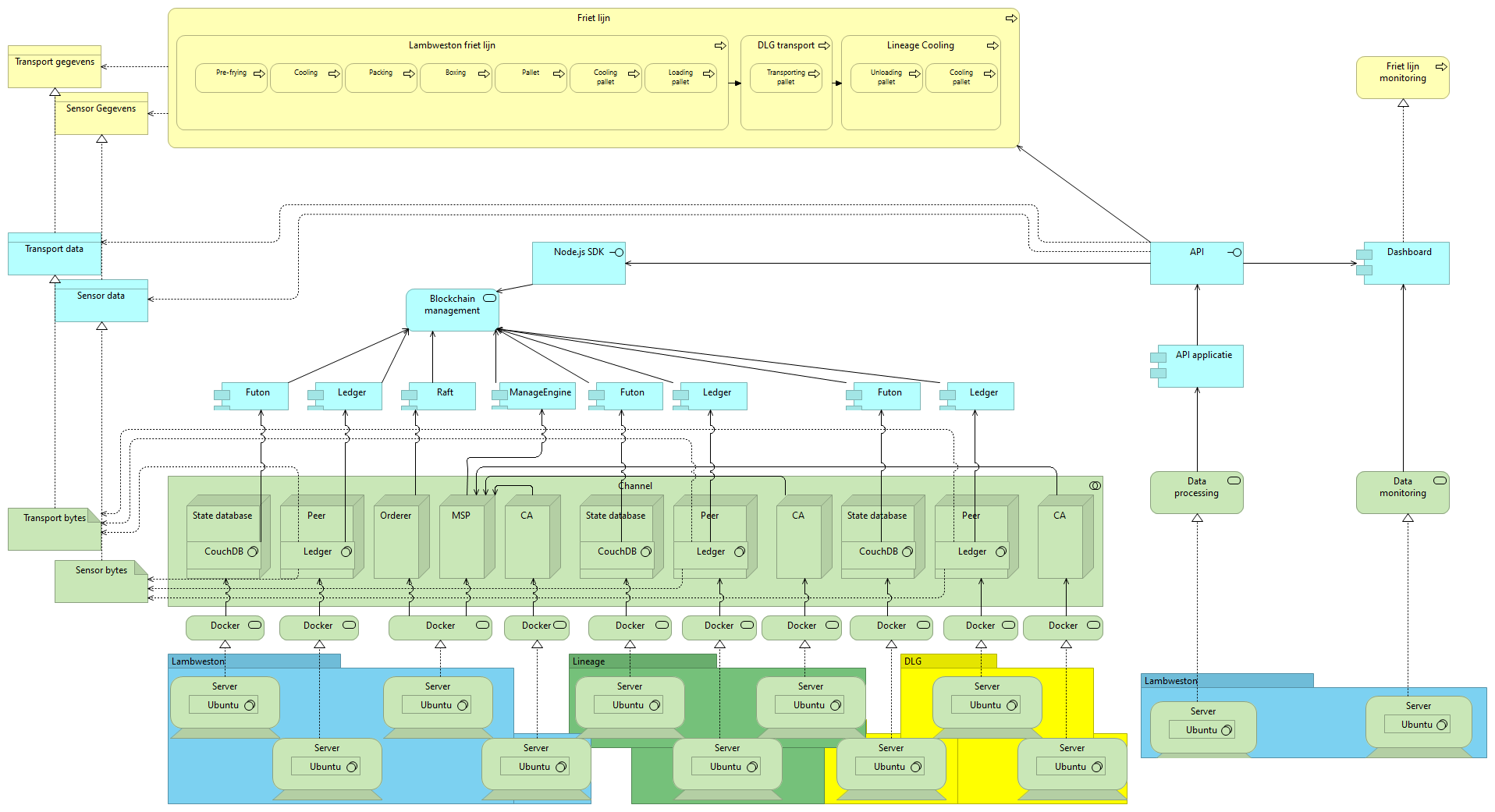


Figure 3‑1: Archi drawing infrastructure

The Archi picture above is a visualization of the Hyperledger Fabric network. The visualization has three layers, business layer (Yellow), application layer (Blue) and the technical layer (Green). The technical layer shows all hardware and the software that runs on this hardware. The servers realize the docker virtualization service and on this service docker containers run. These docker containers make use of applications shown in the blue layer and with these applications the business layer executes the process. The visualization of the Hyperledger Fabric as shown above is a full design of a production network and thus will be reduces for a proof of concept to a few servers running the entire proof of concept.

## Layer 3 drawing

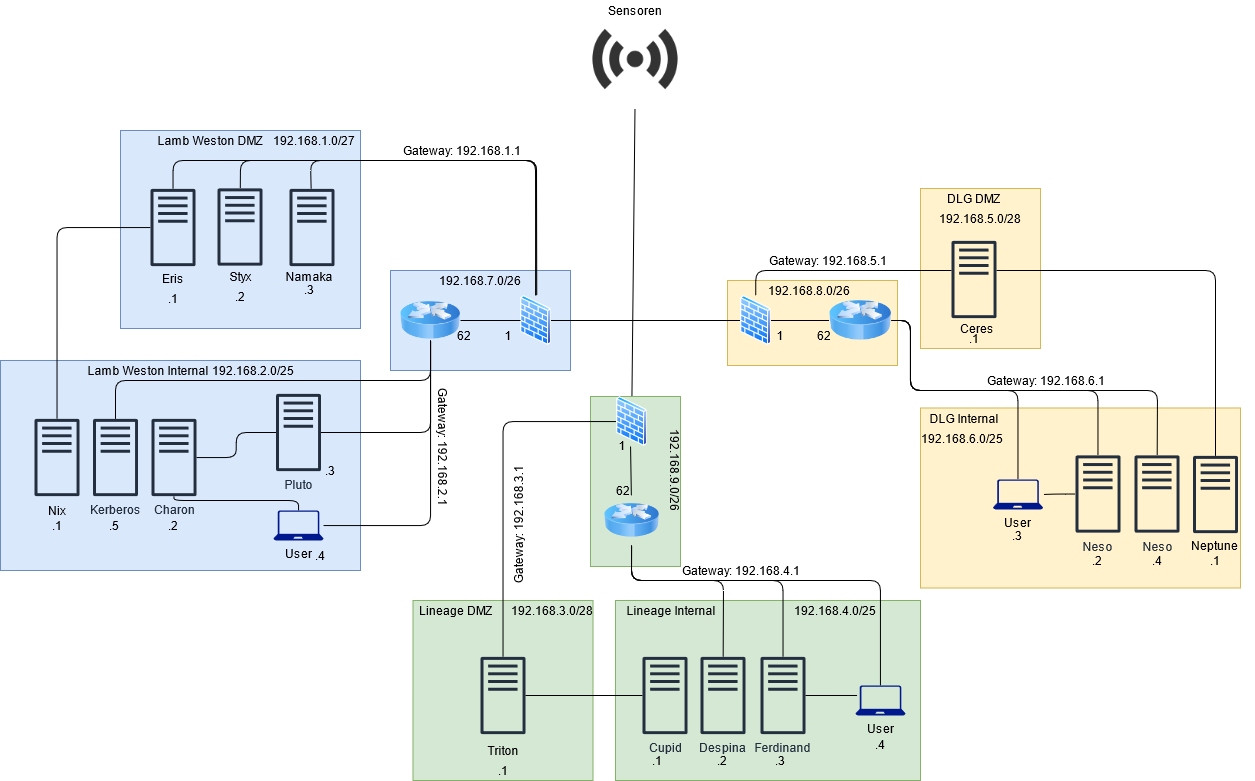


Figure 3‑2: Layer 3 drawing

Figure 3-2 shows the layer 3 drawing of the designed Hyperledger Fabric blockchain infrastructure for Lamb Weston, Lineage and DLG. The drawing is based on a design from the research paper *Proof of Concept of Blockchain Technology in the Field of Finance Using Hyperledger Fabric 1.0* [3]*.* The design has been changed to be used with Hyperledger Fabric version 2.3, and to be applicable to this certain use case.

The network is divided into three locations Lamb Weston, Lineage and DLG. The locations will be connected using a site-to-site VPN. Each location is divided into a firewall and router zone, demilitarized zone (DMZ), and an internal network zone.

Each location contains a blockchain server which will run the Hyperledger Fabric peer, chaincode and ledger. The DMZ of Lamb Weston also contains an orderer server and membership service provider (MSP) server. The orderer orders the different blockchain transactions. The MSP is a directory server which contains all digital certificates of participating nodes. The MSP will be used to identify and validate all the nodes participating in the network.

Each internal network contains a certificate authority server used to provide the digital certificates to the MSP and a CouchDB server, which will serve as the state database. The CouchDB server is connected to the blockchain server to retrieve the current key value. It will also contain a server used to provide a webserver interface to the end user. The interface will show all blockchain data.

The internal network of Lamb Weston contains the API server used to retrieve and store sensor data in the blockchain.

The naming convention can be found in chapter 3.3.

## Layer 2 drawing

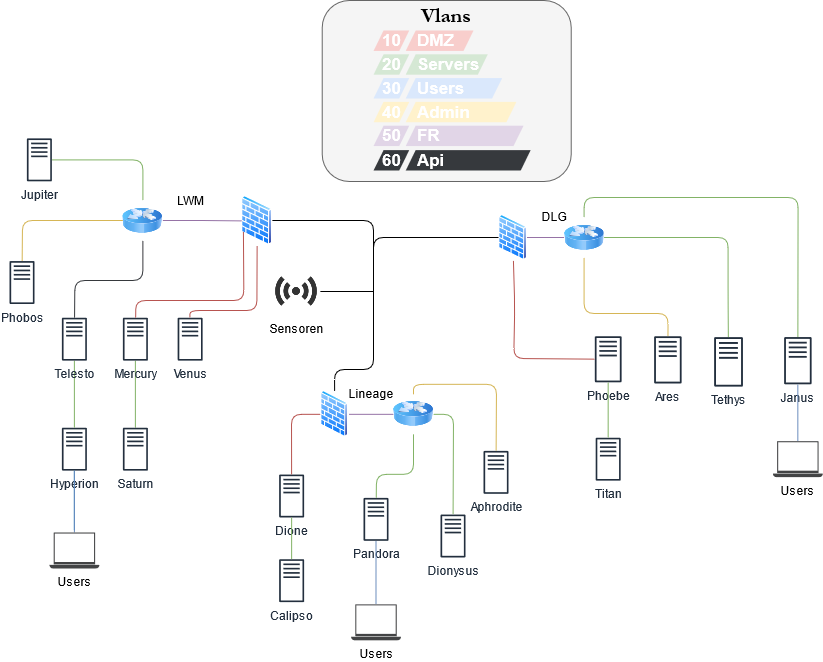


Figure 3‑3: Layer 2 drawing

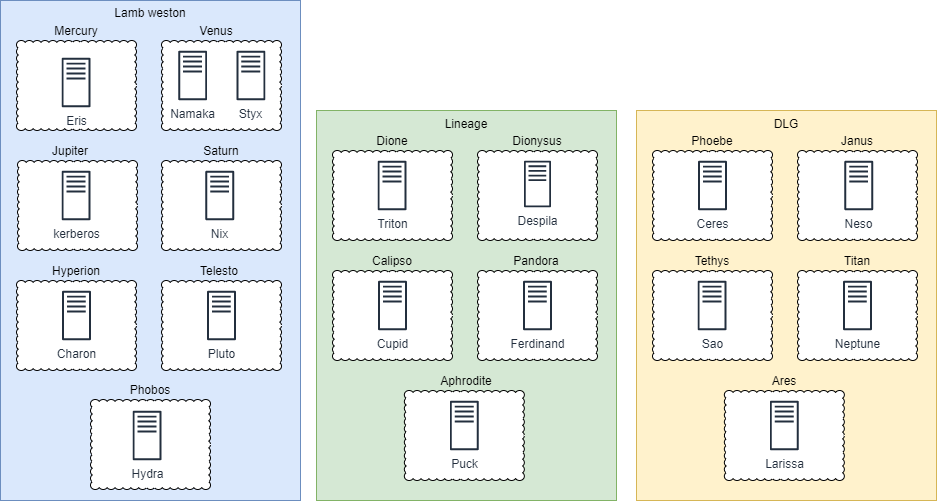
****

Figure 3‑4: Virtualization layer 2

Figure 3-3 shows the layer 2 drawing of the designed Hyperledger Fabric blockchain infrastructure for Lamb Weston, Lineage and DLG. It shows which servers and VLAN’s will be used.   
Figure 3-4 shows which virtual services will run on the servers.

## Subnet summary

Table 3‑1: Subnet summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Network name | Adresses | Networkadres | Prefix | VLAN |
| DMZ | 30 | 192.168.[1,3,5].0 | /27 | 10 |
| Servers | 126 | 192.168.[2,4,6].0 | /25 | 20 |
| Users | n/a | n/a | n/a | 30 |
| Admin | n/a | n/a | n/a | 40 |
| FR | 6 | 192.168.[7,8,9].0 | /29 | 50 |
| Api | 1 | 192.168.2.3 | /25 | 60 |

## Naming conventions

To keep server naming convenient, all servers will be named according to the naming convention. The naming convention sequence will look like this:

Table 3‑2: Naming conventions

|  |  |
| --- | --- |
| Lamb Weston | |
| Function | Name |
| Blockchain server | Mercury |
| MSP + orderer | Venus |
| Certificate Authority | Jupiter |
| CouchDB (state database) | Saturn |
| Webserver Interface | Hyperion |
| API | Telesto |
| Admin | Phobos |

|  |  |
| --- | --- |
| Lamb Weston virtual names | |
| Function | Name |
| Blockchain | Eris |
| MSP | Namaka |
| Orderer | Styx |
| Certificate Authority | Kerberos |
| CouchDB (state database) | Nix |
| Webserver Interface | Charon |
| API | Pluto |
| Admin | Hydra |

|  |  |
| --- | --- |
| DLG | |
| Function | Name |
| Blockchain server | Phoebe |
| Webserver interface | Janus |
| Certificate Authority | Tethys |
| CouchDB (state database) | Titan |
| Admin | Ares |

|  |  |
| --- | --- |
| DLG virtual names | |
| Function | Name |
| Blockchain | Ceres |
| Certificate Authority | Sao |
| CouchDB (state database) | Neptune |
| Webserver Interface | Neso |
| Admin | Larissa |

|  |  |
| --- | --- |
| Lineage | |
| Function | Name |
| Blockchain | Dione |
| Certificate Authority | Dionysus |
| CouchDB (state database) | Calipso |
| Webserver interface | Pandora |
| Admin | Aphrodite |

|  |  |
| --- | --- |
| Lineage virtual names | |
| Function | Name |
| Blockchain | Triton |
| Certificate Authority | Despina |
| CouchDB (state database) | Cupid |
| Webserver Interface | Ferdinand |
| Admin | Puck |

|  |  |
| --- | --- |
| Network location | Reference |
| Demilitarized zone | DMZ |
| Internal network | INT |

## Securitymatrix

Table 3-3 shows all ports which will be used by the Hyperledger Fabric applications and network.

Table 3‑3: Security matrix used ports

|  |  |
| --- | --- |
| Service | Port |
| HTTPS | 443 |
| Kafka | 9092 |
| Fabric CA | 7054 |
| CouchDB | 5984 |
| MSP | 8052 |

Table 3-4 shows what ports must be allowed between the different zones.

Table 3‑4: Security matrix allowed ports

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | LWM DMZ | LWM INT | DLG DMZ | DLG INT | LNG DMZ | LNG INT | To |
| LWM DMZ | --------------- | 443, 5984, 7054 | 443 | X | 443 | X |  |
| LWM INT | 443, 5984, 7054 | --------------- | X | X | X | X |  |
| DLG DMZ | 443, 9092, 8052 | X | --------------- | 443, 5984, 7054 | 443 | X |  |
| DLG INT | X | X | 443, 5984, 7054 | --------------- | X | X |  |
| LNG DMZ | 443, 9092, 8052 | X | 443 | X | --------------- | 443, 4369, 7054 |  |
| LNG INT | X | X | X | X | 443, 5984, 7054 | --------------- |  |
| From |  |  |  |  |  |  |  |

# Infrastructure services

This chapter will describe all services which will be used to build the Hyperledger Fabric blockchain network for Lamb Weston, Lineage and DLG. The services are based on the OIAm open repository.   
The required (virtual) hardware, software, specifications, and connections with and impact on adjacent infrastructure services will be specified for each service. The services will be based on requirements from the Requirement Traceability Matrix [1].

**The hardware specifications of every docker container can be found in the Facilities Deployment.Docker.**

## Technical components and specifications for Datazone Protection.Firewall

### Description

This section covers the firewall.

### (Virtual) hardware

Three Fortinet FortiGate 30E firewalls will be used. One for each company (LWM, Lineage and DLG).

### Software

The firewalls will run the following software:

|  |  |
| --- | --- |
| Operating system | Version |
| FortiOS | 7.0 |

### Specifications

The following firewall rules will be used:

|  |  |  |  |
| --- | --- | --- | --- |
| Lamb Weston Firewall | | | |
| Port | Protocol | Allow/deny | |
| 443 | TCP | Allow |
| 9092 | TCP | Allow |
| 7054 | TCP | Allow |
| 5984 | TCP | Allow |
| 8052 | TCP | Allow |

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Identity and Permissions Management.Hyperledger*** | **Identity and Permissions Management.Hyperledger** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. All traffic from the internet gets filtered and monitored before it can enter the network. All traffic from the network gets filtered before it can leave the network. It ensures that only allowed traffic enters and leaves the network. |
| ***Relational Data Management.Datastore*** | **Relational Data Management.Datastore** hosts the blockchain server and state database. The blockchain servers hosts the ledgers (records all transactions) and smart contracts (the SLA). |
| ***Message Handling.Orderer*** | ***Message* Handling*.Orderer***is used to order the blockchain transactions. |
| ***Facilities Deployment.Docker*** | ***Facilities Deployment.Docker***provides the docker containers used to run Hyperledger Fabric services. |
| ***Application Hosting.Webinterface*** | **Application Hosting.Webinterface** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| ***Data Transport.Router*** | **Data Transport.Router** connect all adjacent services to the internet and each other. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| R42 | Each network should possess the ability to open and close ports. | Each network will have a Fortinet FortiGate 30E Firewall. |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Adaptability | N/A | N/A |
| Scalability | N/A | N/A |
| Manageability | N/A | N/A |
| Accountability | N/A | N/A |
| Availability | N/A | N/A |
| Integrity | N/A | N/A |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| N/A | N/A | N/A |

## Technical components and specifications for Data Transport.Router

### Description

This section covers the router.

### (Virtual) hardware

Three MikroTik RB1100AHx4 routers will be used. One for each company (LWM, Lineage and DLG).

### Software

The routers will run the following software:

|  |  |
| --- | --- |
| Operating system | Version |
| RouterOS | 6.48.2 |

### Specifications

To connect the different sites together site-to-site IPsec (IKEv2) tunnels will be used.   
The package *security* must be installed on all routers for the tunnels to work.   
Configuration for the tunnels can be found on the MikroTik wiki, the following specifications will be used:

|  |  |
| --- | --- |
| Specification LWM | Value |
| dh-group | modp2048 |
| enc-algorithm | aes-128 |
| Profile name | ikev2-sitelwm |
| Peer-address | 145.53.10.0/32 |
| Src-address | 192.168.1.0/27 192.168.2.0/25 |
| Src-port | Any |
| Dst-address | 192.168.3.0/28 192.168.4.0/25  192.168.5.0/28 192.168.6.0/25 |
| Dst-port | Any |
| secret | XXXX |

|  |  |
| --- | --- |
| Specification Lineage | Value |
| dh-group | modp2048 |
| enc-algorithm | aes-128 |
| Profile name | ikev2-sitelng |
| Peer-address | 145.53.20.0/32 |
| Src-address | 192.168.3.0/28 192.168.4.0/25 |
| Src-port | Any |
| Dst-address | 192.168.1.0/27 192.168.2.0/25  192.168.5.0/28 192.168.6.0/25 |
| Dst-port | Any |
| secret | XXXX |

|  |  |
| --- | --- |
| Specification DLG | Value |
| dh-group | modp2048 |
| enc-algorithm | aes-128 |
| Profile name | ikev2-sitedlg |
| Peer-address | 145.53.30.0/32 |
| Src-address | 192.168.5.0/28 192.168.6.0/25 |
| Src-port | Any |
| Dst-address | 192.168.3.0/28 192.168.4.0/25  192.168.1.0/27 192.168.2.0/25 |
| Dst-port | Any |
| secret | XXXX |

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| **Identity and Permissions Management.Hyperledger** | **Identity and Permissions Management.Hyperledger** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. | **Data Transport.Router** connects all adjacent services to the internet and each other. This services routes all the data of the adjacent services so they can communicate. |
| **Relational Data Management.Datastore** | **Relational Data Management.Datastore** hosts the blockchain server and state database. The blockchain servers hosts the ledgers (records all transactions) and smart contracts (the SLA). |
| ***Message* Handling*.Orderer*** | ***Message* Handling*.Orderer***is used to order the blockchain transactions. |
| ***Facilities Deployment.Docker*** | ***Facilities Deployment.Docker***provides the docker containers used to run Hyperledger Fabric services. |
| **Application Hosting.Webinterface** | **Application Hosting.Webinterface** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| **Datazone Protection.Firewall** | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| R41 | Each network should possess the ability to route internet traffic. | Each network will have a MikroTik RB1100AHx4 router. |
| R43 | A secure connection must be established between sites | A secure IPsec (IKEv2) tunnel will be established between each site. |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Adaptability | N/A | N/A |
| Scalability | N/A | N/A |
| Manageability | N/A | N/A |
| Accountability | N/A | N/A |
| Availability | N/A | N/A |
| Integrity | N/A | N/A |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| N/A | N/A | N/A |

## Technical components and specifications for Identity and Permissions Management.Hyperledger

### Description

This section covers the Certificate Authority and Membership Service Provider.

### Software

|  |  |
| --- | --- |
| Software | Version |
| Hyperledger/fabric-ca | 2.3 |

### Specifications

#### Certificate Authority

The server certificates used in the PoC are pre-generated. In a real-life production environment these certificates must be generated by the company themselves.

To use the pre-generated certificates the folder /var/mynetwork/certs must be created on each server. The GitHub folders ‘crypto-config’ and ‘config’ need to be copied to this folder.

**Populate\_hostname.sh**For each organization the path to the certificate authority needs to be set. An example for org1 can be found below.

ORG1\_CA\_PATH=$(ls /var/mynetwork/certs/crypto-config/peerOrganizations/org1.example.com/ca/ | grep "\_sk")

sed "$FLAG" "s#- FABRIC\_CA\_SERVER\_CA\_KEYFILE=/etc/hyperledger/fabric-ca-server-config/.\*#- FABRIC\_CA\_SERVER\_CA\_KEYFILE=/etc/hyperledger/fabric-ca-server-config/$ORG1\_CA\_PATH#g" $SERVICE\_ORG1\_COMPOSE\_PATH

These configurations will be applied for org1, org2, and org3 if the ‘populate\_hostname.sh’ script is run.

**Docker-compose-orderer.yml**For each organization you need to map the crypto-config and config folder so the docker container can find the required certificates and configuration files. An example for org1 can be seen below.

services:

volumes:

- /var/mynetwork/certs/crypto-config/ordererOrganizations/example.com/users:/var/hyperledger/users

- /var/mynetwork/certs/config/:/var/hyperledger/config

**Docker-compose-peer.yml**For each organization you need to map the crypto-config and config folder. An example for org1 can be seen below.

services:

volumes:

- /var/mynetwork/certs/crypto-config/peerOrganizations/org1.example.com/users:/var/hyperledger/users

- /var/mynetwork/certs/config/:/var/hyperledger/config

**Docker-compose-services.yml**The docker-compose-services.yml file is used to configure and run the docker containers. The configuration for the certificate authority needs to be defined in this yaml file for each organization. An example for org1 can be found below.

services:

ca\_org1:

networks:

Kontgoods:

aliases:

- ca\_org1

deploy:

replicas: 1

restart\_policy:

condition: on-failure

delay: 5s

max\_attempts: 3

placement:

constraints:

- node.hostname == Lambweston

image: hyperledger/fabric-ca:latest

hostname: ca.org1.example.com

environment:

- FABRIC\_CA\_HOME=/etc/hyperledger/fabric-ca-server

- FABRIC\_CA\_SERVER\_CA\_NAME=ca\_org1

- FABRIC\_CA\_SERVER\_CA\_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem

- FABRIC\_CA\_SERVER\_CA\_KEYFILE=/etc/hyperledger/fabric-ca-server-config/bb41461d957bc74999916b78b9b6ad4aea4bb2bed4b5897c7fda9185e86d5679\_sk

ports:

- "7054:7054"

command: sh -c 'fabric-ca-server start -b admin:adminpw -d'

volumes:

- /var/mynetwork/certs/crypto-config/peerOrganizations/org1.example.com/ca/:/etc/hyperledger/fabric-ca-server-conf

#### Membership Service Provider

For each organization, the following settings need to be configured to correctly configure the docker containers.

**Docker-compose-orderer.yml**

services:environment:

- ORDERER\_GENERAL\_LOCALMSPID=OrdererMSP

- ORDERER\_GENERAL\_LOCALMSPDIR=/var/hyperledger/msp

volumes:

- /var/mynetwork/certs/crypto-config/ordererOrganizations/example.com/orderers/orderer0.example.com/msp:/var/hyperledger/msp

**Docker-compose-peer.yml**

For each peer, the following configurations needs to be set.

services:

environment:

- CORE\_PEER\_MSPCONFIGPATH=/var/hyperledger/msp

- CORE\_LOGGING\_MSP=DEBUG

- CORE\_PEER\_LOCALMSPID=Org1MSP

volumes:

- /var/mynetwork/certs/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/var/hyperledger/msp

**Docker-compose-services.yml**

services:

environment:

- CORE\_PEER\_LOCALMSPID=Org1MSP

- CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Relational Data Management.Datastore*** | ***Relational Data Management.Datastore*** hosts the blockchain server and state database. The blockchain servers hosts the ledgers (records all transactions) and smart contracts (the SLA). | **Identity and Permissions Management.Hyperledger** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. |
| ***Datazone Protection.Firewall*** | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. |
| ***Facilities Deployment.Docker*** | ***Facilities Deployment.Docker***provides the docker containers used to run Hyperledger Fabric services. |
| ***Application Hosting.Webinterface*** | ***Application Hosting.Webinterface*** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| ***Data Transport.Router*** | ***Data Transport.Router*** connect all adjacent services to the internet and each other. |
| ***Message Handling.Orderer*** | **Message Handling.Orderer** Ordering service provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. Clients connect to the channel and may broadcast messages on the channel which are then delivered to all peers. The channel supports atomic delivery of all messages, that is, message communication with total-order delivery and (implementation specific) reliability. [4] |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| R35 | There must be a service that can validate and authenticate users for corresponding actions. | Each organization will have a Hyperledger Fabric Membership Service Provider used to validate and authenticate users for corresponding actions. The MSP’s will be configured by the scripts. |
| R37 | The must be a service that can generate user certificates. | Each organization will have a Hyperledger Fabric Certificate Authority used to generate user certificates. The CA’s will be deployed by the scripts. |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Adaptability | N/A | N/A |
| Scalability | N/A | N/A |
| Manageability | N/A | N/A |
| Accountability | N/A | N/A |
| Availability | N/A | N/A |
| Integrity | N/A | N/A |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| N/A | N/A | N/A |

## Technical components and specifications for Relational Data Management.Datastore

### Description

This section covers the Blockchain Server and State database.

### Software

|  |  |
| --- | --- |
| Software | Version |
| Hyperledger/fabric-couchdb | 3.1.1 |

### Specifications

In this chapter all specifications regarding the datastore will be specified.

#### CouchDB

Settings that must be changed in the container deployment to ensure correct working of the CouchDB as the state database.

##### CouchDB containers

To ensure all nodes in the network have the correct hostnames run the Hyperledger-CondGoods/network/populate\_hostname.sh script. All correct settings for the CouchDB containers are in the docker-compose-services.yml files in the Hyperledger-CondGoods/network/ under the corresponding org tabs. Running these scripts makes sure the services are installed and deployed correctly. Running these scripts is normally done via the Hyperledger-CondGoods/network/start\_all.sh script. The CouchDB containers are deployed twice per organization to make the use of high availability possible. We will go in depth about the specific settings from the CouchDB docker containers.

The placements constraint makes sure the docker container is deployed on the right server. The container is deployed with the latest Hyperledger Fabric (HLF) CouchDB instance. After this de CouchDB container get named with the hostname tag and connected to a specified network.

**Org1**

services:

couchdb0\_org1:

deploy:

replicas: 1

placement:

constraints:

- node.hostname == Lambweston

image: hyperledger/fabric-couchdb:latest

hostname: couchdb0\_org1

networks:

- Kontgoods

**Org2**

services:

couchdb0\_org2:

deploy:

replicas: 1

restart\_policy:

condition: on-failure

delay: 5s

max\_attempts: 3

placement:

constraints:

- node.hostname == DLG

image: hyperledger/fabric-couchdb:latest

hostname: couchdb0\_org2

networks:

- Kontgoods

**Org3**

services:

couchdb0\_org3:

deploy:

replicas: 1

restart\_policy:

condition: on-failure

delay: 5s

max\_attempts: 3

placement:

constraints:

- node.hostname == Lineage

image: hyperledger/fabric-couchdb:latest

hostname: couchdb0\_org3

networks:

- Kontgoods

##### Hyperledger Fabric Peer containers

These settings point the peer to the state database to use and the port on which the container is listening. These settings are deployed using the docker-compose-peer.yml scripts in the Hyperledger-CondGoods/network/(org1, org2, org3) folder.

**Org1**

services:

environment:

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=couchdb0\_org1:5984

depends\_on:

- couchdb0

ports:

- 7051:7051

- 7053:7053

**Org2**

services:

environment:

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=couchdb0\_org2:5984

depends\_on:

- couchdb0

ports:

- 9051:7051

- 9053:7053

**Org3**

services:

environment:

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=couchdb0\_org3:5984

depends\_on:

- couchdb0

ports:

- 11051:7051

- 11053:7053

#### Hyperledger Fabric network

In this section contains all the settings regarding the hosting and settings of the chaincode and smart contracts for the Hyperledger Fabric network.

##### Hyperledger Fabric Peer containers

This section will include all the settings of the Hyperledger Fabric Peers regarding the chaincode and smart contracts.

To install the chaincode all nodes should run the Hyperledger -CondGoods/network/scripts/chaincode/install\_cc.sh script.

**Org1**

services:

environment:

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org1.example.com:7052

**Org2**

services:

environment:

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org2.example.com:7052

**Org3**

services:

environment:

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org3.example.com:7052

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Identity and Permissions Management.Hyperledger*** | ***Identity and Permissions Management.Hyperledger*** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. | ***Data Transport.Router*** connects all adjacent services to the internet and each other. This services routes all the data of the adjacent services so they can communicate.  ***Facilities Deployment.Docker***  All instances of CouchDB and HLF peers run on docker containers.  ***Application Hosting.Webinterface***  The webinterface uses the statedatabase to get worldstates. |
| ***Datazone Protection.Firewall*** | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. |
| ***Facilities Deployment.Docker*** | ***Facilities Deployment.Docker***provides the docker containers used to run Hyperledger Fabric services. |
| ***Application Hosting.Webinterface*** | ***Application Hosting.Webinterface*** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| ***Data Transport.Router*** | ***Data Transport.Router*** connect all adjacent services to the internet and each other. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| R50 | A state database is available in the network | A CouchDB container is deployed with the deploy scripts |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Scalability | The state database supports pagination | A CouchDB container is deployed with the deploy scripts |
| Scalability | The state database supports indexing | A CouchDB container is deployed with the deploy scripts |
| Integrity | Nodes must connect securely with each other | The servers are connected through the firewall |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| The state database can query rich queries | The state database is able to query rich queries with JSON. | A CouchDB container is deployed with the deploy scripts |

## Technical components and specifications for Application Hosting.Webinterface

### Description

This section covers the Webserver and API server related to the Application Hosting.Webinterface infrastructure service.

### (Virtual) hardware

The (virtual) hardware regarding the webserver and API hosting.

#### Apache HTTP Ubuntu Server

|  |  |
| --- | --- |
| Type | Specifications |
| CPU cores | 4 |
| CPU clockspeed | 3.5 GHz |
| Memory | 8GB |
| Storage | 120Gb |
|  |  |
| Network connection | 1x 1Gbit interface |
| Number of nodes | 1 |

#### Node.js Ubuntu server

|  |  |
| --- | --- |
| Type | Specifications |
| CPU cores | 4 |
| CPU clockspeed | 3.5 GHz |
| Memory | 8GB |
| Storage | 120Gb |
|  |  |
| Network connection | 1x 1Gbit interface |
| Number of nodes | 1 |

### Software

Software used for the Application Hosting.Webinterface infrastructure service.

|  |  |
| --- | --- |
| Software | Version |
| Apache HTTP Server | 2.4.47 |
| Node.js | 16.1.0 |
| Ubuntu server | 20.04 |

### Specifications

This section will discuss the specific settings related to the Apache HTTP Server and Node.js.

#### Frontend webserver

To deploy the front-end React app on the Apache2 server some settings have to be changed regarding security and configuration.

##### Apache2 settings

To ensure secure operation some settings must be changed. Underneath is a list with all the settings that should be changed based on a best practice list [5]:

* **Disable directory listing**
* **Disable unnecessary modules**
* **Turn off unnecessary services**
* **Ensure that Apache server-info is disabled**
* **Disable trace HTTP request**
* **Distribute ownership and do not run Apache as ‘root’**

##### Ubuntu server settings

To make the node.js application available to the network settings regarding the interfaces have to be changed according to the network designs in the layer 3 drawing. These settings are:

**Webserver LMW**

IPv4 address = 192.168.2.2

Netmask = 255.255.255.128

Hostname = Chero

**Webserver DLG**

IPv4 address = 192.168.6.2

Netmask = 255.255.255.128

Hostname = Neso

**Webserver Lineage**

IPv4 address =192.168.4.3

Netmask = 255.255.255.128

Hostname = Ferdinand

To ensure secure operation some settings must be changed. Underneath is a list with all the settings that should be changed on all the webservers based on a best practice list [6]:

* Use strong and unique passwords
* Enable automatic updates
* Avoid unnecessary software
* Avoid unnecessary modules
* Close hidden open ports
* Close unnessecary ports

##### Deploying the React App

Because the React app uses frontend routing the line FallbackResource ./index.html must be placed in the /public/.htaccess file. FallbackResource makes sure that index.html is loaded instead of allowing the frontend routing to be applied. To build the react app after development the npm run build command should be run to “build” the app. The generated build folder should be placed in the domain document root folder on the Apache server.

#### Node.js API server

##### Ubuntu server settings

To make the node.js application available to the network settings regarding the interfaces have to be changed according to the network designs in the layer 3 drawing. These settings are:

**API server LMW**

IPv4 address = 192.168.2.3

Netmask = 255.255.255.128

Hostname = Telesto

To ensure secure operation some settings must be changed. Underneath is a list with all the settings that should be changed based on a best practice list [6]:

* Use strong and unique passwords
* Enable automatic updates
* Avoid unnecessary software
* Avoid unnecessary modules
* Close hidden open ports
* Close unnessecary ports

##### Node.js settings

This section

To host the Node.js application the node.js package must be installed on the ubuntu server with:

curl -sL [https://deb.nodesource.com/setup\_8.x -o nodesource\_setup.sh](https://deb.nodesource.com/setup_8.x%20-o%20nodesource_setup.sh)

sudo bash nodesource\_setup.sh

sudo apt install nodejs

To make sure all NPM packages run as intended the build-essential package needs to be installed. Insalling is done by using:

sudo apt install build-essential

To make running the API as a service possible, the pm2 package can be installed with:

sudo npm install pm2@latest -g

After the package install the node.js API build files need to be placed on the ubuntu server and started with pm2.

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Relational Data Management.Datastore*** | To interact with the web interface, the API pulls and pushes data to and from the state database connected to the peers. | ***Application Hosting.Webinterface*** *In the case that the API becomes unavailable the web interface will not be updated with new information from the state database. Furthermore, you wont be able to add data to the state database from the web interface.* |
| ***Datazone Protection.Firewall*** | monitors, filters, and controls all incoming and outgoing traffic. |
| ***Application Hosting.Webinterface*** | The API makes interaction with from the datastore and webinterface possible. |
| ***Data Transport.Router*** | connect all adjacent services to the internet and each other. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| 26 | The dashboard must display the values stored in the blockchain | The API server connects the state database and sensors to the dashboard |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Availability | The dashboard must have an uptime of 85% | The server should have a stable internet and electricity connection |
| Integrity | Nodes must connect securely with each other | The servers are connected through the firewall |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| n.a | n.a | n.a |

## Technical components and specifications for *Message* Handling*.Orderer*

### Description

This section covers the Orderer.

### Software

|  |  |
| --- | --- |
| Software | Version |
| Hyperledger/fabric-orderer | 2.3 |
| Hyperledger/fabric-zookeeper | Latest |
| Hyperledger/fabric-kafka | Latest |

### Specifications

The hostnames and paths for the orderer node, kafka node and zookeeper node need to be set first. This can be done in the ‘populate\_hostname.sh’ script.

**# Orderer  
# Organization 1**sed "$FLAG" "s/- node.hostname == .\*/- node.hostname == $ORG1\_HOSTNAME/g" $ORDERER0\_COMPOSE\_PATH

**# Organization 2**sed "$FLAG" "s/- node.hostname == .\*/- node.hostname == $ORG2\_HOSTNAME/g" $ORDERER1\_COMPOSE\_PATH

**# Organization 3**sed "$FLAG" "s/- node.hostname == .\*/- node.hostname == $ORG3\_HOSTNAME/g" $ORDERER2\_COMPOSE\_PATH

**# Kafka**sed "$FLAG" "s/- node.hostname == .\*/- node.hostname == $ORG1\_HOSTNAME/g" $KAFKA\_COMPOSE\_PATH

# **Zookeeper**sed "$FLAG" "s/- node.hostname == .\*/- node.hostname == $ORG1\_HOSTNAME/g" $ZK\_COMPOSE\_PATH

The kafka services need to be configured. This will be done in a docker-compose yaml file. In this case it will be done in the ‘Docker-compose-kafka.yml’ file.

**# Docker-compose-kafka.yml**

version: '3.2'

networks:

Kontgoods:

external:

name: Kontgoods

services:

kafka0:

deploy:

replicas: 1

placement:

constraints:

- node.hostname == Lambweston

hostname: kafka0

image: hyperledger/fabric-kafka

networks:

Kontgoods:

aliases:

- kafka0

environment:

- KAFKA\_MESSAGE\_MAX\_BYTES=103809024 # 99 \* 1024 \* 1024 B

- KAFKA\_REPLICA\_FETCH\_MAX\_BYTES=103809024 # 99 \* 1024 \* 1024 B

- KAFKA\_UNCLEAN\_LEADER\_ELECTION\_ENABLE=false

- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=Kontgoods

- KAFKA\_BROKER\_ID=0

- KAFKA\_MIN\_INSYNC\_REPLICAS=2

- KAFKA\_DEFAULT\_REPLICATION\_FACTOR=3

- KAFKA\_ZOOKEEPER\_CONNECT=zookeeper0:2181,zookeeper1:2181,zookeeper2:2181

- KAFKA\_ZOOKEEPER\_CONNECTION\_TIMEOUT\_MS=36000

- KAFKA\_ZOOKEEPER\_SESSION\_TIMEOUT\_MS=36000

The zookeeper services need to be configured. This will be done in a docker-compose yaml file. In this case it will be done in the ‘Docker-compose-zookeeper.yml’ file.

**# Docker-compose-zookeeper.yml**

version: '3'

networks:

Kontgoods:

external:

name: Kontgoods

services:

zookeeper0:

hostname: zookeeper0.example.com

image: hyperledger/fabric-zookeeper

deploy:

replicas: 1

placement:

constraints:

- node.hostname == Lambweston

networks:

- Kontgoods

environment:

- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=Kontgoods

- ZOO\_MY\_ID=1

- ZOO\_SERVERS=server.1=zookeeper0:2888:3888 server.2=zookeeper1:2888:3888 server.3=zookeeper2:2888:3888

zookeeper1:

hostname: zookeeper1.example.com

environment:

- ZOO\_SERVERS=server.1=zookeeper0:2888:3888 server.2=zookeeper1:2888:3888 server.3=zookeeper2:2888:3888

- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=Kontgoods

- ZOO\_MY\_ID=2

deploy:

replicas: 1

placement:

constraints:

- node.hostname == Lambweston

image: hyperledger/fabric-zookeeper

networks:

- Kontgoods

zookeeper2:

deploy:

replicas: 1

placement:

constraints:

- node.hostname == Lambweston

hostname: zookeeper2.example.com

image: hyperledger/fabric-zookeeper

networks:

- Kontgoods

environment:

- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=Kontgoods

- ZOO\_MY\_ID=3

- ZOO\_SERVERS=server.1=zookeeper0:2888:3888 server.2=zookeeper1:2888:3888 server.3=zookeeper2:2888:3888

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Identity and Permissions Management.Hyperledger*** | ***Identity and Permissions Management.Hyperledger*** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. | ***Message Handling.Orderer*** *Ordering service provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. Clients connect to the channel and may broadcast messages on the channel which are then delivered to all peers. The channel supports atomic delivery of all messages, that is, message communication with total-order delivery and (implementation specific) reliability.* [4] |
| ***Relational Data Management.Datastore*** | ***Relational Data Management.Datastore*** hosts the blockchain server and state database. The blockchain servers hosts the ledgers (records all transactions) and smart contracts (the SLA). |
| ***Datazone Protection.Firewall*** | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. |
| ***Facilities Deployment.Docker*** | ***Facilities Deployment.Docker***provides the docker containers used to run Hyperledger Fabric services. |
| ***Application Hosting.Webinterface*** | ***Application Hosting.Webinterface*** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| ***Data Transport.Router*** | ***Data Transport.Router*** connect all adjacent services to the internet and each other. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| R49 | *Atomic delivery of all messages must be available* | The **Message handeling.Orderer** provides a channel that clients and peers can connect with, to broadcast their messages containing transactions; these messages will be delivered atomically. |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| Adaptability | N/A | N/A |
| Scalability | N/A | N/A |
| Manageability | N/A | N/A |
| Accountability | N/A | N/A |
| Availability | N/A | N/A |
| Integrity | N/A | N/A |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| N/A | N/A | N/A |

## Technical components and specifications for Facilities Deployment.Docker

### Description

This section covers the docker containers used to run the infrastructure services.

### (Virtual) hardware

The PoC runs on three Ubuntu 20.04 Servers. The Hyperledger Fabric services will run in docker containers.

|  |  |
| --- | --- |
| Ubuntu 20.04 server | |
| Type | Specifications |
| CPU cores | 2 |
| CPU clockspeed | 2.1 GHz |
| Memory | 4GB |
| Storage | 30Gb |
| Network connection | 1Gbit interface |
| Number of nodes | 3 |

|  |  |
| --- | --- |
| Docker containers | |
| Type | Specifications |
| Number of containers | 26 |

### Software

|  |  |
| --- | --- |
| Software | Version |
| Ubuntu Desktop | 20.04 |

|  |  |
| --- | --- |
| Software | Version |
| Docker | 20.10.2 |
| Docker-compose | 1.29.2 |

The docker image used for each docker container can be found in the corresponding infrastructure service.

### Specifications

**Docker container configuration**The configurations for each docker container will be defined in docker compose yaml files. The specifications for each docker compose file can be found in its corresponding infrastructure service.

**Docker swarm**The docker containers will be connected using docker swarm. The Lambweston node will be the swarm manager; the DLG and Lineage nodes will be swarm workers. The overlay network that will be used has the following specifications.

|  |  |
| --- | --- |
| Specification | Value |
| Subnet | 10.200.10/24 |
| Network name | Kontgoods |

**Used ports**The docker containers and services will use the following ports.

|  |  |  |
| --- | --- | --- |
| Lamb Weston node | | |
| Container | Port | Protocol |
| Orderer | 7050 | TCP |
| Kafka | 9092 | TCP |
| Peer 0 | 7051, 7053 | TCP |
| Peer 1 | 8051, 8053 | TCP |
| Peer 1 and 2 chaincode listen address | 7052 | TCP |
| Certificate Authority | 7054 | TCP |
| CouchDB | 5984 | TCP |

|  |  |  |
| --- | --- | --- |
| Lineage node | | |
| Container | Port | Protocol |
| Orderer | 8050 | TCP |
| Kafka | 9092 | TCP |
| Peer 0 | 9051, 9053 | TCP |
| Peer 1 | 10051, 10053 | TCP |
| Peer 1 and 2 chaincode listen address | 7052 | TCP |
| Certificate Authority | 8054 | TCP |
| CouchDB | 5984 | TCP |

|  |  |  |
| --- | --- | --- |
| DLG node | | |
| Container | Port | Protocol |
| Orderer | 9050 | TCP |
| Kafka | 9092 | TCP |
| Peer 0 | 11051, 1053 | TCP |
| Peer 1 | 12051, 12053 | TCP |
| Peer 1 and 2 chaincode listen address | 7052 | TCP |
| Certificate Authority | 9054 | TCP |
| CouchDB | 5984 | TCP |

##### Ubuntu server settings

To ensure secure operation some settings must be changed. Underneath is a list with all the settings that should be changed based on a best practice list [6]:

* Use strong and unique passwords
* Enable automatic updates
* Avoid unnecessary software
* Avoid unnecessary modules
* Close hidden open ports
* Close unnessecary ports

### Connections with and impact on adjacent infrastructure services

|  |  |  |
| --- | --- | --- |
| Adjacent infrastructure service | Description | Impact |
| ***Identity and Permissions Management.Hyperledger*** | ***Identity and Permissions Management.Hyperledger*** hosts the certificate authority and MSP. The certificate authority issues certificates to register identities in the blockchain. The MSP validates and issues certificates. Authenticates the users. | ***Facilities Deployment.Docker***provides the docker containers used to run all Hyperledger Fabric services. |
| ***Relational Data Management.Datastore*** | ***Relational Data Management.Datastore*** hosts the blockchain server and state database. The blockchain servers hosts the ledgers (records all transactions) and smart contracts (the SLA). |
| ***Datazone Protection.Firewall*** | **Datazone Protection.Firewall** monitors, filters, and controls all incoming and outgoing traffic. |
| ***Message Handling.Orderer*** | ***Message Handling.Orderer*** *Ordering service provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. Clients connect to the channel and may broadcast messages on the channel which are then delivered to all peers. The channel supports atomic delivery of all messages, that is, message communication with total-order delivery and (implementation specific) reliability.* [4] |
| ***Application Hosting.Webinterface*** | ***Application Hosting.Webinterface*** hosts the API used for retrieving and storing sensor data. Also hosts the webserver interface used to inspect the blockchain data. |
| ***Data Transport.Router*** | ***Data Transport.Router*** connect all adjacent services to the internet and each other. |

### Requirement processing

#### Functional requirements processing

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Technical Processing |
| N/A | N/A | N/A |

#### Quality requirements processing

|  |  |  |
| --- | --- | --- |
| Quality attribute | Value | Technical Processing |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| N/A | N/A | N/A |
| N/A | N/A | N/A |

#### Principle requirements processing

|  |  |  |
| --- | --- | --- |
| Principle | Description | Processing |
| R30 | Docker must be used to deploy the infrastructure | Docker version 20.10.2 will be used to run the Hyperledger Fabric infrastructure services. Docker compose version 1.29.2 will be used in combination with docker compose yaml files to configure the docker containers. |
| R31 | Docker swarm must be used | Docker swarm will be used to connect the docker containers together. The Lambweston node will be the swarm manager, the DLG and Lineage node will be the swarm workers. |

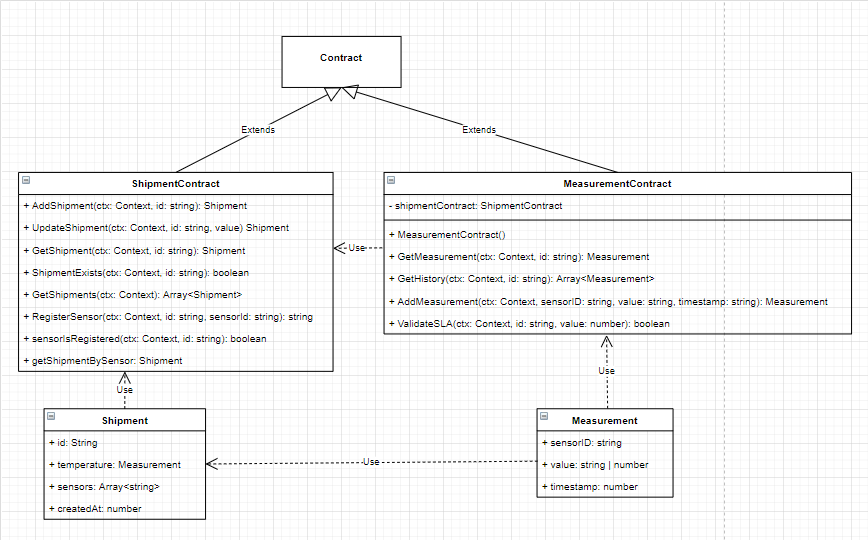
# Application services

The software for the Hyperledger Fabric Network is split in to three different application that work together to give the optimal experience. They can also work independent from each other for example the chaincode could be assessed without the API. So that one application does not affect everything. The software is split into the following applications:

* **Chaincode (smart contracts):** the chaincode handles the business logics by allowing applications to retrieve and insert data in to the blockchain. This is achieved by defining smart contracts.
* **Application Programmer Interface (API):** the API handles the communication between the sensors/dashboard and the blockchain through the chaincode. This allows us to have one central endpoint to communicate to, instead of having multiple application accessing the chaincode with their own gateway.
* **Dashboard (front-end):** the dashboard application’s primary task is to visualize the data inside the blockchain for the user. In this dashboard it will also be possible to add additional shipments and attach sensors to shipments.

## Chaincode

Hyperledger Fabric provides multiple SDKs (Software Development Kit) to communicate with the blockchain Ledger. With these SDKs, it is possible to interact with the ledger by building chaincodes. Chaincodes define the way smart contracts handle inserting and retrieving data from the ledger and smart contracts define the business logic. Hyperledger Fabric offers a high-level and a low-level chaincode SDK for Node, Java and GO. The high-level API provides the contract interface for developing and implementing smart contracts. The low-level API provides the chaincode interface for also managing the chaincode logic and the communication with the Hyperledger Fabric peers for advanced use cases.



The image above displays a UML class diagram for the contracts in the conditioned goods use case. The use case uses two different smart contracts to handle shipments and measurements. Both contracts extend the default Contract interface provided by the Hyperledger Fabric SDK. This interface is required by Hyperledger Fabric to build smart contracts and implement even listeners. The shipment contract handles the creation and modification of shipments. It also allows the user to append sensors to the shipment. The measurement contract provides a way for sensors to add measurement to shipments and makes It possible to retrieve them. This contract also uses the shipment contract to fetch the shipment that is related to the measurements and sensors. The Shipments and measurements inside the contracts are based on their related class.

Every function within a contract has access to the context (ctx). This allows the functions to interact with the ledger and retrieve or add data to it.

**Shipments**

The shipment class provide the basic information of a shipment including a reference id, a temperature value (measurement object) and a list of sensors. This class could be extended with additional information if that is necessary.

**Measurements**

The measurement class provides the basis for the measurements within the shipment. Currently, it only is used to add a temperature to a shipment but could also be used to implement other measurements like a location. This class provides information such as the id of the sensor, the value, and the timestamp of a measurement.

### Buffers

To retrieve and submit data to in Hyperledger Fabric, the input and output needs to be transformed from a buffer (stream of binary data) to an object or vice versa. Hyperledger Fabric uses these buffers (protobuf) to stay language and platform neutral. This allows Hyperledger Fabric to support numerous languages (Node.js, Go and Java) and platforms.

The data needs to be transformed into these buffers and back to readable data to interact with the blockchain. This can be accomplished by converting the buffer to a string and parsing it back to an object with JSON. Submitting to the blockchain can be done in similar way by transforming the object to a JSON string and creating a buffer out of it.

### Validating the Service Level Agreement

The data provided by the sensors need to be validated against the requirements in the Service Level Agreement (SLA). The smart contract needs to validate if the incoming data meet the values defined in the SLA. This is achieved by using a small basic algorithm, descript in the pseudocode below.

// Measurement value should always be a number.

ValidateTemperature(value):

// Get the previous measurement.

Initialize prev = getPreviousMeasurement()

If (value is not between the min and the max value defined in the SLA)

If (previous value does not exist or previous value is not between the min and the max value defined in the SLA)

// Send email.

The algorithm above will check if the provided value is within the values defined in the SLA. If this is not the case the algorithm will check if the previous value was within this range. If this previous value does not exist or the previous value was inside the range the algorithm will send a notification with a warning. In all other cases the algorithm will not perform an action, this is to prevent the receiver from being spammed with notifications or the SLA has a valid state.

### Indexing

The state database inside the Hyperledger Fabric Network will be CouchDB. This is partly because is support JSON and rich queries. With rich queries a developer can write more advanced queries using operators such as sort and regex.

To optimize the performance of the CouchDB queries it is recommended to defined index files. Indexes prevent CouchDB from examining every row when executing a query. This will make the queries run faster dan more efficient. However, this is not required but strongly recommended. Rich queries such as sort queries will not run without an index defined. A sample index is listed below.

// Index for sorting on the createdAt field.

{

“index”: {

// Names of the fields to be queried

“fields”: [“createdAt”],

},

// Name of the design document (optional but recommended)

“ddoc”: “createAtIndex”,

// Name of the index.

“name”: “createAt”,

// Always “JSON” (because we are using the JSON format to store the data).

“type”: “json”

}

Index are defined inside the META-INF/statedb/couchdb/indexes folder within the chaincode. The indexes inside this folder will be installed when the chaincode is deployed.

## API

The application will make use of a RESTful web API, which will be built in a back-end web application framework for NodeJS. The API will be built as an express app, which will we deployed on its own server.

The Api will be designed so that there is way to communicate with the Hyperledger Fabric network without having to build a gateway for every application that wants to use the blockchain. It would be unwise to set up a gateway to the network on every sensor in the supply chain, it would also be al lot unnecessary work to build connection profiles and certificates to every sensor. In the end and maybe the most important reason would be that this prevents any security issues that may arise with having sensors that are directly connected to the gateway. Instead, we have an sensor connected to an Api that is connected to the gateway of Hyperledger Fabric.

### Cors

The Api will make use of cross-origin resource sharing (CORS), which allows the Api to indicate which origins other than its own is allowed to make use of its resources. Cors is an HTTP-header based mechanism that allows a server to indicate other origins than its own from which a browser should permit the loading of resources. This also means that when the application is finished that organisations, and or any other application with a different origin should be registered with CORS.

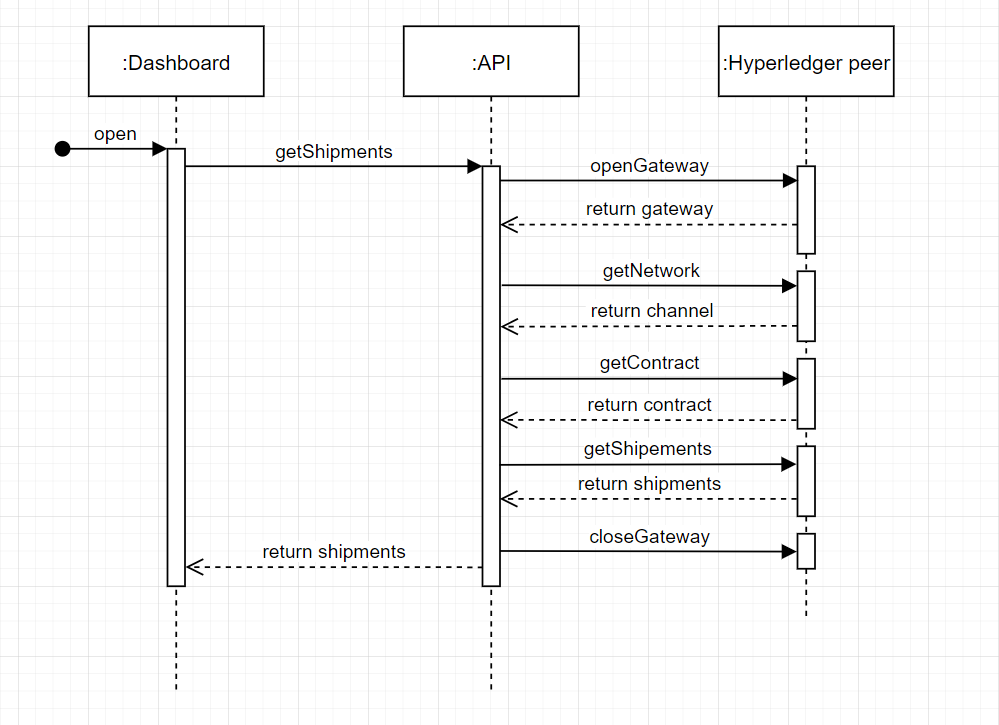
### Dotenv

The Api will make use of Dotenv, which is a zero-dependency module that can stores global variables in a .env file, which can be called by a recallable variable called process.env. This makes it possible to store global variables in a single well-ordered file, it also allows you to separate secret variables from the source code. This is done by GIT-ignoring the Dotenv file and making an Dotenv -example file with all the necessary variables, but all the secret variables can be kept empty. This way a different person can copy this file, rename it to Dotenv, and fill in his own secret variables.

### Hyperledger Gateway

To communicate with a peer within Hyperledger fabric a gateway needs to be setup. With the gateway it is possible to communicate directly with the channels and chaincode inside the Hyperledger fabric network.

In the sequence diagram below displays how the dashboard will retrieve the data from the blockchain through the API. The dashboard will perform a http request to the API asking for the data. The API receives this request and will open a gateway connection to a Hyperledger Fabric peer to retrieve this data. After the gateway is setup, the API will retrieve the channel and the contract. When the API has collected all the necessary information it can start handling the request and retrieve the information the dashboard requested. The API will use the retrieved contract to query the data and return it to the dashboard as a response.



### Route resolving

To add different routes for an API, Express.js expects developers to register them manual through a function. Because this is not optimal if the API contains numerous routes and schemas. Therefore, the API will use a way to configure routes by linking routes, schemas, and functions together in one object before passing them to a function that will register the routes. The route object consists of a route URL, a http type, a schema object, and the route function. An example route can be found below.

[“/shipments”]: {type: “get”, schema: getShipmentsSchema, func: getShipments}

This information is then passed to a function that will register all the routes and validate the schema of a route when a request is made. This function will look something like the pseudo code defined below.

For each (route inside the routes object)

Initialize key, schema, func = route

// Register route in express.

app[type](key checkSchema(schema), (req, res) => {

Initialize errors = validationResult();

// Check if there are validation errors.

If (there are errors)

// Throw error

// Call route function.

func(req, res);

});

This function will register the routes and validate if the incoming request with the schema. If all the information within the body or the route parameters are valid the function will execute the function of this route and provide information back. If the information is invalid the API will throw an error.

### Route schema validation

To check if API requests contain only valid data, route schema validation will be used using a package called Express Validator. Express Validator is an express middleware that can be used to validate and sanitize functions. The API requests are validated using schemas. Schemas are being used because it is the most compact and easy way to configure validation for the data provided to the API. An example of a schema could be something like the example below.

// Get Shipment schema

Id: {

// Validate data in the route params.

In: [“params”],

// Return the error if route param is invalid.

errorMessage: “id is required”,

// Check if length is at least 1.

isLength: {

options: {min: 1},

}

// Trim the input.

trim: true

}

Figure 5‑1: Shipment schema example

This schema trims received parameters and checks if they contain expected data. In this case the schema checks if an ID is not empty. If it is not the case the function will throw an error and the request will fail.

### Pagination

Pagination with Hyperledger fabric and CouchDB (key value Json database) works with bookmarks. The reason why this is, is because blockchain is a linked list. This means it not possible to jump to an index of a page because there are no indexes or pages. To make this work we need to provide a bookmark as reference for CouchDB to start at this bookmark and fetch the next couple of records. This also means that the limit keyword is not support, since Hyperledger Fabric will take care of managing the page size. This makes is perfect for infinite scroll.

## Dashboard

The dashboard of the Hyperledger fabric network will be displayed on a dashboard. The dashboard shall be built as an React JS web application, React JS is a JavaScript library used in web development to build interactive elements on websites. With React it is possible to build user interfaces using declarative code, this means we code what we want and not how to do it.

Diagram

Description automatically generatedWith React is it possible to build single page applications and mobile apps, but it could also be used to build complex and large apps if you utilize it with other libraries.

Figure 5-3: Activity Diagram Dashboard web application

### User Activity

As seen by the activity diagram below it is apparent that every page/activity will be accessible from every other page/activity. The user will always start at the dashboard, this being the front page of the application. From here on out the user will have a sidebar with all the possible pages/activities, which can be clicked at any point in time of using the application.

### Hooks

A hook is a special JavaScript function that lets you ‘hook into’ React features. When you are writing a component in react and realize you need to add some state to the component, it would previously be required to convert the component in a class. With hooks however it is possible to declare a state inside the function itself. For example the ‘useState’ hook makes it possible to add react state to a component without having to write a class for it.

It is possible to create your own custom hooks too, a custom hook is a JavaScript function whose name start with “use” and that may call other hooks. You can write custom hooks that cover a wide range of use cases like handling, animation, declarative subscriptions, timers, and many other possibilities.

### Charts

In the dashboard Chart.js will be used to create a visual representation of the measurement history inside the blockchain. To make it possible to show the data in a chart based on the sensor id, the data structure of the history needs to be redefined by an algorithm before passing it to the chart. The data provided by the history API call will be restructure to the following layout.

{

// Label of the dataset.

Label: string;

// Data of the dataset.

Data: Array<{

// Timestamp

x: Date;

// Measurement value.

y: number;

}>;

// Background color of the line.

backgroundColor: string;

// Border color of the line.

borderColor: string;

// Fill line.

fill: boolean

}

The data will be restructure to data structure above by the following algorithm defined in simple pseudocode.

// Value is an array of measurement objects

OrderData(value):

Initialize datasets = {}

// Sort measurements on timestamp.

For each (value in measurements)

If (value is not inside datasets)

// Create a new dataset object and add it to datasets

Else

// Add measurement value to the already existing dataset (inside the data array).

Return datasets as an array.

This algorithm will start by sorting the provided array of measurements by their timestamp. After this it will loop though all the measurement objects inside the array provided by the first parameter. For each measurement it will check if its sensor id is already in the dataset, if this is not the case it will add a new dataset object for this sensor id. When the sensor id is already inside the dataset object it will add the measurement value to the existing dataset. When this is done the object will be converted to an array of dataset objects.

# References

|  |  |
| --- | --- |
| [1] | T. Imming, S. van de Kerkhof, T. Mansveld, B. Güzel en G. Wesselig, „Requirement traceability matrix,” Zwolle, 2021. |
| [2] | Volkerdon, „MoSCoW prioritization technique,” 2018. [Online]. Available: https://www.volkerdon.com/pages/moscow-prioritisation. |
| [3] | S. Toriyama, T. Okabe en S. K. Y. Tanaka, „Proof of Concept of Blockchain Technology in the Field of Finance Using Hyperledger Fabric 1.0,” NEC Technical Journal, 2018. |
| [4] | H. fabric, „Architecture Explained,” IBM, [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release/arch-deep-dive.html. [Geopend 28 4 2021]. |
| [5] | Monitis, „Top 7 Apache security best practices,” Monitis, 2 06 2016. [Online]. Available: https://blog.monitis.com/blog/top-7-apache-security-best-practices/. [Geopend 27 05 2021]. |
| [6] | Y. Davydov, „Linux Server Security: 10 Linux Hardening & Security Best Practices,” thesslstore.com, 7 8 2020. [Online]. Available: https://www.thesslstore.com/blog/linux-server-security-linux-hardening-best-practices/. [Geopend 27 05 2021]. |
| [7] | The linux foundation, „Using CouchDB,” Hyperledger, 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.2/couchdb\_tutorial.html. [Geopend 28 April 2021]. |