|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Project acronym: **IoTrust**  Project title: **Secure trust bootstrapping and peer to peer network connection in the Internet of Things**  Third Party: **DW/ODINS**   |  |  | | --- | --- | | Digital Worx GmbH (DW) - Germany |  | | Odin Solutions SL (ODINS) - Spain |  |   Deliverable D.2  Open Standards based IoTrust Development   |  |  | | --- | --- | | **Deliverables leader:** | **Odins Solutions SL (ODINS)** | | **Authors:** | Silke Capo, Mihaly Virag, Mirko Ross, Rafael Marin-Perez, Antonio Skarmeta Gomez, | | **Due date:** | 31-03-2021 | | **Actual submission date:** | 25-03-2021 | | **Dissemination level:** | Public / confidential | |

Abstract: Please provide a brief description

**Disclaimer**

The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Commission. The European Commission is not responsible for any use that may be made of the information contained therein.

**Copyright**

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the NGI Consortium. In addition, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.

This document may change without notice.

**Table of contents**

[1 Introduction 1](#_Toc26358819)

[2 Activities carried out to complete the deliverable 1](#_Toc26358820)

[3 Technical description 1](#_Toc26358821)

[4 Conclusions and next steps 1](#_Toc26358822)

[Appendix 2](#_Toc26358823)

Nb: The deliverable structure below is only provided for guidance and you may adapt in a free form manner the structure to fit your needs.

# Introduction

Shortly introduce the objective of the deliverable within the overall implementation of your project, explaining how it is inter-related to other deliverables, outputs or milestones.

# Activities carried out to complete the deliverable

Shortly summarise the activities undertaken to produce the deliverable and how you addressed any technical or other unforeseen issues that may have arisen.

# Technical description

Describe briefly the key technical characteristics of the deliverable and explain how they are related to the final results expected to be achieved by the project.

You can choose to include or annex relevant documents, mock-up, weblinks, screenshots, etc).

## LoRaWAN Network Deployment

The employed LoRaWAN module in IoTrust is the RN2483[[1]](#endnote-1) by Microchip. The RN2483 is a market ready integrated module that implements a LoRaWAN Class A and C stack. The RN2483 is a certified LoRaWAN device, this is, the LoRa Alliance guarantees the RN2483's compatibility with any LoRaWAN network that follows the official LoRaWAN Specification v1.0.2[[2]](#endnote-2). This solution was preferred over other LoRaWAN chip models due its popularity in both academia and industry. The communications with the module are done via ASCII commands over a UART, which enables accelerates development and debugging tasks.

The employed LoRaWAN server for IoTrust, the ChirpStack.io project, runs using the micro-service technology Docker. This improves portability and guarantees that the deployment will not suffer from broken library compatibility or missing dependencies due to all the container images being stored in Docker's official repositories. The official ChirpStack developers provide deployment configuration files[[3]](#footnote-1) via its own repository. This design choice dramatically facilitates the deployment and management of IoTrust solutions in different scenarios, due to Docker's wide-spread availability.

## Secure Lightweight Bootstrapping

### SCHC

Static Context Header Compression (SCHC)[[4]](#endnote-3) is one of the key enabling technologies for the IoTrust project. It is a technology aimed at enabling the interoperability of devices using Low-Power Wide-Area Networks (LP-WANs) with the Internet. In order to do so, SCHC applies a series of header compression and packet fragmentation steps over that allow the transmission of IPv6 packets over low bandwidth technologies. It leverages on the key idea that end-devices run specific applications during their entire lifetime. Thus, the data-flows employed by the device are typically well-known during the design stage. This enables the creation of a data table, known as context in SCHC jargon, that will indicate the header compression and packet fragmentation mechanisms how each header field is processed. If the target values of the header fields are known beforehand, this shifts their contents away from the network into to the static context, avoiding their transmission over radio, thus saving battery power and bandwidth. The more header field values are known, the better the compression will be. Instead of building the context during execution, it is defined during the development phase. Thus, it is expected that each project and application employing SCHC will have its own configuration customized to the deployment needs. Furthermore, different end-devices might have different context within the scope of the same project, depending on the final application that is to be run.

LP-WANs implement a thin network layer that does not take interoperability into consideration. However, SCHC can be tailored to the specifics of each LP-WAN technology as specified in different Internet Drafts authored by the IETF lpwan Work Group. Currently, these Internet Drafts define the compression and fragmentation procedure for NB-IoT, SigFox, and LoRaWAN. In IoTrust we take into consideration the specifics of the Internet Draft for implementing SCHC over LoRaWAN[[5]](#endnote-4), as defined by the IETF lpwan Work Group. LoRaWAN application payloads include a header field called fPort. This is an octet employed locally by the user to differentiate applications or verticals. Hence, in the specific case of SCHC, it is employed to transport the first byte of the SCHC packet. This saves a byte of each transmission, since the fPort is mandatory.

## FUOTA and Trust Monitoring

It is the second critical component of the IoTrust project after the bootstrapping. It focuses on facilitating security patches to the IoT devices and generate dynamic trust score. This is all accomplished by multiple open-source technical components. The following sections give more details about them.

### Blockchain

The Blockchain is employed to store critical device and firmware information in an immutable and distributed ledger. It is chain of blocks which are sequentially linked with cryptographic methods. A change in distributed ledger is tracked by transactions. Transactions are cryptographically signed instruction to transfer funds, deploy smart contract and execute a function of smart contract. A list of transactions is generated using consensus and stored in a block. A cryptographic hash is generated for each block based on the content of the block. Each block also contains the cryptographic hash of its parent block. This makes the distributed ledger immutable. When a block is tempered, its newly calculated hash mismatches with the hash stored in its child block. Therefore, it is virtually impossible to temper the data in distributed ledger. The Figure 1 illustrates a series of blocks linked together with their parent block’s hash.

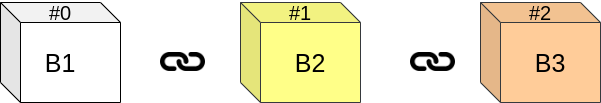


Figure 1

Asvin uses Hyperledger Besu based blockchain. Besu is an open-source Ethereum client written in Java. It is developed under Apache 2.0 license and is maintained by Linux Foundation. Asvin has deployed an Enterprise grade private permissioned blockchain. It is more efficient and has high-performance transaction processing than a public blockchain.

The distributed ledger is exploited to store and retrieve critical device and security patches information in the IoTrust project. The device and firmware metadata include device and firmware identities, mac address, firmware’s content identifier and message digest. The content identifier is used to download a firmware file from the IPFS server. The message digest plays a crucial role in determining integrity of the file.

The Firmware Update smart contract is written in solidity. The smart contract is a collection of code to perform operations on device and firmware data. Typically, these operations are adding, deleting, updating device and firmware metadata. The smart contract is executed in the Ethereum Virtual Machine. Where certain amount of ETH is paid for each write operation.

The besu node deployed and controlled by asvin exposes its functionalities using JSON-RPC and web sockets. The IoT Agent forwards the device registration, firmware update requests to the blockchain using these API endpoints.

The IoTrust project leverages the potential of distributed ledger technology to build a secured, scalable and resilient IoT solution.

### IPFS

InterPlanatery File System (IPFS) is a peer-to-peer storage network. IoTrust project employs this component to store and retrieve security patches for IoT devices. The core principle of IPFS is content based addressing. A file in IPFS is addressed based on its content rather than its location unlike HTTP.

An address of a file in the IPFS is called content identifier (CID). The CID is a cryptographic hash of the content of a file. Therefore, different files with same content have the same CIDs. IPFS supports multiple hashing algorithm. It is powered by the Multihash project. The sha-256 hashing algorithm is used for the IoTrust project. A typical CID looks similar to following address.



IPFS takes advantages of directed acyclic graph (DAG) data structure. In DAG, each node has a unique cryptographic hash generated based on its content. In a graph, content of a node depends on the content of its child nodes. Therefore, if a child node changes then the ripple goes up until the root node in the graph. IPFS builds a Merkle DAG representation of a file by firstly splitting into equal size of *blocks*. These blocks represent different parts of a file. Which can be individually fetched and authenticated.

A private IPFS cluster is utilized for the IoTrust project. It is a group of IPFS nodes. It solves the problem of data redundancy and make the data highly-available. The cluster exposes its functionalities as HTTP API end points. The IoT agent makes these request to IPFS and forward the information to the Network Server. The Figure 2 depicts a IPFS cluster.

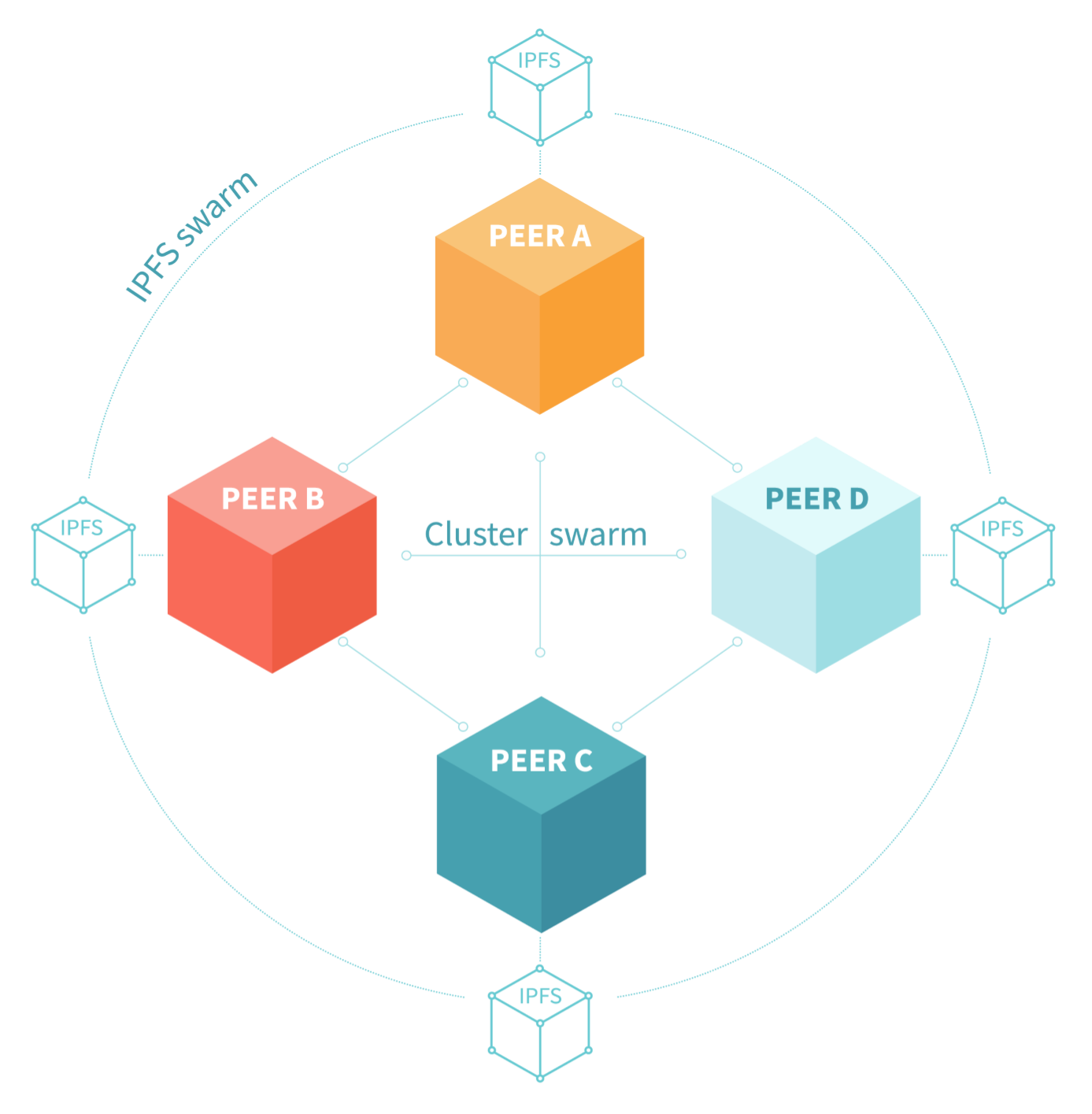


Figure 2: IPFS cluster

QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco

### Cloud Deployment

The IoT Agent, Network Server and asvin’s components Blockchain, IPFS, Customer Platform and Version Controller are all hosted on a cloud server. The cloud server runs Ubuntu 18.04 operating system. All the components are built as docker images and deployed on the cloud using docker compose. The components run as docker containers in an isolated environment.

### Docker Compose

It is tools for defining and running multi-container Docker applications. There applications can be defined in a YAML file as services. Typically, this file is named docker-compose.yml file. An example of docker-compose file used for Network Server is given in the Figure 3.

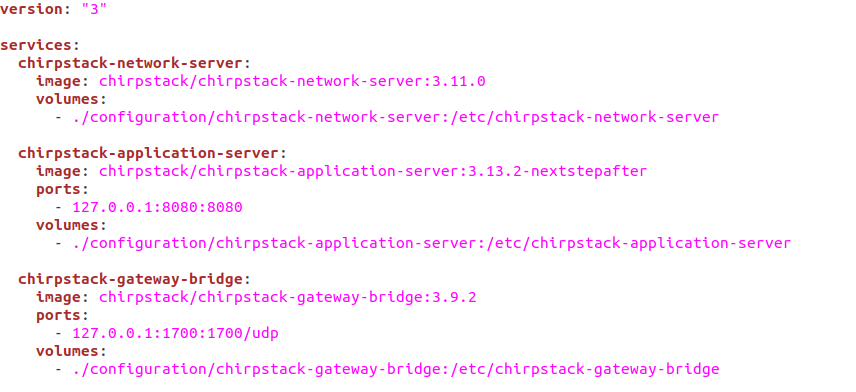


Figure 3: Docker-compose file

It also facilitates a *docker-compose* command line utility to create, start, and stop services defined in the YAML file. Some of the commands are shown below in the Figure 4.

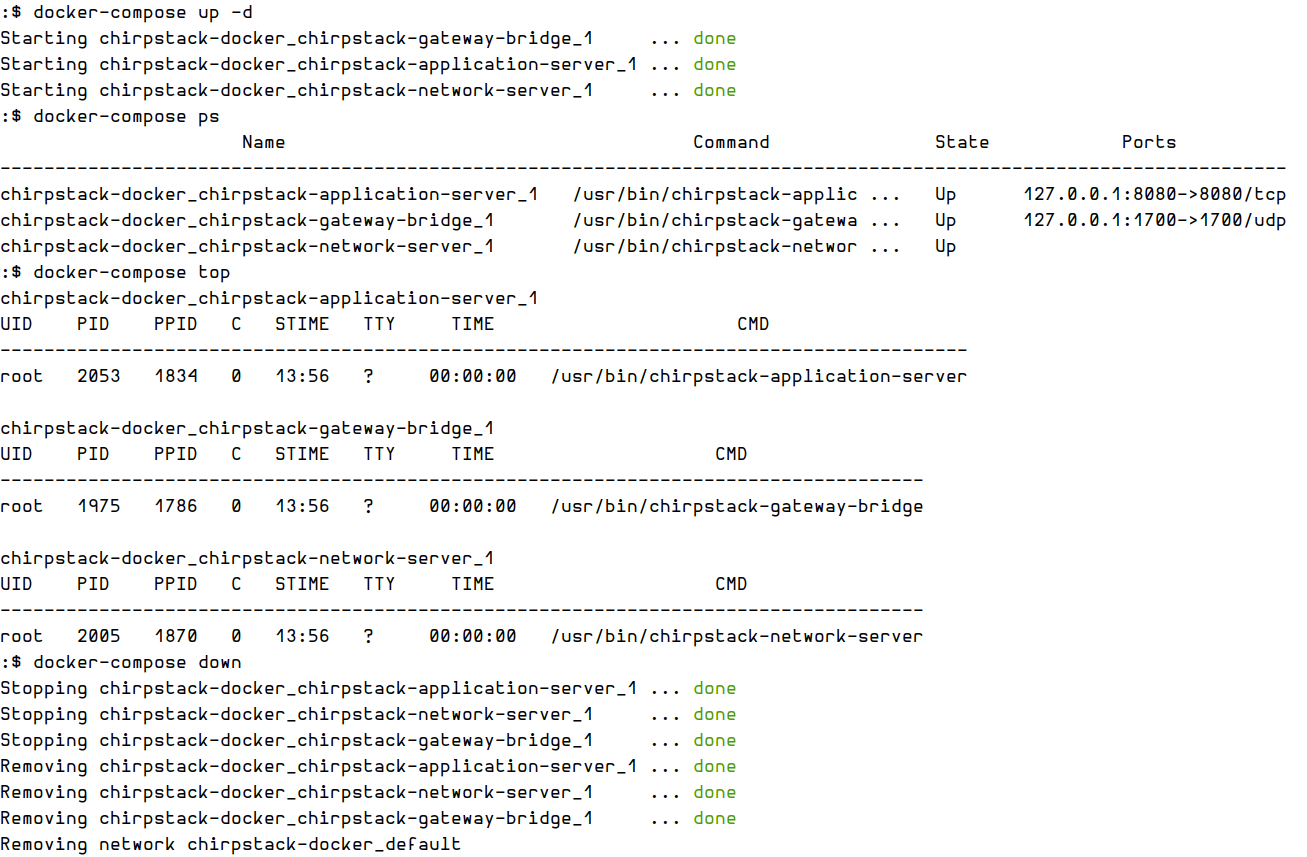


Figure 4

### Reverse Proxy Server

A Reverse Proxy server is an intermediary server which forwards incoming HTTP client requests and responses to and from appropriate application servers running behind it. A reverse proxy facilitates an additional level of abstraction and control. It assures congestion free traffic flow between HTTT clients and servers. The reverse proxy is depicted in the Figure 5.

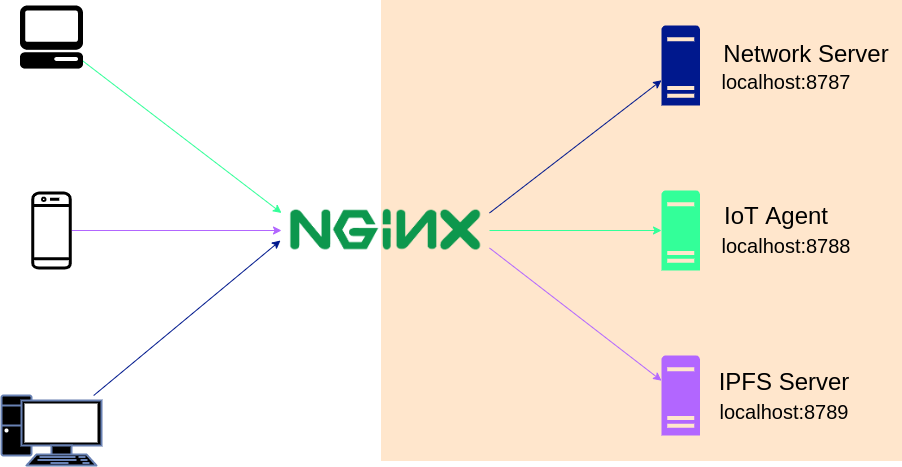


Figure 5: NGINX reverse proxy

In the IoTrust project, NGINX is utilized as a reverse proxy server. It provides load balancing, web acceleration and higher security to the IoTrust components.

Installation

NGINX can be installed in an Ubuntu machine using the prebuilt packages. It is defined in the default Ubuntu repository.

NGINX official installation guidei can be followed to install it on other kinds of operating systems.

Domain Name

Domain names are addresses of network domain or Internet Protocol resources. They are used in the IoTrust project to identify its components. Each component deployed on cloud server under the IoT project has be given domain name. For an instance *iot-agent.iotrust.com*. It is required in the configuration of nginx.

Nginx configurations

The nginx server can be configured in multiple ways to contain the information of backend application servers. A config file can be created in sites-available, sites-enabled or conf.d folders. In the IoTrust project, it is configured in *conf.d* directory. In a Ubuntu server, it is located in */etc/nginx* directory. Create a file in */etc/nginx/conf.d* folder with the name *<domain-name>.conf*. For an example if domain name is *ot-agent.iotrust.com* thenfilename would be *iot-agent.iotrust.com.conf.* It will contain configuration similar to shown below.

server {

listen 80;

server\_name <server-domain-name>;

location / {

proxy\_pass [http://127.0.0.1:<port](http://127.0.0.1:%3cport)>;

}

}

One can test the configuration with sudo nginx –t command and reload the nginx daemon with sudo nginx –s reload command. After this, you should be able to access a component in your browser with its domain name. At this point, you would realize the connection between your browser and server is not secure. it is over HTTP. The next section describes the steps to install Let's Encrypt SSL certificate for the server.

Let’s Encrypt Client

SSL certificates are required to establish a HTTPS connection between client and server. The Let’s Encrypt client can be utilized to generate SSL certificates for free. The Let’s Encrypt is a non-profit Certificate Authority that provides TLS certificates. A client software is need to generate such certificates on server automatically. On Ubuntu, it can be installed using the prebuilt packages.

sudo apt-get install certbotsudo

sudo apt-get install python-certbot-nginx

Generate SSL certificate

Certificate Generation

The SSL/TLS certificates can be generated on Ubuntu server with following command.

sudo certbot --nginx -d <server-doman-name>

For an example

sudo certbot --nginx -d iot-agent.iotrust.com

User has to answer some question with Yes/no after executing this command. If everything goes smooth, it generates the SSL certificates for the server and update the respective nginx conf file. In this example, it would update iot-agent.iotrust.com.conf file.

Certificate Renewal

A cron job can be setup to update SSL certificate automatically on an Ubuntu Server. To create a cron job open the crontab file with crontab -e command and write following content.

0 12 \* \* \* /usr/bin/certbot renew --quiet

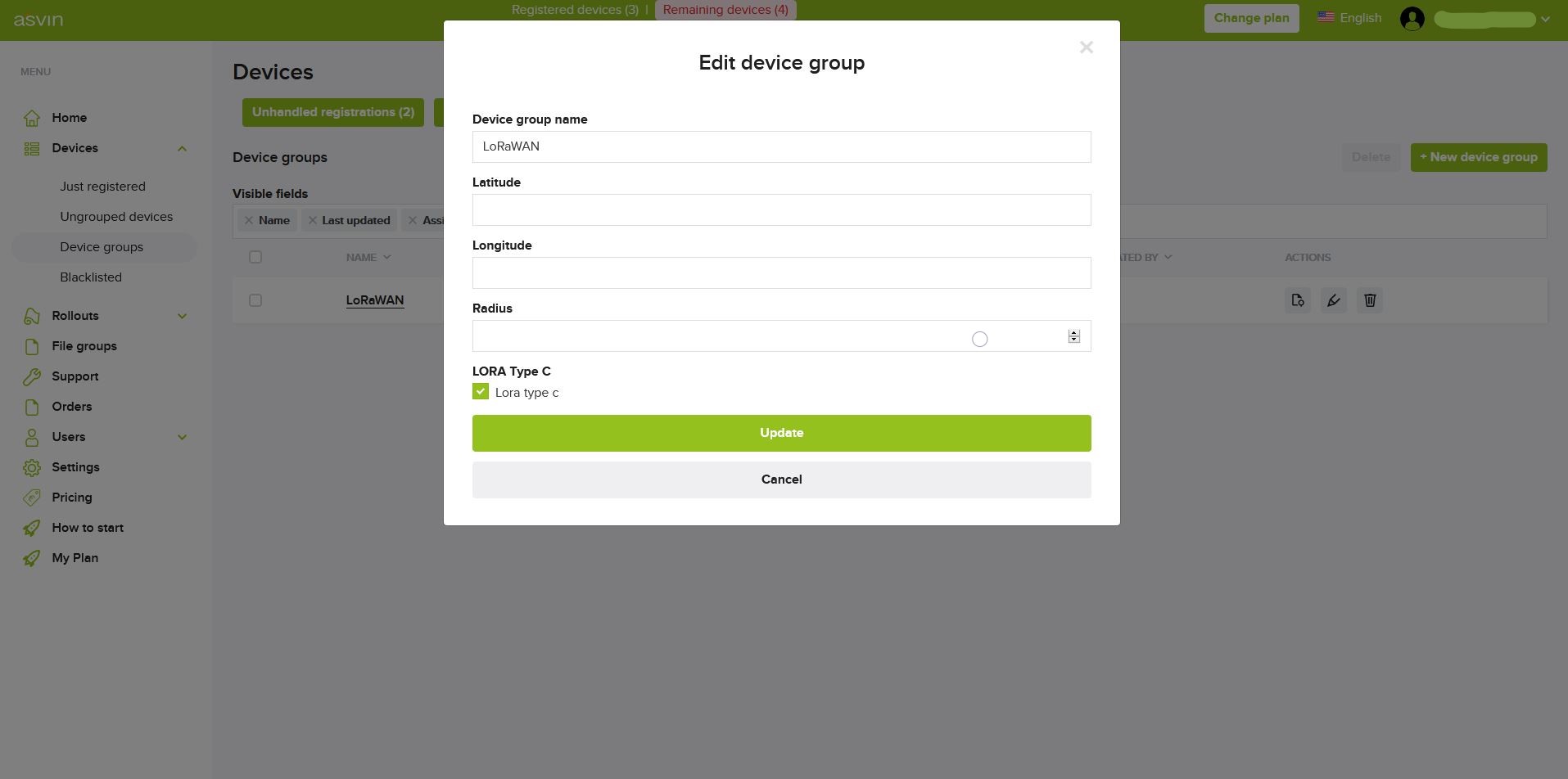
### Customer Platform

The customer platform facilitates a dashboard to manage and control IoT devices and their firmware. It acts as an abstraction layer to hide the complexities and sophistication of the Distributed Ledger technology. This abstraction is facilitated using a cluster of servers backed with database server. To cope up with gigantic number of IoT devices a load balancer is installed. The load balancer streamlines the connection from customers and IoT edge devices to the server.

The customer platform delivers following functionalities:

1. Device Management:

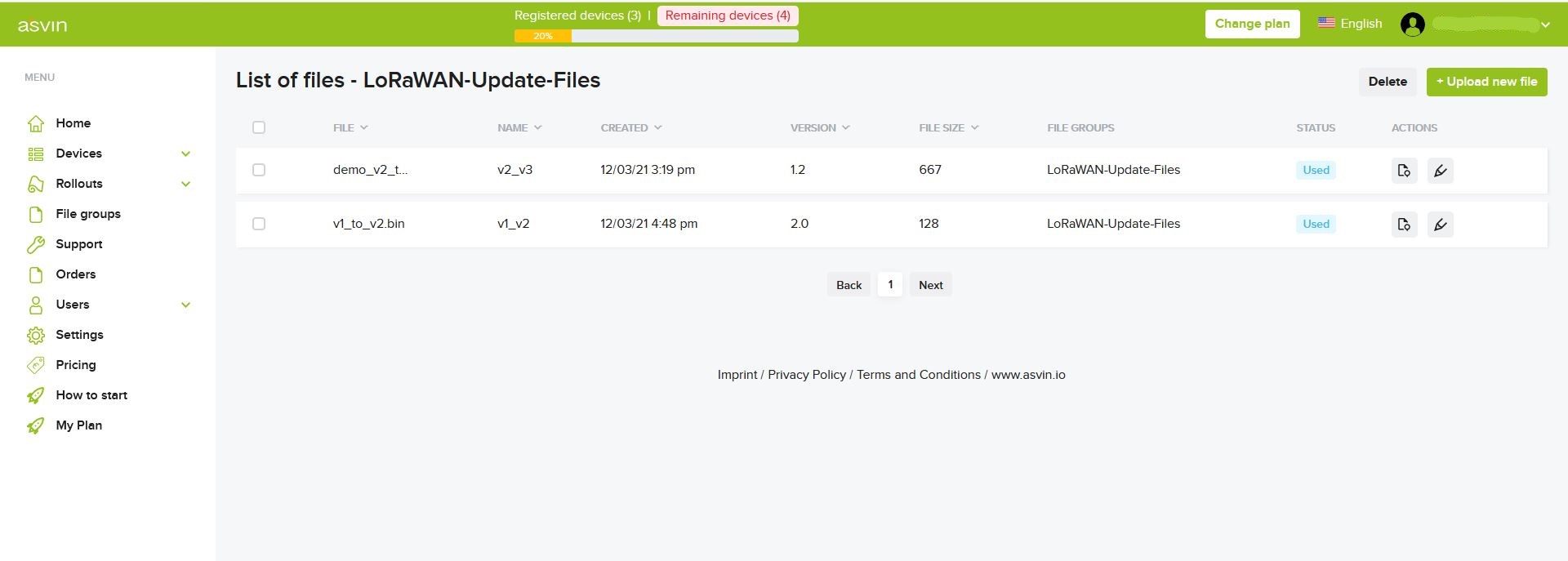
The IoT operator will be able to use the dashboard to register edge devices, group and manage all devices and device classes, and review edge device health statistics, among other features.



1. Firmware and Rollout Management:

In the frontend, the user will upload firmware update files. These new patch files are stored on the platform's IPFS server. Uploading and removing firmware to and from a file group are both possible through the portal.

The operator can schedule a rollout of the latest patch files to the device groups.



1. Update Ledger:

Interacts with Hyperledger blockchain server to update device and firmware database.

1. Update Version Control Server

It keeps version control server updated with information of latest firmware.

### Version Controller

The version controller server consists of multiple nodes which have same copy of web server and hosts identical web services. Each node in the cluster is fully functional web server and can serve a request independently. Each node has different IP address, but they are hidden from edge devices. An abstraction layer is used on top of the cluster to hide complexity. This abstraction layer makes use of round-robin DNS technique for load balancing, fault tolerance and load distribution. The server accepts DNS requests and responds to them by forwarding it to a computing machine in the cluster. A machine from the cluster is chosen in round-robin fashion.

The server performs following tasks.

1. Response to Edge Devices

In order to register on the asvin platform, the edge devices send request to version controller with the secured data. The edge devices also poll the version controller to check for new update. The server responds to edge devices with information of valid firmware and roll-out id if any update is scheduled.

1. Latest Firmware Version List

It maintains real time information of different versions of firmware available on data storage servers. The version controller has a list of available firmware on asvin.io platform for all edge devices. It keeps the list updated by interacting with Customer platform server.

# Conclusions and next steps

Outline any conclusions on the results achieved and any lessons learned for the next stage of the project.

Describe briefly the next steps in the project development and how you will build on this deliverable to complete the work.

Appendix

* E.g. mock-ups, screenshots
* References
* Etc.

1. https://www.microchip.com/wwwproducts/en/RN2483 [↑](#endnote-ref-1)
2. Alliance, L., Sornin, N., Luis, M., Eirich, T., Kramp, T., & Hersent, O. (2016). LoRaWAN TM Specification v1.0.2. *LoRaTM Alliance*. https://lora-alliance.org/resource-hub/lorawantm-specification-v102 [↑](#endnote-ref-2)
3. https://github.com/brocaar/chirpstack-docker [↑](#footnote-ref-1)
4. Minaburo, A., Toutain, L., Gomez, C., & Barthel, D. (2020). *SCHC: Generic Framework for Static Context Header Compression and Fragmentation* (Request for Comments, Issue 8724). RFC Editor. https://doi.org/10.17487/RFC8724 [↑](#endnote-ref-3)
5. Gimenez, O., & Petrov, I. (2021). *Static Context Header Compression (SCHC) over LoRaWAN* (Issue draft-ietf-lpwan-schc-over-lorawan-14). Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-schc-over-lorawan-14 [↑](#endnote-ref-4)