
Asvin Documentation

Release 0.0.1

Asvin

May 02, 2022

CONTENTS

1	Introduction	3
2	Key Concepts	5
3	Getting Started	15
4	Tutorials	21
5	Security Principles	35
6	Architecture Reference	37
7	REST API	39
8	Contribution Welcomed	57
9	Glossary	69
10	Releases	71
	HTTP Routing Table	73

Note: Please make sure you are looking at the documentation that matches the version of the software you are using. See the version label at the top of the navigation panel on the left. You can change it using selector at the bottom of that navigation panel.



An Enterprise-grade Platform-as-a-Service (PasS) to manage, distribute and monitor firmware updates, and produce a real-time threat landscape for registered IoT devices.

INTRODUCTION

There was a time when making coffee required a broad scope of tasks. First you would go to the market to buy freshly roasted, aromatic coffee beans, then grind them and while the coffee was brewing you would retrieve the newspaper from the front door and start your day while waiting for the coffee to ‘catch up’. Fast-forward to 2021 and our “smart” coffee machine monitors our armband to determine if we are awake so that by the time we reach the kitchen our artisanal coffee beans have been freshly roasted, ground and brewed to our exact preferences whilst we make our way down to the coveted first cup. Additionally, that coffee maker can now monitor how much coffee remains in the hopper, can order our favorite beans from a preferred seller, and automatically makes the payment using credit card details saved on the machine.

We are surrounded by many smart devices such as our coffee machine. Equipped with wireless communication protocols such as WiFi, Bluetooth, NFC, LoRaWAN, etc. these smart devices talk to one another and are connected to the internet and are referred to as Internet for Things (IoT) devices. IoT devices are now ubiquitous to our lives, whether found in our homes, cities, workplaces, or in hospitals. We live in a time where smart devices communicate and interact with us more often than other human beings. By 2030, the number of IoT devices and IoT market size are projected to reach 125 billion and 1.3 trillion USD respectively.



IoT devices simplify processes and make our lives more convenient, on the other hand, these devices are vulnerable to cyber-security threats. For instance, if someone hacks our benign-looking smart coffee machine it could reveal our WLAN credentials, credit card details, sleeping patterns and much more personal data but it can also be used as a bot

to launch cyber security attacks on other digital resources. By example, Mirai-2016 turned over six million network connected devices to a botnet to launch DDoS attacks. These types of attacks have been outpacing the exponential growth of IoT devices for a myriad of reasons including using default or weak passwords, small encryption keys, or failing to download security patches and softwares updates.

One of the major security concerns is that IoT devices do not have built-in Over-the-Air (OTA) firmware update mechanisms. IoT devices are manufactured and managed with a set and forget policy, meaning IoT devices are configured at the beginning of their lifecycle and then left alone to defend against security attacks. Technologies are evolving so fast that it is paramount to have a well-defined, secure process to continuously assess the threat landscape of IoT devices and update the firmware regularly with latest security patches.

In the last couple of years, there have been major developments in the area of update platforms for IoT devices. Some solutions, such as Eclipse hawkBit, Mbed, and Mender, address the existing problem but their solutions are limited to a specific microcontroller architecture, software stacks, and industries are designed and developed as centralized solutions which are prone to single point failure and do not scale very well. The following sections describe how the asvin platform differentiates itself from other IoT update solutions.

1.1 Asvin Platform

asvin platform is an enterprise-grade, cloud platform designed with Distributed Ledger Technology (DLT) to cascade security patches and generate a chain of trust for IoT devices. The asvin platform is powered by **distributed** and **decentralized** technologies and utilizes Interplanetary Filesystem (IPFS) to store and manage the firmware files of IoT devices. All firmware files and patches are distributed across multiple IPFS nodes while the respective devices' critical metadata information is saved on an immutable distributed ledger (DL) as is each event taking place on the asvin platform. These technologies increase the fault tolerance of the platform making it both more accessible as well as offering greater resiliency.

It is one thing to have a working prototype of an IoT product and another to administer and update millions of smart devices. The most desirable feature of any IoT solution is scalability, and asvin delivers - effectively and efficiently.

The asvin platform has a highly **customizable** and **modular** architecture and has been designed and developed to support IoT applications in diverse industries with its pluggable modules optimized according to the specific use case.

Unlike other FUOTA solutions the asvin platform is a **universal** solution, it is not restrictive in nature for any hardware and software stack. asvin enables the innovation and versatility of any IoT application built with Atmel AVR to ARM Cortex M7 ranging in applications from agriculture to financial services.

KEY CONCEPTS

2.1 Overview

The asvin platform is underpinned by multiple decentralized technologies for distributing security patches for IoT devices with a higher degree of security, scalability, confidentiality and resiliency and supports the full range of IoT devices and their applications. The platform hides the complexities and intricacies of IoT devices to provide firmware updates securely while identifying the threat landscape and protecting the firmware and software within a secure chain of trust.

2.1.1 What is Firmware Update?

IoT device

An IoT device is a piece of hardware typically embedded with microcontrollers, sensors, communication modules, input/output peripherals, and software for the objective of collecting useful data from multiple processes and exchanging it with other connected devices over the internet. IoT devices are employed virtually everywhere - in smart homes, manufacturing plants, cities, hospitals, transporting goods, workplaces, and out in space.

Firmware

Embedded software running on an IoT device is called firmware. IoT devices have a very small footprint and, as such, cannot run full-fledged operating systems on them. IoT devices based on Arduino, ESP, TTGO, WeMoS development boards run as a standalone program. Deterministic multi-threaded programs can be run on more powerful boards like Nucleo, NuMaker, PSoC using embedded RTOS such as Mbed, FreeRTOS.

Firmware Update

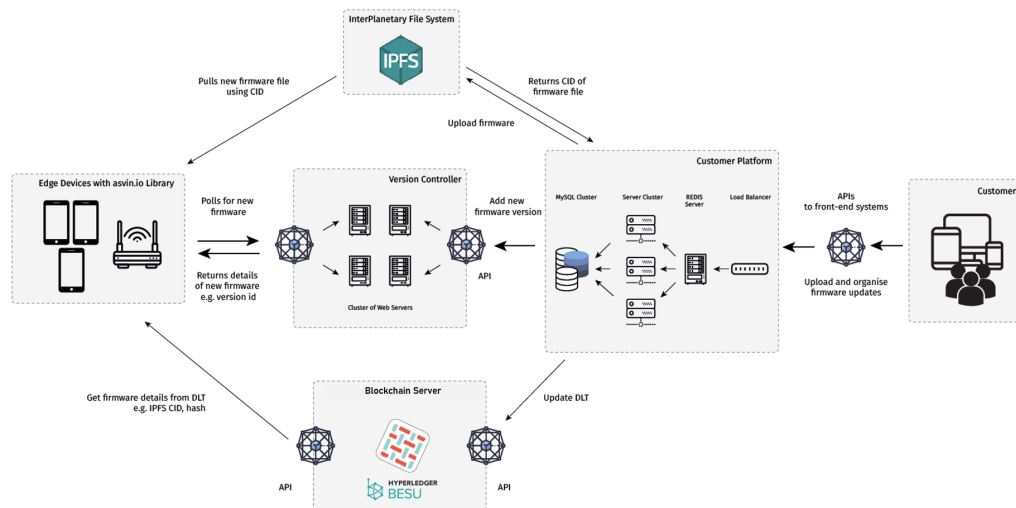
A firmware update is the process of somehow changing the software running on IoT devices, this can be done by updating the complete firmware or applying delta changes.

Why is Firmware Update Necessary?

We live in a world where technology is both a boon and curse. On one hand, technology simplifies our processes while the exploitation of its vulnerabilities by BlackHats and state-sponsored actors result in huge financial losses and impact the quality of life for millions. A proactive cybersecurity policy which includes regular updates and security patches and the ability to provide isolation of vulnerable IoT devices is critical.

2.2 Asvin Model

In this section we will take a brief look at asvin's architecture and briefly explain its components. Some relevant concepts and keywords are also mentioned below.



2.2.1 Customer Platform

asvin provides a user-friendly customer platform to abstract the complexities of the background processes to register devices, setup & schedule firmware rollouts, define group actions on registered devices, and to blacklist devices. The customer platform also displays information related to firmware updates and provides the current version of edge devices.

2.2.2 Version controller

The version controller maintains updated information of firmware available for a particular edge device or device group on the asvin.io platform. It is one of the core components of the asvin architecture.

2.2.3 Interplanetary FileSystem

Interplanetary FileSystem or IPFS is a distributed peer-to-peer hypermedia protocol designed for storing and sharing data in a distributed file system. IPFS serves as one of the core components of the asvin architecture. asvin uses IPFS protocol to store firmware, updates and security patches. Check out the detailed notes on IPFS [here](#).

Content Identifier

Data exchanges on decentralised platforms like [IPFS](#) depends upon content-based addressing rather than local addressing to securely locate and identify data. A Content Identifier (CID) is a self-describing content address identifier which is a cryptographic hash, typically SHA-256, which is 32 bytes. The CID for a binary image file stored on an IPFS network might be **QmcRD4wkPPi6dig81r5sLdrtD1gDCL4zgpEj9CfuRrGbZF**.

2.2.4 Blockchain

The asvin platform uses blockchain technology to store all of the transactions executed on the asvin.io network including: device registration, firmware upload, device update, firmware update, user registration, etc. All these transactions are connected with hashes and stored in blocks which are again linked to a secured hash. For the asvin platform we use the [Hyperledger Fabric](#) and [Hyperledger besu](#) blockchains.

Distributed Ledger

Distributed Ledger Technology (DLT) is a digital system for recording the transaction of assets in which the details of specific transactions are recorded in multiple places. Unlike traditional databases, distributed ledgers have no central data storage or administration functionality. The above mentioned blockchain is a type of DLT.

Smart Contracts

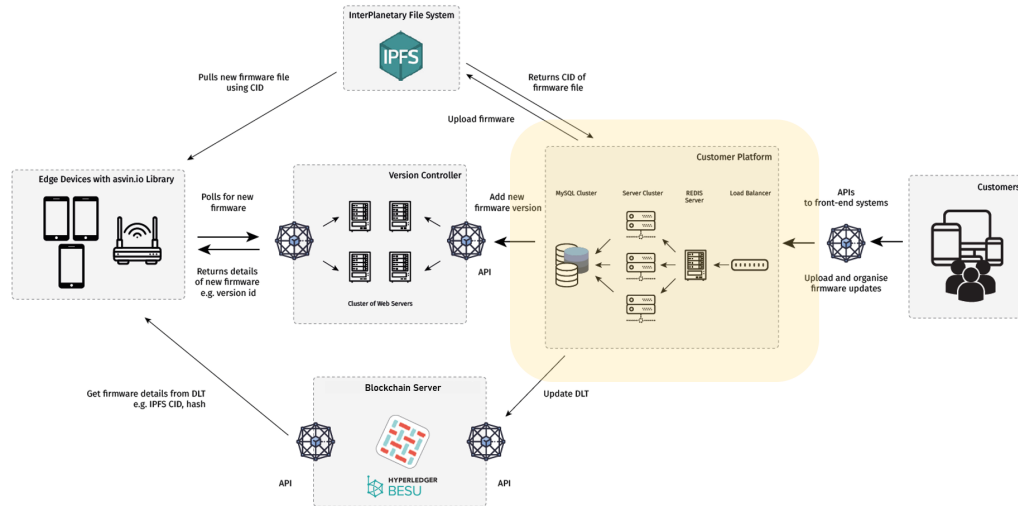
A smart contract is a computer program or a digital transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement. The objective of a smart contract is to reduce the need of trusted intermediators, arbitration, enforcement costs, fraud losses, as well as the risk mitigation of malicious attacks and accidental exceptions.

2.2.5 Edge Devices

Edge devices are end-points in the asvin.io architecture and in an IoT network which control, manage and solve a specific physical task: For instance, edge devices turn on a smart washing machine in a house, monitoring temperature and humidity in a chemical plant or an air quality sensor installed in a city. These devices have microcontrollers and sensors at their core and with their small footprints these edge devices are easy to manage in remote areas under extreme environmental conditions. Examples of edge devices in the Industrial Internet of Things (IIoT) include process monitoring sensors, smart meters, Lora nodes, smoke detectors, etc.

2.3 Customer Platform

In this section we briefly explain the **Customer Platform** component of asvin architecture.



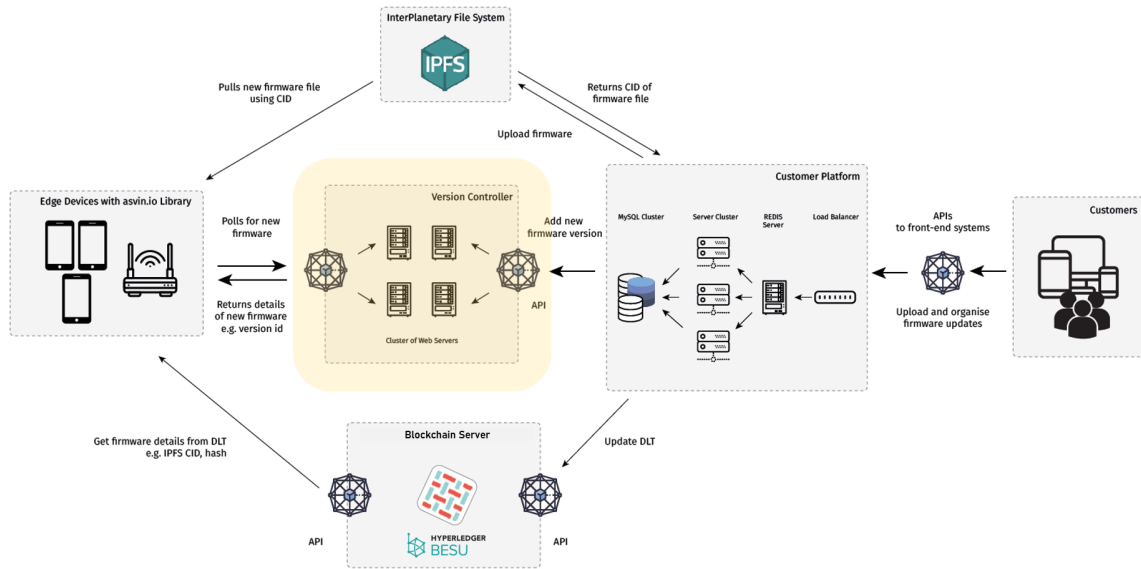
At asvin, we believe in a simple and user-friendly solutions. The asvin.io customer platform provides an abstraction layer to hide the complexities and sophistication of the Distributed Ledger Technology. This abstraction is facilitated using a cluster of servers backed with a database server. A load balancer ensures that the volume of IoT devices being maintained on the asvin.io platform encounter no latency in the connection between the customer's network and their IoT edge devices to the asvin.io server.

Beehive - The customer Platform delivers following functions:

1. **Service Dashboard** It serves as a portal for the IoT device operators to directly interact and control the update and patch status of their edge devices. The portal drives the functionalities to upload firmware, delete firmware, manage edge devices, check health statistics of edge devices etc.
2. **Upload Firmware** Uploads firmware provided by customer to IPFS Server.
3. **Update Ledger** Interacts with Hyperledger blockchain server to update firmware database.
4. **Update Version Control Server** It keeps version control server updated with information of latest firmware.

2.4 Version Controller

In this section we will take a brief look at the **Version Controller** component of asvin architecture. Version Controller (VC) maintains updated information of firmware available for a particular edge device or device group on the asvin.io platform.



To manage billions of IoT edge devices asvin.io employs a distributed service cluster infrastructure. For edge devices this complexity is invisible, and they interact with the cluster as they would do with a single machine. The version controller server consists of multiple nodes which have copy of web server and host identical web services. Each node in the cluster is a fully functional web server with a unique IP address and can service any request independently but they are not visible to the edge devices. An abstraction layer consisting of a round-robin DNS2 technique for load balancing, fault tolerance and load distribution is used on top of the cluster to hide this complexity. The server accepts DNS requests and responds by forwarding them on to a computing machine in the cluster. A machine from the cluster is then chosen in round-robin fashion.

The key features of the version controller server are following.

High-availability

The two layers architecture of the version controller server provides defense against single point failure. In the event a node in the cluster malfunctions the asvin.io framework remains operational.

Scalability

The cluster is highly scalable and stable making it easy to install a new node in the cluster to enhance the performance of the VC server.

Efficiency

The VC provides resilience to the asvin.io network and distributes the workload across multiple web servers in the cluster improving network performance, reducing latency and collisions during periods of high demand. The Version controller provides resilience to the asvin.io network.

The server performs following tasks.

1. Response to Edge Devices

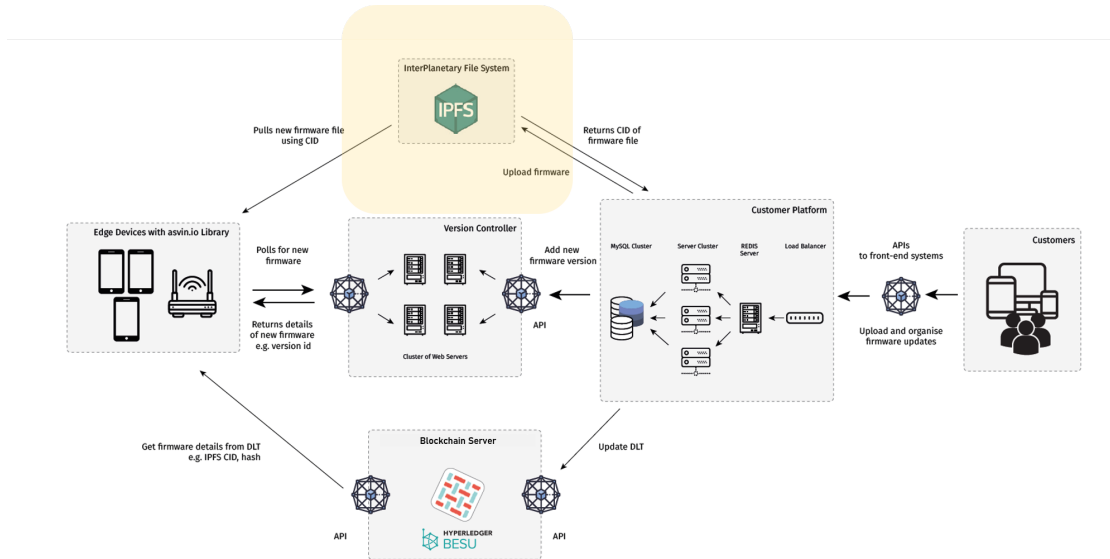
The edge devices poll the version controller to check for new update and the server responds to edge device with information of a valid firmware.

2. Latest Firmware Version List

The Version Controller maintains real time information of different versions of firmware available on data storage servers. It also 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.

2.5 Interplanetary FileSystem

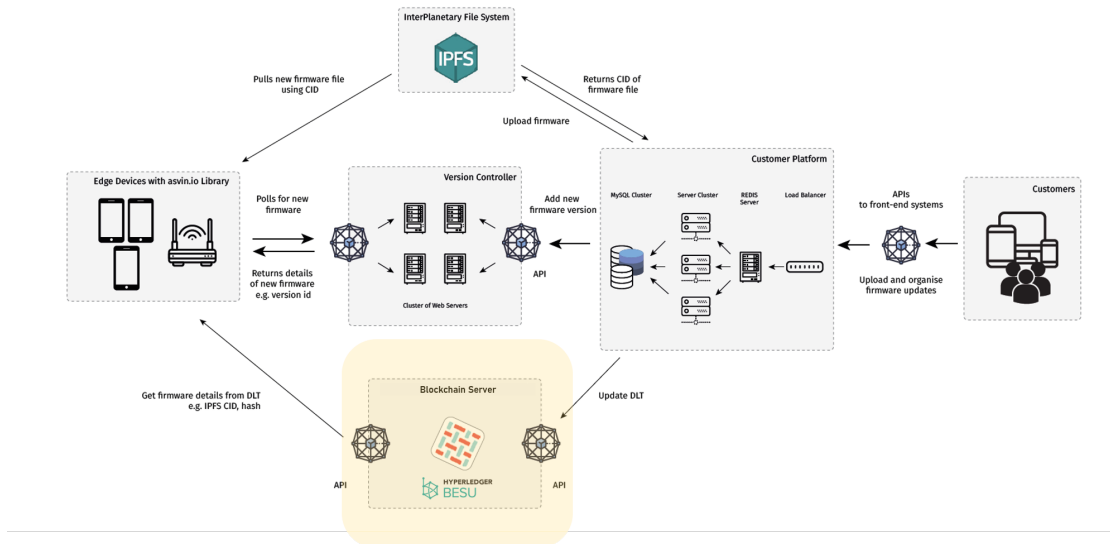
In this section we will see take a brief look at the **Interplanetary FileSystem (IPFS)** component of asvin architecture and explain it briefly.



asvin.io uses Interplanetary File System protocol to store firmware and patches. IPFS is a content-addressable peer to peer method for storing and sharing hypermedia in a distributed file system while eliminating duplicate files and redundancy. It uses a network infrastructure that enables asvin.io to store unalterable firmware data. When a firmware file is stored on the network a hash is generated based on content of the firmware and stored on blockchain network. Later on, the same hash is used by edge devices to pull the firmware from data storage. This forms a generalized Merkle directed acyclic graph(DAG). Each node of Merkle DAG is connected with a secured Hash. When a node is added in the DAG, its hash is computed based on hash of its local content and hashes of its children's name instead of their content. Once it is created, it is impossible to alter a node in the network. IPFS has no single point of failure. asvin.io provides a distributed content delivery system which provides an extra layer of security and prevents DDoS attacks. asvin.io's SDK, which is run on edge devices, enables the functionality to interact with data storage servers. Once an edge device gets information from the version controller regarding the newly available firmware, it uses this information to download appropriate firmware from the distributed CDN.

2.6 Blockchain

In this section we will take a brief look at the **Blockchain** component of asvin architecture and explain it briefly.



Blockchain is a type of Distributed Ledger Technology (DLT) which contains a growing list of records known as **blocks**. Each block contains the **cryptographic hash** of the previous block. By design, blockchains are resistant to modification of the data stored on them.

In the asvin architecture, we have built the blockchain network based on [Hyperledger Fabric](#) and [Hyperledger besu](#). Customers can choose to use either of the two blockchains for their deployments. These blockchain networks provide a ledger that stores all transactions executed on the asvin.io network, e.g: device register, firmware upload, device update, firmware update, and user registration. Each of these transactions are paired with unique hashes and are stored in blocks which are again linked with a secured hash. This process provides security and immutability to the ledger. The asvin.io platform is designed for unlimited scalability with a cluster of servers which runs Fabric & Besu networks to administer the requirements to maintain functionality to millions of IoT devices.

2.6.1 Hyperledger Fabric

The pluggable modules of the Fabric network allow maximum flexibility to asvin.io infrastructure, enabling asvin.io to develop bespoke solutions for its IoT customers. The cluster is developed using [Docker swarm](#).

The fabric has following modules and services:

1. **Peer:** an entity in the Fabric network which receives request from applications, runs a chaincode, validates transactions and maintains ledger.
2. **Orderer:** This service orders all the transactions happening in the Fabric network and forward them to peers to be validated
3. **MSP:** provides digital identities to every member of the Fabric network.
4. **Storage:** Fabric network uses CouchDB to store the current state of ledger data.

2.6.2 Hyperledger Besu

Hyperledger Besu is an open source Ethereum client. It can be run on the Ethereum public network or on private permissioned networks.

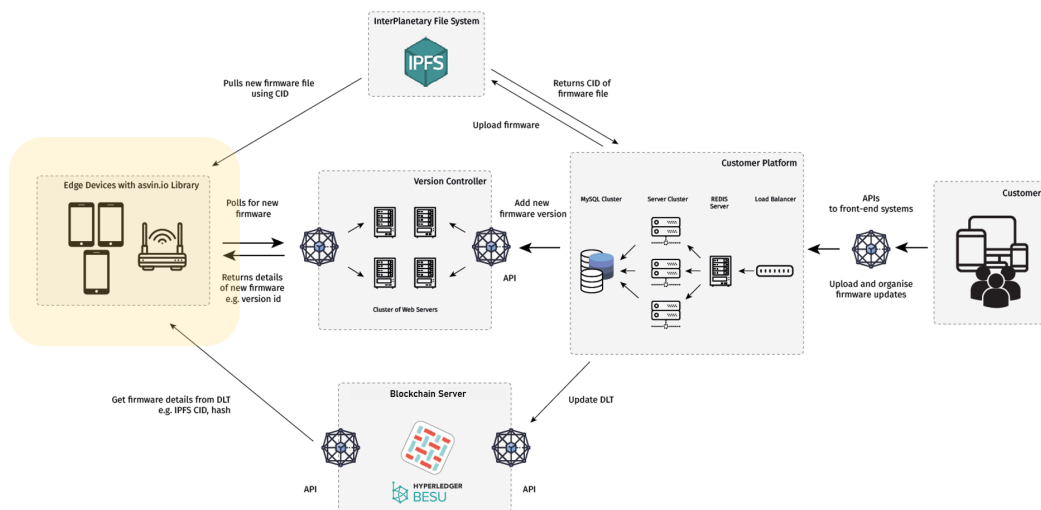
Hyperledger Besu's features include:

1. The Ethereum Virtual Machine (EVM) is the Turing complete virtual machine that allows the deployment and execution of smart contracts via transactions within an Ethereum blockchain.
2. Consensus Algorithms are involved in transaction validation, block validation, and block production (i.e., mining in Proof of Work).
3. RocksDB key-value database to persist chain data locally is used for storage.
4. P2P networking via Ethereum's devp2p network protocols for inter-client communication and an additional sub-protocol for IBFT2.
5. User-facing mainnet Ethereum and EEA JSON-RPC APIs over HTTP and WebSocket protocols as well as a GraphQL API.
6. Node performance is monitored using Prometheus or the debug_metrics JSON-RPC API method. Network Performance is monitored with Alethio tools such as Block Explorer and EthStats Network Monitor.
7. Privacy in Hyperledger Besu refers to the ability to keep transactions private between the involved parties. Other parties cannot access the transaction content, sending party, or list of participating parties. Besu uses a Private Transaction Manager to implement privacy.
8. A permissioned network allows only specified nodes and accounts to participate by enabling node permissioning and/or account permissioning on the network.

The operating system level virtualization is achieved on blockchain server using docker. Each service in the network runs in a separate docker container and these containers are hosted on multiple machines on the cluster. The communication among containers is achieved using docker swarm. The whole blockchain network is developed and run using docker swarm technology.

2.7 Edge Devices

Following is a brief explanation of the **Edge Devices** component of asvin architecture.



The edge devices are end points in the asvin.io architecture and, in an IoT network, are tasked to control, manage and solve a physical task. Usually, many edge devices shall have a small footprint and must be easy to manage in remote areas under often extreme environmental conditions. For example: industrial process monitoring sensors, smart meters, Lora node etc. asvin provides libraries for a wide range of edge devices to facilitate the technical requirements necessary to interact with the asvin.io infrastructure securely and efficiently. The asvin libraries act as an interface between the edge devices and the asvin.io platform. The asvin.io SDK makes it easy to use APIs to interact with the asvin.io update infrastructure. The SDKs are developer friendly and allow for new IoT devices to connect with asvin.io infrastructure quickly to take direct advantage of a ready-to-run, update distribution network. The key features of asvin SDK is that It is a generic solution and not limited to any specific hardware platform e.g. Arduino, ESP, STM, Arm Cortex-M3, M4 etc. and software protocols.

The asvin.io libraries installed on edge device provides following functionalities:

1. Device Update Management

- Securely register devices on asvin.io platform

2. Check for Firmware Updates

- Regularly poll for updates in configurable time intervals

3. Download and Install Firmware

- Download and store updated versions of firmware
- Install the firmware based on traffic and availability of edge device

4. Collect Health Statistics

- Send timely health updates of edge device on asvin.io platform e.g. firmware version in use
- asvin.io platform generates insights from the data which can be used to extend life cycle of IoT devices and for preventive maintenance.

GETTING STARTED

3.1 Prerequisites

Before your embark on a journey to explore asvin getting started and tutorials sections, it is necessary to install the following tools on your machine.

3.1.1 Postman

Download and install the latest version of [postman](#). The postman will be used to make http requests to asvin servers. We have a [postman collection](#) of asvin APIs. You could create a test workspace and import the collection. It is required for the getting started documentation. The following video shows the procedure.

The Postman is enough to execute all the steps listed in the getting started section. If you want to try any of the tutorials then install the tools mentioned below.

3.1.2 Git

Download and install the latest version of [git](#).

3.1.3 cURL

Download and install the latest version of [cURL](#) tool.

3.2 Account Registration

asvin Beehive provides an interactive dashboard to visualize and manages devices, firmware, and rollouts. It is necessary to create an account on the asvin Beehive to use asvin services. The process is quite simple. All you need is a valid email address. Go to the register page, input your email address, select couple of check boxes and click on Register now. It is shown below.

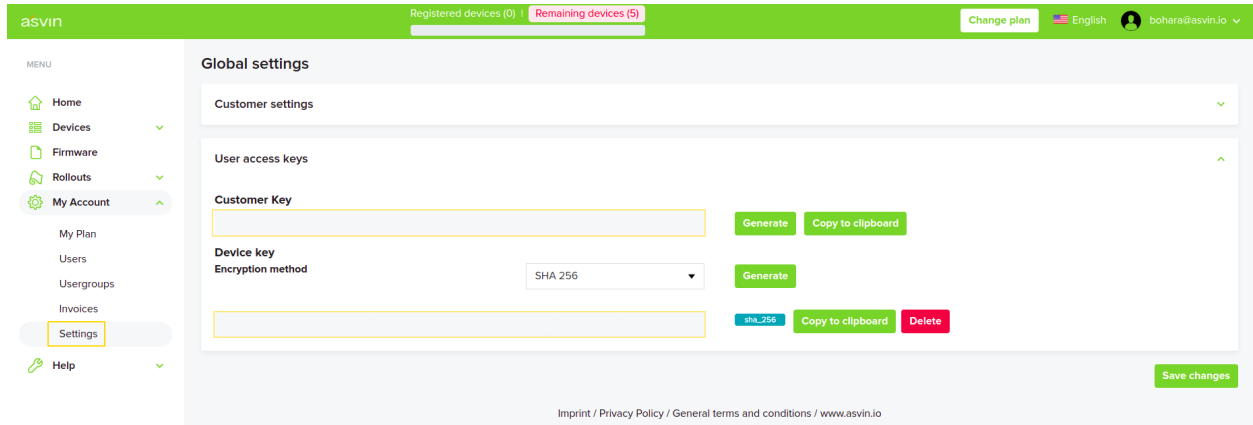
When everything goes well, you will receive an email for setting up password for your account. Click on Account Activation, it will take you to a web page. There you can create your password. You need to take care of asvin's password guidelines. Your password should contain

- at least 10 characters
- at least one big letter and one small letter
- at least one number

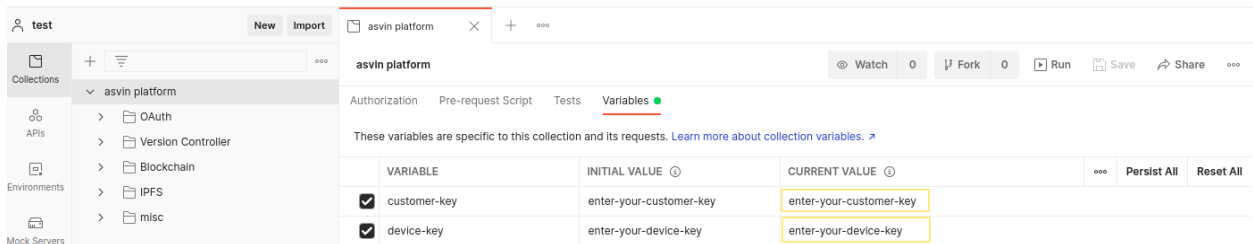
- at least one special character

Once you enter valid password and click on Create a password button, it will land you to the home page of the dashboard. The process is shown below.

asvin uses access keys to manage unauthorized access to the api end-points. Theses access keys are customer key and device key. The keys are generated automatically when an account is created for the first time. You can access the keys in My Account -> Settings section as shown in the picture below.



The keys will be used to get Jason Web Token (JWT) from OAuth server. Copy the access keys and set the respective collection variables in the Postman as shown in the image below.



3.3 Device Registration

At this stage, you have created an account on the asvin Beehive, received the customer and device keys, and imported asvin api collection on postman application. It is time to register your first device. Go to you postman collection and open Register Device POST request located in the Version Controller folder. In the request, enter desired device name, current firmware version, and valid MAC address. Next, all you have to do is click on big blue button named Send. You will received device inserted message if the request is accepted. You can go the dashboard and check it in Devices menu. It will appear under Lobby sub-menu. The process is shown in video below.

Next, you need to create a device group and add the device in it to complete the registration. You will not be able to create rollout for the device until you add it in a device group. Click on New device group button under Devices->Grouped menu to add a device group. Once you have a device group, go to Lobby, select the device and click on Group to add it in a desired device group. The procedure to add a device group and grouping a device is shown below.

At the end of this task, you will have a device registered, grouped in a device group and stored on distributed ledger.

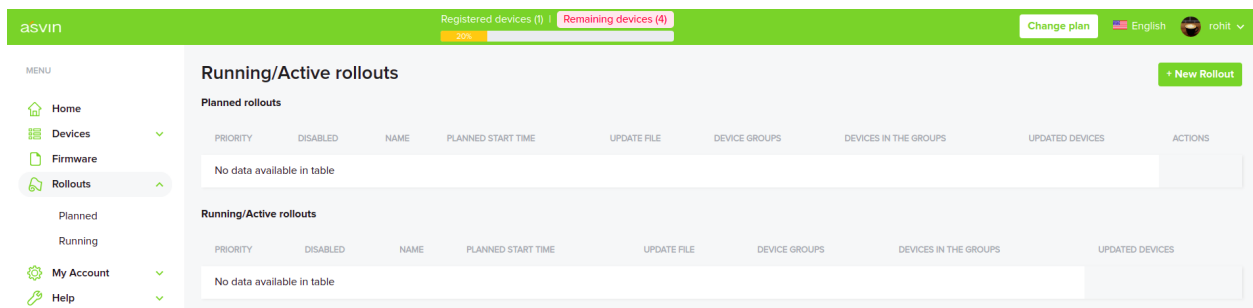
3.4 Firmware Upload

If you have completed the device registration process, then you must have a device in your account. Next, you need a firmware for the device. This can be achieved in two steps. Firstly, create a firmware group by clicking on create new file group button under Firmware menu. When you have a firmware group with a desired name then click on show button or on the name. It will show you list of firmware in the group. Obviously, it is empty now. Secondly, create a simple firmware and click on Upload new file button to upload it. The steps are shown in the video below.

You will have a firmware group and firmware in it at completion of the task. The firmware metadata will be stored on the distributed ledger and firmware will be saved on the private distributed network powered by IPFS protocol.

3.5 Rollout Creation

Once you have a you registered a device and upload a firmware successfully, it is time to create and start a rollout. You can access all your rollouts under Rollouts menu. It contains list of planned and running rollouts. Initially, the list will be empty as shown in the figure below.

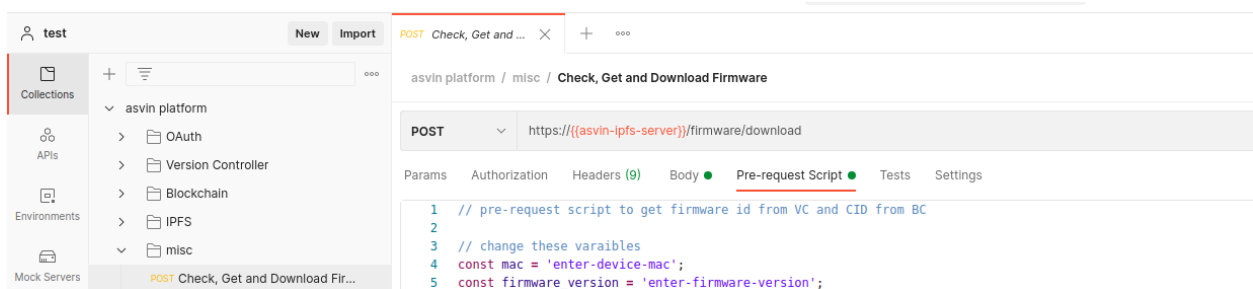


Click on New Rollout button to create a rollout. It will open a dialog box where you need to select relevant device group, firmware file and start time. If everything goes well, you will have a rollout in the Planned list. You can click on Start button to make the rollout active. The procedure is shown below in the video.

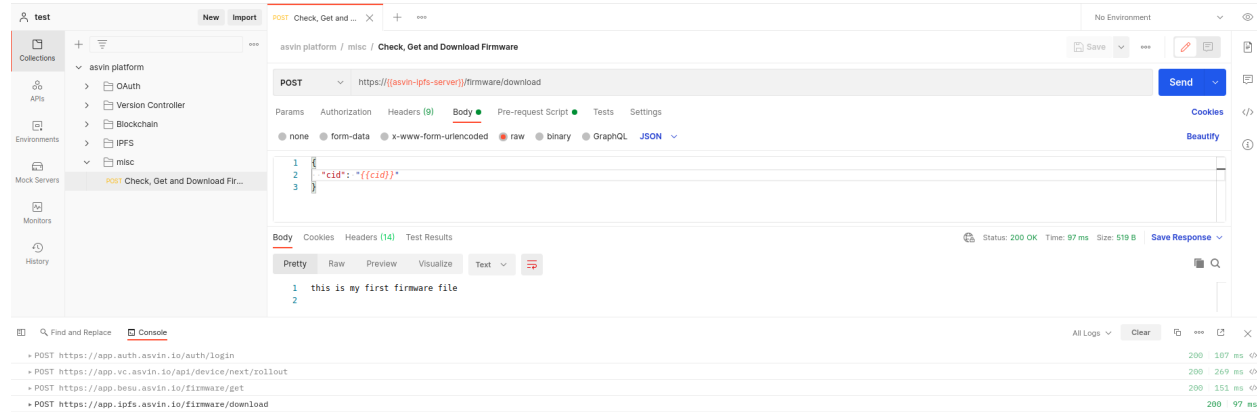
An active rollout is the result of the task. The devices can use it to get firmware details which are utilized to download firmware from IPFS server.

3.6 Firmware Download

After you have an active rollout in your account, you can download the associated firmware. We have created a request in misc folder in the asvin postman collection. The request is named Check, Get and Download Firmware. Firstly, open the request and go to Pre-request Script tab. Secondly, enter valid MAC address and firmware version as shown in the image below. Lastly, click on Send button to send the request.



The one click firmware download request is more complicated than it seems. Under the hood, it contains 4 more requests. First of all, it sends login request to OAuth server to get JWT. The JWT is used in all subsequent requests for authorization. After that it checks for an active rollout. If a rollout exist the Version Controller returns firmware and rollout ids. The firmware id is utilized to get firmware CID from the Blockchain Server. Finally, it sends download firmware request with CID to the IPFS server. You can see the details of all requests in the Postman Console. It is depicted in the image below.



You can also make the requests manually one by one. It is illustrated in the video below.

You will have your first firmware downloaded at the completion of the task.

3.7 Rollout Success

Now that you have downloaded a firmware, you can install it on the device. Once the installation is completed and device works as expected, you should send rollout success request to the Version Controller. A rollout is associated with a device group. Therefore, an active rollout shows the number of devices in the groups and updated devices as shown in the picture below.

Running/Active rollouts							
PRIORITY ^	DISABLED	NAME ^	PLANNED START TIME ^	UPDATE FILE	DEVICE GROUPS	DEVICES IN THE GROUPS	UPDATED DEVICES
Immediately	No	my-first-rollout	21/05/21 8:27 pm	my-first-firmware	my-first-dg	1	0

In the beginning, an active rollout has 0 updated devices. It gets updated when a rollout success request is sent with valid parameters. You can send the request using postman collection. It is in the Version Controller folder with the name Rollout Success. The request requires 4 parameters in the body. They are device mac address, firmware version and rollout ID. You must have received rollout ID in response to the next rollout request. The response looks similar to the following.

```
{
  "rollout_id": "63",
  "rollout_name": "my-first-rollout",
  "priority": "1",
  "start_date": "2021-05-21 18:27:31",
  "version": "1.0",
  "firmware_id": "57"
}
```

The rollout success request is shown in the video below.

Once the rollout success request is completed, the updated device count is changed in the rollout as shown in the image below.

Running/Active rollouts							
PRIORITY ^	DISABLED	NAME v	PLANNED START TIME v	UPDATE FILE	DEVICE GROUPS	DEVICES IN THE GROUPS	UPDATED DEVICES
Immediately	No	my-first-rollout	21/05/21 8:27 pm	my-first-firmware	my-first-dg	1	1

Before we start, it is required that you install the *Prerequisites* on the machine on which you will execute the instructions given in the getting started documentation. Once you are done with the prerequisites, you are well equipped to work with and experience asvin platform. Next, follow the steps given below.

- *Create and activate your account* on *Customer Platform* and get global access keys
- *Register* your first device and add it in a device group
- *Make a file group* and add your first firmware in it
- *Plan a rollout* for your device with the first firmware
- *Check next rollout, get CID and download firmware* from VC, BC and IPFS server respectively
- Send *Rollout success* request

TUTORIALS

This section includes tutorials for the usage of the Asvin platform.

4.1 Over the air updates with ESP8266 (nodemcu) boards

In this tutorial we will see the demonstration of OTA updates using the Asvin IoT platform and nodemcu board based on the ESP8266 chipset.



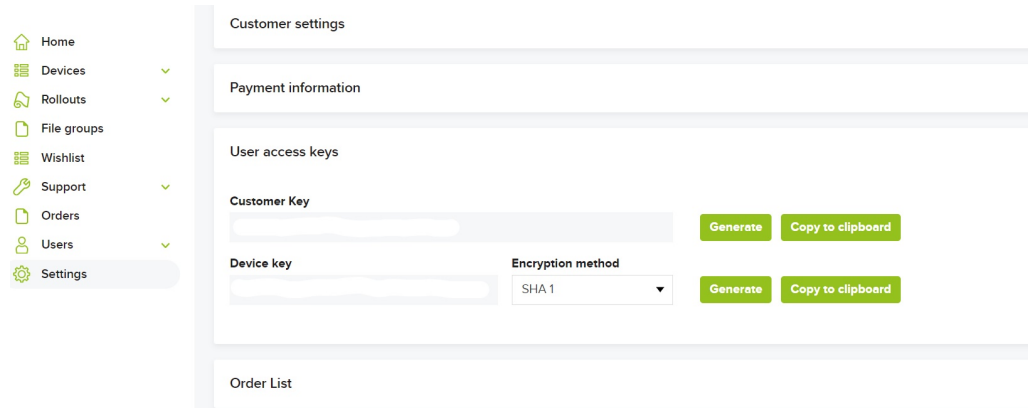
4.1.1 Requirements

1. nodemcu esp8266 board
2. Micro USB cable
3. asvin platform subscription
4. PlatformIO VScode extension

4.1.2 Getting started

To get started head to [asvin's Github repository](#) and clone it. The code is written under PlatformIO. Import the project using VScode's PlatformIO extension.

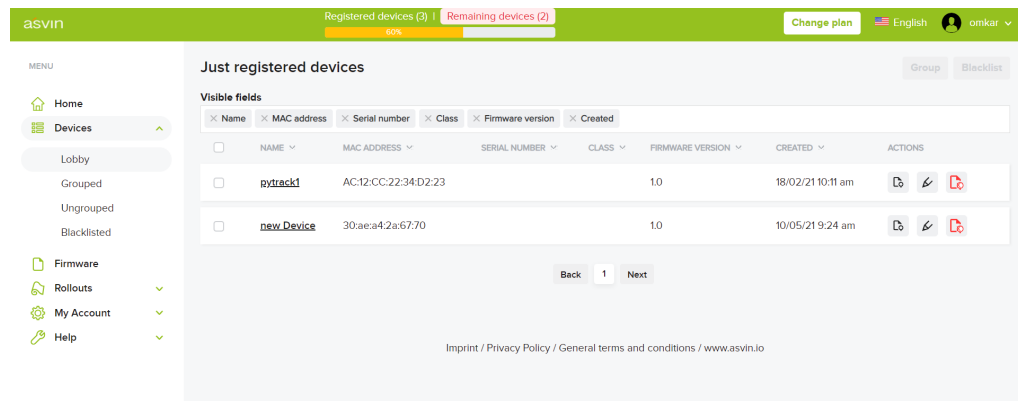
- To proceed further you will need to edit few of the parameters in the code.
 - Open the main.cpp file in the editor and add credentials for your device



- Device & Customer keys: This will be found on the asvin platform under the settings tab. These keys will be used to generate an auth token.
- After this step build the code for arduino or platformIO and upload to on the ESP8266 board
 - This sketch uses the popular [WifiManager library](#) to manage WiFi credentials. Upon booting the ESP8266 will start a WiFi hotspot with the name “AutoConnectAP”. The user should connect to it with cellphone/laptop and enter in their credentials. These credentials will be stored in the ESP flash memory and will be stored as the default. These credentials can be changed later on.
 - Setting up OTA**

The asvin IoT platform provides secure OTA updates for IoT devices. Lets see how we can setup OTA updates

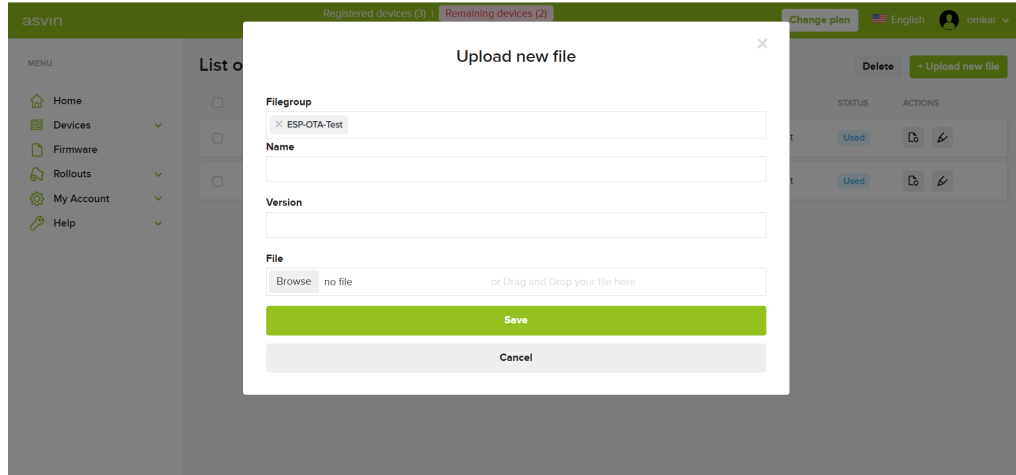
- Register Device:** When you start and upload your sketch on the ESP8266 board, the board will start executing and calling the defined API routes. The first API it calls is [Register Device](#), After this API is successfully called, you will see your device appear under the “Just Registered Devices” section of the platform under devices.



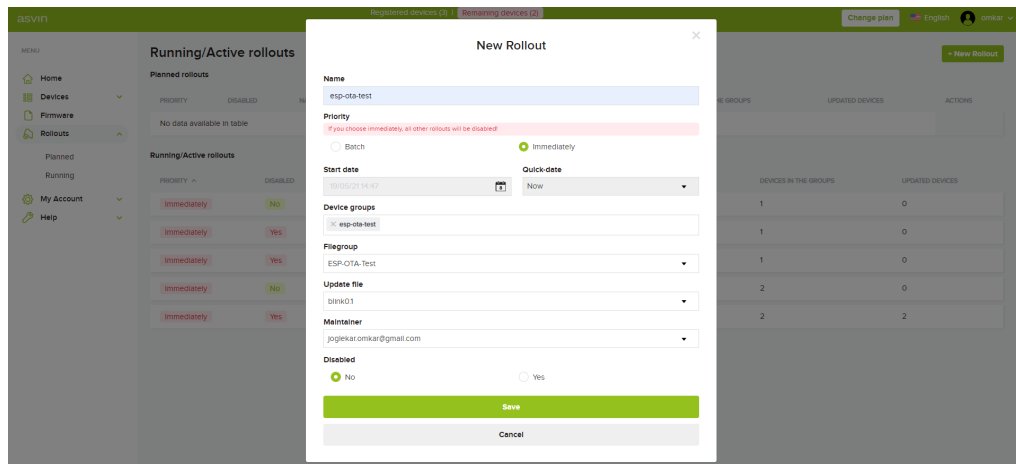
- Device Groups:** Asvin's IoT platform provides updates for a group of devices. Let us create a group called OTA test. We can add our ESP device to this group . Under Devices > Device

groups click on “*New Device Group*”. After this navigate back to the “Just Registered” Devices, click Device Grouping and add the device to the newly created device group.

3. *File Groups*: Once our device is assigned to a file group. Let’s upload a file we want to provide as an OTA update. Usually this is `<file_name>.bin`. Let us upload esp-ota-blink.bin file to the filegroup ESP_OTA_Test



4. *Rollout*: In this step we will setup a rollout to deliver OTA an update of the file specified above to our ESP8266 device. In the rollout section let’s start by creating a rollout. Fill in the options as shown in the screenshot. Choose either batch or immediate update. There is an option to choose a time or to do an update immediately. Select the file to be rolled out as an update and click *Save*.

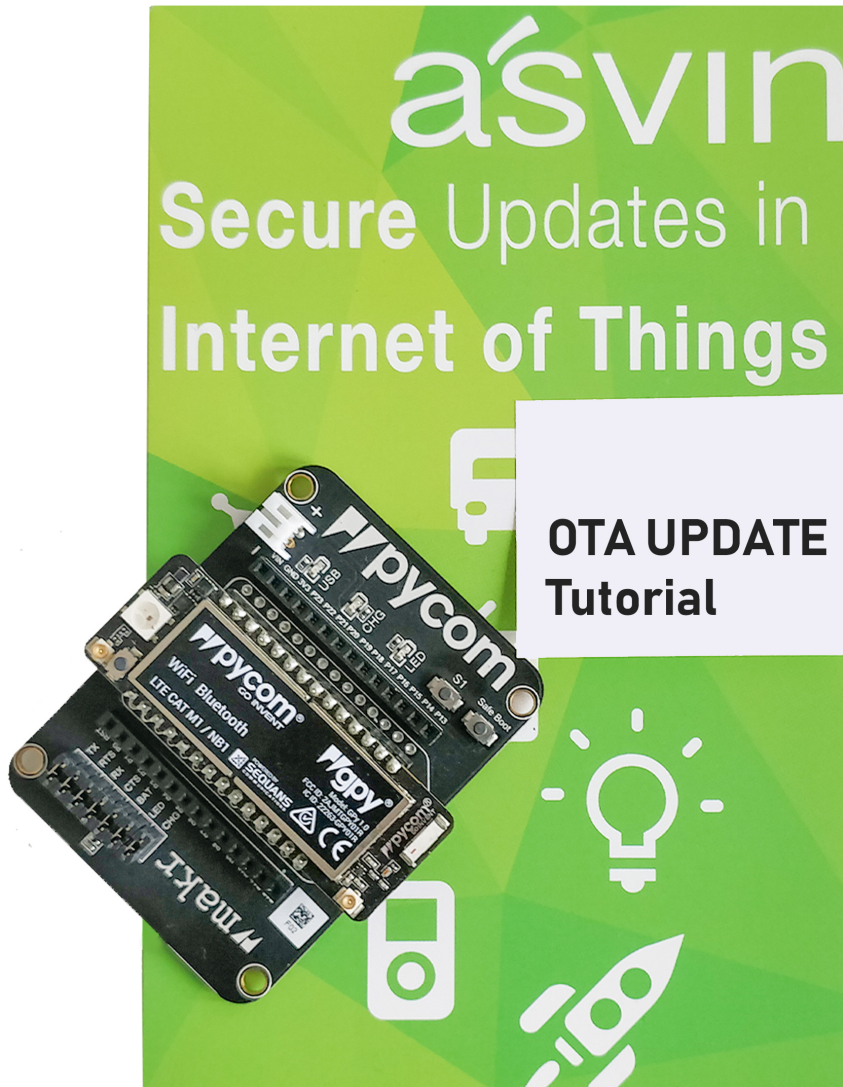


5. The rollout is now enabled. Next time our device queries the *Next Rollout* API, the rollout will be available and further API’s will be called inside the ESP device. The ESP device will update itself after this with the file we uploaded earlier. In this case we will see the LED blinking on our ESP board
6. Once the rollout is completed the new file will be running on the board. In this case we rolled out a Blink LED file. The board will call the *Rollout Success* API, which is the part of the esp-ota-blink.bin file that we uploaded earlier.
7. The change in the firmware version of the device is also updated on the [asvin platform](#).

Thus we have successfully completed the OTA rollout. The Complete code and files can be found at asvin’s Github repository [Github repository](#)

4.2 Over the air updates with Pycom Gpy boards

In this tutorial we will see the demonstration of OTA updates using the asvin IoT platform and the *Gpy* board from Pycom.



4.2.1 Requirements

1. Pycom [Gpy](#) board
2. Pycom [Expansion](#) board
3. Micro USB cable
4. Asvin platform subscription
5. [Pymakr](#) VScode extension

4.2.2 Getting started

To get started head to [asvin's Github repository](#) and clone it. Open the *pycom-ota-updates* folder in VScode. Make sure to have the updated firmware on your Pycom Gpy as well as the *expansion* board.

1. Description of Files:

This repository contains few python sketches. Below is a brief description of them

lib/OTA.py This library enables OTA updates. It is discussed in detail below.

connect_wifi.py This script is a wrapper around the pycom Wlan() library

asvin.py This file contains functions to call various API's from the Asvin Platform.

config.py This file contains various user configuration options and are discussed in the next section.

2. To proceed further you will need to edit a few of the parameters in the config.py file.

- Open the config.py file in the editor and add credentials for your device

```

8
9 # LED colors
10 LED_ERROR = 0xff0000 #RED
11 LED_blink_WIFI = 0x00ff00 #Green
12 LED_SLEEP = 0x0000ff #Blue
13
14 # Asvin Credentials
15 customer_key="< Enter Customer key from Asvin platform >"
16 device_key = "< Enter Device key from Asvin platform >"
17 platformemail= "< Enter email address registerd on Asvin platform >"
18 platformpassword = "< Enter password registerd on Asvin platform >"
19
20 # URL's
21 register= "https://vcprod.asvin.io/api/device/register"
22 checkRollout= "https://vcprod.asvin.io/api/device/next/rollout"
23 checkRolloutSuccess= "https://vcprod.asvin.io/api/device/success/rollout"
24 bc_Login= "https://bc.asvin.io:6767/auth/login"
25 bc_GetFirmware= "https://bc.asvin.io:6767/firmware/get"
26 ipfs_Login= "https://ipfsprod.asvin.io/auth/login"
27 ipfs_Download= "https://ipfsprod.asvin.io/firmware/download"
28
29 # Wifi Credentials
30 wifissid= "< Enter Wifi SSID >"
31 wifipassword= "< Enter Wifi Password >"
32
33

```

- Under the Asvin Credentials populate the following fields
 - customer_key: Enter your Customer key from the asvin platform
 - device_key: Enter your Device key from asvin platform
- Under WiFi Credentials, fill in the SSID and password.
- Optionally, you can also set the LED color for various funtions from the config file.

3. After this step, upload the project on the Pycom Board.

4. The code goes through the following steps:

- Connect to WiFi.

- Check if the previous *rollout* was successful.
- Next it will register the device by calling the *Register Device* API.
- Then the code will check if a rollout exists
- If a rollout exists the code will try to download and perform the update

5. Setting up OTA

Follow the steps below along with the `../getting-started/customer-platform` guide.

1. **Register Device:** The device will be automatically registered on booting
2. **Device Groups:** Setup a device group on the Asvin IoT platform.
3. **File Groups:** In the case of Pycom target devices, there are certain modifications to be done to files before uploading them to a filegroup for rollout. Users must add the following two lines at the start of every file they want to upload over the air.

```
path="/flash/config.py"
version = "0.0.1"
"""
Asvin OTA Config File
"""
```

In this case the *Path* variable is the path of the variable inside the Pycom's filesystem. The *version* is the user defined version number of the existing file.

4. **Rollout:** Setup the rollout as mentioned in the *Getting Started* guide. In this case it is important to follow the guidelines mentioned under *File Groups*.

Thus we have successfully completed the OTA rollout for the Pycom Gpy board. The Complete code and files can be found at asvin's Github repository [Github repository](#)

4.3 Over-The-Air update of ESP32

This tutorial demonstrates the process of securely updating ESP32 device firmware Over-The-Air with asvin platform.

4.3.1 Requirements

1. ESP32 board
2. Micro USB cable
3. asvin platform subscription
4. PlatformIO VScode extension

4.3.2 Setup

To get started head to [asvin's Github repository](#) and clone it. Open the folder ESP32-OTA in Visual Studio code. The code is written under PlatformIO.

1. Rename `credentials_copy.h` to `credentials.h` in the `src` folder.
2. Update `customer_key`, `device_key` which can be obtained from asvin platform as shown below.

3. Then build the application and flash it to ESP32.
4. This sketch uses the popular [WifiManager library](#) to manage WiFi credentials. Upon booting the ESP32 will start a WiFi hotspot with the name “AutoConnectAP”. The user should connect to it with cellphone/laptop and enter in their WiFi credentials. These credentials will be stored in the ESP flash memory and will be stored as the default. These credentials can be changed later on.

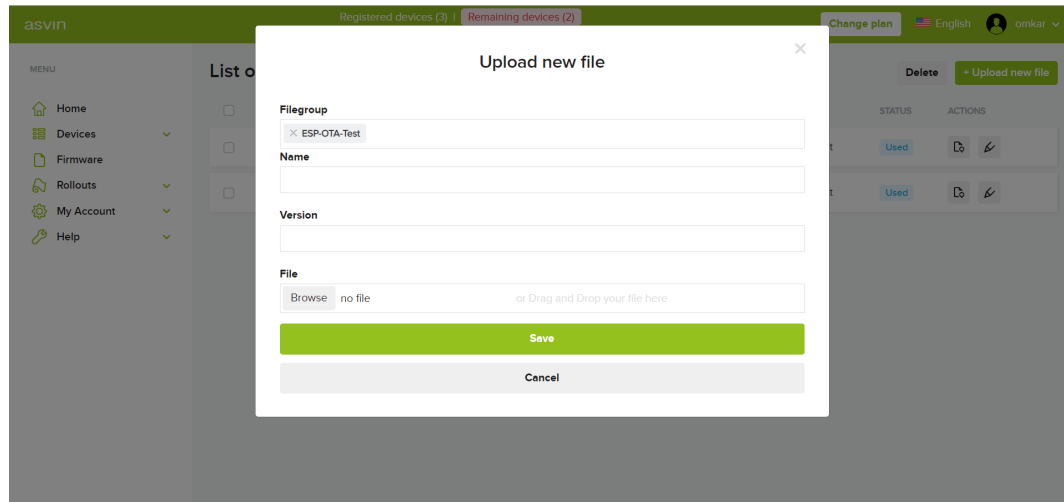
4.3.3 OTA update procedure

The asvin platform provides secure OTA updates for IoT devices. The user can follow the below easy steps to update their IoT edge devices.

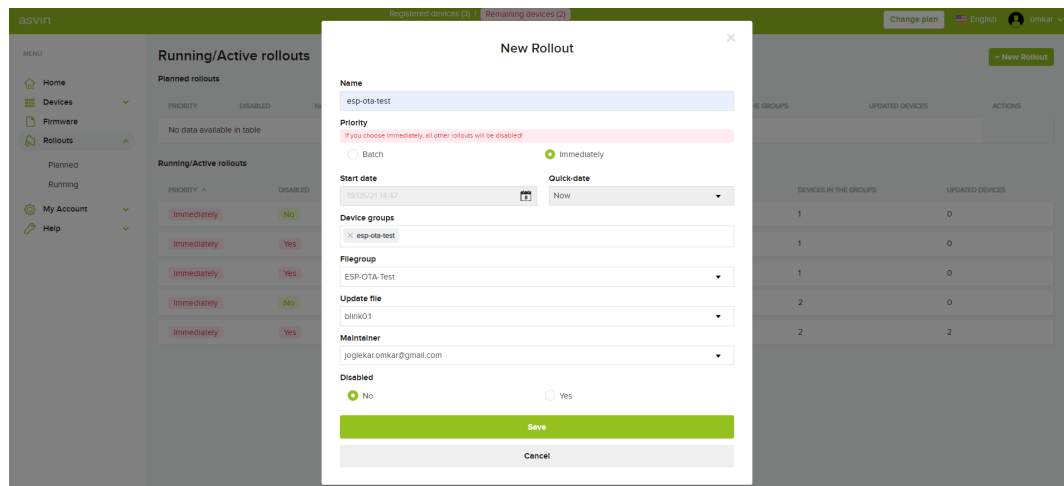
1. *Register Device*: When you upload your sketch on the ESP32 board and start it, the device will start executing and calling set of defined API routes. The device first obtains the oauth token from the asvin [oauth API](#). This token will be valid for next 10 mins. After obtaining the oauth token, the device calls the API *Register Device*. Once this API call is successful, you will see your device appear under the “Lobby” subsection of the “Devices” section in the platform.

Visible fields	Name	MAC address	Serial number	Class	Firmware version	Created	Actions
<input type="checkbox"/>	pytrack1	AC:12:CC:22:34:D2:23			1.0	18/02/21 10:11 am	
<input type="checkbox"/>	new Device	30:ae:a4:2a:67:70			1.0	10/05/21 9:24 am	

2. *Device Groups*: asvin's IoT platform provides updates for a group of devices. Let us create a group called ESP32Devices. We can add our ESP device to this group . Under *Devices -> Grouped* click on “*New Device Group*”. Then give the group name and save it. After this navigate back to the “Lobby”, click Device Grouping and add the device to the newly created device group.
3. *File Groups*: Now we have to upload the file we want to provide as an OTA update. Usually this is `<file_name>.bin`. Let us upload esp-ota-blink.bin file to the filegroup ESP_OTA_Test.



4. *Rollout*: In this step we will setup a rollout to deliver OTA update of the file specified above to our ESP32 device. In the *Rollouts* section let's start by creating a rollout. Fill in the options as shown in the below figure. Choose either batch or immediate update. There is an option to choose a time or to do an update immediately. Select the file to be rolled out as an update and click *Save*.



5. The rollout is now enabled. Next time our device queries the *Next Rollout* API, the rollout will be available and further API's will be called inside the ESP device. The ESP device will update itself by downloading the file from asvin IPFS server. After successful update, we will see the LED blinking on the ESP board.
6. Once the rollout is completed the new application will be running on the board. In this case we rolled out a Blink LED application. The board will call the *Rollout Success* API, which is the part of the esp-ota-blink.bin file that we uploaded earlier.
7. The change in the firmware version of the device is also updated on the [asvin platform](#) .

Thus we have successfully completed the OTA rollout. The Complete code and files can be found at asvin's Github repository [Github repository](#)

4.4 Fed4FIRE+ Experiments

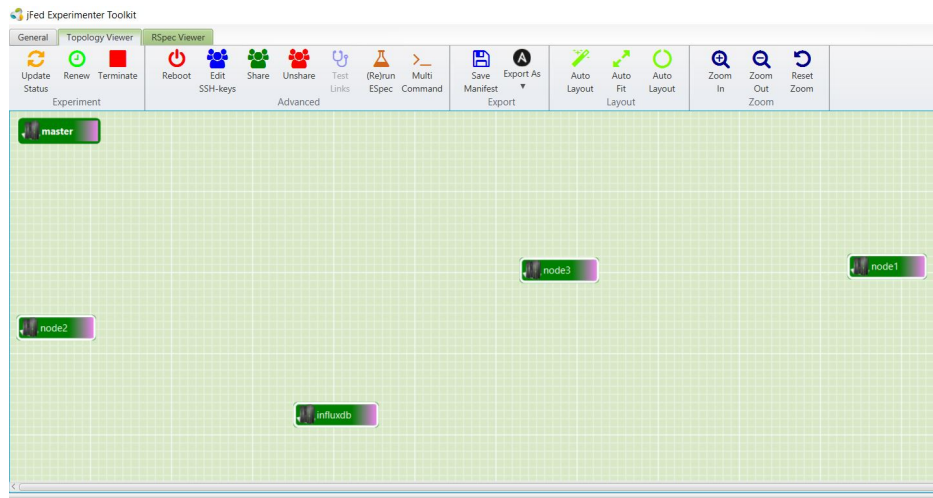
The tutorial demonstrates the steps required to start and manage the Fed4FIRE+ experiments for executing stress tests of the asvin.io platform by simulating virtual edge devices with asvin API stack. This procedure facilitates the detailed analysis of the scalability and reliability of the platform.

4.4.1 Requirements

1. jFed Experimenter Toolkit
2. ESPEC Generator
3. asvin API stack image
4. Grafana json file

4.4.2 Getting Started

1. **Starting the Experiment** To start and monitor the Fed4FIRE+ experiments we use the [jFed Experimenter Toolkit](#). It is a user-friendly tool with a simple and self-explanatory interface. Login to the experimenter tool with Fed4FIRE+ account. Then experiments can be created and started by utilizing the drag and drop options to form a network of nodes. Alternatively, the Experiment Specifications (ESpec) can be generated and uploaded in the toolkit to start the experiment.



To create the ESpec, the ESPEC Generator tool is needed which is available in [ESPEC Generator Github repository](#). This tool can be used to generate ESpec for only Virtual Wall1 and Wall2 testbeds. It includes all the installation scripts of Kubernetes cluster, influx DB and other tools which are required for the stress test of the platform.


```

generate-espec> python2.exe .\main.py --help
usage: main.py [-h] [--nodes [NODES]] [--no-control-server] [--gateway]
               [--wall {wall1,wall2}]

Generate espec for kubernetes

optional arguments:
  -h, --help            show this help message and exit
  --nodes [NODES]       amount of nodes in the generated espec, not including
                        the master node
  --no-control-server    Do not include the code to provision and setup a
                        control server with influx, grafana, private docker
                        registry and control website
  --gateway              add a gateway + apache server for delay testing
  --wall {wall1,wall2}  Target Virtual Wall, defaults to wall2
generate-espec> python2.exe .\main.py --wall wall2 --node 100

```

Before running the ESPEC generator, the python requirements have to be installed by running the line below in the generator folder.

```
pip install -r requirements.txt
```

Using the generated ESPEC, one can start the experiment on Fed4FIRE+ testbeds. The experiment will have a master node, influxdb node, and all other worker nodes. If ESPEC is generated for 5 nodes, then the experiment will have a master node, influxdb node, and 5 worker nodes running.

To reserve more than 15 nodes in an experiment, it is advised to inform the Fed4FIRE+ team before initiating the experiment.

2. **Setting up Control Server** After successfully starting the experiment with a Kubernetes cluster, the user shall download the control server from the [Experiment-webserver Github repository](#) and deploy it on the master node of the Kubernetes cluster.

Follow the steps given in the repository. Then the control server website will be accessible by going to the public IPv6 address of the server.

3. **Deploying asvin API stack image** The example python code running the API stack for simulating the edge device is provided in [asvin Github repository](#). The user has to provide the credentials for the Blockchain server and IPFS Login, User Key and Device key in the UserDetails.json file.

The image takes 2 user inputs:

- Number of threads to run

- The server (production or staging)

By default it starts with 1 thread and uses staging server details

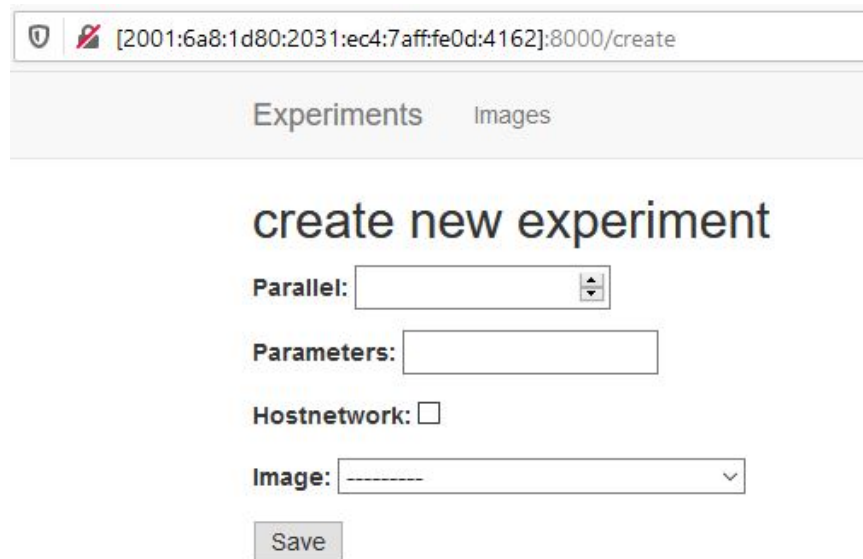
Files asvincurl.py and Dockerfile are zipped together to .tar.gz

```
tar cvfz asvin_stage2.tar.gz asvincurl.py Dockerfile
```

The control server has a web interface through which the user can create a docker image using the tar file generated, which will then be deployed to the Docker registry.

4. **Monitoring the Experiment** In the Experiment Monitoring interface, a new experiment can be created using one of the Docker images from the Docker registry.

While creating the experiments, you should provide the runtime arguments for the python code. Otherwise, the code runs with the default arguments as mentioned previously. Also, the user should mention the number of pods (parallels) to run on the Kubernetes cluster.



[2001:6a8:1d80:2031:ec4:7aff:fe0d:4162]:8000/create

Experiments Images

create new experiment

Parallel:

Parameters:

Hostnetwork: ☐

Image:

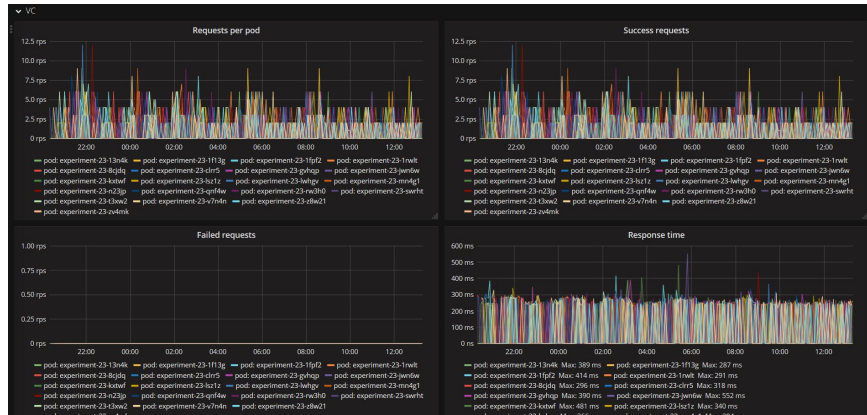
Save

The number of parallel pods running on the cluster can be changed anytime while the experiment is running.

5. **Analysis of results in Grafana** The asvin API stack image running in the experiment saves the following values in the influxdb server.

1. Total requests to Version controller, Blockchain, and IPFS servers
2. Total successfully served requests from Version controller, Blockchain, and IPFS servers
3. Total failed requests from Version controller, Blockchain, and IPFS servers
4. Response times of each requests to all 3 servers
5. Successful firmware updates

In Grafana, these values are fetched from the influxdb server and visualized as time-series graphs to analyze the robustness of asvin platform. The [sample json file](#) can be used to create a Grafana dashboard.



SECURITY PRINCIPLES

This section includes security principles of the asvin platform.

5.1 PUF Based Device ID

Wikipedia” A physical unclonable function or PUF, is a physical object that for a given input and conditions (challenge), provides a physically-defined “digital fingerprint” output that serves as a unique identifier”

PUF takes advantage of submicron variations that occur naturally during semiconductor fabrication. These variations occur in the physical parameters such as length, width, thickness, etc of semiconductor materials. a detailed blog is posted [here](#).

For this demo we have used the E1 development board from OKDO. This development board is based on LPC55S69xx microcontroller from NXP. The microcontroller contains onboard Physical Unclonable Function (PUF) using dedicated SRAM. Follow this blog [here](#) for a detailed [setup guide](#).

Later on we also describe the process of key generation [here](#) based on PUF technology. This generated key can be used in your applications in various ways. We at asvin are using the generated ID as a unique device ID while making requests to the asvin BeeHive IoT update platform.

5.2 API Endpoint Security

asvin components expose their services using RESTful API endpoints. They are secured using Jason Web Token(JWT). It is required to obtain a JWT from OAuth server. Only thereafter the endpoints can accessed successfully. The [Login](#) API endpoint is used to get JWT from OAuth.

5.2.1 Device Signature

The `device_signature` used in the The [Login](#) API is a hashed-based message authentication code (MAC). It consists of cryptographic hash function (HMAC-SHA256) and secret key. In psuedocode, it can be illustrated as `HMAC-SHA256(key, message)`. Here, message is `timestamp+device_key` and key is `customer_key`. So, the `device_signature` is calculated as

```
device_signature = HMAC-SHA256(customer_key, timestamp+device_key)
```

The `customer_key` and `device_key` are acquired from [Customer Platform](#). One needs to make a account there. The code block below shows the `device_signature` generation.

Bash

Python

JavaScript

```
#!/bin/bash
customer_key="my-customer-key"
device_key="my-device-key"
timestamp=$(date +%s)
device_signature=$(echo -n $timestamp$device_key | openssl dgst -sha256 -hmac $customer_
↪key)
echo $device_signature
```

```
import hmac
import hashlib
from time import time
customer_key = "my-customer-key"
device_key = "my-device-key"
timestamp = str(math.floor(time()))
device_signature = hmac.new(customer_key, msg=timestamp+device_key, digestmod=hashlib.
↪sha256).hexdigest().upper()
print device_signature
```

```
const CryptoJS = require("crypto-js");
const dateNow = new Date();
const customerKey = "my-customer-key";
const deviceKey = "my-device-key";
const timestamp = Math.floor(dateNow.getTime() / 1000);
const deviceSignature = CryptoJS.HmacSHA256(timestamp + deviceKey, customerKey).
↪toString(CryptoJS.digest);
console.log(deviceSignature)
```

ARCHITECTURE REFERENCE

REST API

This section defines REST API's supported by all components of Asvin. To test the API end points use the [Swagger API Doc](#) or [Postman Collection](#).

7.1 OAuth APIs

This section shows the Rest API end-points of OAuth Server.

Table of contents

- [Login](#)

7.1.1 Login

This API end point returns a token which should be added to the **x-access-token** header, for all other API requests.

POST /auth/login

The `device_key` and `customer_key` are obtained from the asvin dashboard. The `timestamp` is unix epoch. The `device_signature` is `HMAC-SHA256`. The *Device Signature* section contains calculation details.

Request Headers

- `Content-Type` – `application/json`

Example request:

cURL

JavaScript

Python

PHP

```
$ curl --location --request POST 'https://oauth-server/auth/login' \
--header 'Content-Type: application/json' \
--data-raw '{
  "device_key": "your-device-key",
  "timestamp": 1620045991
  "device_signature": "your-device-signature"
}'
```

```

var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://oauth-server/auth/login',
  'headers': {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({"device_key":"your-device-key","timestamp": 1620045991,
  ↪ "device_signature":"your-device-signature"})
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});

```

```

import requests
url = "https://oauth-server/auth/login"
payload={"device_key":"your-device-key","timestamp":1620045991,"device_
  ↪ signature":"your-device-signature"}
headers = {
  'Content-Type': 'application/json'
}
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)

```

```

<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://oauth-server/auth/login');
$request->setRequestMethod('POST');
$body = new http\Message\Body;
$body->append('{
  "device_key": "your-device-key",
  "timestamp": 1620045991,
  "device-signature": "your-device-signature",
}');
$request->setBody($body);
$request->setOptions(array());
$request->setHeaders(array(
  'Content-Type' => 'application/json'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();

```

Example response:

```

{
  "token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
  ↪ eyJpYXQiOiJlMDYybnDk4ODYsImV4cCI6MTYwNjMxMDQ4Nn0.CCWvzR124OGf5FFOFAObQDPNrlmtI_
  ↪ kaObtu0X-
  ↪ eNfPjUaHv5kfjGzZl4PUVXT0idSC4SjXFLACqOgyY7gb1UiHI3S47KvhIdCLgte8BvEIyIwLLj4rD4mdWT4NeRkP67-
  ↪ AXUG9IVM7_6XaGB-xmVLD-cLKFiMLH7wAnEDx051gOgbc05CP-1LQKuc2ApYPnDwtJMbKLlcQ-
  ↪ f7k81ouiiOWK0sB-cXq8yqt85WV4BJADhTDbvm3kjAQ5AE0pi7cU_
  ↪ sxh4JG4RaFKz7mNAanvHTw7LbZmP6tcvcf-bvcqTkbb0nkstXCD6300mBe4D44gV
  ↪ 70ehM1HF7xUS6nYpnIw"

```


(continued from previous page)

}

Response Headers

- *Content-Type* – application/json
- *X-RateLimit-Limit* – 10
- *X-RateLimit-Remaining* – 9
- *X-RateLimit-Reset* – 1617352926

Status Codes

- *200 OK* – OK
- *429 Too Many Requests* – Too many requests in this time frame.
- *500 Internal Server Error* – Error on Server

7.2 Version Controller APIs

This section shows the Rest API end-points of Version Controller.

Table of contents

- *Register Device*
- *Delete Device*
- *Get Device*
- *Get All Devices*
- *Get Report*
- *Next Rollout*
- *Rollout Success*

7.2.1 Register Device

POST /api/device/register

Register a device.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
$ curl --location --request POST 'https://vc-server/api/device/register' \
--header 'Content-Type: application/json' \
--header 'X-Access-Token: <JWT-TOKEN>' \
--data-raw '{
  "mac": "your-device-mac",
  "firmware_version": "1.0",
  "name": "device-name",
}'
```

```
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://vc-server/api/device/register',
  'headers': {
    'Content-Type': 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
  },
  body: JSON.stringify({"mac":"your-device-mac","firmware_version":"1.0","name":
↪ "device-name"})
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests
url = "https://vc-server/api/device/register"
payload = "{\n\t\"mac\": \"your-device-mac\", \n\t\"firmware_version\": \"1.0\", \n\t\"
↪ name\": \"device-name\" \n}"
headers = {
  'Content-Type': 'application/json',
  'X-Access-Token': '<JWT-TOKEN>'
}

response = requests.request("POST", url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://vc-server/api/device/register');
$request->setRequestMethod('POST');
$body = new http\Message\Body;
$body->append('{
  "mac": "your-device-mac",
  "firmware_version": "1.0",
  "name": "device-name",
}');
$request->setBody($body);
$request->setOptions(array());
```

(continues on next page)

(continued from previous page)

```

$request->setHeaders(array(
    'Content-Type' => 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();

```

Example response:

```

{
  "message": "Device inserted!"
}

```

```

{
  "message": "Device exists!"
}

```

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – No error
- 404 Not Found – Not Found
- 401 Unauthorized – JWT is not valid

7.2.2 Delete Device

DELETE /api/device/

Delete a device.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

```

$ curl --location --request DELETE 'https://vc-server/api/device/' \
--header 'Content-Type: application/json' \
--header 'X-Access-Token: <JWT-TOKEN>' \
--data-raw '{
  "mac": "your-device-mac",
}'

```

Example response:

```
{  
  "message": "Device deleted!"  
}
```

```
{  
  "message": "Device does't exist!"  
}
```

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – No error
- 404 Not Found – Not Found
- 401 Unauthorized – JWT is not valid

7.2.3 Get Device

GET /api/device/:mac

Get a device.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

```
$ curl --location --request GET 'https://vc-server/api/device/<device-mac>' \  
  --header 'X-Access-Token: <JWT-TOKEN>'
```

Example response:

```
{  
  "mac": "device-mac",  
  "name": "device-name",  
  "firmware-version": "device-firmware-version"  
}
```

```
{  
  "message": "Device does't exist!"  
}
```

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – No error

- 404 Not Found – Not Found
- 401 Unauthorized – JWT is not valid

7.2.4 Get All Devices

GET /api/device/

Get all devices.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

```
$ curl --location --request GET 'https://vc-server/api/device/' \
--header 'X-Access-Token: <JWT-TOKEN>'
```

Example response:

```
[
  {
    "mac": "device-mac",
    "name": "device-name",
    "firmware-version": "device-firmware-version"
  },
  {
    "mac": "device-mac",
    "name": "device-name",
    "firmware-version": "device-firmware-version"
  }
]
```

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – No error
- 404 Not Found – Not Found
- 401 Unauthorized – JWT is not valid

7.2.5 Get Report

GET /api/report/:type

Get report in json, xml, and pdf format.

reqheader Content-Type application/json

reqheader X-Access-Token JWT-TOKEN

Example request:

cURL

```
$ curl --location --request GET 'https://vc-server/api/report/<json/xml/pdf>'
→ '\
--header 'X-Access-Token: <JWT-TOKEN>'
```

Example response:

```
[
  {
    "mac": "device-mac",
    "name": "device-name",
    "firmware-version": "device-firmware-version"
  },
  {
    "mac": "device-mac",
    "name": "device-name",
    "firmware-version": "device-firmware-version"
  }
]
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <element>
    <firmware-version>device-firmware-version</firmware-version>
    <mac>device-mac</mac>
    <name>device-name</name>
  </element>
  <element>
    <firmware-version>device-firmware-version</firmware-version>
    <mac>device-mac</mac>
    <name>device-name</name>
  </element>
</root>
```

:resheader Content-Type: application/json

:statuscode 200: No error

:statuscode 404: Not Found

:statuscode 401: JWT is not valid

7.2.6 Next Rollout

POST /api/device/next/rollout

Check next rollout

reqheader Content-Type application/json

reqheader X-Access-Token JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
curl --location --request POST 'https://vc-server/api/device/next/rollout' \
--header 'Content-Type: application/json' \
--header 'X-Access-Token: <JWT-TOKEN>' \
--data-raw '{
  "mac": "your-device-mac",
  "firmware_version": "1.0"
}'
```

```
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://vc-server/api/device/next/rollout',
  'headers': {
    'Content-Type': 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
  },
  body: JSON.stringify({"mac":"your-device-mac","firmware_version":"1.0"})
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests
url = "https://vc-server/api/device/next/rollout"
payload = "{\n\t\"mac\": \"your-device-mac\", \n\t\"firmware_version\": \"1.0\"}"
headers = {
  'Content-Type': 'application/json',
  'X-Access-Token': '<JWT-TOKEN>'
}
response = requests.request("POST", url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://vc-server/api/device/next/rollout');
$request->setRequestMethod('POST');
$body = new http\Message\Body;
$body->append('{
    "mac": "your-device-mac",
    "firmware_version": "1.0"
}');
$request->setBody($body);
$request->setOptions(array());
$request->setHeaders(array(
    'Content-Type' => 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();
```

Example response:

```
{
  "rollout_id": "84",
  "rollout_name": "new-demo-rollout",
  "priority": "1",
  "start_date": "2021-04-02 09:30:00",
  "version": "2.0",
  "firmware_id": "11"
}
```

If no rollout exists:

```
{}
```

resheader Content-Type application/json

statuscode 200 No error

statuscode 404 Not Found

statuscode 401 JWT is not valid

7.2.7 Rollout Success

POST /api/device/success/rollout

Inform rollout status

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
curl --location --request POST 'https://vc-server/api/device/success/rollout' \
--header 'Content-Type: application/json' \
--header 'X-Access-Token: <JWT-TOKEN>' \
--data-raw '{
  "mac": "your-device-mac",
  "firmware_version": "2.0"
  "rollout_id": "84"
}'
```

```
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://vc-server/api/device/success/rollout',
  'headers': {
    'Content-Type': 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
  },
  body: JSON.stringify({"mac":"your-device-mac","firmware_version":"2.0","rollout_id
→":"84"})
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests
url = "https://vc-server/api/device/success/rollout"
payload = "{\n\t\"mac\": \"your-device-mac\", \n\t\"firmware_version\": \"2.0\", \n\t
→\"rollout_id\": \"84\" \n}"
headers = {
  'Content-Type': 'application/json',
  'X-Access-Token': '<JWT-TOKEN>'
}
response = requests.request("POST", url, headers=headers, data = payload)
print(response.text.encode('utf8'))
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://vc-server/api/device/success/rollout');
$request->setRequestMethod('POST');
$body = new http\Message\Body;
$body->append('{
  "mac": "your-device-mac",
  "firmware_version": "2.0",
```

(continues on next page)

(continued from previous page)

```
"rollout_id": "84"
}');
$request->setBody($body);
$request->setOptions(array());
$request->setHeaders(array(
    'Content-Type' => 'application/json',
    'X-Access-Token': '<JWT-TOKEN>'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();
```

Example response:

```
{
  "message": "Successfully inserted!"
}
```

```
{
  "message": "Existing Record"
}
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – No error
- 404 Not Found – Not Found
- 401 Unauthorized – JWT is not valid

7.3 Blockchain APIs

This section shows the Rest API end-points of Blockchain.

Table of contents

- *Get Device*
- *Get Firmware*

7.3.1 Get Device

GET /device/:id

Get a device.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
$ curl 'https://bc-server/device/:id' \
-H 'Content-Type: application/json' \
-H 'X-Access-Token: <JWT-TOKEN>
```

```
var request = require('request');
var options = {
  'method': 'GET',
  'url': 'https://bc-server/device/:id',
  'headers': {
    'X-Access-Token': '<JWT-TOKEN>'
  }
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests

url = "https://bc-server/device/:id"

headers = {
  'X-Access-Token': '<JWT-TOKEN>'
}
response = requests.request("GET", url, headers=headers)
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://bc-server/device/:id');
$request->setRequestMethod('GET');

$request->setOptions(array());
$request->setHeaders(array(
```

(continues on next page)

(continued from previous page)

```
'X-Access-Token' => '<JWT-TOKEN>'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();
```

Example response:

```
{
  "mac": "AC:AC:CC:CC:34:34",
  "fwId": "3",
  "dType": "ESP"
}
```

Response Headers

- *Content-Type* – application/json

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found

7.3.2 Get Firmware

GET `/firmware/:id`

Get a device.

Request Headers

- *Content-Type* – application/json
- *X-Access-Token* – JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
$ curl 'https://bc-server/device/:id' \
-H 'X-Access-Token: <JWT-TOKEN>
```

```
var request = require('request');
var options = {
  'method': 'GET',
  'url': 'https://bc-server/firmware/:id',
  'headers': {
    'X-Access-Token': '<JWT-TOKEN>'
  }
};
```

(continues on next page)

(continued from previous page)

```
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests

url = "https://bc-server/firmware/:id"

headers = {
  'X-Access-Token': '<JWT-TOKEN>'
}
response = requests.request("GET", url, headers=headers)
print(response.text)
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://bc-server/firmware/:id');
$request->setRequestMethod('GET');
$request->setOptions(array());
$request->setHeaders(array(
  'X-Access-Token' => '<JWT-TOKEN>'
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();
```

Example response:

```
{
  "md": "6f5902ac237024bdd0c176cb93063dc4",
  "cid": "QmWATWQ7fVPP2EFGu71UkfnqhYXDYH566qy47CnJDgvs8u",
  "version": "1.9",
}
```

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found

7.4 IPFS APIs

This section shows the Rest API end-points of IPFS.

Table of contents

- *Download Firmware*

7.4.1 Download Firmware

GET /firmware:id

Get a device.

Request Headers

- *Content-Type* – application/json
- *x-access-token* – JWT-TOKEN

Example request:

cURL

JavaScript

Python

PHP

```
$ curl -X GET 'https://ipfs-server/firmware/:cid' \
-H 'x-access-token: <JWT-TOKEN>' \
--header 'Content-Type: application/json'
```

```
var request = require('request');
var options = {
  'method': 'GET',
  'url': 'https://ipfs-server/firmware/:cid',
  'headers': {
    'x-access-token': '<JWT-TOKEN>',
  },
};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

```
import requests

url = "https://ipfs-server/firmware/:cid"

headers = {
  'x-access-token': '<JWT-TOKEN>',
}
```

(continues on next page)

(continued from previous page)

```
response = requests.request("GET", url, headers=headers)
print(response.text)
```

```
<?php
$client = new http\Client;
$request = new http\Client\Request;
$request->setRequestUrl('https://ipfs-server/firmware/:cid');
$request->setRequestMethod('GET');
$body = new http\Message\Body;

$request->setOptions(array());
$request->setHeaders(array(
    'x-access-token' => '<JWT-TOKEN>',
));
$client->enqueue($request)->send();
$response = $client->getResponse();
echo $response->getBody();
```

Example response:

Response Headers

- Content-Type – application/json

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found

CONTRIBUTION WELCOMED

The asvin project was started by a few but has been developed and maintained by many. At asvin, we are a big supporter of open source projects and have also kept the whole asvin platform open for the community. We welcome your contribution to the project. Let's all build the asvin platform together for all.

We follow the *Asvin Code of Conduct* to keep the playing field equal and fair for all to work together. We request you to review it before you report a bug or make pull requests. The aim of it to maintain decorum.

The source files of the asvin platform are stored on the GitHub repositories. Follow the *GitHub Contribution Guidelines* to start contributing in the project.

Note: Please do not modify without first getting approval from m.ross@asvin.io

8.1 Asvin Code of Conduct

Asvin GmbH developed a platform to distribute firmware updates for IoT devices. We also encourage external developers to contribute in the project. Which means it is an open-source project where participants can choose to work together, and in that process experience differences in language, location, nationality, and experience. In such a diverse environment, misunderstandings and disagreements happen, which in most cases can be resolved informally. In rare cases, however, behavior can intimidate, harass, or otherwise disrupt one or more people in the community, which Asvin will not tolerate.

A Code of Conduct is useful to define accepted and acceptable behaviors and to promote high standards of professional practice. It also provides a benchmark for self evaluation and acts as a vehicle for better identity of the organization.

This code (CoC) applies to any member of the Asvin community – developers, participants in meetings, teleconferences, mailing lists, conferences or functions, etc. Note that this code complements rather than replaces legal rights and obligations pertaining to any particular situation.

8.1.1 Statement of Intent

Asvin is committed to maintain a positive *work environment*. This commitment calls for a workplace where *participants* at all levels behave according to the rules of the following code. A foundational concept of this code is that we all share responsibility for our work environment.

8.1.2 Code

1. Treat each other with *respect*, professionalism, fairness, and sensitivity to our many differences and strengths, including in situations of high pressure and urgency.
2. Never *harass* or *bully* anyone verbally, physically or *sexually*.
3. Never *discriminate* on the basis of personal characteristics or group membership.
4. Communicate constructively and avoid *demeaning* or *insulting* behavior or language.
5. Seek, accept, and offer objective work criticism, and *acknowledge* properly the contributions of others.
6. Be honest about your own qualifications, and about any circumstances that might lead to conflicts of interest.
7. Respect the privacy of others and the confidentiality of data you access.
8. With respect to cultural differences, be conservative in what you do and liberal in what you accept from others, but not to the point of accepting disrespectful, unprofessional or unfair or *unwelcome behavior* or *advances*.
9. Promote the rules of this Code and take action (especially if you are in a *leadership position*) to bring the discussion back to a more civil level whenever inappropriate behaviors are observed.
10. Stay on topic: Make sure that you are posting to the correct channel and avoid off-topic discussions. Remember when you update an issue or respond to an email you are potentially sending to a large number of people.
11. Step down considerably: Members of every project come and go, and the Asvin is no different. When you leave or disengage from the project, in whole or in part, we ask that you do so in a way that minimizes disruption to the project. This means you should tell people you are leaving and take the proper steps to ensure that others can pick up where you left off.

8.1.3 Glossary

Demeaning Behavior

is acting in a way that reduces another person's dignity, sense of self-worth or respect within the community.

Discrimination

is the prejudicial treatment of an individual based on criteria such as: physical appearance, race, ethnic origin, genetic differences, national or social origin, name, religion, gender, sexual orientation, family or health situation, pregnancy, disability, age, education, wealth, domicile, political view, morals, employment, or union activity.

Insulting Behavior

is treating another person with scorn or disrespect.

Acknowledgement

is a record of the origin(s) and author(s) of a contribution.

Harassment

is any conduct, verbal or physical, that has the intent or effect of interfering with an individual, or that creates an intimidating, hostile, or offensive environment.

Leadership Position

includes group Chairs, project maintainers, staff members, and Board members.

Participant

includes the following persons:

- Developers
- Member representatives
- Staff members
- Anyone from the Public partaking in the Asvin work environment (e.g. contribute code, comment on our code or specs, email us, attend our conferences, functions, etc)

Respect

is the genuine consideration you have for someone (if only because of their status as participant in Asvin, like yourself), and that you show by treating them in a polite and kind way.

Sexual Harassment

includes visual displays of degrading sexual images, sexually suggestive conduct, offensive remarks of a sexual nature, requests for sexual favors, unwelcome physical contact, and sexual assault.

Unwelcome Behavior

Hard to define? Some questions to ask yourself are:

- how would I feel if I were in the position of the recipient?
- would my spouse, parent, child, sibling or friend like to be treated this way?
- would I like an account of my behavior published in the organization's newsletter?
- could my behavior offend or hurt other members of the work group?
- could someone misinterpret my behavior as intentionally harmful or harassing?
- would I treat my boss or a person I admire at work like that ?

Summary: if you are unsure whether something might be welcome or unwelcome, don't do it.

Unwelcome Sexual Advance

includes requests for sexual favors, and other verbal or physical conduct of a sexual nature, where:

- submission to such conduct is made either explicitly or implicitly a term or condition of an individual's employment,
- submission to or rejection of such conduct by an individual is used as a basis for employment decisions affecting the individual,
- such conduct has the purpose or effect of unreasonably interfering with an individual's work performance or creating an intimidating hostile or offensive working environment.

Workplace Bullying

is a tendency of individuals or groups to use persistent aggressive or unreasonable behavior (e.g. verbal or written abuse, offensive conduct or any interference which undermines or impedes work) against a co-worker or any professional relations.

Work Environment

is the set of all available means of collaboration, including, but not limited to messages to mailing lists, private correspondence, Web pages, chat channels, phone and video teleconferences, and any kind of face-to-face meetings or discussions.

Incident Procedure

To report incidents or to appeal reports of incidents, send email to m.ross@asvin.io. Please include any available relevant information, including links to any publicly accessible material relating to the matter. Every effort will be taken to ensure a safe and collegial environment in which to collaborate on matters relating to the Project. In order to protect the community, the Project reserves the right to take appropriate action, potentially including the removal of an individual from any and all participation in the project. The Project will work towards an equitable resolution in the event of a misunderstanding.

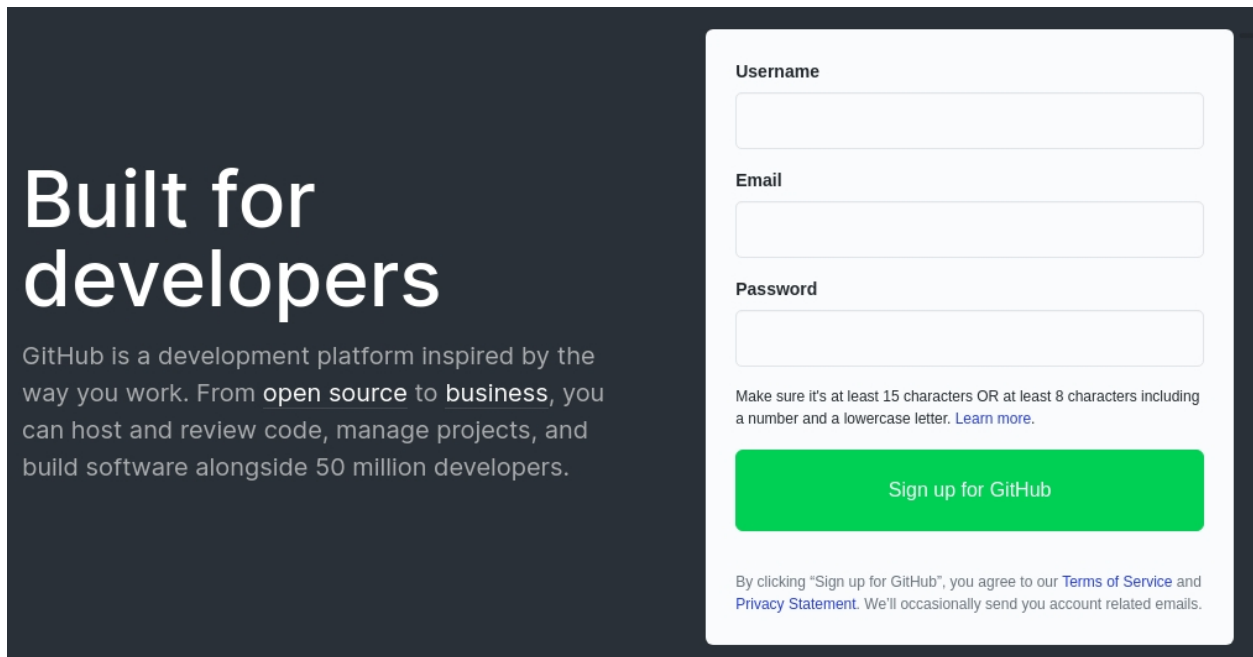
8.1.4 Credits

This code is based on the [Hyperledger Code of Conduct](#).

8.2 GitHub Contribution Guidelines

8.2.1 Create Github Account

The code base of asvin platform is developed and maintained on GitHub. One needs to create a GitHub account to start contributing for the Asvin project. In order to create the account go to [GitHub](#) and sign up with username, email and password.

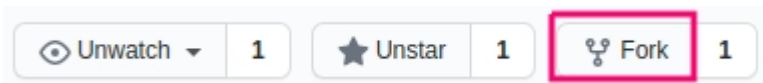


8.2.2 Fork Repository

Asvin comprised of multiple components which are developed in respective repositories. When you start contributing it is recommended to fork the repository, make your changes and submit a pull request. It helps to keep the source code in master branch maintainable, clean and stable. The forking of a repository results in an identical copy of the repository in your personal account. With that, you will have full control over the repository to make changes in the source code. Once, you are satisfied with your changes you can submit a pull request to Asvin's official repository.

How to fork a repository?

- Open your browser and go to the Asvin repository that you want to fork
- Click on fork button on top right corner



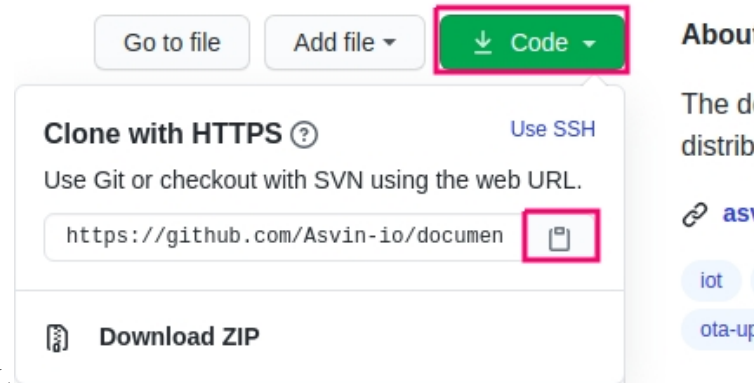
- Select your GitHub account and automatically the repository will be forked and will land you in cloned repository in your account

8.2.3 Clone Repository

After the forking process, you have a repository in your account. The next step is to clone the repository to your local system. After cloning process, you will have source files in your local machine and could open in your favorite IDE for development.

How to clone a repository?

- Open your browser and go to the repository in your account that you want to clone



- Click on Code button on middle right and copy the web URL
- Open your terminal and go to the directory where you want to clone
- Use `git clone <web-url>` e.g. `git clone https://github.com/b-rohit/documentation.git`
- Go to repository directory and add original Asvin repository as remote upstream repository

For an example Framework repository

```
cd documentation
git remote add upstream https://github.com/Asvin-io/documentation.git
```

- You can check all your remote repositories with following command

```
git remote -v
```

For an example output of the command for Framework repository

```
origin      https://github.com/b-rohit/documentation.git (fetch)
origin      https://github.com/b-rohit/documentation.git (push)
upstream    https://github.com/Asvin-io/documentation.git (fetch)
upstream    https://github.com/Asvin-io/documentation.git (push)
```

Now, everything is set to start developing and making the Asvin platform better.

8.2.4 Create Feature Branch

All Asvin component repositories have a main branch called **master**. The **master** branch contains the stable code of the component. The main idea behind using a feature branch for development is to keep the **master** branch unaffected from broken code in new feature. A developer should create a feature branch in its fork repository to develop new feature, to fix an issue, to purpose changes etc. A feature branch provides encapsulation from the main codebase. That means the **master** branch is isolated from errors caused by broken code of a new commit. It keeps the main codebase clean and stable.

How to create feature branch ?

- Fetch commits, files and refs from the upstream repository

```
git fetch upstream
```

- Checkout to master branch

```
git checkout master
```

- Merge upstream changes to local master branch

```
git merge upstream/master
```

- Push changes to remote master in forked repo

```
git push origin master
```

- Now, you have your origin/master and upstream/master synced. This process ensures there are no discrepancies between two branches and new feature branch is an exact copy of them.

```
git checkout -b <feature_branch_name>
```

With this, you have a new feature branch, where you can make changes.

8.2.5 Push changes to your Forked Repository

When you are done creating a new feature or fixing an issue in your feature branch, it is time to commit and push your changes to forked repository. This process will save the state in remote branch in forked repository. Later, this state will be used to create pull request to Asvin component repository.

How to push ?

- Add your modified, delete and new files to the index

```
git add <file1> <file2>
```

- You can now commit the current contents of the index. This will capture a snapshot of the project's currently staged changes. Your commit message must contain the following information:
 - one line summary of the changes in this commit as title, followed by an empty line
 - explain why this change is needed, and how you approached it in the commit message body. This helps reviewers better understand your code and often speeds up the review process.

```
git commit -s
```

- Push changed to your forked repository

```
git push origin <feature_branch_name>
```

At the end of this process you have developed a new feature or fixed an issue and pushed it to your forked repository. At this moment, your changes in the forked repository are ready to be integrated with Asvin component repository. This can be done by making a pull request to Asvin repository.

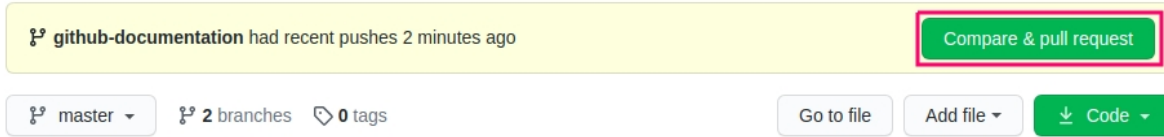
8.2.6 Opening a Pull Request in GitHub

Once you have pushed your changes to the feature branch in your forked repo, you can now open a pull request against the original Asvin component repository. This is going to be easiest of all tasks you have completed previously in this thread.

How to open Pull request ?

- Navigate to your forked repository https://github.com/<user-name>/<forked_repository>.

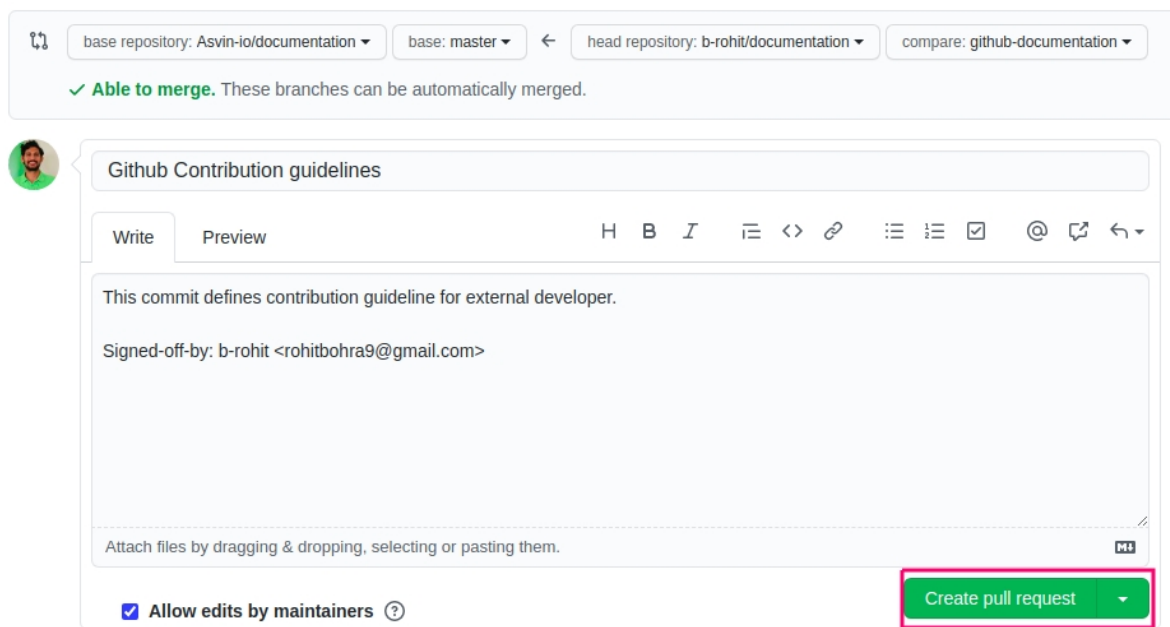
- For new changes, automatically it will be shown that there are some differences between forked and original repository. Click on Compare & pull request.



- You will be navigated to original Asvin repository. Here, you can change title and comment message if you want and click on Create pull request.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



Excelente, you just open your first pull request in Asvin project. The pull request will be reviewed and merged in the original Asvin repository.

8.2.7 Delete Feature Branch

When you are finished with your feature and your pull request is accepted and merged in the original Asvin repository, it is time to clean your forked repository. You need to delete the feature branch from your local and remote forked repository.

How to delete local and remote feature branch ?

- Delete your local feature branch

```
git branch -d <feature_branch_name>
```

- Push your changes to delete remote branch

```
git push --delete origin <feature_branch_name>
```


8.2.8 Sync Your Forked Repository with Original

Asvin is completely community based project. Therefore, the main codebase will keep getting commits from other developers. That means, you need to sync your forked repository with the original Asvin repository. By doing so, you will avoid merge conflicts along the way.

How to sync?

- Fetch changes from upstream repository

```
git fetch upstream
```

- Rebase the the local origin/master with upstream/master

```
git rebase upstream/master
```

- Push the changes to the forked repository

```
git push origin master
```

8.3 Documentation style guide

This section describes style guide for writing the documentation. It is based on [Sphinx based documentation style guide](#).

8.3.1 Filenames

Use only lowercase alphabetic characters and - (minus) symbol.

8.3.2 Whitespaces

Indentation

Indent with 2 spaces.

Except:

- `toctree` directive requires a 3 spaces indentation.

Blank lines

One blank line before each section (e.g. H1, H2 ..). See [Headings](#) for an example.

One blank line to separate directives.

```
Some text before.
```

```
.. note::
```

```
    Some note.
```

Exception: directives can be written without blank lines if they are only one line long.

```
.. note:: A short note.
```

8.3.3 Line length

Limit all lines to a maximum of 145 characters.

8.3.4 Headings

Use the following symbols to create headings:

1. # with overline
2. * with overline
3. =
4. -
5. ^
6. "

As an example:

```
#####  
H1: document title  
#####
```

Introduction text.

Sample H2

Sample content.

Another H2

Sample H3

Sample H4

Sample H5
^ ^ ^ ^ ^ ^ ^ ^ ^

Sample H6
" " " " " " " " " " " " " " " "

(continues on next page)

(continued from previous page)

And some text.

If you need more than heading level 4 (i.e. H5 or H6), then you should consider creating a new document.

There should be only one H1 in a document.

Note: See also [Sphinx's documentation about sections](#)¹.

8.3.5 Code blocks

Use the code-block directive **and** specify the programming language. As an example:

```
.. code-block:: python

    import this
```

8.3.6 Links and references

Use links and references footnotes with the target-notes directive. As an example:

```
#####
Some document
#####

Some text which includes links to `Example website`_ and many other links.

`Example website`_ can be referenced multiple times.

(... document content...)

And at the end of the document...

*****
References
*****

.. target-notes::

.. _`Example website`: http://www.example.com/
```

¹ <http://sphinx.pocoo.org/rest.html#sections>

8.3.7 References

8.4 Types of Contribution

8.4.1 As a user:

- Feature proposal
- Enhancement proposal
- Bug reporting
- Testing

8.4.2 As a Developer

- Fixing open issues
- Make feature/enhancement proposal and implement them
- Improve documentation

8.5 Maintainers

The Asvin project is driven by a dynamic team. All the development activities, such as reviewing and merging pull requests, new feature proposals, and road map release, are headed by the core [maintainers](#).

GLOSSARY

CHAPTER

TEN

RELEASES

HTTP ROUTING TABLE

/api

GET /api/device/, 45
GET /api/device/:mac, 44
GET /api/report/:type, 46
POST /api/device/next/rollout, 47
POST /api/device/register, 41
POST /api/device/success/rollout, 48
DELETE /api/device/, 43

/auth

POST /auth/login, 39

/device

GET /device/:id, 51

/firmware

GET /firmware/:id, 52

/firmware:id

GET /firmware:id, 54