OpenWISP

version 22.05

OpenWISP Community

July 02, 2025

Contents

OpenWISP Documentation

Contents:

Quick Start

Table of Contents:

1. Install the OpenWISP server application

If you want to find out how to deploy OpenWISP for production usage, follow the step by step ansible-openwisp2 tutorial.

Alternatively, there's also docker-openwisp (in alpha stage) which can be especially useful for testing purposes.

For testing purposes vagrant-openwisp2 can be used as well.

2. Install openwisp-config on your devices

Here's a guide on how to install openwisp-config on OpenWRT and connect it to OpenWISP.

If you don't have a physical OpenWRT-compatible device at hand, you can install OpenWRT in a VirtualBox VM, that's also covered by the guide.

3. Install openwisp-monitoring on your devices

If you want to take advantage of the features offered by the monitoring module of OpenWISP, read the Monitoring Quickstart Guide.

4. Watch video demonstrations

5. Look for help

See support channels.

Connect OpenWRT to OpenWISP

This page will guide you through the installation of openwisp-config on a device which supports OpenWRT.

If you don't have a physical device available but you still want to try out OpenWISP, you can use a Virtual Machine.

Table of Contents:

1. Install OpenWISP

Refer to the instructions described in Install the OpenWISP server application.

2. Flash OpenWRT on a device

If you have a physical OpenWRT compatible hardware, follow the instructions in the official OpenWRT flashing guide.

If you don't have a physical device, you can install OpenWrt on a VirtualBox Virtual Machine.

Note

It's required to enable SSH access and connect the device or VM to the internet.

Note that when using Virtualbox, both Adapter1 and Adapter2 use "Adapter Type: Intel PRO/1000 MT Desktop". Also, please do not use the same IP Address that you used for the local OpenWISP website you hosted before. That suggested change applies only when you boot into the OpenWRT device as per the description of the above link (for example, if you set 192.168.56.2 as the IP Address of your local OpenWISP website, please use another IP such as 192.168.56.3 for the IP Address of the OpenWRT device).

3. Install openwisp-config

Installation

Install openwisp-config on your OpenWRT system. For this guide.

We recommend to install one of the latest stable builds from downloads.openwisp.io, copy the URL of the *ipk* file you want to download in your clipboard and then run the following commands on your OpenWrt device:

```
opkg update
opkg install <URL-you-just-copied>
```

If you're running at least OpenWRT 19.07, you can install openwisp-config from the official OpenWRT packages:

```
opkg update
opkg install openwisp-config
```

Configuration

Once openwisp-config is installed, we need to configure it to connect to our OpenWISP2 controller. To do that, edit the config file located at `/etc/config/openwisp`.

You will see the default config file, something like the following (if your instance lacks some of the lines in the following, please add them):

```
# For more information about the config options please see the README
# or https://github.com/openwisp/openwisp-config#configuration-options

config controller 'http'
    #option url 'https://openwisp2.mynetwork.com'
    #option interval '120'
    #option verify_ssl '1'
    #option shared_secret ''
    #option consistent_key '1'
    #option mac_interface 'eth0'
    #option management_interface 'tun0'
    #option merge_config '1'
    #option test_config '1'
    #option test_script '/usr/sbin/mytest'
    #option hardware_id_script '/usr/sbin/read_hw_id'
    #option hardware_id_key '1'
    option uuid ''
    option key ''
    # curl options
    #option connect_timeout '15'
    #option max_time '30'
    #option capath '/etc/ssl/certs'
    #option cacert '/etc/ssl/certs/ca-certificates.crt'
    # hooks
    #option pre_reload_hook '/usr/sbin/my_pre_reload_hook'
    #option post_reload_hook '/usr/sbin/my_post_reload_hook'
```

Uncomment and change the following fields:

- `url`: the hostname of your OpenWISP2 controller (for example, if you are hosting your OpenWISP server locally and you set the IP Address to "192.168.56.2", the URL would be `https://192.168.56.2`).
- `verify_ssl`: set to `'0'` if your controller's SSL certificate is self-signed; in production you will need a valid SSL certificate to keep your instance secure
- `shared_secret`: you can retrieve this from OpenWISP2 admin panel, in the Organization settings. The list of organizations is available at `/admin/openwisp_users/organization/`.

Note

When testing or developing using the Django development server directly from your computer, make sure the server listens on all interfaces (`./manage.py runserver 0.0.0.0:8000`) and then just point openwisp to use your local IP address (e.g. `http://192.168.1.34:8000`)

Save the file and start openwisp-config:

```
/etc/init.d/openwisp_config restart
```

Your OpenWRT instance should register itself to your openwisp2 controller. Check the devices menu on the admin panel to make sure your OpenWRT instance is registered.

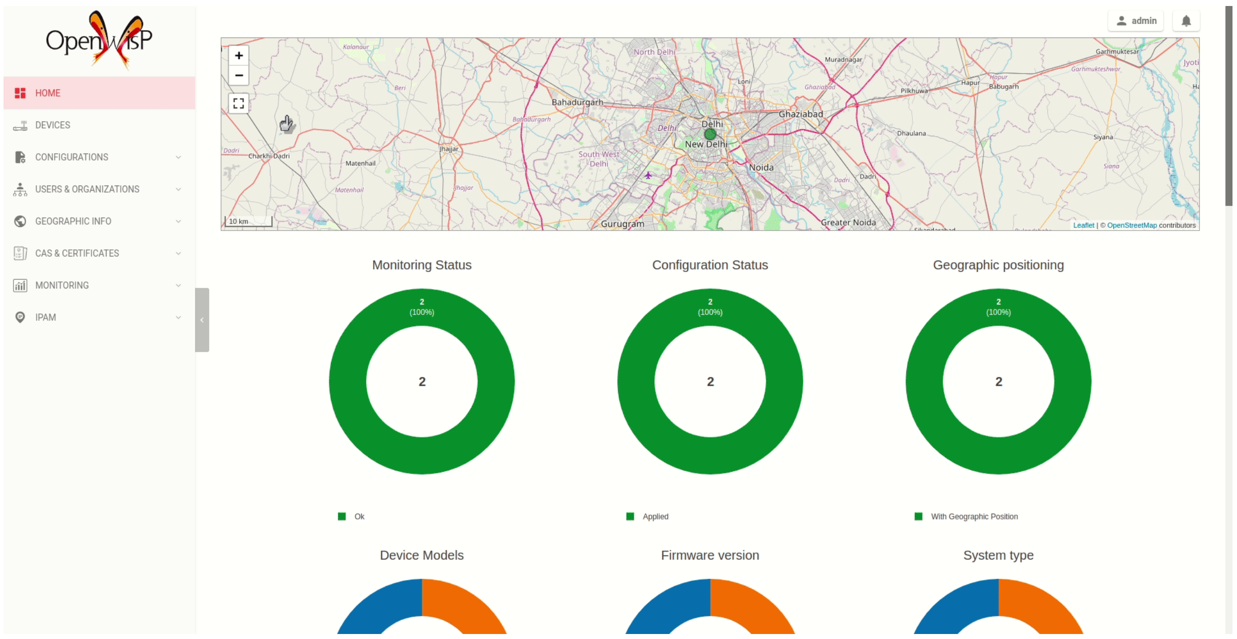
Compile your own OpenWRT image

You may want to compile a custom OpenWRT image to save time in configuring new devices. By compiling a custom image, you can preinstall openwisp-config, including your configurations (e.g. `url` and `shared_secret`), so that you won't have to go through the configuration process again.

This will make you save a lot of time if you need to manage many devices!

A guide on how to compile a custom OpenWRT image available in the openwisp-config documentation.

Monitoring



OpenWISP Monitoring has been introduced in OpenWISP 22.05 and focuses on allowing users to know the status of their devices at any given time.

Table of Contents:

Deploy instructions

The monitoring module is enabled by default in the OpenWISP 22.05 ansible role.

It's also available in docker-openwisp although its usage is not recommended for production usage yet, unless the reader is willing to invest effort in adapting the docker images and configurations to overcome any roadblocks encountered.

Quick Start Guide

This guide assumes you have the OpenWISP Monitoring module enabled in your OpenWISP server application and you have already followed the steps to install openwisp-config on your OpenWRT devices.

Install monitoring packages on the device

Install the openwrt-openwisp-monitoring packages on your device.

These packages collect and send the monitoring data from the device to OpenWISP Monitoring and are required to collect metrics like interface traffic, WiFi clients, CPU load, memory usage, etc.

Make sure OpenWISP can reach your devices

In order to perform active checks and other actions like triggering the push of configuration changes, executing shell commands or performing firmware upgrades, **the OpenWISP server needs to be able to reach the network devices**.

There are mainly two deployment scenarios for OpenWISP:

1. the OpenWISP server is deployed on the public internet and the devices are geographically distributed across different locations: **in this case a management tunnel is needed**
2. the OpenWISP server is deployed on a computer/server which is located in the same Layer 2 network (that is, in the same LAN) where the devices are located. **in this case a management tunnel is NOT needed**

1. Public internet deployment

This is the most common scenario:

- the OpenWISP server is deployed to the public internet, hence the server has a public IPv4 (and IPv6) address and usually a valid SSL certificate provided by Mozilla Letsencrypt or another SSL provider
- the network devices are geographically distributed across different locations (different cities, different regions, different countries)

In this scenario, the OpenWISP application will not be able to reach the devices **unless a management tunnel** is used, for that reason having a management VPN like OpenVPN, Wireguard or any other tunneling solution is paramount, not only to allow OpenWISP to work properly, but also to be able to perform debugging and troubleshooting when needed.

In this scenario, the following requirements are needed:

- a VPN server must be installed in a way that the OpenWISP server can reach the VPN peers, for more information on how to do this via OpenWISP please refer to the following sections:
  - OpenVPN tunnel automation
  - Wireguard tunnel automation

  If you prefer to use other tunneling solutions (L2TP, Softether, etc.) and know how to configure those solutions on your own, that's totally fine as well.

  If the OpenWISP server is connected to a network infrastructure which allows it to reach the devices via preexisting tunneling or Intranet solutions (e.g.: MPLS, SD-WAN), then setting up a VPN server is not needed, as long as there's a dedicated interface on OpenWrt which gets an IP address assigned to it and which is reachable from the OpenWISP server.
- The devices must be configured to join the management tunnel automatically, either via a preexisting configuration in the firmware or via an OpenWISP Template.
- The openwisp-config agent on the devices must be configured to specify the `management_interface` option, the agent will communicate the IP of the management interface to the OpenWISP Server and OpenWISP will use the management IP for reaching the device.

For example, if the *management interface* is named `tun0`, the openwisp-config configuration should look like the following example:

```
# In /etc/config/openwisp on the device

config controller 'http'
    # ... other configuration directives ...
    option management_interface 'tun0'
```

2. LAN deployment

When the OpenWISP server and the network devices are deployed in the same L2 network (e.g.: an office LAN) and the OpenWISP server is reachable on the LAN address, OpenWISP can then use the **Last IP** field of the devices to reach them.

In this scenario it's necessary to set the "OPENWISP_MONITORING_MANAGEMENT_IP_ONLY" setting to `False`.

Find out more about OpenWISP Monitoring

For more information about the features offered by OpenWISP Monitoring and the related OpenWrt packages we refer to the following sections of their respective READMEs.

OpenWISP Monitoring Python/Django module

- List of the available features
- Passive vs Active Metric Collection
- Device Health Status
- Default Metrics
- Available Checks
- Rest API
- Django Settings

OpenWISP Monitoring OpenWrt packages

- Configuration options
- Collecting vs Sending
- Compiling openwisp-monitoring
- Debugging

Configuration Templates

Templates are designed to store configuration that can be reused by some or all the devices in the system.

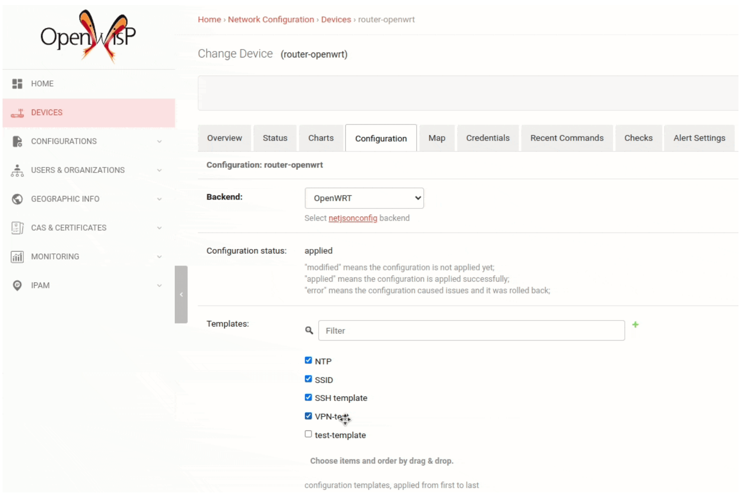
Updating the configuration stored in a template allows to update the configuration of all the devices that have that template assigned.

This means that configuration can be defined only once for multiple devices, and if the need to update a specific piece of configuration arises, it can be easily achieved by updating the template.

Table of Contents:

Template ordering and override

A device can use multiple templates, **the order in which templates are assigned to each device matters**: templates assigned last can override templates assigned earlier, the order can be changed by drag and dropping the template in the device configuration page as in the animated screenshot below.



Change Device (router-openwrt)

Configuration: router-openwrt

Backend: OpenWRT
Select netjsonconfig backend

Configuration status: applied
"modified" means the configuration is not applied yet;
"applied" means the configuration is applied successfully;
"error" means the configuration caused issues and it was rolled back;

Templates:
- [x] NTP
- [x] SSID
- [x] SSH template
- [x] VPN-test
- [ ] test-template

Choose items and order by drag & drop.

configuration templates, applied from first to last

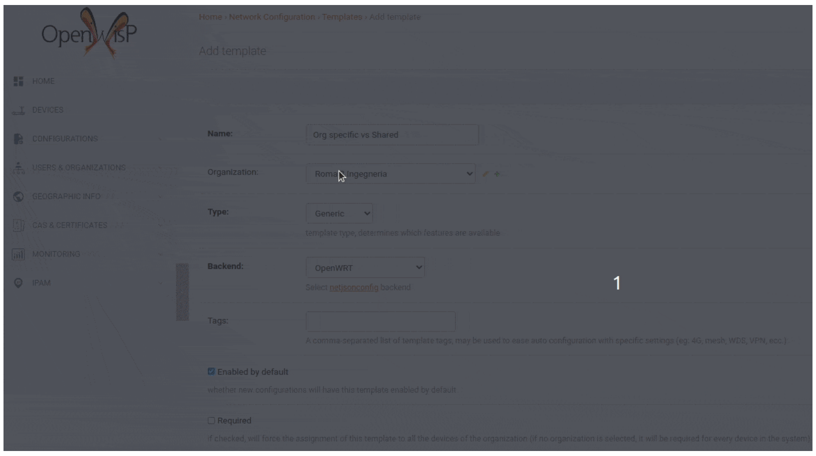
The device configuration can also override what is defined in templates.

Overriding means redefining a specific configuration section in a way that overwrites the template.

Overriding involves some form of duplication of information, which is not great, it should be used as a last resort. The recommended way to define parts of the configuration that are specific for each device is to use Configuration variables.

Shared templates vs organization specific

Templates can be *organization specific* or *shared* (no organization specified).



Home › Network Configuration › Templates › Add template

Add template

Name: Org specific vs Shared

Organization: Roma - Ingegneria

Type: Generic

template type, determines which features are available

Backend: OpenWRT

Select netjsonconfig backend

Tags:

A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

[✓] Enabled by default

whether new configurations will have this template enabled by default

[ ] Required

if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

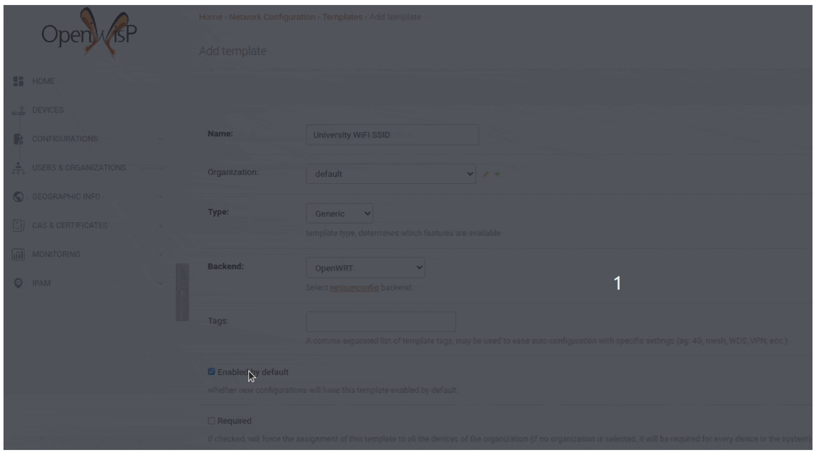
Organization specific templates will be available and usable only within the same organization which they are assigned to.

If no organization is specified when creating a template, a shared template will be created, **shared templates are available to any organization in the system**.

Here are a few typical use cases of shared templates:

- Management VPN
- Authorized SSH keys belonging to network administrators
- Crontab with generic periodic management operations

Default Templates



Home › Network Configuration › Templates › Add template

Add template

Name: University WiFi SSID

Organization: default

Type: Generic

template type, determines which features are available

Backend: OpenWRT

Select netjsonconfig backend

Tags:

A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

[✓] Enabled by default

whether new configurations will have this template enabled by default

[ ] Required

if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

When templates are flagged as **"Enabled by default"**, they will be automatically assigned to new devices.

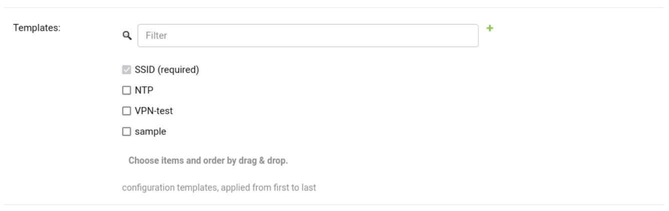
This is a very powerful feature: **once default templates are correctly configured to implement the use case you need, you will only have to register a device into OpenWISP for it to auto-configure itself**.

Moreover, you can change the default templates any time you need, which is the reason this feature has replaced the practice of storing default configuration in firmware images (which would need to be recompiled and redistributed): with default templates, the default firmware image only needs to contain the bare minimum configuration to connect to OpenWISP, once the device connects to OpenWISP it will download and apply the default templates without the need of manual intervention from the network operators.

An organization specific template flagged as default will be automatically assigned to any new device which will be created in the same organization.

A shared default template instead will be automatically assigned to all the new devices which will be created in the system, regardless of organization.

Required Templates



Templates:

Filter

- [✓] SSID (required)
- [ ] NTP
- [ ] VPN-test
- [ ] sample

Choose items and order by drag & drop.

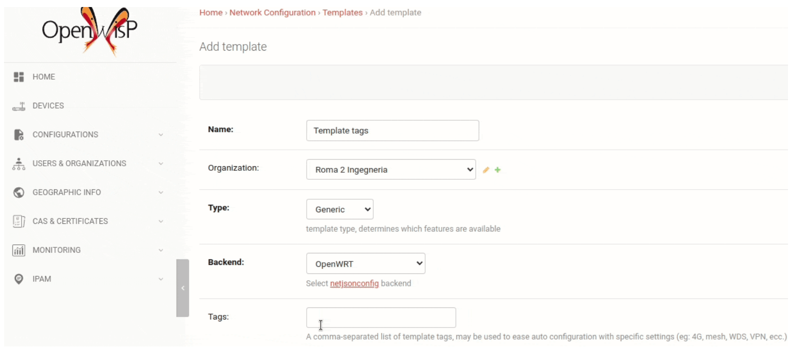
configuration templates, applied from first to last

Required templates are similar to Default templates but cannot be unassigned from a device configuration, they can only be overridden.

They will be always assigned earlier than default templates, so they can be overridden if needed.

In the example above, the "SSID" template is flagged as "(required)" and its checkbox is always checked and disabled.

Template tags



Home › Network Configuration › Templates › Add template

Add template

Name:	Template tags
Organization:	Roma 2 Ingegneria
Type:	Generic template type, determines which features are available
Backend:	OpenWRT Select netjsonconfig backend
Tags:	 A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

Default Templates are an incredibly useful tool, but they're limited: **only one set of default templates can be created**.

In some cases, you may have multiple set of default settings to use, let's explain this with a practical example: you may have 2 different device types in your network:

- Mesh routers: they connect to one another, forming a wireless mesh network
- Dumb access points: they connect to the mesh routers on the LAN port and offer internet access which is routed via the mesh network by the routers

In this example case, the default configuration to use in each device type can greatly differ.

In such a setup, default templates would only contain configuration which is common to both device types, while configuration which is specific for each type would be stored in specific templates which are then tagged with specific keywords:

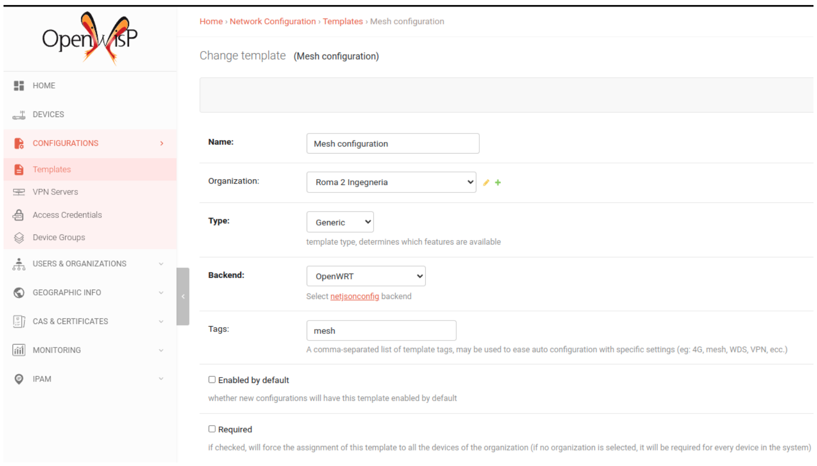
- `mesh`: tag to use for mesh configuration
- `dumb-ap`: tag to use for dumb AP configuration

The openwisp-config configuration of each device type must specify the correct tag before each device registers in the system.

Here's the sample `/etc/config/openwisp` configuration for mesh devices:

```
config controller 'http'
    option url 'https://openwisp2.mynetwork.com'
    option shared_secret 'mySharedSecret123'
    option tags 'mesh'
```

Once devices with the above configuration will register into the system, any template tagged as `mesh` (as in the screenshot below) will be assigned to them.



Home › Network Configuration › Templates › Mesh configuration

Change template (Mesh configuration)

Name: Mesh configuration

Organization: Roma 2 Ingegneria

Type: Generic

template type, determines which features are available

Backend: OpenWRT

Select netjsonconfig backend

Tags: mesh

A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☐ Enabled by default

whether new configurations will have this template enabled by default

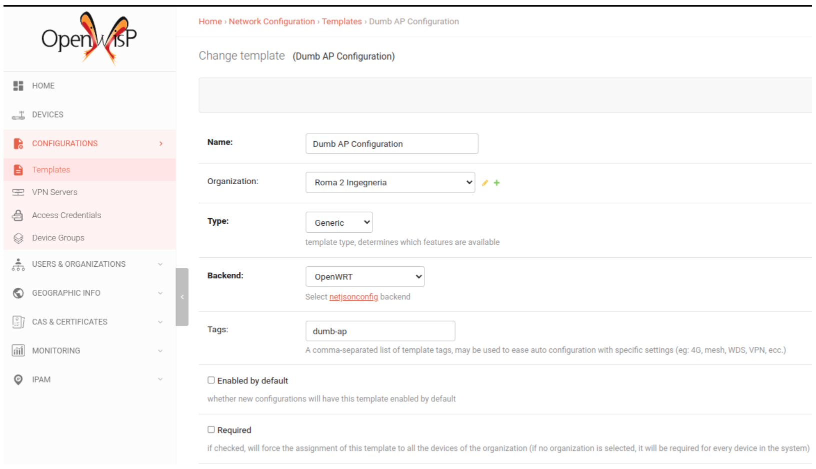
☐ Required

if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

The sample `/etc/config/openwisp` configuration for dumb access points is the following:

```
config controller 'http'
    option url 'https://openwisp2.mynetwork.com'
    option shared_secret 'mySharedSecret123'
    option tags 'dumb-ap'
```

Once devices with the above configuration will register into the system, any template tagged as `dumb-ap` (as in the screenshot below) will be assigned to them.



Home › Network Configuration › Templates › Dumb AP Configuration

Change template (Dumb AP Configuration)

Name: Dumb AP Configuration

Organization: Roma 2 Ingegneria

Type: Generic

template type, determines which features are available

Backend: OpenWRT

Select netjsonconfig backend

Tags: dumb-ap

A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☐ Enabled by default

whether new configurations will have this template enabled by default

☐ Required

if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

How to Use Configuration Variables

Sometimes the configuration is not exactly equal on all the devices, some parameters are unique to each device or need to be changed by the user.

In these cases it is possible to use configuration variables in conjunction with templates, this feature is also known as *configuration context*, think of it like a dictionary which is passed to the function which renders the configuration, so that it can fill variables according to the passed context.

The different ways in which variables are defined are described below.

Table of Contents:

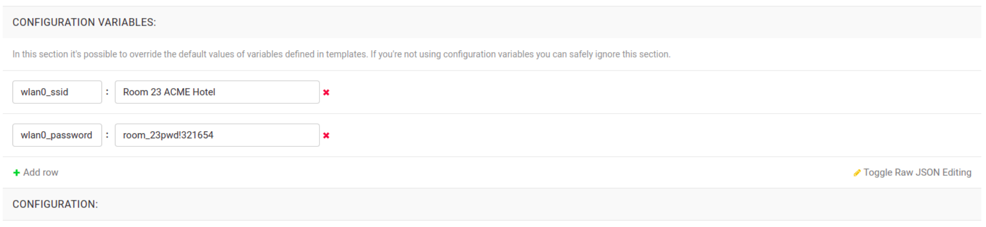
Predefined device variables

Each device gets the following attributes passed as configuration variables:

- `id`
- `key`
- `name`
- `mac_address`

User defined device variables

In the device configuration section you can find a section named "Configuration variables" where it is possible to define the configuration variables and their values, as shown in the example below:


CONFIGURATION VARIABLES:

In this section it's possible to override the default values of variables defined in templates. If you're not using configuration variables you can safely ignore this section.

wlan0_ssid : Room 23 ACME Hotel

wlan0_password : room_23pwd!321654

Template default values

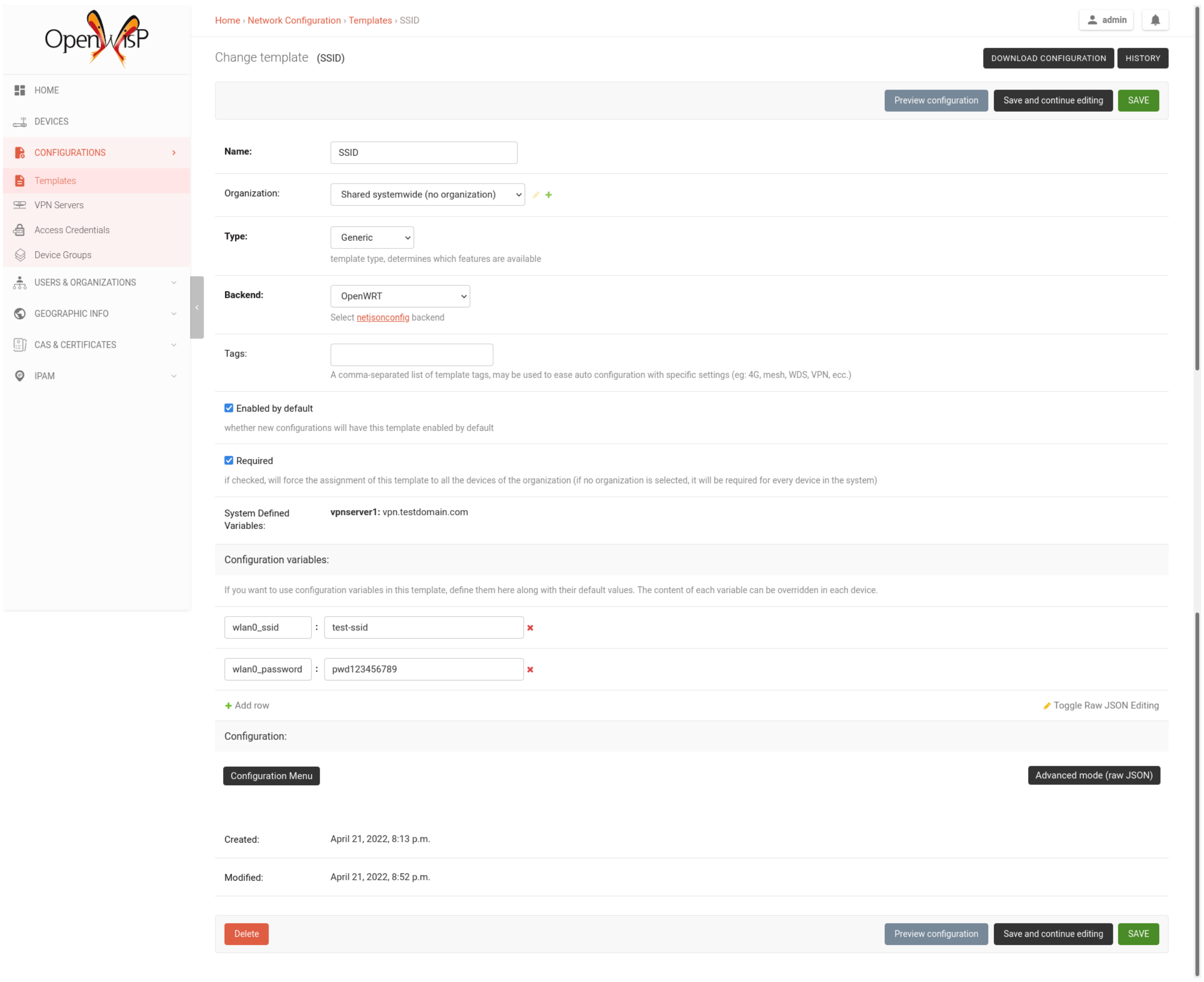
It's possible to specify the default values of variables defined in a template.

This allows to achieve 2 goals:

1. pass schema validation without errors (otherwise it would not be possible to save the template in the first place)
2. provide good default values that are valid in most cases but can be overridden in the device if needed

These default values will be overridden by the User defined device variables.

The default values of variables can be manipulated from the section "configuration variables" in the edit template page:



Change template (SSID)

Name:	SSID
Organization:	Shared systemwide (no organization)
Type:	Generic template type, determines which features are available
Backend:	OpenWRT Select netjsonconfig backend
Tags:	 A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☑ Enabled by default
whether new configurations will have this template enabled by default

☑ Required
if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

System Defined Variables: **vpnserver1:** vpn.testdomain.com

Configuration variables:

If you want to use configuration variables in this template, define them here along with their default values. The content of each variable can be overridden in each device.

wlan0_ssid	test-ssid
wlan0_password	pwd123456789

Configuration:

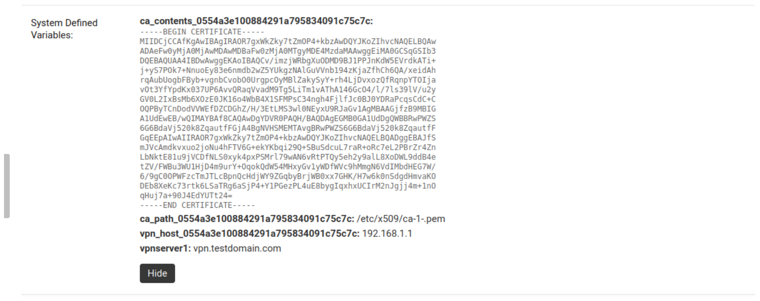
Created:	April 21, 2022, 8:13 p.m.
Modified:	April 21, 2022, 8:52 p.m.

Global variables

Variables can also be defined globally using the OPENWISP_CONTROLLER_CONTEXT setting.

System defined variables

Predefined device variables, global variables and other variables that are automatically managed by the system (e.g.: when using templates of type VPN-client) are displayed in the admin UI as *System Defined Variables* in read-only mode.



System Defined Variables:

ca_contents_0554a3e100884291a795834091c75c7c:

```
-----BEGIN CERTIFICATE-----
MIIDCjCCAfKgAwIBAgIRAOR7gxWkZky7tZmOP4+kbzAwDQYJKoZIhvcNAQELBQAw
ADAeFw0yMjA0MjAwMDAwMDBaFw0zMjA0MTgyMDE4MzdaMAAwggEiMA0GCSqGSIb3
DQEBAQUAA4IBDwAwggEKAoIBAQCv/imzjWRbgXuODMD9BJ1PPJnKdW5EVrdkATi+
j+yS7POk7+NnuoEy83e6nmdb2wZ5YUkgzNAlGuVVnb194zKjaZfhCh6QA/xeidAh
rqAubUogbFByb+vgnbCvobO0UrgpcOyMBlZakySyY+rh4LjDvxozQfRqnpYTOIja
vOt3YfYpdKx037UP6AvvQRaqVvadM9Tg5LiTm1vAThA146GcO4/l/7ls39lV/u2y
GV0L2IxBsMb6XOzE0JK16o4WbB4X1SFMPsC34ngh4FjlfJc0BJ0YDRaPcqsCdC+C
OQPByTCnDodVVWEfDZCDGhZ/H/3EtLMS3wl0NEyxU9RJaGv1AgMBAAGjfzB9MBIG
A1UdEwEB/wQIMAYBAf8CAQAwDgYDVR0PAQH/BAQDAgEGMB0GA1UdDgQWBBRwPWZS
6G6BdaVj520k8ZqautfFGjA4BgNVHSMEMTAvgBRwPWZS6G6BdaVj520k8ZqautfF
GqEEpAIwAIIRAOR7gxWkZky7tZmOP4+kbzAwDQYJKoZIhvcNAQELBQADggEBAJfS
mJVcAmdkvxuo2joNu4hFTV6G+ekYKbqi29Q+SBuSdcuL7raR+oRc7eL2PBrZr4Zn
LbNktE81u9jVCDfNLS0xyk4pxPSMrl79wAN6vRtPTQy5eh2y9alL8XoDWL9ddB4e
tZV/FWBu3WU1HjD4m9urY+OqokQdW54MHxyGv1yWDfWVc9hMmgN6VdIMbdHEG7W/
6/9gC0OPWFzcTmJTLcBpnQcHdjWY9ZGqbyBrjWB0xx7GHK/H7w6k0nSdgdHmvaKO
DEb8XeKc73rtk6LSaTRg6aSjP4+Y1PGezPL4uE8bygIqxhxUCIrM2nJgjj4m+1nO
qHuj7a+90J4EdYUTt24=
-----END CERTIFICATE-----
```

ca_path_0554a3e100884291a795834091c75c7c: /etc/x509/ca-1-.pem

vpn_host_0554a3e100884291a795834091c75c7c: 192.168.1.1

vpnserver1: vpn.testdomain.com

Example usage of variables

Here's a typical use case, the WiFi SSID and WiFi password. You don't want to define this for every device, but you may want to allow operators to easily change the SSID or WiFi password for a specific device without having to re-define the whole wifi interface to avoid duplicating information.

This would be the template:

```
{
    "interfaces": [
        {
            "type": "wireless",
            "name": "wlan0",
            "wireless": {
                "mode": "access_point",
                "radio": "radio0",
                "ssid": "{{wlan0_ssid}}",
                "encryption": {
                    "protocol": "wpa2_personal",
                    "key": "{{wlan0_password}}",
                    "cipher": "auto"
                }
            }
        }
    ]
}
```

These would be the default values in the template:

```
{
    "wlan0_ssid": "SnakeOil PublicWiFi",
    "wlan0_password": "Snakeoil_pwd!321654"
}
```

The default values can then be overridden at device level if needed, e.g.:

```
{
    "wlan0_ssid": "Room 23 ACME Hotel",
    "wlan0_password": "room_23pwd!321654"
}
```

OpenVPN tunnel Automation

In this guide we will explore how to set up the automatic management of **OpenVPN tunnels**.

If you're interested in **Wireguard tunnels** see Wireguard and Wireguard over VXLAN tunnel automation.

Table of Contents:

Installing OpenVPN Server and importing the OpenVPN configuration

We will be installing OpenVPN Server using this ansible role Stouts.openvpn and then import the VPN configuration into OpenWISP. If you have already setup your VPN server or would like to install the VPN server via some other way, you can skip to Step 4.

Note

This process is not automated yet.

1. Install Ansible and required Ansible roles

Install ansible (version 2.5 or higher) **on your local machine** (not on the server!) if you haven't done already.

To **install ansible** we suggest you follow the official ansible installation guide .

After having installed ansible, **you need to install git** (example for linux debian/ubuntu systems):

```
sudo apt-get install git
```

After having ansible and git installed, install the required roles:

```
ansible-galaxy install git+https://github.com/Stouts/Stouts.openvpn,3.0.0 nkakouros.easyrsa
```

2. Create hosts file and ansible playbook

Create an ansible inventory file named `hosts` **on your local machine and not in the server** with the following contents:

```
[openvpn]
your_server_domain_or_ip
```

For e.g. if your server ip is `192.168.56.2`:

```
[openvpn]
192.168.56.2
```

In the same directory where you created the `host` file, create a file named `playbook.yml` which contains the following:

```
- hosts: openvpn
  vars:
    # EasyRSA
    easyrsa_generate_dh: true
    easyrsa_servers:
      - name: server
    easyrsa_clients: []
    easyrsa_pki_dir: /etc/easyrsa/pki

    # OpenVPN
    openvpn_keydir: "{{ easyrsa_pki_dir }}"
    openvpn_clients: []
    openvpn_use_pam: false
  roles:
    - role: nkakouros.easyrsa
    - role: Stouts.openvpn
```

Hint

You can further customize the Configuration using the role variables. Get Info about other options in EasyRSA and OpenVPN

3. Run the Playbook

Run the ansible playbook using:

```
ansible-playbook -i hosts playbook.yml -b -k -K --become-method=su
```

4. Importing the CA and the Server Certificate

To import the CA and Server Certificate, you need to access your server via `ssh` or any other method that suits you.

You need to change your directory to `/etc/easyrsa/pki/`

Note

If you face `-bash: cd: /etc/easyrsa/pki: Permission denied` you may need to login as root user.

Importing the CA:

On your openwisp dashboard go to `/admin/pki/ca/add/`

In **Operation Type** choose Import Existing

Get your CA Certificate from `ca.crt` file and Private Key from `private/ca.key` and then enter them in the respective fields.

Importing the Server Certificate:

On your openwisp dashboard go to `/admin/pki/cert/add/`

In **Operation Type** choose Import Existing and in **CA** choose the CA you just created.

Get your Server Certificate from `issued/server.crt` file and Server Private Key from `private/server.key` and then enter them in the respective fields.

4. Creating VPN Server

On your openwisp dashboard go to `/admin/config/vpn/add/`

In **Host** enter you Server IP, in **Certification Authority** select the CA you created and in **X509 Certificate** select the certificate you created.

Now under **Configuration**, open **Configuration Menu** and deselect Property Files. For **VPN1** Change Server (Bridged) to the Type of your VPN Server. The VPN Server installed using the guide above is a Routed Server so change the Type to Server (Routed). The Process to setup a Bridged Server is identical to that of Routed Server.

Change the rest of the Configuration of the VPN according to the configuration in `/etc/openvpn/server.conf`

Tip

You can check if your VPN Configuration is similar to the `server.conf` file using the **Preview Configuration** option at the Top.

Preparing the configuration template for VPN Clients

Create VPN Template

On your openwisp dashboard go to `/admin/config/template/add/`.

Change **Type** to VPN-client. For **VPN** select the VPN you created in the previous steps.

You can further toggle Enabled by default and Auto certificate options according to your needs.

Save the template. You can now tweak the Client VPN configuration. Now can add the template to your devices.

Auto Client Certificates

Option: `Auto certificate`

Default: `True`

Auto Client Certificates feature allows you to automatically generate generate client certificates for your Device.

Default Templates

Option: `Enabled by default`

Default: `False`

Default templates are automatically added to newly created devices of the organization of the template. If no organization is specified, the template is added to all devices of all the organizations.

Wireguard and Wireguard over VXLAN tunnel automation

In this guide we will explore how to set up the automatic management of Wireguard or Wireguard over VXLAN tunnels.

Table of Contents:

How to setup WireGuard tunnels

Follow the procedure described below to setup WireGuard tunnels on your devices.

Note: This example uses **Shared systemwide (no organization)** option as the organization for VPN server and VPN client template. You can use any organization as long as VPN server, VPN client template and Device has same organization.

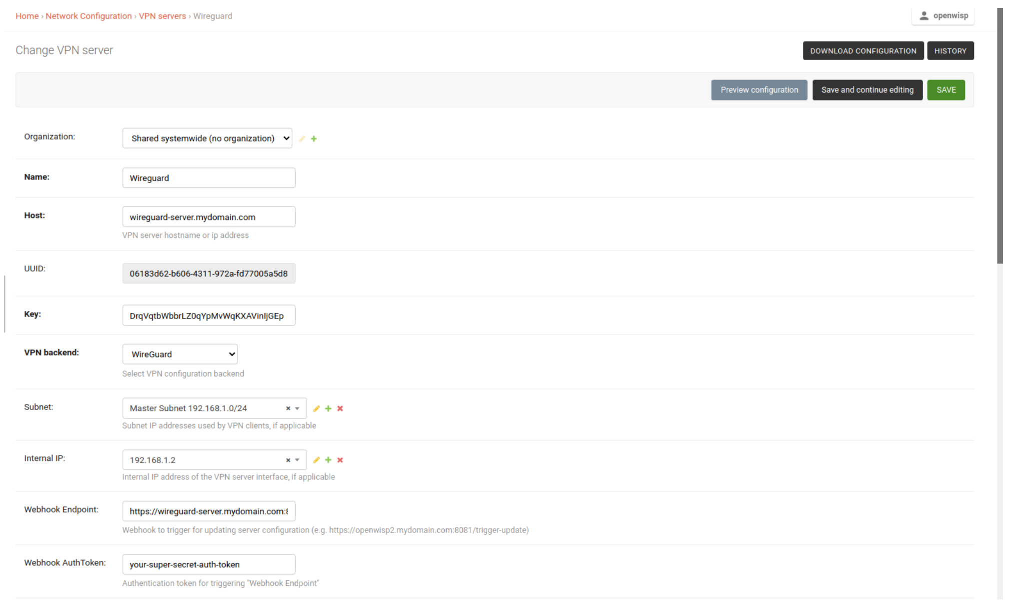
1. Create VPN server configuration for WireGuard

1. Visit `/admin/config/vpn/add/` to add a new VPN server.
2. We will set **Name** of this VPN server `Wireguard` and **Host** as `wireguard-server.mydomain.com` (update this to point to your WireGuard VPN server).
3. Select `WireGuard` from the dropdown as **VPN Backend**.
4. When using WireGuard, OpenWISP takes care of managing IP addresses (assigning an IP address to each VPN peer). You can create a new subnet or select an existing one from the dropdown menu. You can also assign an **Internal IP** to the WireGuard Server or leave it empty for OpenWISP to configure. This IP address will be used by the WireGuard interface on server.
5. We have set the **Webhook Endpoint** as `https://wireguard-server.mydomain.com:8081/trigger-update` for this example. You will need to update this according to you VPN upgrader endpoint. Set **Webhook AuthToken** to any strong passphrase, this will be used to ensure that configuration upgrades are requested from trusted sources.

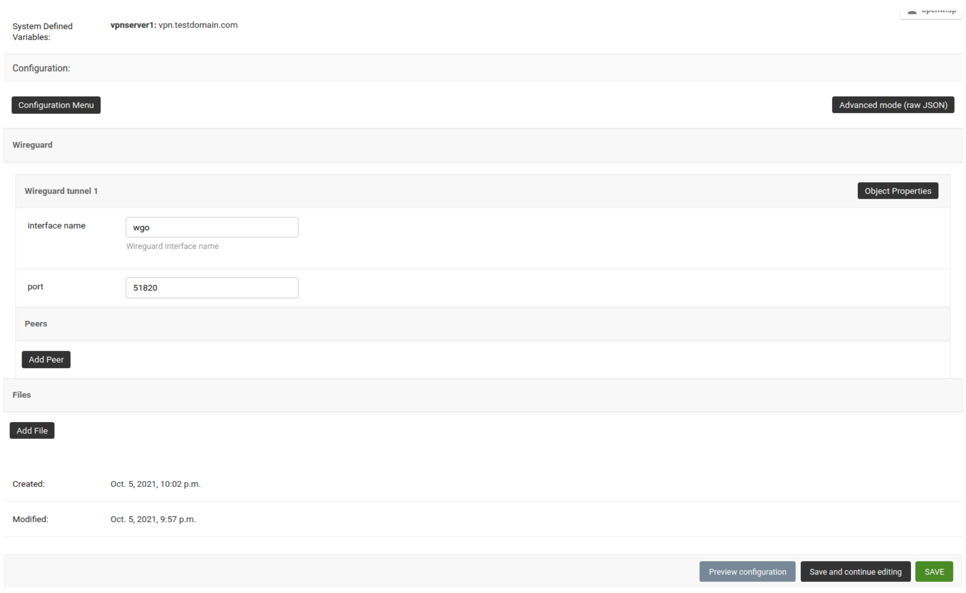
   Note

   If you are following this tutorial for also setting up WireGuard VPN server, just substitute `wireguard-server.mydomain.com` with hostname of your VPN server and follow the steps in next section.

6. Under the configuration section, set the name of WireGuard tunnel `1` interface. We have used `wg0` in this example.


Home › Network Configuration › VPN servers › Wireguard
openwisp
Change VPN server
DOWNLOAD CONFIGURATION
HISTORY
Preview configuration
Save and continue editing
SAVE

Organization:	Shared systemwide (no organization)
Name:	Wireguard
Host:	wireguard-server.mydomain.com VPN server hostname or ip address
UUID:	06183d62-b606-4311-972a-fd77005a5d8
Key:	DrqVqtbWbbrLZ0qYpMvWqKXAVinIjGEp
VPN backend:	WireGuard Select VPN configuration backend
Subnet:	Master Subnet 192.168.1.0/24 Subnet IP addresses used by VPN clients, if applicable
Internal IP:	192.168.1.2 Internal IP address of the VPN server interface, if applicable
Webhook Endpoint:	https://wireguard-server.mydomain.com:8 Webhook to trigger for updating server configuration (e.g. https://openwisp2.mydomain.com:8081/trigger-update)
Webhook AuthToken:	your-super-secret-auth-token Authentication token for triggering "Webhook Endpoint"



System Defined Variables: **vpnserver1:** vpn.testdomain.com

Configuration:

Wireguard

Wireguard tunnel 1

interface name: wgo
Wireguard interface name

port: 51820

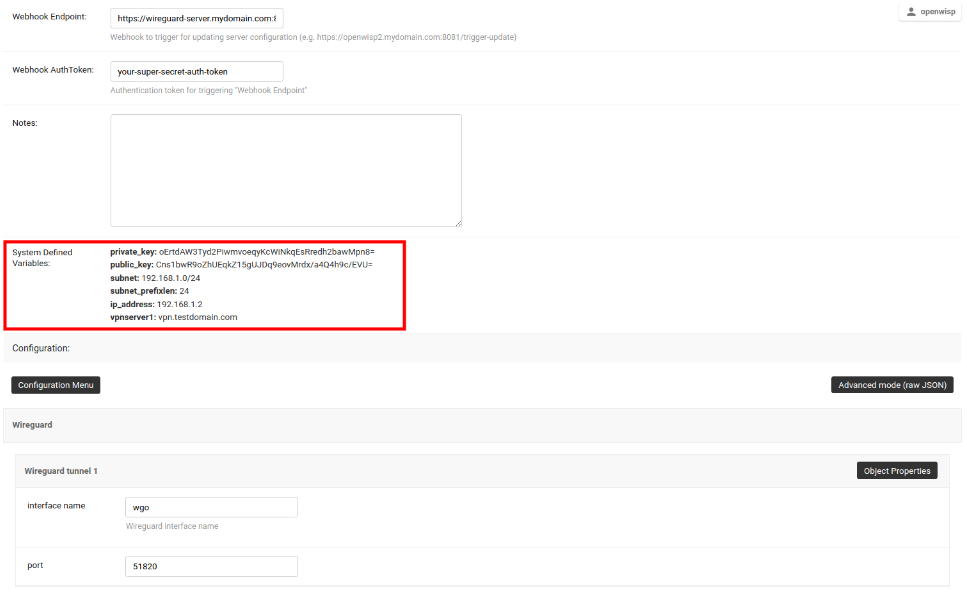
Peers

Files

Created: Oct. 5, 2021, 10:02 p.m.

Modified: Oct. 5, 2021, 9:57 p.m.

7. After clicking on **Save and continue editing**, you will see that OpenWISP has automatically created public and private key for WireGuard server in **System Defined Variables** along with internal IP address information.



Webhook Endpoint: https://wireguard-server.mydomain.com:8
Webhook to trigger for updating server configuration (e.g. https://openwisp2.mydomain.com:8081/trigger-update)

Webhook AuthToken: your-super-secret-auth-token
Authentication token for triggering "Webhook Endpoint"

Notes:

System Defined Variables:
private_key: oErtdAW3Tyd2PiwmvoeqyKcWiNkqEsRredh2bawMpn8=
public_key: Cns1bwR9oZhUEqkZ15gUJDq9eovMrdx/a4Q4h9c/EVU=
subnet: 192.168.1.0/24
subnet_prefixlen: 24
ip_address: 192.168.1.2
vpnserver1: vpn.testdomain.com

Configuration:

Wireguard

Wireguard tunnel 1

interface name: wgo
Wireguard interface name

port: 51820

2. Deploy Wireguard VPN Server

If you haven't already setup WireGuard on your VPN server, this will be a good time do so.

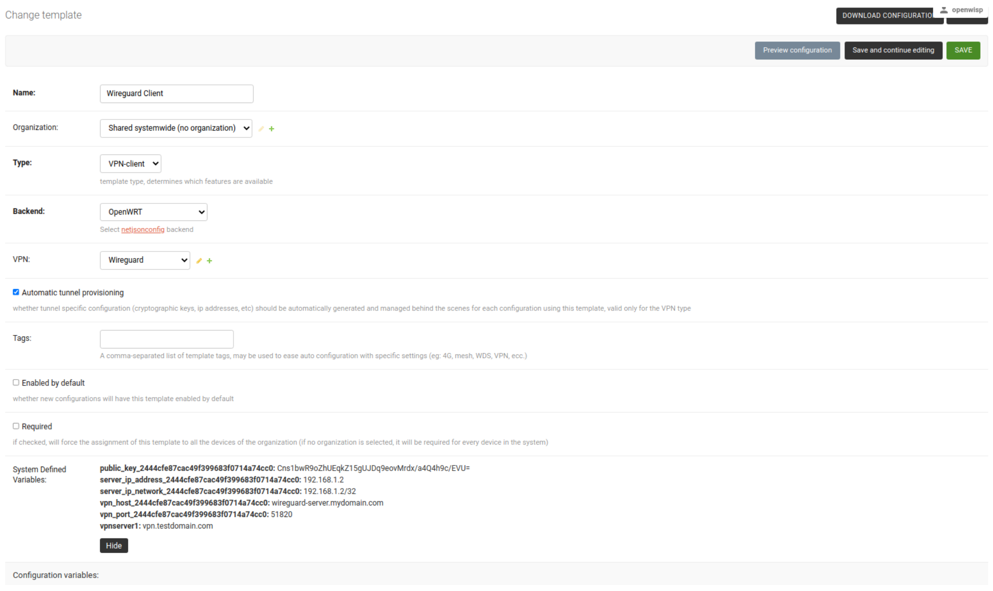
We recommend using the ansible-wireguard-openwisp role for installing WireGuard since it also installs scripts that allows OpenWISP to manage WireGuard VPN server.

Pay attention to the VPN server attributes used in your playbook. It should be same as VPN server configuration in OpenWISP.

3. Create VPN client template for WireGuard VPN Server

1. Visit `/admin/config/template/add/` to add a new template.
2. Set `Wireguard Client` as **Name** (you can set whatever you want) and select `VPN-client` as **type** from the dropdown list.
3. The **Backend** field refers to the backend of the device this template can be applied to. For this example, we will leave it to `OpenWRT`.
4. Select the correct VPN server from the dropdown for the **VPN** field. Here it is `Wireguard`.

5. Ensure that **Automatic tunnel provisioning** is checked. This will make OpenWISP to automatically generate public and private keys and provision IP address for each WireGuard VPN client.
6. After clicking on **Save and continue editing** button, you will see details of *Wireguard* VPN server in **System Defined Variables**. The template configuration will be automatically generated which you can tweak accordingly. We will use the automatically generated VPN client configuration for this example.



Name: Wireguard Client

Organization: Shared systemwide (no organization)

Type: VPN-client
template type, determines which features are available

Backend: OpenWRT
Select netjsonconfig backend

VPN: Wireguard

☑ Automatic tunnel provisioning
whether tunnel specific configuration (cryptographic keys, ip addresses, etc) should be automatically generated and managed behind the scenes for each configuration using this template, valid only for the VPN type

Tags:
A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☐ Enabled by default
whether new configurations will have this template enabled by default

☐ Required
if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

System Defined Variables:
public_key_2444cfe87cac49f399683f0714a74cc0: Cns1bwR9oZhUEqkZ15gUJDq9eovMrdx/a4Q4h9c/EVU=
server_ip_address_2444cfe87cac49f399683f0714a74cc0: 192.168.1.2
server_ip_network_2444cfe87cac49f399683f0714a74cc0: 192.168.1.2/32
vpn_host_2444cfe87cac49f399683f0714a74cc0: wireguard-server.mydomain.com
vpn_port_2444cfe87cac49f399683f0714a74cc0: 51820
vpnserver1: vpn.testdomain.com

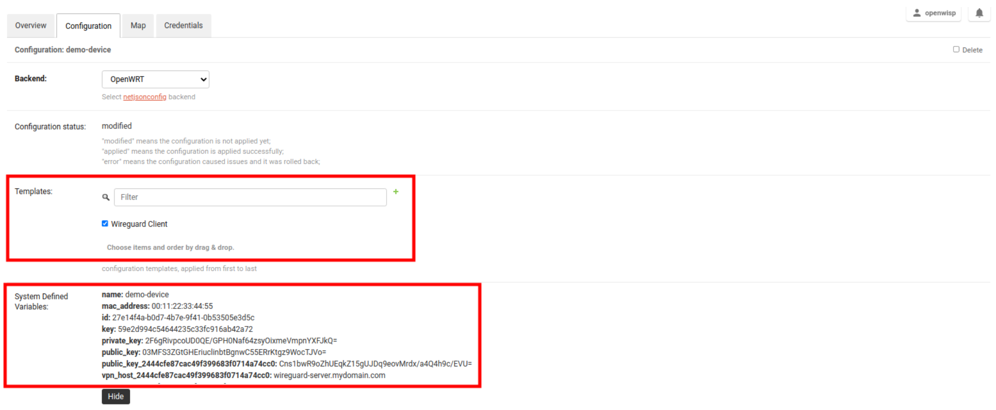
Configuration variables:

4. Apply Wireguard VPN template to devices

Note

This step assumes that you already have a device registered on OpenWISP. Register or create a device before proceeding.

1. Open the **Configuration** tab of the concerned device.
2. Select the *WireGuard Client* template.
3. Upon clicking on **Save and continue editing** button, you will see some entries in **System Defined Variables**. It will contain internal IP address, private and public key for the WireGuard client on the device along with details of WireGuard VPN server.



Configuration: demo-device

Backend: OpenWRT
Select netjsonconfig backend

Configuration status: modified
"modified" means the configuration is not applied yet;
"applied" means the configuration is applied successfully;
"error" means the configuration caused issues and it was rolled back;

Templates:
☑ Wireguard Client
Choose items and order by drag & drop.
configuration templates, applied from first to last

System Defined Variables:
name: demo-device
mac_address: 00:11:22:33:44:55
id: 27e14f4a-b0d7-4b7e-9f41-0b53505e3d5c
key: 59e2d994c54644235c33fc916ab42a72
private_key: 2F6gRivpcoUD0QE/GPH0Naf64zsyOixmeVmpnYXFJkQ=
public_key: 03MFS3ZGtGHEriuclinbtBgnwC55ERrKtgz9WocTJVo=
public_key_2444cfe87cac49f399683f0714a74cc0: Cns1bwR9oZhUEqkZ15gUJDq9eovMrdx/a4Q4h9c/EVU=
vpn_host_2444cfe87cac49f399683f0714a74cc0: wireguard-server.mydomain.com

Voila! You have successfully configured OpenWISP to manage WireGuard tunnels for your devices.

How to setup VXLAN over WireGuard tunnels

By following these steps, you will be able to setup layer 2 VXLAN tunnels encapsulated in WireGuard tunnels which work on layer 3.

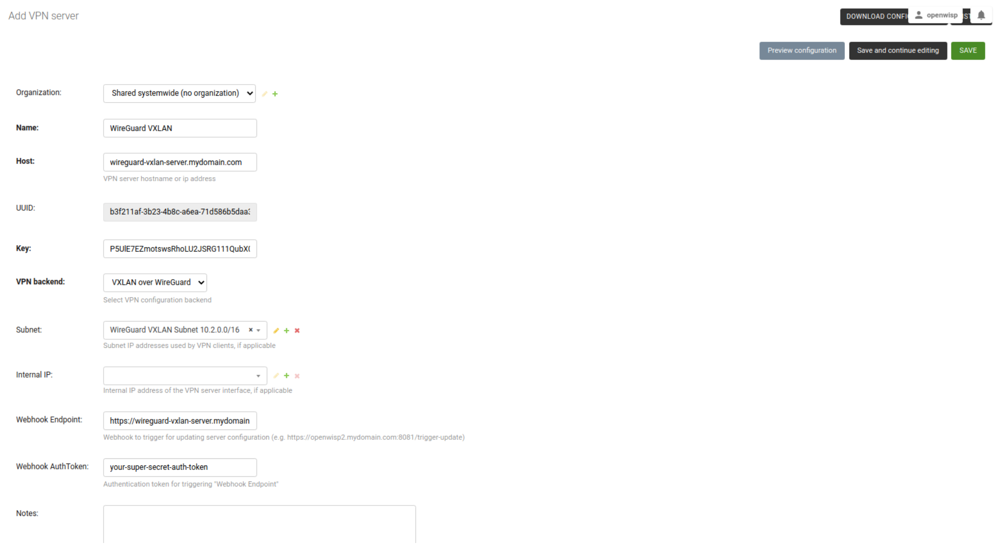
Note: This example uses **Shared systemwide (no organization)** option as the organization for VPN server and VPN client template. You can use any organization as long as VPN server, VPN client template and Device has same organization.

1. Create VPN server configuration for VXLAN over WireGuard

1. Visit `/admin/config/vpn/add/` to add a new VPN server.
2. We will set **Name** of this VPN server `Wireguard VXLAN` and **Host** as `wireguard-vxlan-server.mydomain.com` (update this to point to your WireGuard VXLAN VPN server).
3. Select `VXLAN over WireGuard` from the dropdown as **VPN Backend**.
4. When using VXLAN over WireGuard, OpenWISP takes care of managing IP addresses (assigning an IP address to each VPN peer). You can create a new subnet or select an existing one from the dropdown menu. You can also assign an **Internal IP** to the WireGuard Server or leave it empty for OpenWISP to configure. This IP address will be used by the WireGuard interface on server.
5. We have set the **Webhook Endpoint** as `https://wireguard-vxlan-server.mydomain.com:8081/trigger-update` for this example. You will need to update this according to you VPN upgrader endpoint. Set **Webhook AuthToken** to any strong passphrase, this will be used to ensure that configuration upgrades are requested from trusted sources.

> **Note**
>
> If you are following this tutorial for also setting up WireGuard VPN server, just substitute `wireguard-server.mydomain.com` with hostname of your VPN server and follow the steps in next section.

6. Under the configuration section, set the name of WireGuard tunnel 1 interface. We have used `wg0` in this example.



Organization: Shared systemwide (no organization)

Name: WireGuard VXLAN

Host: wireguard-vxlan-server.mydomain.com
VPN server hostname or ip address

UUID: b3f211af-3b23-4b8c-a6ea-71d586b5daa

Key: P5UlE7EZmotswsRhoLU2JSRG111QubX

VPN backend: VXLAN over WireGuard
Select VPN configuration backend

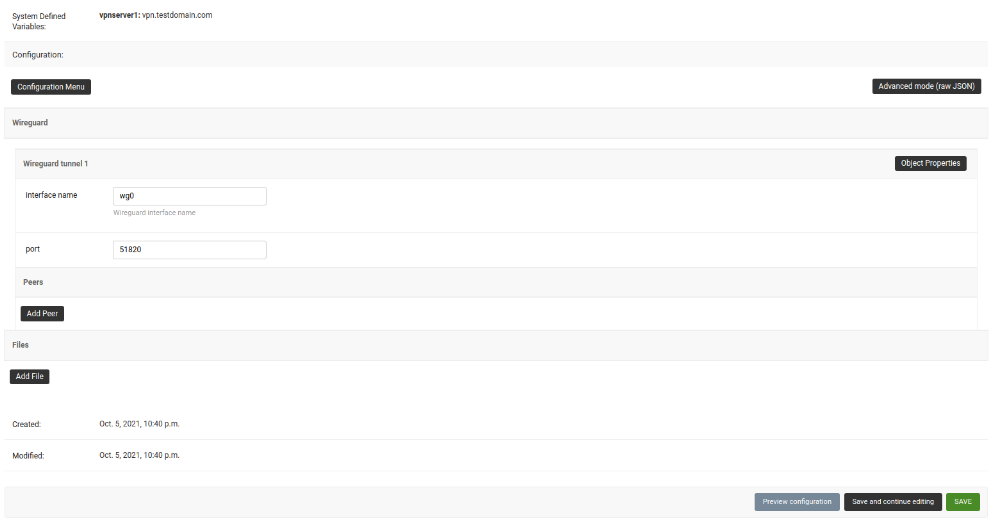
Subnet: WireGuard VXLAN Subnet 10.2.0.0/16
Subnet IP addresses used by VPN clients, if applicable

Internal IP:
Internal IP address of the VPN server interface, if applicable

Webhook Endpoint: https://wireguard-vxlan-server.mydomain
Webhook to trigger for updating server configuration (e.g. https://openwisp2.mydomain.com:8081/trigger-update)

Webhook AuthToken: your-super-secret-auth-token
Authentication token for triggering "Webhook Endpoint"

Notes:



System Defined Variables: **vpnserver1:** vpn.testdomain.com

Configuration:

Wireguard

Wireguard tunnel 1

interface name: wg0

port: 51820

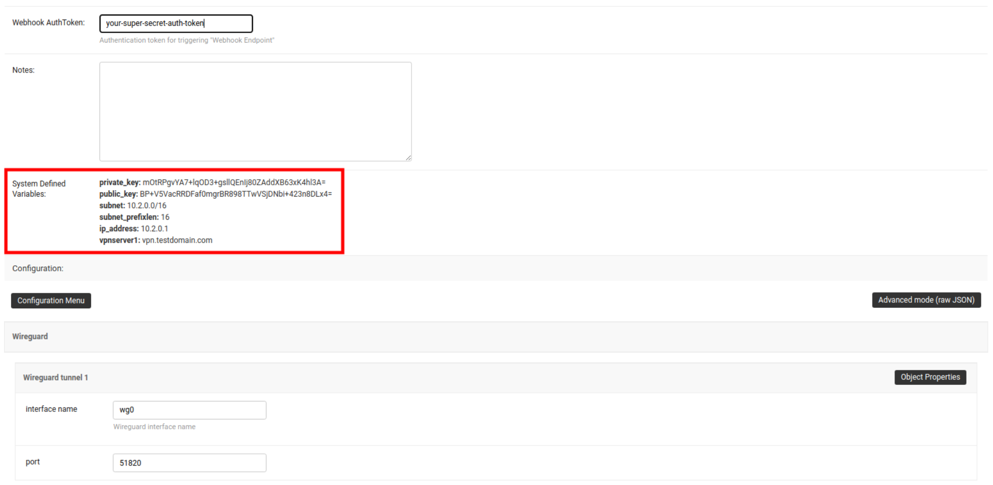
Peers

Files

Created: Oct. 5, 2021, 10:40 p.m.

Modified: Oct. 5, 2021, 10:40 p.m.

7. After clicking on **Save and continue editing**, you will see that OpenWISP has automatically created public and private key for WireGuard server in **System Defined Variables** along with internal IP address information.



Webhook AuthToken: your-super-secret-auth-token

Authentication token for triggering "Webhook Endpoint"

Notes:

System Defined Variables:
private_key: mOtRPgvYA7+lqOD3+gsllQEnIj80ZAddXB63xK4hl3A=
public_key: BP+V5VacRRDFaf0mgrBR898TTwVSjDNbi+423n8DLx4=
subnet: 10.2.0.0/16
subnet_prefixlen: 16
ip_address: 10.2.0.1
vpnserver1: vpn.testdomain.com

Configuration:

Wireguard

Wireguard tunnel 1

interface name: wg0

port: 51820

2. Deploy Wireguard VXLAN VPN Server

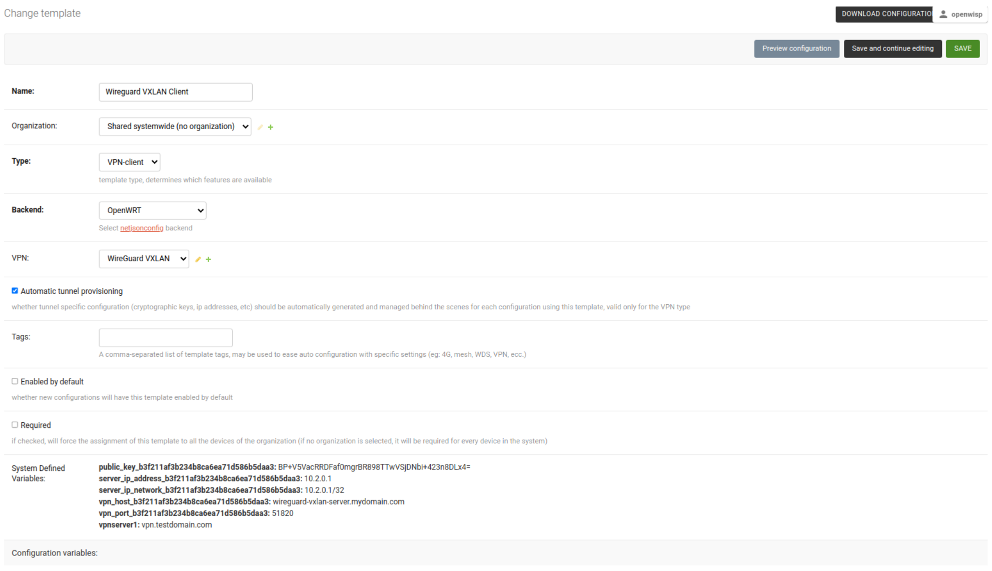
If you haven't already setup WireGuard on your VPN server, this will be a good time do so. We recommend using the ansible-wireguard-openwisp role for installing WireGuard since it also installs scripts that allows OpenWISP to manage WireGuard VPN server along with VXLAN tunnels.

Pay attention to the VPN server attributes used in your playbook. It should be same as VPN server configuration in OpenWISP.

3. Create VPN client template for WireGuard VXLAN VPN Server

1. Visit `/admin/config/template/add/` to add a new template.
2. Set `Wireguard VXLAN Client` as **Name** (you can set whatever you want) and select `VPN-client` as **type** from the dropdown list.
3. The **Backend** field refers to the backend of the device this template can be applied to. For this example, we will leave it to `OpenWRT`.
4. Select the correct VPN server from the dropdown for the **VPN** field. Here it is `Wireguard VXLAN`.
5. Ensure that **Automatic tunnel provisioning** is checked. This will make OpenWISP to automatically generate public and private keys and provision IP address for each WireGuard VPN client along with VXLAN Network Identifier (VNI).

6. After clicking on **Save and continue editing** button, you will see details of *Wireguard VXLAN* VPN server in **System Defined Variables**. The template configuration will be automatically generated which you can tweak accordingly. We will use the automatically generated VPN client configuration for this example.


Change template

Name: Wireguard VXLAN Client

Organization: Shared systemwide (no organization)

Type: VPN-client

template type, determines which features are available

Backend: OpenWRT

Select netjsonconfig backend

VPN: WireGuard VXLAN

☑ Automatic tunnel provisioning

whether tunnel specific configuration (cryptographic keys, ip addresses, etc) should be automatically generated and managed behind the scenes for each configuration using this template, valid only for the VPN type

Tags:

A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☐ Enabled by default

whether new configurations will have this template enabled by default

☐ Required

if checked, will force the assignment of this template to all the devices of the organization (if no organization is selected, it will be required for every device in the system)

System Defined Variables:

public_key_b3f211af3b234b8ca6ea71d586b5daa3: BP+V5VacRRDFaf0mgrBR898TTwVSjDNbi+423n8DLx4=
server_ip_address_b3f211af3b234b8ca6ea71d586b5daa3: 10.2.0.1
server_ip_network_b3f211af3b234b8ca6ea71d586b5daa3: 10.2.0.1/32
vpn_host_b3f211af3b234b8ca6ea71d586b5daa3: wireguard-vxlan-server.mydomain.com
vpn_port_b3f211af3b234b8ca6ea71d586b5daa3: 51820
vpnserver1: vpn.testdomain.com

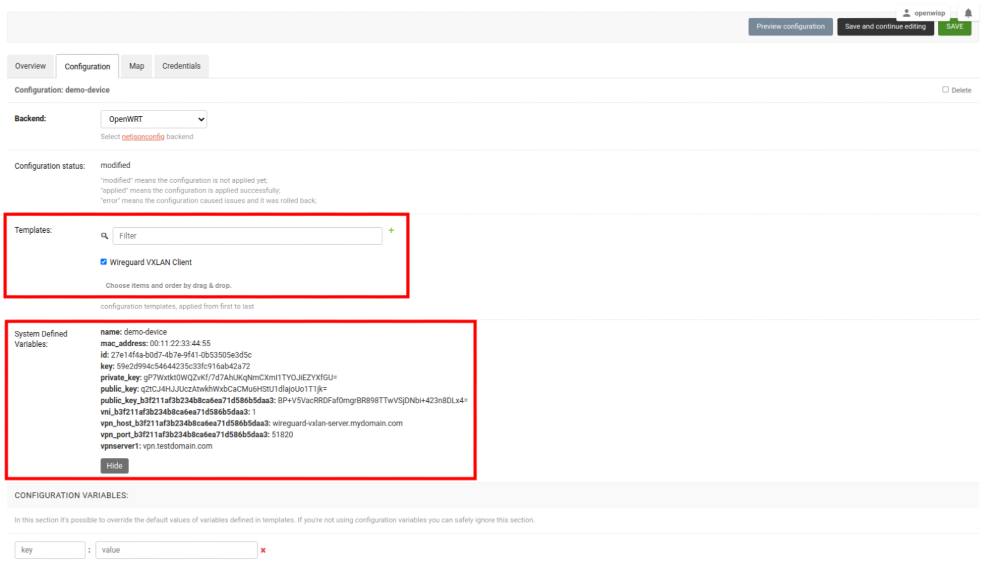
Configuration variables:

4. Apply Wireguard VXLAN VPN template to devices

Note

This step assumes that you already have a device registered on OpenWISP. Register or create a device before proceeding.

1. Open the **Configuration** tab of the concerned device.
2. Select the *WireGuard VXLAN Client* template.
3. Upon clicking on **Save and continue editing** button, you will see some entries in **System Defined Variables**. It will contain internal IP address, private and public key for the WireGuard client on the device and details of WireGuard VPN server along with VXLAN Network Identifier (VNI) of this device.



Configuration: demo-device

Backend: OpenWRT

Select netjsonconfig backend

Configuration status: modified

"modified" means the configuration is not applied yet;
"applied" means the configuration is applied successfully;
"error" means the configuration caused issues and it was rolled back;

Templates:

☑ Wireguard VXLAN Client

Choose items and order by drag & drop.

configuration templates, applied from first to last

System Defined Variables:

name: demo-device
mac_address: 00:11:22:33:44:55
id: 27e14f4a-b0d7-4b7e-9f41-0b53505e3d5c
key: 59e2d994c54644235c33fc916ab42a72
private_key: gP7Wxtkt0WQZvKf/7d7AhUKqNmCXmI1TYOJiEZYXfGU=
public_key: q2tCJ4HJJUczAtwkhWxbCaCMu6HStU1dlajoUo1T1jk=
public_key_b3f211af3b234b8ca6ea71d586b5daa3: BP+V5VacRRDFaf0mgrBR898TTwVSjDNbi+423n8DLx4=
vni_b3f211af3b234b8ca6ea71d586b5daa3: 1
vpn_host_b3f211af3b234b8ca6ea71d586b5daa3: wireguard-vxlan-server.mydomain.com
vpn_port_b3f211af3b234b8ca6ea71d586b5daa3: 51820
vpnserver1: vpn.testdomain.com

CONFIGURATION VARIABLES:

In this section it's possible to override the default values of variables defined in templates. If you're not using configuration variables you can safely ignore this section.

key : value

Voila! You have successfully configured OpenWISP to manage VXLAN over WireGuard tunnels for your devices.

How to Configure Push Updates

Follow the procedure described below to enable secure SSH access from OpenWISP to your devices, this is required to enable push updates (whenever the configuration is changed, OpenWISP will trigger the update in the background) and/or firmware upgrades (via the additional module openwisp-firmware-upgrader).

Note

If you have installed OpenWISP with the ansible-openwisp2 role then you can skip the following steps. The Ansible role automatically creates a default template to update `authorized_keys` on networking devices using the default access credentials.

1. Generate SSH key

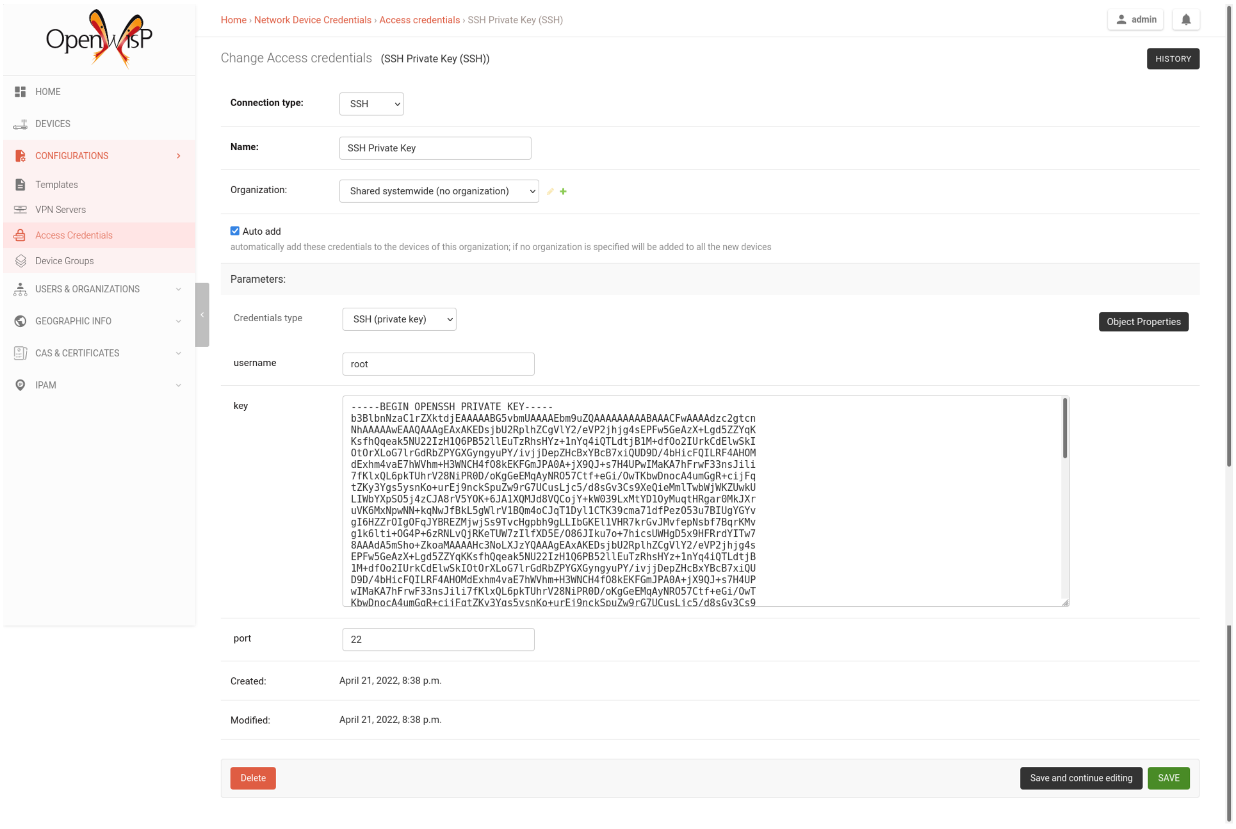
First of all, we need to generate the SSH key which will be used by OpenWISP to access the devices, to do so, you can use the following command:

```
echo './sshkey' | ssh-keygen -t rsa -b 4096 -C "openwisp"
```

This will create two files in the current directory, one called `sshkey` (the private key) and one called `sshkey.pub` (the public key).

Store the content of these files in a secure location.

2. Save SSH private key in OpenWISP (access credentials)



Home › Network Device Credentials › Access credentials › SSH Private Key (SSH)

Change Access credentials (SSH Private Key (SSH))

Connection type: SSH

Name: SSH Private Key

Organization: Shared systemwide (no organization)

☑ Auto add

automatically add these credentials to the devices of this organization; if no organization is specified will be added to all the new devices

Parameters:

Credentials type: SSH (private key)

username: root

key:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAACFwAAAAdzc2gtcn
NhAAAAAwEAAQAAAgEAxAKEDsjbU2RplhZCgVlY2/eVP2jhjg4sEPFw5GeAzX+Lgd5ZZYqK
KsfhQqeak5NU22IzH1Q6PB52llEuTzRhsHYz+1nYq4iQTLdtjB1M+dfOo2IUrkCdElwSkI
OtOrXLoG7lrGdRbZPYGXGyngyuPY/ivjjDepZHcBxYBcB7xiQUD9D/4bHicFQILRF4AHOM
dExhm4vaE7hWVhm+H3WNCH4fO8kEKFGmJPA0A+jX9QJ+s7H4UPwIMaKA7hFrwF33nsJili
7fKlxQL6pkTUhrV28NiPR0D/oKgGeEMqAyNRO57Ctf+eGi/OwTKbwDnocA4umGgR+cijFq
tZKy3Ygs5ysnKo+urEj9nckSpuZw9rG7UCusLjc5/d8sGv3Cs9XeQieMmlTwbWjWKZUwkU
LIWbYXpSO5j4zCJA8rV5YOK+6JA1XQMJd8VQCojY+kW039LxMtYD1OyMuqtHRgar0MkJXr
uVK6MxNpwNN+kqNwJfBkL5gWlrV1BQm4oCJqT1Dyl1CTK39cma71dfPezO53u7BIUgYGYv
gI6HZZrOIgOFqJYBREZMjwjSs9TvcHgpbh9gLLIbGKEl1VHR7krGvJMvfepNsbf7BqrKMv
g1k6lti+OG4P+6zRNLvQjRKeTUW7zIlfXD5E/O86JIku7o+7hicsUWHgD5x9HFRrdYITw7
8AAAdA5mSho+ZkoaMAAAAHc3NoLXJzYQAAAgEAxAKEDsjbU2RplhZCgVlY2/eVP2jhjg4s
EPFw5GeAzX+Lgd5ZZYqKKsfhQqeak5NU22IzH1Q6PB52llEuTzRhsHYz+1nYq4iQTLdtjB
1M+dfOo2IUrkCdElwSkIOtOrXLoG7lrGdRbZPYGXGyngyuPY/ivjjDepZHcBxYBcB7xiQU
D9D/4bHicFQILRF4AHOMdExhm4vaE7hWVhm+H3WNCH4fO8kEKFGmJPA0A+jX9QJ+s7H4UP
wIMaKA7hFrwF33nsJili7fKlxQL6pkTUhrV28NiPR0D/oKgGeEMqAyNRO57Ctf+eGi/OwT
KbwDnocA4umGgR+cijFqtZKy3Ygs5ysnKo+urEj9nckSpuZw9rG7UCusLjc5/d8sGv3Cs9
```

port: 22

Created: April 21, 2022, 8:38 p.m.

Modified: April 21, 2022, 8:38 p.m.

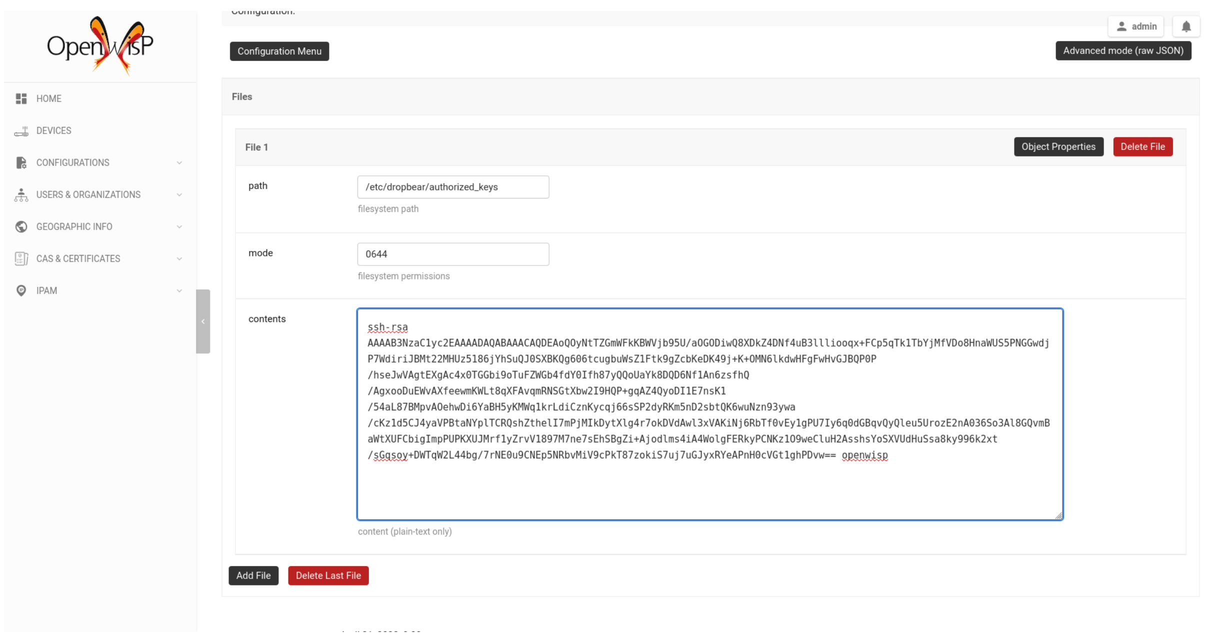
From the first page of OpenWISP click on "Access credentials", then click on the **"ADD ACCESS CREDENTIALS"** button in the upper right corner (alternatively, go to the following URL: `/admin/connection/credentials/add/`).

Select SSH as `type`, enable the **Auto add** checkbox, then at the field "Credentials type" select "SSH (private key)", now type "root" in the `username` field, while in the `key` field you have to paste the contents of the private key just created.

Now hit save.

The credentials just created will be automatically enabled for all the devices in the system (both existing devices and devices which will be added in the future).

3. Add the public key to your devices



path: /etc/dropbear/authorized_keys
filesystem path

mode: 0644
filesystem permissions

contents:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDEAoQOyNtTZGmWFkKBWVjb95U/aOGODiwQ8XDkZ4DNf4uB3llliooqx+FCp5qTk1TbYjMfVDo8HnaWUS5PNGGwdj
P7WdiriJBMt22MHUz5186jYhSuQJ0SXBKQg606tcugbuWsZ1Ftk9gZcbKeDK49j+K+OMN6lkdwHFgFwHvGJBQP0P
/hseJwVAgtEXgAc4x0TGGbi9oTuFZWGb4fdY0Ifh87yQQoUaYk8DQD6Nf1An6zsfhQ
/AgxooDuEWvAXfeewmKWLt8qXFAvqmRNSGtXbw2I9HQP+gqAZ4QyoDI1E7nsK1
/54aL87BMpvAOehwDi6YaBH5yKMWq1krLdiCznKycqj66sSP2dyRKm5nD2sbtQK6wuNzn93ywa
/cKz1d5CJ4yaVPBtaNYplTCRQshZthelI7mPjMIkDytXlg4r7okDVdAwl3xVAKiNj6RbTf0vEy1gPU7Iy6q0dGBqvQyQleu5UrozE2nA036So3Al8GQvmB
aWtXUFCbigImpPUPKXUJMrf1yZrvV1897M7ne7sEhSBgZi+Ajodlms4iA4WolgFERkyPCNKz1O9weCluH2AsshsYoSXVUdHuSsa8ky996k2xt
/sGqsoy+DWTqW2L44bg/7rNE0u9CNEp5NRbvMiV9cPkT87zokiS7uj7uGJyxRYeAPnH0cVGt1ghPDvw== openwisp
```

content (plain-text only)

Now we need to instruct your devices to allow OpenWISP accessing via SSH, in order to do this we need to add the contents of the public key file created in step 1 (`sshkey.pub`) in the file `/etc/dropbear/authorized_keys` on the devices, the recommended way to do this is to create a configuration template in OpenWISP: from the first page of OpenWISP, click on "Templates", then and click on the **"ADD TEMPLATE"** button in the upper right corner (alternatively, go to the following URL: `/admin/config/template/add/`).

Check **enabled by default**, then scroll down the configuration section, click on "Configuration Menu", scroll down, click on "Files" then close the menu by clicking again on "Configuration Menu". Now type `/etc/dropbear/authorized_keys` in the `path` field of the file, then paste the contents of `sshkey.pub` in `contents`.

Now hit save.

There's a catch: you will need to assign the template to any existing device.

4. Test it

Once you have performed the 3 steps above, you can test it as follows:

1. Ensure there's at least one device turned on and connected to OpenWISP, ensure this device has the "SSH Authorized Keys" assigned to it.
2. Ensure the celery worker of OpenWISP Controller is running (e.g.: `ps aux | grep celery`)
3. SSH into the device and wait (maximum 2 minutes) until `/etc/dropbear/authorized_keys` appears as specified in the template.
4. While connected via SSH to the device run the following command in the console: `logread -f`, now try changing the device name in OpenWISP
5. Shortly after you change the name in OpenWISP, you should see some output in the SSH console indicating another SSH access and the configuration update being performed.

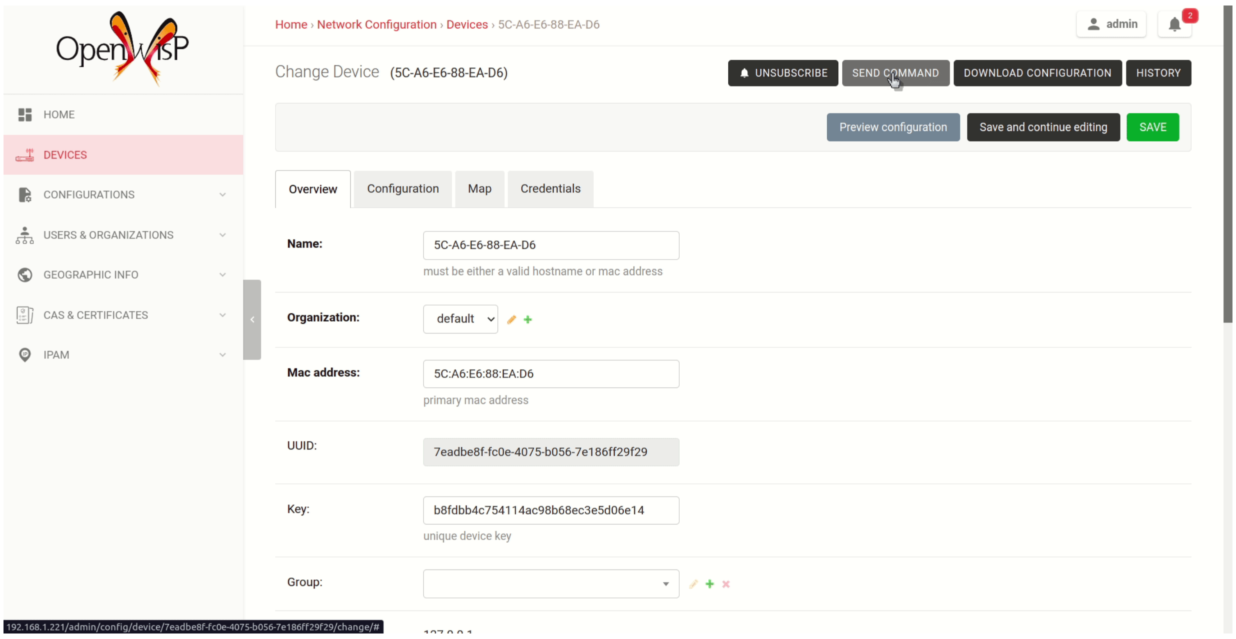
Sending Commands to Devices

Default command options

By default, there are three options in the **Send Command** dropdown:

1. Reboot
2. Change Password
3. Custom Command

While the first two options are self-explanatory, the **custom command** option allows you to execute any command on the device as shown in the example below.



Change Device (5C-A6-E6-88-EA-D6)

Name: 5C-A6-E6-88-EA-D6
must be either a valid hostname or mac address

Organization: default

Mac address: 5C:A6:E6:88:EA:D6
primary mac address

UUID: 7eadbe8f-fc0e-4075-b056-7e186ff29f29

Key: b8fdbb4c754114ac98b68ec3e5d06e14
unique device key

Group:

192.168.1.221/admin/config/device/7eadbe8f-fc0e-4075-b056-7e186ff29f29/change/#

Note

In order for this feature to work, a device needs to have at least one **Access Credential** (see How to Configure Push Updates).

The **Send Command** button will be hidden until the device has at least one **Access Credential**.

If you need to allow your users to quickly send specific commands that are used often in your network regardless of your users' knowledge of Linux shell commands, you can add new commands by following instructions in "How to define new options in the commands menu" section below.

If you are an advanced user and want to register commands programmatically, then refer to "Register / Unregistering commands" section.

How to define new options in the commands menu

It is possible to define new custom commands which are added to the menu of available commands, this allows to make it easy for users perform additional management actions without having to be Linux/Unix experts and know the exact shell command syntax (because the system generates the command for them based on the input received via the UI).

We can do so by using the `OPENWISP_CONTROLLER_USER_COMMANDS` django setting (see How to Edit Django Settings).

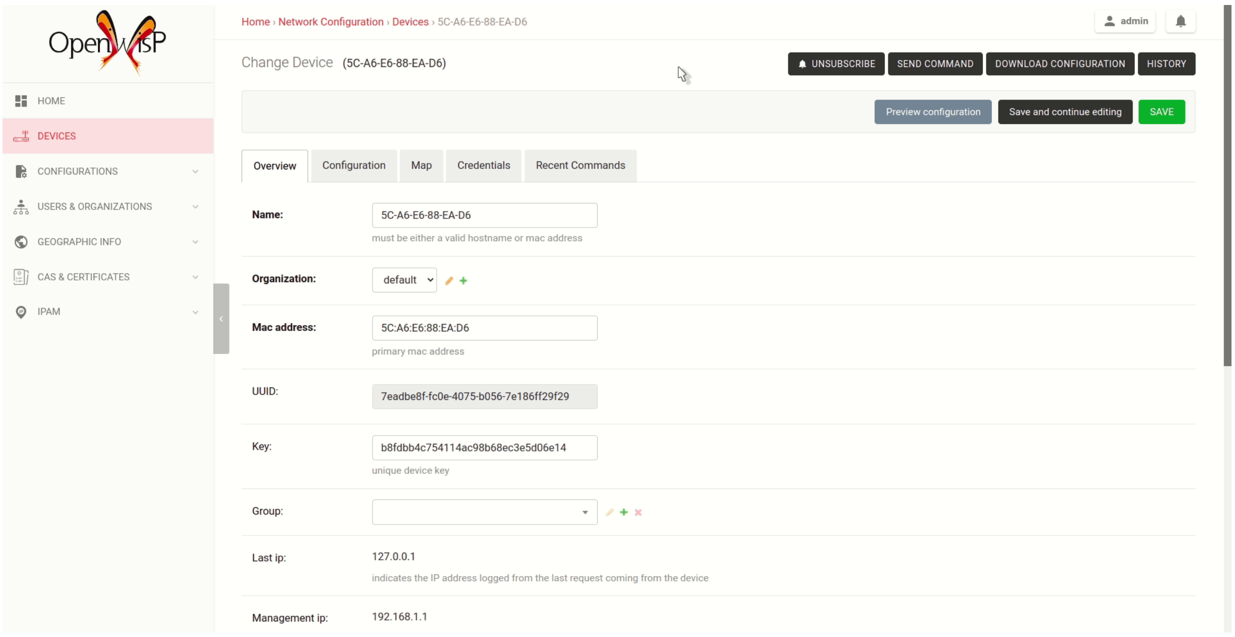
The following example defines a simple command that can `ping` an input `destination_address` through a network interface, `interface_name`.

```python
# In yourproject/settings.py


def ping_command_callable(destination_address, interface_name=None):
    command = f"ping -c 4 {destination_address}"
    if interface_name:
        command += f" -I {interface_name}"
    return command


OPENWISP_CONTROLLER_USER_COMMANDS = [
    (
        "ping",
        {
            "label": "Ping",
            "schema": {
                "title": "Ping",
                "type": "object",
                "required": ["destination_address"],
                "properties": {
                    "destination_address": {
                        "type": "string",
                        "title": "Destination Address",
                    },
                    "interface_name": {
                        "type": "string",
                        "title": "Interface Name",
                    },
                },
                "message": "Destination Address cannot be empty",
                "additionalProperties": False,
            },
            "callable": ping_command_callable,
        },
    )
]
```

The above code will add the "Ping" command in the user interface as show in the GIF below:



Change Device (5C-A6-E6-88-EA-D6)

Name: 5C-A6-E6-88-EA-D6
must be either a valid hostname or mac address

Organization: default

Mac address: 5C:A6:E6:88:EA:D6
primary mac address

UUID: 7eadbe8f-fc0e-4075-b056-7e186ff29f29

Key: b8fdbb4c754114ac98b68ec3e5d06e14
unique device key

Group:

Last ip: 127.0.0.1
indicates the IP address logged from the last request coming from the device

Management ip: 192.168.1.1

The `OPENWISP_CONTROLLER_USER_COMMANDS` setting takes a `list` of `tuple` each containing two elements. The first element of the tuple should contain an identifier for the command and the second element should contain a `dict` defining configuration of the command.

Command Configuration

The `dict` defining configuration for command should contain following keys:

1. `label`

A `str` defining label for the command used internally by Django.

2. `schema`

A `dict` defining JSONSchema for inputs of command. You can specify the inputs for your command, add rules for performing validation and make inputs required or optional.

Here is a detailed explanation of the schema used in above example:

```
{
    # Name of the command displayed in *Send Command* widget
    "title": "Ping",
    # Use type *object* if the command needs to accept inputs
    # Use type *null* if the command does not accepts any input
    "type": "object",
    # Specify list of inputs that are required
    "required": ["destination_address"],
    # Define the inputs for the commands along with their properties
    "properties": {
        "destination_address": {
            # type of the input value
            "type": "string",
            # label used for displaying this input field
            "title": "Destination Address",
        },
        "interface_name": {
            "type": "string",
            "title": "Interface Name",
        },
    },
    # Error message to be shown if validation fails
    "message": "Destination Address cannot be empty",
    # Whether specifying addtionaly inputs is allowed from the input form
    "additionalProperties": False,
}
```

This example uses only handful of properties available in JSONSchema. You can experiment with other properties of JSONSchema for schema of your command.

3. `callable`

A `callable` or `str` defining dotted path to a callable. It should return the command (`str`) to be executed on the device. Inputs of the command are passed as arguments to this callable.

The example above includes a callable(`ping_command_callable`) for `ping` command.

Subnet division rules

The subnet division rule is a feature introduced in OpenWISP 22.05 in the Controller module, this feature aims at making it easy to provision an arbitrary number of subnets and IP addresses to each device, these provisioned subnets and addresses are then added to the configuration variables (as system defined variables) of the device and can be used in its configuration or even in configuration templates.

Table of Contents:

Enabling the subnet division rule app

If you are using the official OpenWISP ansible role, **this feature is not enabled by default**, to enable it just set the `openwisp2_controller_subnet_division` configuration variable to `true` in your playbook.

On docker-openwisp2 this feature is enabled by default.

Configuring the automatic provisioning of subnets and IPs

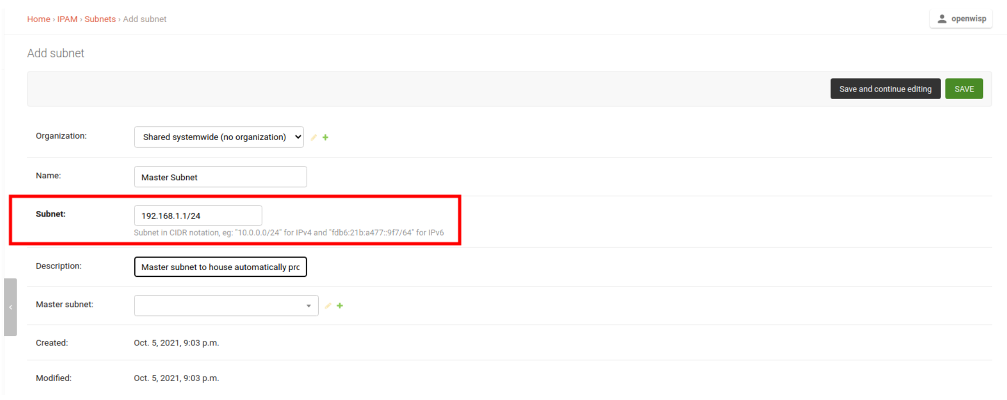
This guide will help you configure automatic provisioning of subnets and IPs for devices.

1. Create a Subnet and a Subnet Division Rule

Create a master subnet under which automatically generated subnets will be provisioned.

Tip

Choose the size of the subnet appropriately considering your use case.



Home › IPAM › Subnets › Add subnet

Add subnet

Organization:	Shared systemwide (no organization)
Name:	Master Subnet
Subnet:	192.168.1.1/24 Subnet in CIDR notation, eg: "10.0.0.0/24" for IPv4 and "fdb6:21b:a477::9f7/64" for IPv6
Description:	Master subnet to house automatically prc
Master subnet:	
Created:	Oct. 5, 2021, 9:03 p.m.
Modified:	Oct. 5, 2021, 9:03 p.m.

On the same page, add a **subnet division rule** that will be used to provision subnets under the master subnet.

The type of subnet division rule controls when subnets and IP addresses will be provisioned for a device.

The subnet division rule types currently implemented are described below.

Device Subnet Division Rule

This rule type is triggered whenever a device configuration (`config.Config` model) is created for the organization specified in the rule.

Creating a new rule of "Device" type will also provision subnets and IP addresses for existing devices of the organization automatically.

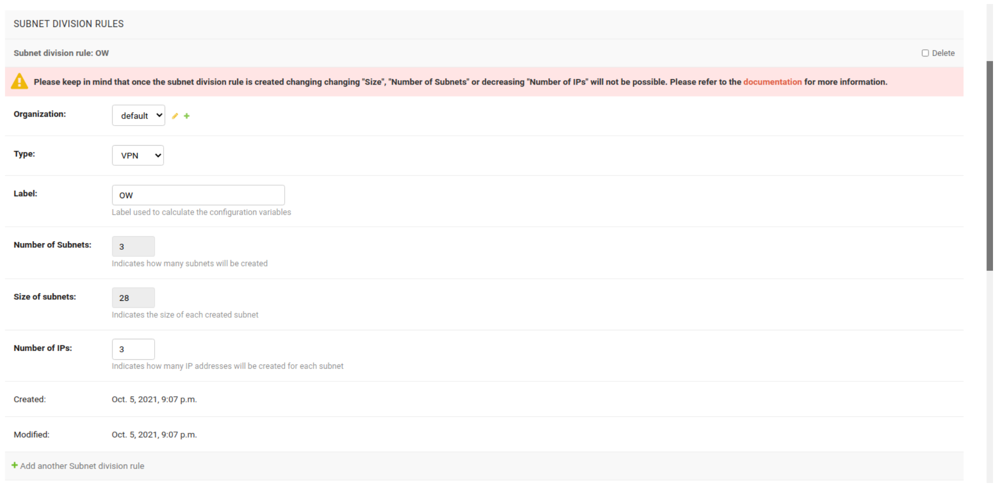
Important

Keep in mind that a device without a defined configuration object will not trigger this rule.

VPN Subnet Division Rule

This rule is triggered when a VPN client template is assigned to a device, provided the VPN server to which the VPN client template relates to has the same subnet for which the subnet division rule is created.

Note: This rule will only work for **WireGuard** and **VXLAN over WireGuard** VPN servers.



SUBNET DIVISION RULES

Subnet division rule: OW

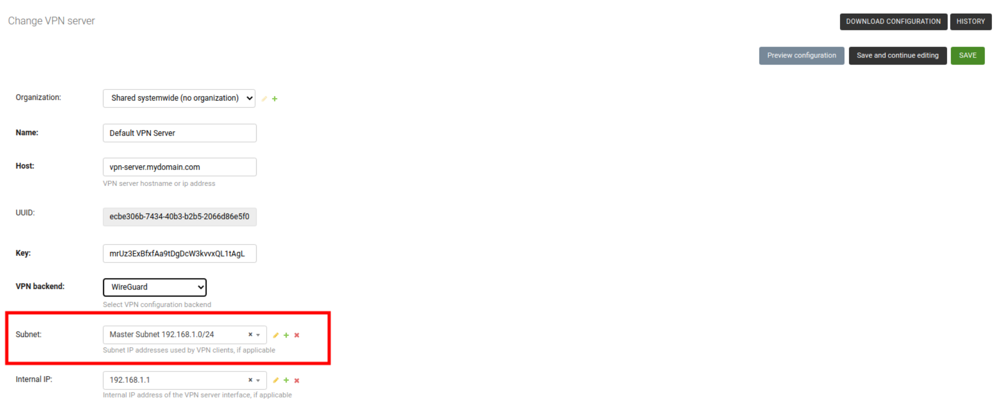
Please keep in mind that once the subnet division rule is created changing changing "Size", "Number of Subnets" or decreasing "Number of IPs" will not be possible. Please refer to the documentation for more information.

Organization:	default
Type:	VPN
Label:	OW Label used to calculate the configuration variables
Number of Subnets:	3 Indicates how many subnets will be created
Size of subnets:	28 Indicates the size of each created subnet
Number of IPs:	3 Indicates how many IP addresses will be created for each subnet
Created:	Oct. 5, 2021, 9:07 p.m.
Modified:	Oct. 5, 2021, 9:07 p.m.

In this example, **VPN subnet division rule** is used.

2. Create a VPN Server

Now create a VPN Server and choose the previously created **master subnet** as the subnet for this VPN Server.

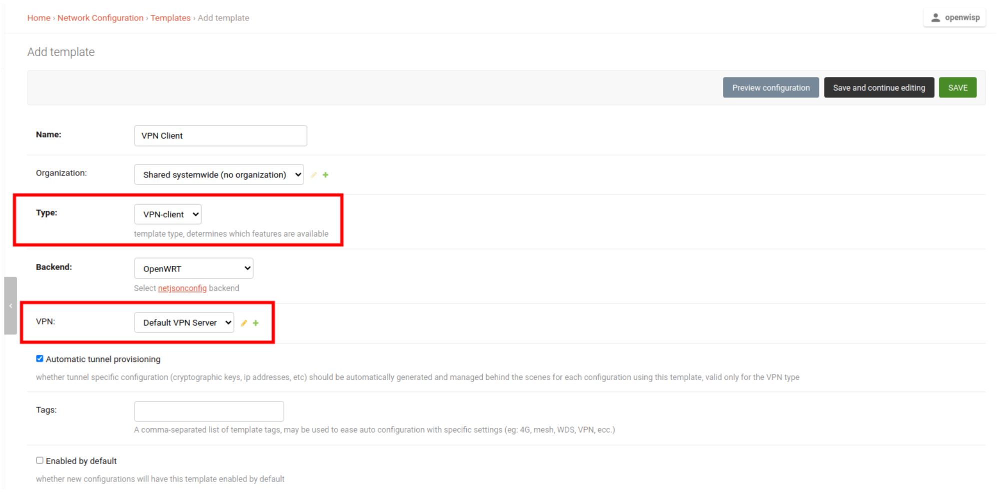


Change VPN server

Organization:	Shared systemwide (no organization)
Name:	Default VPN Server
Host:	vpn-server.mydomain.com VPN server hostname or ip address
UUID:	ecbe306b-7434-40b3-b2b5-2066d86e5f0
Key:	mrUz3ExBfxfAa9tDgDcW3kvvxQL1tAgL
VPN backend:	WireGuard Select VPN configuration backend
Subnet:	Master Subnet 192.168.1.0/24 Subnet IP addresses used by VPN clients, if applicable
Internal IP:	192.168.1.1 Internal IP address of the VPN server interface, if applicable

3. Create a VPN Client Template

Create a template, setting the **Type** field to **VPN Client** and **VPN** field to use the previously created VPN Server.



Home › Network Configuration › Templates › Add template
openwisp
Preview configuration
Save and continue editing
SAVE

Add template

Name:	VPN Client
Organization:	Shared systemwide (no organization)
Type:	VPN-client template type, determines which features are available
Backend:	OpenWRT Select netjsonconfig backend
VPN:	Default VPN Server

☑ Automatic tunnel provisioning

whether tunnel specific configuration (cryptographic keys, ip addresses, etc) should be automatically generated and managed behind the scenes for each configuration using this template, valid only for the VPN type

Tags:	A comma-separated list of template tags, may be used to ease auto configuration with specific settings (eg: 4G, mesh, WDS, VPN, ecc.)

☐ Enabled by default

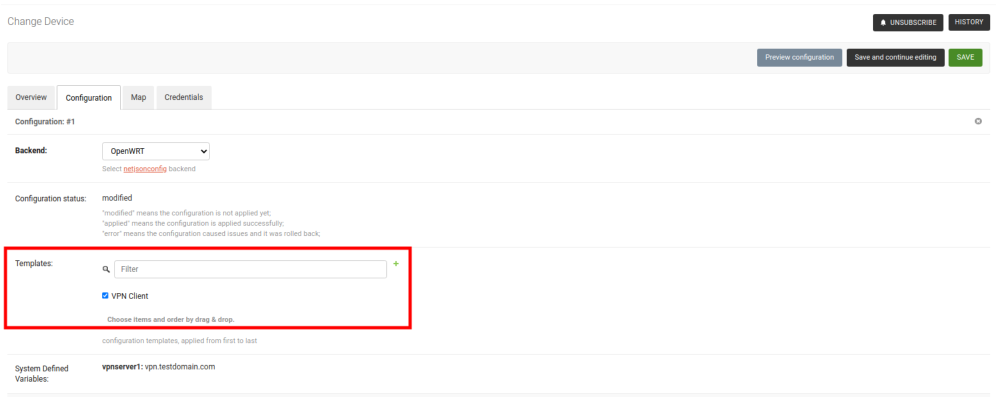
whether new configurations will have this template enabled by default

Tip

You can also check the **Enable by default** field if you want to automatically apply this template to devices that will register in future.

4. Apply VPN Client Template to Devices

With everything in place, you can now apply the VPN Client Template to devices.

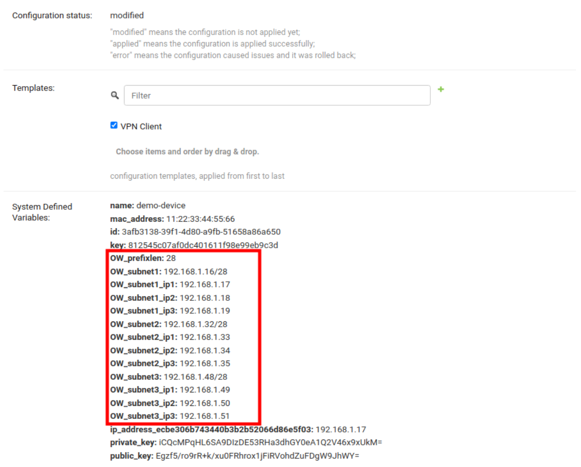


Change Device

Configuration: #1

Backend:	OpenWRT Select netjsonconfig backend
Configuration status:	modified "modified" means the configuration is not applied yet; "applied" means the configuration is applied successfully; "error" means the configuration caused issues and it was rolled back;
Templates:	Filter ☑ VPN Client Choose items and order by drag & drop. configuration templates, applied from first to last
System Defined Variables:	**vpnserver1:** vpn.testdomain.com

After saving the device, you should see all provisioned Subnets and IPs for this device under System Defined Variables.



Configuration status:	modified "modified" means the configuration is not applied yet; "applied" means the configuration is applied successfully; "error" means the configuration caused issues and it was rolled back;
Templates:	Filter ☑ VPN Client Choose items and order by drag & drop. configuration templates, applied from first to last
System Defined Variables:	**name:** demo-device **mac_address:** 11:22:33:44:55:66 **id:** 3afb3138-39f1-4d80-a9fb-51658a86a650 **key:** 812545c07af0dc401611f98e99eb9c3d **OW_prefixlen:** 28 **OW_subnet1:** 192.168.1.16/28 **OW_subnet1_ip1:** 192.168.1.17 **OW_subnet1_ip2:** 192.168.1.18 **OW_subnet1_ip3:** 192.168.1.19 **OW_subnet2:** 192.168.1.32/28 **OW_subnet2_ip1:** 192.168.1.33 **OW_subnet2_ip2:** 192.168.1.34 **OW_subnet2_ip3:** 192.168.1.35 **OW_subnet3:** 192.168.1.48/28 **OW_subnet3_ip1:** 192.168.1.49 **OW_subnet3_ip2:** 192.168.1.50 **OW_subnet3_ip3:** 192.168.1.51 **ip_address_ecbe306b743440b3b2b52066d86e5f03:** 192.168.1.17 **private_key:** iCQcMPqHL6SA9DIzDE53RHa3dhGY0eA1Q2V46x9xUkM= **public_key:** Egzf5/ro9rR+k/xu0FRhrox1jFiRVohdZuFDgW9JhWY=

Voila! You can now use these variables in configuration of the device. Refer to How to use configuration variables section of this documentation to learn how to use configuration variables.

Important notes for using Subnet Division

- In the above example Subnet, VPN Server, and VPN Client Template belonged to the **default** organization. You can use **Systemwide Shared** Subnet, VPN Server, or VPN Client Template too, but Subnet Division Rule will be always related to an organization. The Subnet Division Rule will only be triggered when such VPN Client Template will be applied to a Device having the same organization as Subnet Division Rule.
- You can also use the configuration variables for provisioned subnets and IPs in the Template. Each variable will be resolved differently for different devices. E.g. `OW_subnet1_ip1` will resolve to `10.0.0.1` for one device and `10.0.0.55` for another. Every device gets its own set of subnets and IPs. But don't forget to provide the default fall back values in the "default values" template field (used mainly for validation).
- The Subnet Division Rule will automatically create a reserved subnet, this subnet can be used to provision any IP addresses that have to be created manually. The rest of the master subnet address space **must not** be interfered with or the automation implemented in this module will not work.
- The above example used VPN subnet division rule. Similarly, device subnet division rule can be used, which only requires creating a subnet and a subnet division rule.

Limitations of Subnet Division

In the current implementation, it is not possible to change *"Size"*, *"Number of Subnets"* and *"Number of IPs"* fields of an existing subnet division rule due to following reasons:

Size

Allowing to change size of provisioned subnets of an existing subnet division rule will require rebuilding of Subnets and IP addresses which has possibility of breaking existing configurations.

Number of Subnets

Allowing to decrease number of subnets of an existing subnet division rule can create patches of unused subnets dispersed everywhere in the master subnet. Allowing to increase number of subnets will break the continuous allocation of subnets for every device. It can also break configuration of devices.

Number of IPs

Allowing to decrease number of IPs of an existing subnet division rule will lead to deletion of IP Addresses which can break configuration of devices being used. It **is allowed** to increase number of IPs.

If you want to make changes to any of above fields, delete the existing rule and create a new one. The automation will provision for all existing devices that meets the criteria for provisioning. **WARNING**: It is possible that devices get different subnets and IPs from previous provisioning.

Firmware Upgrades

OpenWISP Firmware Upgrader has been introduced in OpenWISP 20 and focuses on allowing users to maintain the firmware of their network devices up to date.

Table of Contents:

Deploy instructions

See Enabling the firmware upgrader module on the OpenWISP 22.05 ansible role documentation.

This module is also available in docker-openwisp although its usage is not recommended for production usage yet, unless the reader is willing to invest effort in adapting the docker images and configurations to overcome any roadblocks encountered.

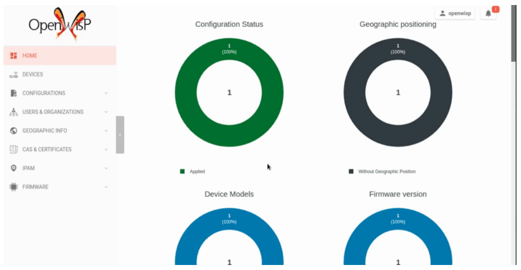
Quick Start Guide

Requirements:

- Devices running at least OpenWRT 12.09 Attitude Adjustment, older versions of OpenWRT have not worked at all in our tests
- Devices must have enough free RAM to be able to upload the new image to `/tmp`

1. Create a category

Create a category for your firmware images by going to *Firmware management > Firmware categories > Add firmware category*, if you use only one firmware type in your network, you could simply name the category "default" or "standard".



If you use multiple firmware images with different features, create one category for each firmware type, e.g.:

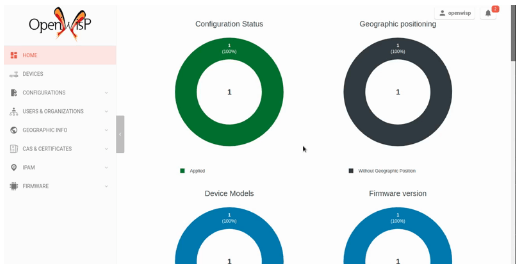
- WiFi
- SDN router
- LoRa Gateway

This is necessary in order to perform mass upgrades only on specific firmware categories when, for example, a new *LoRa Gateway* firmware becomes available.

2. Create the build object

Create a build a build object by going to *Firmware management > Firmware builds > Add firmware build*, the build object is related to a firmware category and is the collection of the different firmware images which have been compiled for the different hardware models supported by the system.

The version field indicates the firmware version, the change log field is optional but we recommend filling it to help operators know the differences between each version.



An important but optional field of the build model is **OS identifier**, this field should match the value of the **Operating System** field which gets automatically filled during device registration, e.g.: `OpenWrt 19.07-SNAPSHOT r11061-6ffd4d8a4d`. It is used by the firmware-upgrader module to automatically create `DeviceFirmware` objects for existing devices or when new devices register.

A `DeviceFirmware` object represent the relationship between a device and a firmware image, it basically tells us which firmware image is installed on the device.

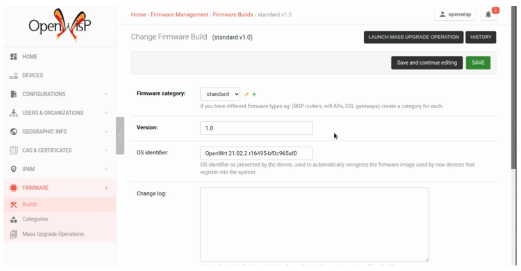
To find out the exact value to use, you should either do a test flash on a device and register it to the system or you should inspect the firmware image by decompressing it and find the generated value in the firmware image.

If you're not sure about what **OS identifier** to use, just leave it empty, you can fill it later on when you find out.

Now save the build object to create it.

3. Upload images to the build

Now is time to add images to the build, we suggest adding one image at time. Alternatively the REST API can be used to automate this step.



Home › Firmware Management › Firmware Builds › standard v1.0

Change Firmware Build (standard v1.0)

Firmware category: standard

if you have different firmware types eg: (BGP routers, wifi APs, DSL gateways) create a category for each.

Version: 1.0

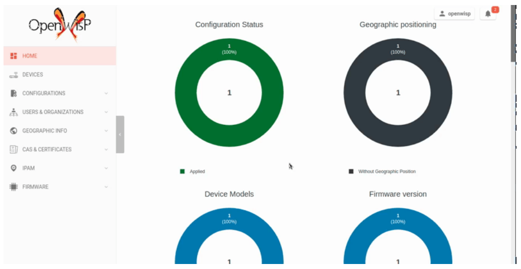
OS identifier: OpenWrt 21.02.2 r16495-bf0c965af0

OS identifier as presented by the device, used to automatically recognize the firmware image used by new devices that register into the system

Change log:

If you use a hardware model which is not listed in the image types, if the hardware model is officially supported by OpenWRT, you can send us a pull-request to add it, otherwise you can use the setting OPENWISP_CUSTOM_OPENWRT_IMAGES to add it.

4. Perform a firmware upgrade to a specific device



Once a new build is ready, has been created in the system and its image have been uploaded, it will be the time to finally upgrade our devices.

To perform the upgrade of a single device, navigate to the device details, then go to the "Firmware" tab.

If you correctly filled **OS identifier** in step 2, you should have a situation similar to the one above: in this example, the device is using version `1.0` and we want to upgrade it to version `2.0`, once the new firmware image is selected we just have to hit save, then a new tab will appear in the device page which allows us to see what's going on during the upgrade.

Right now, the update of the upgrade information is not asynchronous yet, so you will have to reload the page periodically to find new information.

This will be addressed in a future release.

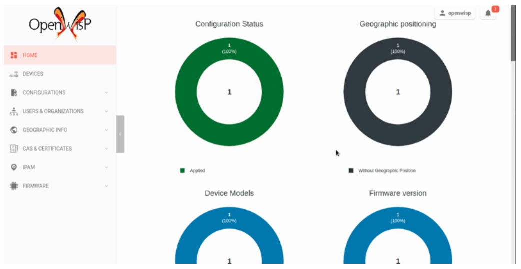
5. Performing mass upgrades

First of all, please ensure the following preconditions are met:

- the system is configured correctly
- the new firmware images are working as expected
- you already tried the upgrade of single devices several times.

At this stage you can try a mass upgrade by doing the following:

- go to the build list page
- select the build which contains the latest firmware images you want the devices to be upgraded with
- click on "Mass-upgrade devices related to the selected build".



At this point you should see a summary page which will inform you of which devices are going to be upgraded, you can either confirm the operation or cancel.

Once the operation is confirmed you will be redirected to a page in which you can monitor the progress of the upgrade operations.

Right now, the update of the upgrade information is not asynchronous yet, so you will have to reload the page periodically to find new information.

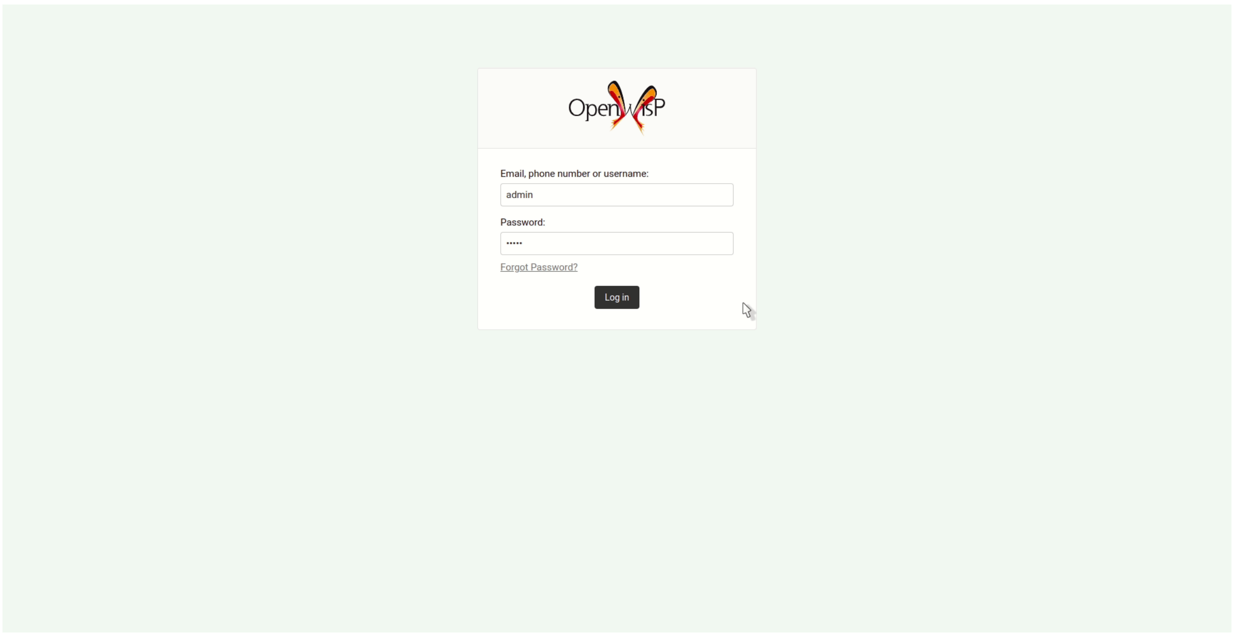
This will be addressed in a future release.

Find out more about OpenWISP Firmware Upgrader

For more information about the features offered by OpenWISP Firmware Upgrader we refer to the following sections of its documentation:

- List of the available features
- Automatic device firmware detection
- Writing Custom Firmware Upgrader Classes
- Rest API
- Django Settings

Network Topology



OpenWISP Network Topology is a network topology collector and visualizer web application and API, it allows to collect network topology data from different networking software (dynamic mesh routing protocols, OpenVPN), store it, visualize it, edit its details, it also provides hooks (a.k.a Django signals) to execute code when the status of a link changes.

When used in conjunction with OpenWISP Controller and OpenWISP Monitoring, it makes the monitoring system faster in detecting change to the network.

Table of Contents:

Deploy instructions

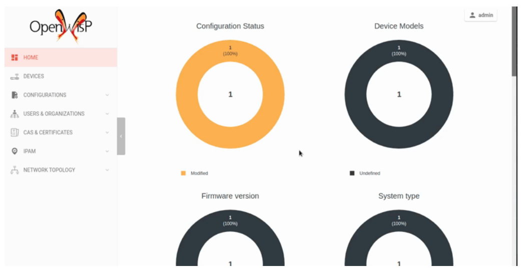
See Enabling the network topology module on the OpenWISP 22.05 ansible role documentation.

This module is also available in docker-openwisp although its usage is not recommended for production usage yet, unless the reader is willing to invest effort in adapting the docker images and configurations to overcome any roadblocks encountered.

Quick Start Guide

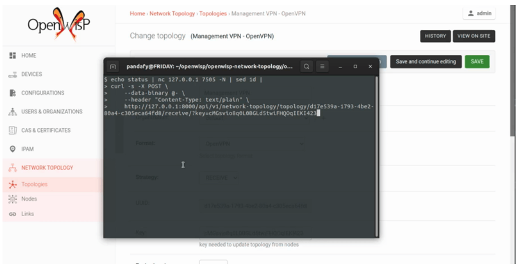
This module works by periodically collecting the network topology graph data of the supported networking software or formats. The data has to be either fetched by the application or received in POST API requests, therefore after deploying the application, additional steps are required to make the data collection and visualization work, read on to find out how.

Creating a topology



1. Create a topology object by going to *Network Topology* > *Topologies* > *Add topology*.
2. Give an appropriate label to the topology.
3. Select the *topology format* from the dropdown menu. The *topology format* determines which parser should be used to process topology data.
4. Select the *Strategy* for updating this topology.
    - If you are using FETCH strategy, then enter the URL for fetching topology data in the *Url* field.
    - If you are using RECEIVE strategy, you will get the *URL* for sending topology data. The *RECEIVE* strategy provides an additional field *expiration time*. This can be used to add delay in marking missing links as down.

Sending data for topology with RECEIVE strategy


OpenWISP
Home › Network Topology › Topologies › Management VPN - OpenVPN
admin
Change topology (Management VPN - OpenVPN)
HISTORY
VIEW ON SITE
Save and continue editing
SAVE
HOME
DEVICES
CONFIGURATIONS
USERS & ORGANIZATIONS
CAS & CERTIFICATES
IPAM
NETWORK TOPOLOGY
Topologies
Nodes
Links
pandafy@FRIDAY: ~/openwisp/openwisp-network-topology/o...
key needed to update topology from nodes

```
$ echo status | nc 127.0.0.1 7505 -N | sed 1d |
> curl -s -X POST \
>     --data-binary @- \
>     --header "Content-Type: text/plain" \
>     http://127.0.0.1:8000/api/v1/network-topology/topology/d17e539a-1793-4be2-
80a4-c305eca64fd8/receive/?key=cMGsvio8q0L0BGLd5twiFHQOqIEKI423
```

1. Copy the *URL* generated by OpenWISP for sending the topology data.

   E.g., in our case the URL is `http://127.0.0.1:8000/api/v1/network-topology/topology/d17e539a-1793-4be2-80a4-c305eca64fd8/receive/?key=cMGsvio8q0L0BGLd5twiFHQOqIEKI423`.

2. Create a script (e.g.: `/opt/send-topology.sh`) which sends the topology data using `POST`, in the example script below we are sending the status log data of OpenVPN but the same code can be applied to other formats by replacing `cat /var/log/openvpn/tun0.stats` with the actual command which returns the network topology output:

```
#!/bin/bash

# Get OpenVPN topology data from OpenVPN management interface
cat /var/log/openvpn/tun0.stats |
    # Upload the topology data to OpenWISP
    curl -s -X POST \
        --data-binary @- \
        --header "Content-Type: text/plain" \
        http://127.0.0.1:8000/api/v1/network-topology/topology/d17e539a-1793-4be2-80a4-c305e
```

3. Add the `/opt/send-topology.sh` script created in the previous step to the crontab, here's an example which sends the topology data every 5 minutes:

```
# flag script as executable
chmod +x /opt/send-topology.sh
# open crontab
crontab -e

## Add the following line and save

echo */5 * * * * /opt/send-topology.sh
```

4. Once the steps above are completed, you should see nodes and links being created automatically, you can see the network topology graph from the admin page of the topology change page (you have to click on the *View topology graph* button in the upper right part of the page) or, alternatively, a non-admin visualizer page is also available at the URL `/topology/topology/<TOPOLOGY-UUID>/`.

Find out more about OpenWISP Network Topology

For more information about the features offered by OpenWISP Network Topology we refer to the following sections of its documentation:

- List of the available features
- Collection Strategies
- Integration with OpenWISP Controller and OpenWISP Monitoring
- Rest API
- Django Settings

RADIUS

OpenWISP RADIUS has been introduced in OpenWISP 22.05 and provides many features aimed at public WiFi services.

Table of Contents:

Deploy instructions

See Enabling the RADIUS module on the OpenWISP 22.05 ansible role documentation.

Alternatively you can set it up manually by following these guides:

- Freeradius Setup for Captive Portal authentication
- Freeradius Setup for WPA Enterprise (EAP-TTLS-PAP) authentication

This module is also available in docker-openwisp although its usage is not recommended for production usage yet, unless the reader is willing to invest effort in adapting the docker images and configurations to overcome any roadblocks encountered.

Find out more about OpenWISP RADIUS

For more information about the features offered by OpenWISP RADIUS, we refer to the its documentation:

- Registration of new users
- SMS verification
- Importing users
- Generating users
- Social Login
- Single Sign-On (SAML)
- Enforcing session limits
- REST API
- Django Settings

WiFi Login Pages

Screenshots

Overview

OpenWISP WiFi login pages provides unified and consistent user experience for public/private WiFi services.

In short, this app replaces the classic captive/login page of a WiFi service by integrating the OpenWISP Radius API to provide the following features:

- Mobile first design (responsive UI)
- Sign up
- Optional support for mobile phone verification: verify phone number by inserting token sent via SMS, resend the SMS token
- Login to the wifi service (by getting a radius user token from OpenWISP Radius and sending a POST to the captive portal login URL behind the scenes)
- Session status information
- Logout from the wifi service (by sending a POST to the captive portal logout URL behind the scenes)
- Change password
- Reset password (password forgot)
- Support for Social Login and SAML
- Optional social login buttons (*Facebook*, *Google*, *Twitter*)
- Contact box allowing to show the support email and/or phone number, as well as additional links specified via configuration
- Navigation menu (header and footer) with possibility of specifying if links should be shown to every user or only authenticated or unauthenticated users
- Support for multiple organizations with possibility of customizing the theme via CSS for each organization
- Support for multiple languages
- Possibility to change any text used in the pages
- Configurable Terms of Services and Privacy Policy for each organization
- Possibility of recognizing users thanks to signed cookies, which saves them from having to re-authenticate
- Support for credit/debit card verification and paid subscription plans

For more information please see the OpenWISP WiFi Login Pages documentation.

How to Edit Django Settings

Modules (a.k.a Django Apps)

The OpenWISP server application is composed of a number of modules called Django apps, Django is the underlying web framework on top of which OpenWISP is built.

Some of the Django apps used by OpenWISP are developed and maintained by OpenWISP, other apps are developed and maintained by either Django or third party organizations, but most of these apps are configurable and customizable in different shapes or forms.

The most common way to modify the behavior of a Django app is by editing the settings.py file, a file which holds all the global configuration of the application.

The OpenWISP django modules are highly configurable and over time you may need to edit their settings, these settings are documented in the respective README documentation files of each module, for a list of all the available modules please see Architecture, Modules, Technologies.

Editing settings with the ansible role

The official ansible OpenWISP role provides many configuration variables which offer a convenient way to edit the most widely used settings of OpenWISP.

However, not all the possible settings have a corresponding variable because doing so would be very costly to maintain and make the code more complicated, for that reason the role provides a way to add any python instruction to define and manipulate settings via the `openwisp2_extra_django_settings_instructions` variable, e.g.:

```
# in the playbook variables add:
openwisp2_extra_django_settings_instructions:
  - |
    OPENWISP_NETWORK_TOPOLOGY_NODE_EXPIRATION = 14

    OPENWISP_MONITORING_METRICS = {
        'ping': {
            'alert_settings': {'tolerance': 60}
        },
        'config_applied': {
            'alert_settings': {'tolerance': 60}
        },
        'disk': {
            'alert_settings': {'tolerance': 60}
        },
        'memory': {
            'alert_settings': {'tolerance': 60}
        },
        'cpu': {
            'alert_settings': {
                'threshold': 95,
                'tolerance': 60
            }
        },
    }
```

This allows for great flexibility in configuring and extending OpenWISP, because additional custom modules can be added and configured too.

Editing settings with docker-openwisp

Similarly to the ansible role, the dockerized version of OpenWISP provides mainly two ways of changing settings:

1. The most widely used setting have a dedicated environment variable.
2. For more advanced use cases, it's possible to provide an entirely custom django settings file.

Architecture, Modules, Technologies

Architecture Overview

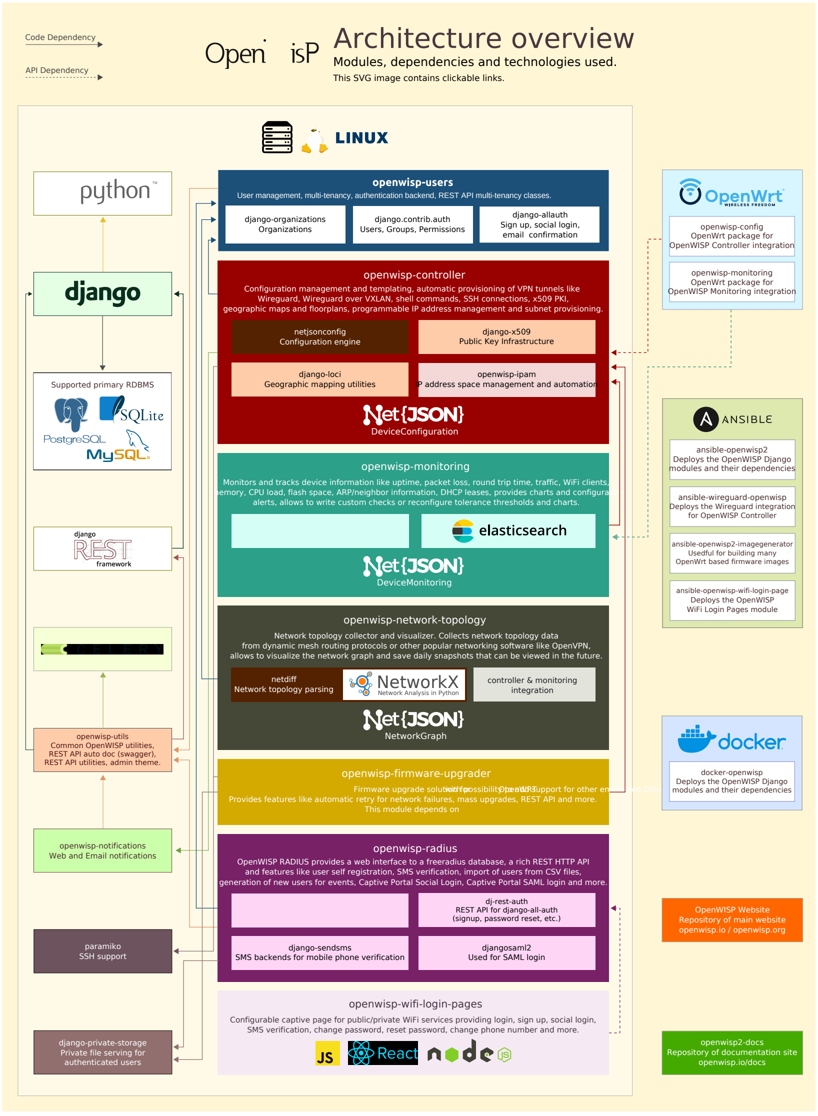
The following SVG image summarizes the architecture of OpenWISP, the main technologies used, the structure of the OpenWISP modules and their most important dependencies and the way they interact with one another.

Note

For the best experience it is recommended to open the image in a new tab of your browser, from there you can also click on the different elements to open the README or website of each module or technology used.

`The Inkscape source file of the architecture diagram is also available for download`.

Inkscape is an open source vector editing software which has been used to produce this diagram.


Code Dependency
API Dependency
OpenWISP
Architecture overview
Modules, dependencies and technologies used.
This SVG image contains clickable links.
LINUX
python
django
Supported primary RDBMS
PostgreSQL
SQLite
MySQL
django REST framework
CELERY
openwisp-utils
Common OpenWISP utilities,
REST API auto doc (swagger),
REST API utilities, admin theme.
openwisp-notifications
Web and Email notifications
paramiko
SSH support
django-private-storage
Private file serving for authenticated users
openwisp-users
User management, multi-tenancy, authentication backend, REST API multi-tenancy classes.
django-organizations
Organizations
django.contrib.auth
Users, Groups, Permissions
django-allauth
Sign up, social login, email confirmation
openwisp-controller
Configuration management and templating, automatic provisioning of VPN tunnels like Wireguard, Wireguard over VXLAN, shell commands, SSH connections, x509 PKI, geographic maps and floorplans, programmable IP address management and subnet provisioning.
netjsonconfig
Configuration engine
django-x509
Public Key Infrastructure
django-loci
Geographic mapping utilities
openwisp-ipam
IP address space management and automation
NetJSON
DeviceConfiguration
openwisp-monitoring
Monitors and tracks device information like uptime, packet loss, round trip time, traffic, WiFi clients, memory, CPU load, flash space, ARP/neighbor information, DHCP leases, provides charts and configurable alerts, allows to write custom checks or reconfigure tolerance thresholds and charts.
elasticsearch
NetJSON
DeviceMonitoring
openwisp-network-topology
Network topology collector and visualizer. Collects network topology data from dynamic mesh routing protocols or other popular networking software like OpenVPN, allows to visualize the network graph and save daily snapshots that can be viewed in the future.
netdiff
Network topology parsing
NetworkX
Network Analysis in Python
controller & monitoring integration
NetJSON
NetworkGraph
openwisp-firmware-upgrader
Firmware upgrade solution for OpenWRT with possibility to add support for other embedded OSes.
Provides features like automatic retry for network failures, mass upgrades, REST API and more.
This module depends on
openwisp-radius
OpenWISP RADIUS provides a web interface to a freeradius database, a rich REST HTTP API and features like user self registration, SMS verification, import of users from CSV files, generation of new users for events, Captive Portal Social Login, Captive Portal SAML login and more.
dj-rest-auth
REST API for django-all-auth
(signup, password reset, etc.)
django-sendsms
SMS backends for mobile phone verification
djangosaml2
Used for SAML login
openwisp-wifi-login-pages
Configurable captive page for public/private WiFi services providing login, sign up, social login, SMS verification, change password, reset password, change phone number and more.
JS
React
node
OpenWrt
WIRELESS FREEDOM
openwisp-config
OpenWrt package for OpenWISP Controller integration
openwisp-monitoring
OpenWrt package for OpenWISP Monitoring integration
ANSIBLE
ansible-openwisp2
Deploys the OpenWISP Django modules and their dependencies
ansible-wireguard-openwisp
Deploys the Wireguard integration for OpenWISP Controller
ansible-openwisp2-imagegenerator
Usedful for building many OpenWrt based firmware images
ansible-openwisp-wifi-login-page
Deploys the OpenWISP WiFi Login Pages module
docker
docker-openwisp
Deploys the OpenWISP Django modules and their dependencies
OpenWISP Website
Repository of main website
openwisp.io / openwisp.org
openwisp2-docs
Repository of documentation site
openwisp.io/docs

s

OpenWISP Modules

Note

If you want to know more about the motivations and philosophy that have shaped the modular architecture of OpenWISP, please see Applying the Unix Philosophy to Django projects: a report from the real world.

Deployment

- Ansible OpenWISP2: Recommended way to deploy OpenWISP on virtual machines.
- Docker OpenWISP (alpha): allows to deploy OpenWISP on dockerized cloud infrastructure. It's still being improved but the basic features of OpenWISP are working.
- Ansible OpenWISP WiFi Login Pages: ansible role that allows to deploy OpenWISP WiFi Login Pages.
- Ansible OpenWISP2 Image Generator: useful to generate many OpenWrt firmware images for different organizations with the OpenWISP packages preinstalled.
- Ansible Wireguard OpenWISP.: ansible role that allows to deploy the Wireguard integration for OpenWISP Controller.

Server Side

- OpenWISP Users: User management, multi-tenancy, authentication backend, REST API utilities and classes to implement multi-tenancy.
- OpenWISP Controller: Configuration management, automatic provisioning of VPN tunnels like OpenVPN, Wireguard, Wireguard over VXLAN, shell commands, SSH connections, x509 PKI management, geographic maps and floor plans, programmable IP address management and subnet provisioning.

  This module depends on several django apps or python libraries developed or maintained by OpenWISP:

  - netjsonconfig: configuration generation, validation and parsing.
  - django-x509: Public Key Infrastructure (management of certification authorities and x509 certificates).
  - django-loci: Geographic and indoor mapping features.
  - openwisp-ipam: IP and Subnet administration.
  - django-rest-framework-gis: GIS utilities for Django REST Framework.
- OpenWISP Monitoring: Monitors and tracks device information like uptime, packet loss, round trip time, traffic, WiFi clients,memory, CPU load, flash space, ARP/neighbor information, DHCP leases, provides charts and configurable alerts, allows to write custom checks or reconfigure tolerance thresholds and charts.
- OpenWISP Network Topology: Network topology collector and visualizer. Collects network topology data from dynamic mesh routing protocols or other popular networking software like OpenVPN, allows to visualize the network graph and save daily snapshots that can be viewed in the future.

  This module depends on a couple of libraries developed and maintained by OpenWISP:

  - netdiff: network topology parsing.
  - netjsongraph.js: Javascript library for network graph visualization.
- OpenWISP Firmware Upgrader: Firmware upgrade solution for OpenWRT with possibility to add support for other embedded OSes. Provides features like automatic retry for network failures, mass upgrades, REST API and more.
- OpenWISP RADIUS: provides a web interface to a freeradius database, a rich REST HTTP API and features like user self registration, SMS verification, import of users from CSV files, generation of new users for events, Captive Portal Social Login, Captive Portal SAML login and more.
- OpenWISP Notifications: provides email and web notifications to OpenWISP. Its main goal is to allow any OpenWISP module to notify users about meaningful events that happen in their network.
- OpenWISP Utils: common utilities and classes shared by all the OpenWISP python modules, it includes a lot of utilities for QA checks and automated testing which are heavily used in the continuous integration builds of most if not all the OpenWISP github repositories.
- OpenWISP WiFi Login Pages: Configurable captive page for public/private WiFi services providing login, sign up, social login, SMS verification, change password, reset password, change phone number and more. It is a frontend of the OpenWISP RADIUS REST API and it's designed to be used by end users of a public WiFi network.

Network Device Side

- OpenWISP Config: OpenWrt package which integrates with OpenWISP Controller.
- OpenWISP Monitoring: OpenWrt package which integrates with OpenWISP Monitoring.

Website and Documentation

- openwisp2-docs: repository for the documentation of OpenWISP, hosted on openwisp.io/docs.
- OpenWISP-Website: repository of the OpenWISP website, hosted on openwisp.org.

Main Technologies Used

Python

Python it's the main programming language used by the server side application (web admin, API, controller, workers).

In the past OpenWISP was built in Ruby On Rails, but we later switched to Python because it's much more suited to networking and it has a wider pool of potential contributors.

Find out more on why OpenWISP chose Python as its main language.

Django

Django is one of the most popular web frameworks for Python language.

It is used extensively in our modules. Django allows rapid development and has a very rich ecosystem.

It's the base framework used in most of the server side modules of OpenWISP.

Find out more on why OpenWISP chose Django as its main web framework.

Django REST Framework

Django REST framework is a powerful and flexible toolkit for building Web APIs based on Django and it's widely used in most of the Django and web based based OpenWISP modules.

Find out more on why OpenWISP chose Django REST Framework to build its REST API.

Celery

Celery is a python implementation of a distributed task queue and is heavily used in OpenWISP to execute background tasks, perform network operations like monitoring checks, configuration updates, firmware upgrades and so on.

OpenWrt

OpenWrt is an linux distribution designed for embedded systems, routers and networking in general.

It has a very skilled community and it is used as a base by many hardware vendors (Technicolor, Ubiquiti Networks, Linksys, Teltonika and many others).

Lua

Lua is a lightweight, multi-paradigm programming language designed primarily for embedded systems and clients.

Lua is cross-platform, since the interpreter is written in ANSI C, and has a relatively simple C API.

It is the official scripting language of OpenWRT and it's used heavily in the OpenWrt packages of OpenWISP: openwisp-config and openwisp-monitoring.

Node.js and React JS

NodeJS is javascript runtime to build JS based applications.

In OpenWISP it's used as a base for frontend applications along with React, like openwisp-wifi-login-pages.

Ansible

Ansible is a very popular software automation tool written in python that is generally used for automating software provisioning, configuration management and application deployment.

We use Ansible to provide automated procedures to deploy OpenWISP, to compile custom OpenWRT images for different organizations, to deploy OpenWISP WiFi Login Pages and to deploy the Wireguard integration for OpenWISP Controller.

Docker

We use docker in docker-openwisp, which aims to ease the deployment of OpenWISP in a containerized infrastructure.

NetJSON

NetJSON is a data interchange format based on JSON designed to ease the development of software tools for computer networks.

RADIUS

RADIUS (Remote Authentication Dial-In User Service) is a networking protocol that used for centralized Authentication, Authorization, and Accounting management of network services.

Freeradius

Freeradius is the most popular open source implementation of the RADIUS protocol and is heavily relied on in OpenWISP RADIUS.

Mesh Networking

A mesh nework is a local network topology in which the infrastructure nodes connect directly, dynamically and non-hierarchically to as many other nodes as possible and cooperate with one another to efficiently route data from/to clients.

OpenWrt supports the standard mesh mode (802.11s) and OpenWISP supports this mode out of the box.

It is also possible to support other popular dynamic open source routing protocols available on OpenWrt like OLSRd2, BATMAN-advanced, Babel, BMX, etc.

InfluxDB

InfluxDB is the default open source timeseries DB used in OpenWISP Monitoring.

Elasticsearch

Elasticsearch is an alternative option which can be used in OpenWISP Monitoring as timeseries DB, although it was designed with different purposes related to storing and retrieving data in a fast and efficient way.

Networkx

Networkx is a network graph analysis library written in Python and used under the hood by netdiff and the OpenWISP Network Topology module.

Relational Databases

Django supports several Relational Database Management Systems.

The most notable ones are:

- PostgreSQL
- MySQL
- SQLite

For production usage we recommend PostgreSQL.

For development we recommend SQLite for it simplicity.

Other notable dependencies

- paramiko (used in OpenWISP Controller and Firmware Upgrader)
- django-allauth (used in OpenWISP Users)
- django-organizations (used in OpenWISP Users)
- django-swappable-models (used in all the major Django modules)
- django-private-storage (used in OpenWISP RADIUS and Firmware Upgrader)
- dj-rest-auth (used in OpenWISP RADIUS)
- django-sendsms (used in OpenWISP RADIUS)
- django-saml2 (used in OpenWISP RADIUS)

Values and Goals of OpenWISP

What is OpenWISP?

OpenWISP is a software platform designed to ease and automate the management of networks, with a special focus on wireless networks, mainly used in public wifi, mesh networks, community networks and IoT scenarios.

OpenWISP 2, launched in December 2016, is the new generation of the software which is gradually replacing OpenWISP 1 and aims to build an ecosystem of applications and tools that make it easy for developers to build custom networking applications in order to bring innovation in the network infrastructure of communities that most need it.

History

See the History page on our website.

Core Values

1. Communication through electronic means is a human right

We believe that **communication through electronic means is a FUNDAMENTAL human right**.

According to Mozilla, 4 billion of people live without internet today.

Having seen the great progress the internet has brought to our society, we are deeply convinced that solving the issue of internet connectivity will help to alleviate the economic disparity that at the beginning of the 21st century is so evident in our world.

For these reasons, **fighting digital divide, both primary (lack of infrastructure) and secondary (lack of know how) is our utmost priority**.

2. Net Neutrality

We believe Net Neutrality to be beneficial to the internet because it allows everyone to have a fair treatment (non discrimination) to their private communications.

The very first public wifi networks that have been built with OpenWISP in Italy follow this principle very strictly: **no content filtering of any type is allowed on these network, no special privilege is given to any private network**.

For this reason **we are against including into our ecosystem and documentation any software tool or tutorial that aims at implementing solutions that go against Net Neutrality**.

3. Privacy

We believe that privacy is very important for a healthy and well functioning society.

The very first public wifi networks that have been built with OpenWISP in Italy follow this principle very strictly: **traffic logs are stored only for the period of time mandated by law and personal data is never sold to third parties**.

For this reason **we are against including into our ecosystem and documentation any software tool or tutorial that aims at implementing solutions that aim to collect user data with the aim of selling it to third parties without the explicit consent of the user**.

4. Open Source, licenses and collaboration

We release all our software under Open Source licenses on github.com/openwisp.

We mainly use two type of licenses:

- **GPLv3**: we use this license for the software modules which we think have a potentially high commercial value for ISPs and private companies, our aim in using this license is to avoid the inclusion of these tools and modules in proprietary closed source solutions, which would result in private companies profiting from the work of our community without contributing back to it
- **BSD3** and **MIT**: we use these two very permissive licenses for experimental and innovative software modules which are very useful but do not deliver that kind of value which can be monetized easily. Therefore we hope that by allowing these modules to be included in proprietary solutions we will avoid having many organizations reinventing the wheel and we hope that a small percentage of the companies and individuals using them will contribute back even if not explicitly forced by the license

We believe in transparency and also work towards in making this very place as more of a **Community** than a **Top Down Organization** by warmly welcoming any new participant, contributor and user.

We want our community to be supportive, friendly and highly collaborative, with the aim of making the software useful to the broadest possible audience - **as long as our core values are not distorted or ignored**.

We encourage anyone who shares our values to get in touch with us via our support channels and contribute to the project however they can, according to their means and available free time.

5. Software reusability means long term sustainability

Long time contributors of **OpenWISP experienced first hand the consequences of dealing with unflexible monolithic applications** which were hardly reusable outside of the narrow scope for which they were designed.

We have seen countless projects born with great promises, developing their code from scratch and then fading into oblivion, only to notice the same vicious cycle begin again some time later in some other area of the globe; think about it: **what a waste of human effort, energy and resources**!

For this reason, **OpenWISP 2 has a strong focus on modularity and reusability** and follows the **best practices developed in the Unix world** as described in The Art of Unix Programming by Eric S. Raymond.

The core OpenWISP 2 modules are licensed and built in a way that makes it possible for developers not involved in OpenWISP to include these modules in their own applications (according to their licenses).

This is leading to the creation of an ecosystem of modern networking software tools which is attracting developers from all over the world.

The mutual interest of the people who use, modify, share, resell and contribute to these modules is our foundation for **long term sustainability**.

Goals

- Help to solve the problem of lack of internet connectivity by making it easy to deploy and manage low cost network infrastructure all over the world
- Bring innovation in the networking software world by emphasizing automation, modularity, reusability, flexibility, extensibility and collaboration
- Create an ecosystem of software tools that can be used to create infinite OpenWISP derivatives that can be used to make human communication through electronic means easier and more affordable
- Alleviate the problem of vendor lock-in by attempting to support multiple operating systems and hardware vendors (although we now officially support only OpenWRT derivatives, but we do have 2 experimental configuration backends for Raspbian and AirOS)
- Provide good documentation both for users and developers
- Create web interfaces that are easy to use even for people who have limited experience with computer networking concepts

Help us to grow

You don't need necessarily to be a programmer in order to help out.

An apparently insignificant action can have a very positive impact on the project and in this page we'll explain why it's in your interest to help the project grow.

Table of Contents:

Are you using OpenWISP for your organization?

If you are using OpenWISP for your company or no profit organization, it's in your best interest to help the project to grow, because the more we grow as a community, the more contributors we'll attract which in turn will help us to improve the software, its documentation and keep alive the support channels.

Even small and apparently meaningless actions can make a big difference if performed by a sufficient number of people.

Note

If you need commercial support for your business, see the paragraph about Commercial support and funding development.

How to help

1. Open new discussion threads

The Github Discussions Forum and the Mailing List are excellent places to ask questions or share information regarding OpenWISP.

Every question and its replies are archived and indexed by search engines, creating a repository of solved problems that people can find over time.

For this reason, **using these channels for support questions should be preferred over the chats**.

Warning

Please be mindful that **over 700 people read these channels** and **discussions are indexed forever**. For these reasons, you should:

- Keep the focus of the discussion technical.
- Avoid irrelevant comments.
- Be mindful about what you write.
- Keep the tone calm and constructive.
- Be respectful to the volunteers who reply in their free time.
- Avoid generating noise.

When subscribing to the mailing list, we suggest choosing one of these options:

- Receive all emails by creating a filter in your mailbox that moves the messages to a dedicated folder.
- Receive a periodic summary (abridged or digest).

2. Send feedback

When you use OpenWISP, you may find ideas about improvements, new features or you may incur in bugs.

It's very helpful to us if you send us your feedback in some way. The preferred way to send feedback is to use the mailing list, but you can send feedback in any way you want.

If you have found a bug we will likely ask you to open a bug report in a specific github repository, if you can follow up with this activity it will be very helpful to us.

3. Stars on github

Unfortunately, when evaluating a project, a disproportionate amount of people look at the github stars as a method of evaluation on how popular a project is and if they don't see many stars they discard the idea of using it.

OpenWISP is composed of many modules and for that reason we don't have a single super popular github repository with thousands of stars, but when new users and developers look at our github organization page they may not get this at first glance and they will start looking for the numbers of stars.

Yes, we know it sounds silly, but since it does not cost you anything, it would be really useful if you could **take a look at our projects on github** and **star the ones you find most interesting**.

4. Documentation

If you find anything in this documentation that you think may be improved, please edit the document on github and send us a pull request, alternatively you can file a bug report or write to the support channels.

5. Social media

If you are using OpenWISP, it's very useful to let the world know about it by sharing a public post on social media using the *#openwisp* hashtag.

We also have a twitter account and a facebook page you can follow to help us share news about our community.

If more people talk about OpenWISP on social media, we increase the chance that those who have the will and technical skills to contribute will hear about its existence.

6. Blogging

Write a blog post about how you are using OpenWISP!

It would be great if you could explain the reasons for which you chose OpenWISP, the traits you like about it and the traits you don't like about it.

This is **VERY** helpful not only for the core developers but also for potential readers that may find your blog post and read about your use case: maybe they have the same use case and they want to know if OpenWISP is a good fit for them.

A concise, straight to the point blog post with some images and screenshots will go a long way in attracting new people into the community.

7. Conferences & Meetups

If you like to share your knowledge at conferences and meetups, you may cite OpenWISP in one of your presentations or lightning talks, you may also show some of its features, if relevant.

8. Participate

By participating actively in the support channels you can also help us a lot: the welcoming level of an open source community is a key factor in attracting a good numbers of contributors.

9. Contribute technically

Are you skilled in one of the following areas?

- technical writing
- python
- networking
- graphic/web design
- frontend development
- OpenWrt
- Freeradius
- linux
- devops

If yes, you can help us greatly. Find out more about this subject in How to contribute to OpenWISP.

10. Commercial support and funding development

If your company uses OpenWISP for its business and needs professional support on custom setups, development of new features or commercial support, **you can hire a specialist which very active in the community** so they can help you achieve what you need.

Hiring a specialist is usually more effective than trying to figure it out alone: specialists know OpenWISP very well, they can suggest what are the best ways to accomplish something with the least effort, with the highest quality and in the least time at the least cost. Moreover, they will produce solutions that can also be shared with the rest of the community and become part of the OpenWISP ecosystem.

Press

In this page we aim to collect the following:

- presentations, blog posts and academic publications in which OpenWISP is either the main subject or it's mentioned
- logos and other design files

Presentations

OpenWISP: a Hackable Network Management System for the 21st Century

Presented by *Federico Capoano* at the IETF Meeting 103 Bangkok:

- slides

django-freeradius at PyCon Italia 2018

Presented by *Fiorella De Luca* at PyCon Italy 2018:

- video
- abstract

OpenWISP 2: the modular configuration manager for OpenWrt

Presented by *Federico Capoano* at OpenWrt Summit 2017 in Prague:

- video
- slides

Applying the Unix Philosophy to Django projects

Presented by *ederico Capoano* at PyCon Italy 2017:

- video
- slides

Opening Proprietary Networks with OpenWISP

Lightning talk by *Federico Capoano* at DjangoCon Europe 2017:

- slides

OpenWISP2 a self hosted solution to control OpenWrt/LEDE devices

Talk by *Federico Capoano* at FOSDEM 2017 in Brussels:

- video
- abstract

Do you really need to fork OpenWrt?

Presented at OpenWrt Summit 2015 in Dublin:

- video

OpenWISP GARR Conference 2011

Interview for GARR Conference presented by *Davide Guerri* (in Italian):

- video

OpenWISP e Progetti WiFi Nazionali

Interview for GARRTV by *Davide Guerri* (in Italian):

- video

Blog Posts

- How Bottom-up Broadband will overcome the 'last mile' problem
- netjsonconfig: convert NetJSON to OpenWrt UCI
- Automate OpenWrt/LEDE firmware generation with Ansible
- django-x509: a reusable django app for PKI management
- Network Topology Visualizer: django-netjsongraph
- Marco and Alessia for an increasingly open network (in Italian)
- Fly with Uniurb and OpenWISP to the Google Summer of Code 2018 (in Italian)
- Uniurb at the Google Summer of Code with OpenWISP2 and Marco (in Italian)

- Post by the Metropolitan City of Rome (in Italian)

Google Summer of Code Blog Posts

2023 Contributors

- ZeroTier Tunnels Support for OpenWISP Controller by *Aryaman* (*Aryamanz29*).

2022 Contributors

- Iperf3 Check for OpenWISP Monitoring by *Aryaman* (*Aryamanz29*).
- Improve netjsongraph.js for its new release by *Vaishnav Nair* (*totallynotvaishnav*).

2021 Students

- OpenWISP REST API by *Manish Kumar Shah* (*manishshah120*).
- OpenWrt OpenWISP Monitoring by *Kapil Bansal* (*devkapilbansal*).
- OpenWISP WiFi Login Pages by *Sankalp* (*codesankalp*).
- Modern UI/UX by *Nitesh Sinha* (*nitehsinha17*).
- Revamp Netengine and add its SNMP capability to OpenWISP Monitoring by *Purhan Kaushik* (*purhan*).

2020 Students

- Introducing OpenWISP Monitoring: Project report by *Hardik Jain* (*nepython*).
- Merge django reusable-apps by *Ajay Tripathi* (*atb00ker*).
- OpenWISP Notifications Module by *Gagan Deep* (*pandafy*).

2019 Students

- Dockerization of OpenWISP by *Ajay Tripathi* (*atb00ker*).
- Project Report: NetJSONGraph.js Library of OpenWISP by *KuTuGu*.

2018 Students

- OpenWISP IPAM: IP Address Management tool for OpenWISP2 by *Anurag Sharma*.

2017 Students

- Adding AirOS support to netjsonconfig by *Edoardo Putti*.
- Building a Javascript Based Configuration UI for OpenWISP by *Nkhoh Gaston Che*.
- OpenWISP 2 Network Topology by *Rohith A. S. R. K*.
- Google Summer of Code 2017 Django-freeradius by *Fiorella De Luca*.
- Raspbian backend for OpenWISP 2 by *Ritwick DSouza*.

Research and publications

- **`Monitoring Community Networks: Report on Experimentations on Community Networks`**
- **`Network Infrastructure as Commons`**
- **`Bottom-up Broadband Initiatives in the Commons for Europe Project`**
- **`Free Europe WiFi by Justel Pizarro`** (in Spanish)
- **`Bottom-up Broadband: Free Software Philosophy Applied to Networking Initiatives`**
- **`Study of community organizations and the creation of a collaborative environment for the initiative "Bottom up Broadband"`** (in Catalan)
- **`Control and management of WiFi networks`** (in Slovenian)
- **IEEE publication**: ProvinciaWiFi: A 1000 hotspot free, public, open source WiFi network
- **`OpenWISP, an original open source solution for the diffusion of wifi services`** (in Italian)

Logos and Graphic material

`OpenWISP Logo (Black Foreground)`

`OpenWISP Logo (White Foreground)`

`OpenWISP Logo (Black Foreground, with openwisp.org)`

Code of Conduct

1. Purpose

OpenWISP aims to be a welcoming organization for contributors with the most varied and diverse backgrounds possible. We are devoted towards providing a friendly, safe and welcoming environment for all, regardless of gender, sexual orientation, ability, ethnicity, socioeconomic status, and religion.

This code of conduct outlines our expectations for all those who participate in our community, as well as the consequences for unacceptable behavior.

We invite all those who participate in OpenWISP to help us create safe and positive experiences for everyone.

2. Open Source Citizenship

An additional purpose of this Code of Conduct is to boost open source citizenship by encouraging participants to recognize and strengthen the relationships between our actions and their effects on our community.

Communities mirror the societies in which they exist and positive action is essential to prevent the many forms of inequality and abuses of power that exist in society.

If you see someone who is making an extra effort to ensure our community is welcoming, friendly, and encourages all participants to contribute to the fullest extent, we want to know.

3. Expected Behavior

The following behaviors are expected and requested of all community members:

- Participate in an authentic and active way. In doing so, you contribute to the health and longevity of this community.
- Exercise consideration and respect in your speech and actions.
- Attempt collaboration before conflict.
- Refrain from demeaning, discriminatory, or harassing behavior and speech.
- Be mindful of your surroundings and of your fellow participants. Alert community leaders if you notice a dangerous situation, someone in distress, or violations of this Code of Conduct, even if they seem inconsequential.

- Remember that community event venues may be shared with members of the public; please be respectful to all patrons of these locations.

4. Unacceptable Behavior

The following behaviors are considered harassment and are unacceptable within our community:

- Violence, threats of violence or violent language directed against another person.
- Sexist, racist, homophobic, transphobic, ableist or otherwise discriminatory jokes and language.
- Posting or displaying sexually explicit or violent material.
- Posting or threatening to post other people’s personally identifying information ("doxing").
- Personal insults, particularly those related to gender, sexual orientation, race, religion, or disability.
- Inappropriate photography or recording.
- Inappropriate physical contact. You should have someone’s consent before touching them.
- Unwelcome sexual attention. This includes, sexual comments or jokes; inappropriate touching, groping, and unwelcome sexual advances.
- Deliberate intimidation, stalking or following (online or in person).
- Advocating for, or encouraging, any of the above behavior.
- Sustained disruption of community events, including talks and presentations.

5. Consequences of Unacceptable Behavior

We do not tolerate harassment of the participants in any form. Unacceptable behavior from any community member, including sponsors and those with decision-making authority, will not be tolerated.

Anyone asked to stop unacceptable behavior is expected to comply immediately.

If a community member engages in unacceptable behavior, the community organizers may take any action they deem appropriate, up to and including a temporary ban or permanent expulsion from the community without warning (and without refund in the case of a paid event).

6. Reporting Guidelines

If you are being harassed, noticed that someone else is being harassed, or have any other concerns, please contact community organizers immediately.

Additionally, community organizers are available to aid community members to engage with local law enforcement or to otherwise help those experiencing unacceptable behavior feel safe. In the situation of in-person events, organizers will also provide escorts as desired by the person experiencing distress.

7. Addressing Grievances

If you feel you have been falsely or unfairly accused of violating this Code of Conduct, you should get in touch with the OpenWISP community managers by sending a short explanation of your grievance.

Your grievance will be handled in accordance with our existing governing policies.

8. Scope

All community participants (contributors, paid or otherwise; sponsors; and other guests) must abide by this Code of Conduct in all forms of communications within the community such as venues, online and in-person as well as in all one-on-one communications pertaining to community business.

This code of conduct and its related procedures also applies to unacceptable behavior occurring outside the scope of community activities when such behavior has the potential to adversely affect the safety and well-being of community members.

9. Contact info

E-mail:

10. License and attribution

This Code of Conduct is distributed under a Creative Commons Attribution-ShareAlike License.

Portions of text derived from the Django Under The Hood.

Contributing guidelines

We are glad and thankful that you want to contribute to OpenWISP.

Please read these guidelines carefully, it will help you and us to save precious time later.

Introduce yourself

It won't hurt to join our main communication channel and introduce yourself, although to coordinate with one another on technical matters we use the development channel. Use these two channels share feedback, share your OpenWISP derivative work, ask questions or announce your intentions.

Look for open issues

Check out these two kanban boards:

- OpenWISP Contributor's Board: lists issues that are suited to newcomers.
- OpenWISP Priorities for next releases, lists issues that are more urgently needed by the community and is frequently used and reviewed by more seasoned contributors.

If there's anything you don't understand regarding the board or a specific github issue, don't hesitate to ask questions in our general chat.

You don't need to wait for the issue to be assigned to you. Just check if there is anyone else actively working on it (e.g.: an open pull request with recent activity). If nobody else is actively working on it, **just announce your intention to work on it by leaving a comment in the issue**.

Priorities for the next release

When we are close to releasing a new major version of OpenWISP, we will encourage all contributors to focus on the **To Do** column of the OpenWISP Priorities for next releases board and filter the issues according to their expertise:

- **Newcomer**: filter by Good first issue or Hacktoberfest.
- **Expert**: filter by Important.

Setup

Once you have chosen an issue to work on, read the `README` of the repository of the module you want to contribute to, follow the setup instructions, each module has its own specific instructions which we highly advise to read carefully.

How to commit your changes properly

Our main development branch is master, it's our central development branch.

You should open a pull request on github. The pull request will be merged only once the CI build completes successfully (automated tests, code coverage check, QA checks, etc.) and after project maintainers have reviewed and tested it.

You can run QA checks locally by running `./run-qa-checks` in the top level directory of the repository you're working on. Every OpenWISP module should have this script (if a module does not have it, please open an issue on github).

1. Branch naming guidelines

Create a new branch for your patch, use a self-descriptive name, e.g.:

```
git pull origin master
# if there's an issue your patch addresses
git checkout -b issues/48-issue-title-shortened

# if there is no issue for your branch, (we suggest creating one anyway)
# use a descriptive name
git checkout -b autoregistration
```

2. Commit message style guidelines

Please follow our commit message style conventions.

If the issue is present on Github, use following commit style:

```
[module/file/feature] Short description #<issue-number>

Long description here.
Fixes #<issue-number>
```

Here's a real world commit message example from one of our modules:

```
[admin] Fixed VPN context in preview #57

Fortunately it was just a frontend JS issue.
The preview instance was getting the UUID of the Device
object instead of the Config object, and that prevented
the system from finding the associated VPN and fill the
context VPN keys correctly.

Fixes #57
```

Moreover, keep in mind the following guidelines:

- commits should be descriptive in nature, the message should explain the nature of the change
- make sure to follow the code style used in the module you are contributing to
- before committing and pushing the changes, test the code both manually and automatically with the automated test suite if applicable
- after pushing your branch code, make a pull-request of that corresponding change of yours which should contain a descriptive message and mention the issue number as suggested in the example above
- make sure to send one pull request for each feature. Whenever changes are requested during reviews, please send new commits (do not amend previous commits), if multiple commits are present in a single pull request, they will be squashed in a single commit by the maintainers before merging
- in case of big features in which multiple related features/changes needs to be implemented, multiple commits (one commit per feature) in a single PR are acceptable.

3. Pull-Request guidelines

After pushing your changes to your fork, prepare a new Pull Request (from now on we will shorten it often to just *PR*):

- from your forked repository of the project select your branch and click "New Pull Request"
- check the changes tab and review the changes again to ensure everything is correct
- write a concise description of the PR, if an issue exists for
- after submitting your PR, check back again whether your PR has passed our required tests and style checks
- if the tests fail for some reason, try to fix them and if you get stuck seek our help on our communication channels
- if the tests pass, maintainers will review the PR and may ask you to improve details or changes, please be patient: creating a good quality open source project takes a bit of sweat and effort; ensure to follow up with this type of operations
- once everything is fine with us we'll merge your PR

4. Avoiding unnecessary changes

Keep your contribution focused and change the least amount of lines of code as possible needed to reach the goal you're working on.

Avoid changes unrelated to the feature/fix/change you're working on.

Avoid changes related to white-space (spaces, tabs, blank lines) by setting your editor as follows:

- always add a blank line at the end of the file
- clear empty lines containing only spaces or tabs
- show white space (this will help you to spot unnecessary white space)

Coding Style Conventions

1. Python code conventions

OpenWISP follows PEP 8 -- Style Guide for Python Code and several other style conventions which can be enforced by using the following tools:

- `openwisp-qa-format`: this command is shipped in openwisp-utils, a dependency used in every OpenWISP python module, it formats the Python code according to the OpenWISP style conventions, it's based on popular tools like: isort and black (**please do not run black directly** but always call `openwisp-qa-format`)
- `./run-qa-checks`: it's a script present in the top level directory of each OpenWISP module and performs all the QA checks that are specific to each module. It mainly calls the `openwisp-qa-check` command, which performs several common QA checks used across all OpenWISP modules to ensure consistency (including flake8), for more info consult the documentation of openwisp-qa-check

Keep in mind that the QA checks defined in the `run-qa-checks` script are also executed in the CI builds, which will fail if any QA check fails.

To fix QA check failures, run `openwisp-qa-format` and apply manual fixes if needed until `./run-qa-checks` runs without errors.

Note

If you want to learn more about our usage of python and django, we suggest reading Hacking OpenWISP: Python and Django

2. Javascript code conventions

- OpenWISP follows standard JavaScript coding style conventions that are generally accepted or the ones that are specified in .jslintrc files; find out more about JSlint here
- please follow this JavaScript Style Guide and Coding Conventions link for proper explanation and wonderful examples

3. OpenWRT related conventions

OpenWISP follows the standard OpenWRT coding style conventions of OpenWRT:

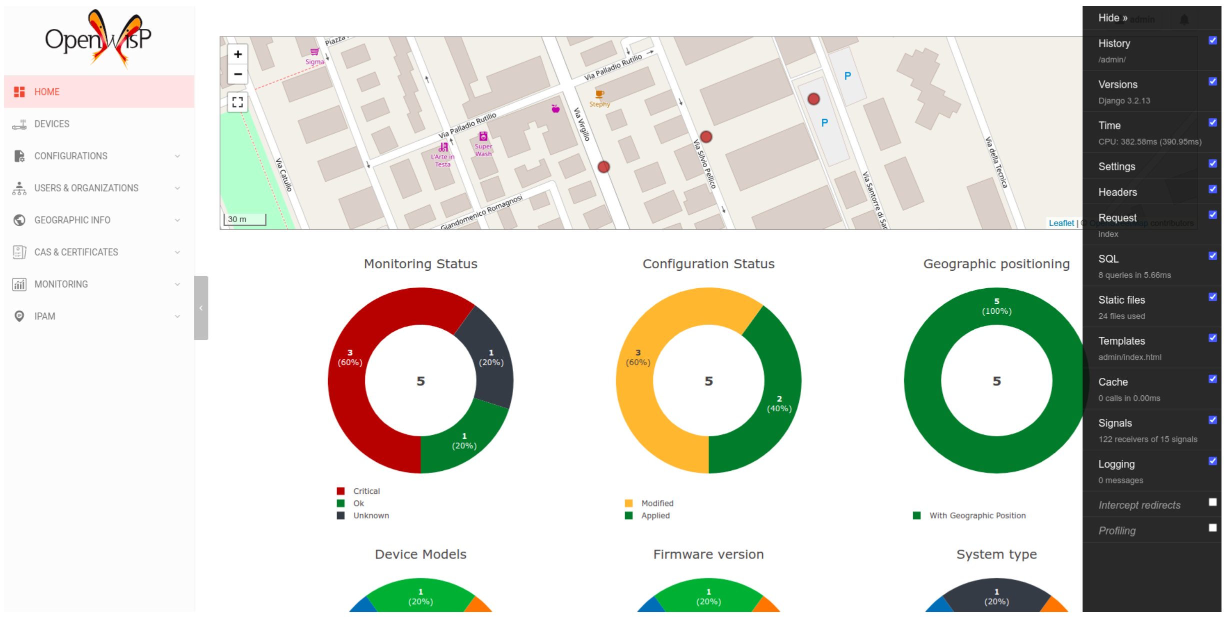
- Working with Patches
- Naming patches
- Adding new files.

Thank You

If you follow these guidelines closely your contribution will have a very positive impact on the OpenWISP project.

Thanks a lot for your patience.

Useful Python & Django Tools for OpenWISP Development



In this page we aim to help users and contributors who want to work on the internal code of OpenWISP in the following ways:

1. By explaining **why OpenWISP uses Python and Django** as its main technologies for the backend application
2. By introducing some Python tools and Django extensions which are **extremely useful during development and debugging**.

Table of Contents:

Why Python?

Note

The first version of OpenWISP was written in Ruby.

OpenWISP 2 was rewritten in Python because Ruby developers were becoming scarce, which led to stagnation. The widespread use of Python in the networking world also played a significant role in this decision.

Python is an interpreted, high-level programming language designed for general-purpose programming, emphasizing productivity, fast prototyping, and high readability.

Python is widely used today, with major organizations like Google, Mozilla, and Dropbox extensively employing it in their systems.

Here are the main reasons why OpenWISP is written in Python:

- It is widely used in the networking and configuration management world. Famous libraries such as networkx, ansible, salt, paramiko, and fabric are written in Python. This allows our users to work with a familiar programming language.
- Finding developers who know Python is not a hard task, which helps the community grow and contributes to the improvement of the OpenWISP software ecosystem over time.
- Python allows great flexibility and extensibility, making OpenWISP hackable and highly customizable. This aligns with our emphasis on software reusability, which is one of the core values of our project.

Resources for learning Python:

- LearnPython.org.
- SoloLearn (a detailed beginner course).

Why Django?

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.

In OpenWISP we chose Django mainly for these reasons:

- It has a rich ecosystem and pluggable apps that allow us to accomplish a lot very quickly.
- It has been battle-tested over many years by a large number of users and high-profile companies.
- Security vulnerabilities are usually privately disclosed to the developers and quickly fixed.
- Being popular, it's easy to find Python developers with experience in Django who can quickly start contributing to OpenWISP.
- Django projects are easily customizable by editing a `settings.py` file. This allows OpenWISP to design its modules so they can be imported into larger, more complex, and customized applications, enabling the creation of tailored network management solutions. **This makes OpenWISP similar to a framework**: users can use the default installation, but if they need a more tailored solution, they can use it as a base, avoiding the need to redevelop a lot of code from scratch.

Resources for learning Django:

- Official Basic Django Tutorial

- DjangoGirls Tutorial (excellent for absolute beginners!)

PS: If you are wondering why the second tutorial mentions the word "Girls," we suggest taking a look at djangogirls.org.

Why Django REST Framework?

Django REST framework is a powerful and flexible toolkit for building Web APIs, used and trusted by internationally recognized companies including Mozilla, Red Hat, Heroku, and Eventbrite.

Here are some reasons why OpenWISP uses Django REST framework:

- Simplicity, flexibility, quality, and extensive test coverage of the source code.
- Powerful serialization engine compatible with both ORM and non-ORM data sources.
- Clean, simple views for resources, using Django's class-based views.
- Efficient HTTP response handling and content type negotiation using HTTP Accept headers.
- Easy publishing of metadata along with querysets.

Resources for learning Django REST Framework:

- Django REST Framework Official Tutorial

Useful Development Tools

IPython and ipdb

IPython (Interactive Python) is a command shell for interactive computing in multiple programming languages, originally developed for Python. It offers introspection, rich media, shell syntax, tab completion, and history.

It provides:

- A powerful interactive shell with syntax highlighting
- A browser-based notebook interface with support for code, text, mathematical expressions, inline plots, and other media
- Support for interactive data visualization and use of GUI toolkits
- Flexible, embeddable interpreters to load into one's own projects
- Tools for parallel computing

More details, including installation and updates, can be found on the official website.

As for ipdb, it allows the use of the `ipython` shell when using the Python debugger (`pdb`).

Try adding this line in a Django project (or an OpenWISP module), for example in a `settings.py` file:

```
import ipdb

ipdb.set_trace()
```

Now load the Django development server and have fun while learning how to debug Python code!

Django Extensions

Django Extensions is a collection of extensions for the Django framework. These include management commands, additional database fields, admin extensions, and much more. We will focus on three of them for now: `shell_plus`, `runserver_plus`, and `show_urls`.

Django Extensions can be installed with:

```
pip install django-extensions
```

shell_plus: Django shell which automatically imports the project settings and the django models defined in the settings.

runserver_plus: the typical `runserver` with the Werkzeug debugger baked in.

`show_urls`: displays the registered URLs of a Django project.

Django Debug Toolbar

The Django Debug Toolbar is a configurable set of panels that display various debug information about the current HTTP request/response and, when clicked, provide more details about the panel's content.

It can be installed with:

```
pip install django-debug-toolbar
```

More information can be found in the django-debug-toolbar documentation.

Using these Tools in OpenWISP

These tools can be added to an OpenWISP development environment to significantly improve the efficiency and experience of development. Here's a guide on how to use them in OpenWISP Controller.

In the `tests/` folder, `local_settings.example.py` should be copied and renamed to `local_settings.py` for customization. This technique can be used in other OpenWISP development environments too.

```
cd tests/
cp local_settings_example.py local_settings.py
```

Follow the installation steps of the OpenWISP Controller module. Run the command `pipenv install --dev`, then run `pipenv run ./manage.py migrate` and `pipenv run ./manage.py createsuperuser`. Ensure SPATIALITE_LIBRARY_PATH is specified in the `local_settings.py` file.

To start the development server with more debugging information, run:

```
python manage.py runserver_plus
```

For an interactive shell, use `ipython` alongside `shell_plus` by running:

```
./manage.py shell_plus --ipython
```

To debug the code, use `ipdb`. For example:

```
ipdb mymodule.py
```

This command will provide a list of lines where errors have been found or lines that can be further optimized.

To use `django-debug-toolbar` for displaying information about processes occurring on the website, some configuration is required. Add the following lines to your `local_settings.py`:

```
from django.conf import settings

settings.INSTALLED_APPS += ["debug_toolbar", "django_extensions"]
settings.MIDDLEWARE += ["debug_toolbar.middleware.DebugToolbarMiddleware"]
INTERNAL_IPS = ["127.0.0.1"]
```

This ensures that the Django Debug Toolbar is displayed. Note that *django_extensions* is already included in `settings.py`.

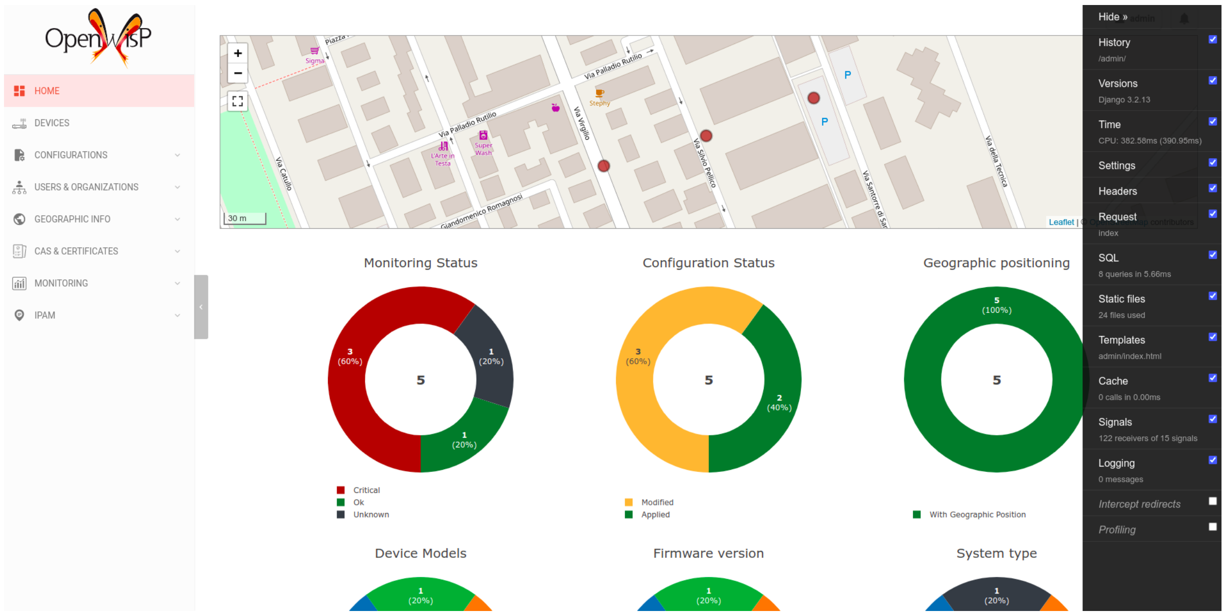
Finally, add the Debug Toolbar's URL to the URLconf of `openwisp-controller` as shown in the installation tutorial, though this should already be present in the last lines of `urls.py`:

```
from django.conf import settings

if settings.DEBUG and "debug_toolbar" in settings.INSTALLED_APPS:
    import debug_toolbar
```

```
urlpatterns.append(url(r"^__debug__/", include(debug_toolbar.urls)))
```

When you open `http://127.0.0.1:8000` in the browser and log in with the credentials created earlier, you should see something like this:



Now that you know the basics, you can experiment and apply these techniques to other OpenWISP modules.

Google Summer of Code

Google Summer of Code

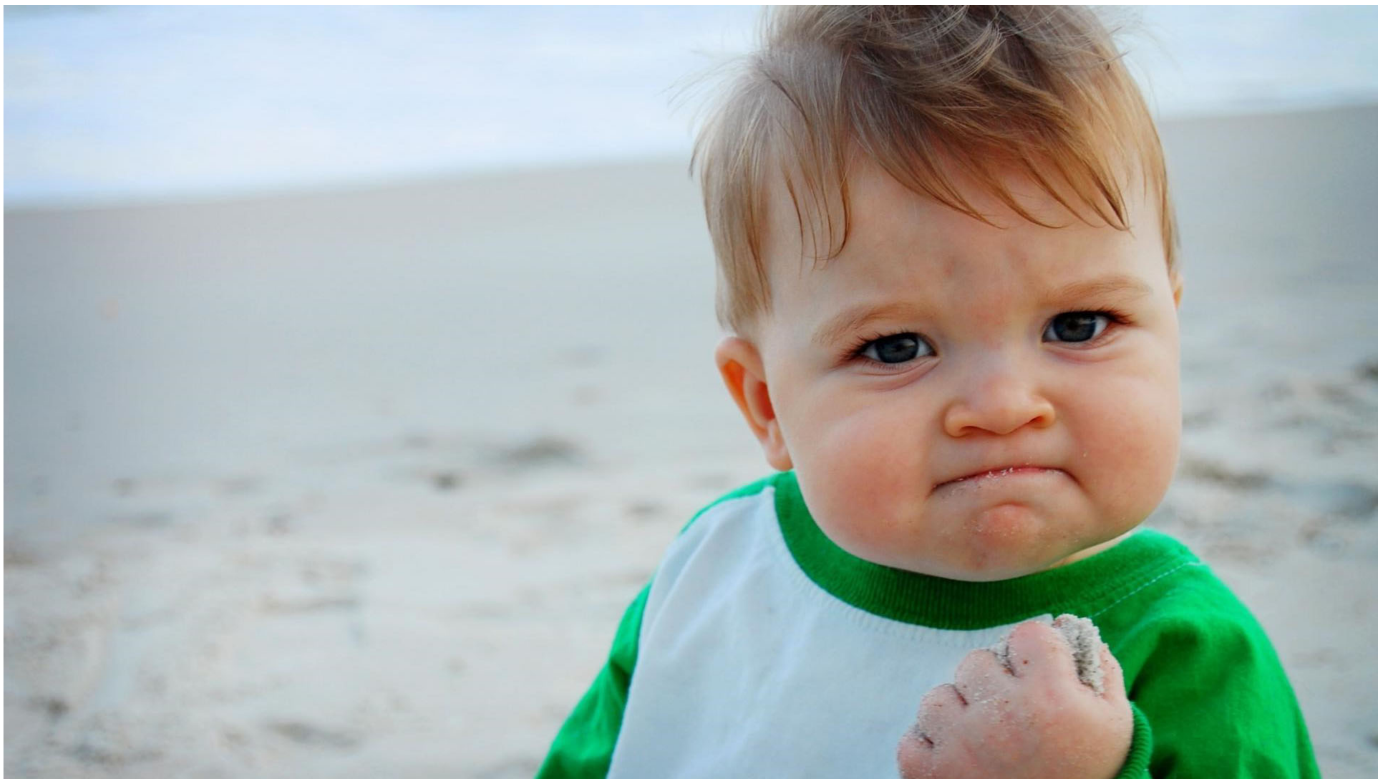
Note

OpenWISP is a mentoring organization for the Google Summer of Code 2022.

If you are reading this page you are probably considering OpenWISP as a possible mentoring organization for the Google Summer of Code, that's great!

If you are looking for a **friendly community** where **your contribution will have a very tangible positive effect from the first day of your participation** and where **you can grow your tech skills at 360°**, then **CONGRATULATIONS!** OpenWISP is the right organization for you.

How to run a successful Google Summer of Code



First of all: PLEASE, PLEASE, read all the information contained in this page (including links!) because this will save everybody involved a lot of time. We would rather spend our time coding than repeating the same stuff over and over.

Have you read the Student manual yet? If not, please do **because it's a MUST if you want to be successful**!

Communication with the rest of the community is vital for a successful Google Summer of Code, please join our communication channels, join our mailing list (we have a dedicated mailing list for GSoC, receive all emails please, and filter them in your mail box so they are moved to an "OpenWISP" folder), present yourself in our general chat, tell us who you are, what your values are, what is attracting to OpenWISP and don't be cold like a robot! Stay human :-).

Traits we look for in applicants

We participate in GSoC because we believe it's a great opportunity for us to give back to Open Source by helping newcomers to get trained and thrive in this industry, but we also do it because we want to grow the pool of maintainers of our project so we can help a greater number of users to use OpenWISP successfully.

Contributors who also become maintainers and start working professionally with OpenWISP are rare, but over time we found out the traits that are good leading indicators for contributors who are likely to become core members of our project, **here are the traits we look for in GSoC applicants which give a higher chance of getting selected**:

- **Genuinely interested in networking**: we look for people who are genuinely attracted in the topics we cover because we believe they are the ones who most likely will benefit from a long term contribution to our project.
- **Participate actively**: they become active participants of the community, not just by submitting pull requests, but also by helping new users or reviewing patches of other less experienced contributors.
- **Put effort in understanding**: they put effort in understanding the problem they need to solve and the outcomes that is expected from them, which means actively researching the problem, expand the project idea with more details, create a prototype, note down a list of questions regarding points that are not clear.
- **Value the time of mentors**: they read carefully the description of issues and put effort in understanding what they have to do, when something is not clear they do not hesitate to explain the problem carefully via email or on github.

- **Parallelize tasks when waiting for a reply**: while they wait for mentors to review or answer their questions, they start tackling other issues for which they have enough information to get started, in order to avoid staying idle.
- **Value quality**: they ensure their work is of the highest quality and does not break existing features of the system thanks to thorough testing before flagging a patch as ready to be merged.

How to become an OpenWISP star

Here's a few quick tricks you can use to become a star in our community:

- read the founding values and goals of OpenWISP, are you on our side?
- study and follow closely the contributing guidelines
- be patient in the interaction with your mentors, we are all volunteers, we are taking our time to mentor you from our free time which we usually spend family and loved ones
- we know our documentation is incomplete and fragmented, we are working hard to fix it; if you find a passage that is not clear or you have an idea about how to improve it, **please let us know!**
- start using OpenWISP 2: install it, run it, play with it; understand its structure
- start contributing (e.g.: fix easy bugs, write documentation, improve tests); look for open issues in our most used repositories on github.com/openwisp (ask in our support channels before starting to code please! we have many legacy repositories that are not under active development anymore)
- if we ask you to open an issue in one of our github repository, please take at least 5 minutes of time to write a proper bug report
- watch the OpenWISP 2 presentation at the recent OpenWRT Summit 2017 and read the slides of this more technical OpenWISP 2 talk
- try using OpenWISP in real use case scenarios (find out if there's a free wifi community near your area), spend time reading its code, ask questions

- try to participate in the community, if a fellow member is in need of help and you know how to help him, please do so, we will reward you

Time to start hacking

If you are not familiar with the following concepts yet, take the time to read these resources, it will help you to speed up your raise to the top!

Programming languages and frameworks:

- Python (book)
- Django (official documentation)
- Lua (video tutorial)
- **Shell**
  (video tutorial)
- Javascript (tutorial)

Networking concepts:

- Introduction to networking terminology

Configuration management:

- Introduction to configuration management
- Writing Ansible playbooks
- Creating Ansible roles from scratch

Project ideas

- Project Ideas 2022

GSoC Project Ideas 2021

Tip

Do you want to apply with us?

We have a page that describes how to increase your chances of success. **Please read it carefully.**

Read our Google Summer of Code guidelines.

Table of Contents:

General suggestions and warnings

- **Project ideas describe the goals we want to achieve but may miss details that have to be defined during the project**: we expect students to do their own research, propose solutions and be ready to deal with uncertainty and solve challenges that will come up during the project
- **Code and prototypes are preferred over detailed documents and unreliable estimates**: rather than using your time to write a very long application document, we suggest to invest in writing a prototype (which means the code may be thrown out entirely) which will help you understand the challenges of the project you want to work on; your application should refer to the prototype or other Github contributions you made to OpenWISP that show you have the capability to succeed in the project idea you are applying for.
- **Students who have either shown to have or have shown to be fast learners for the required hard and soft skills by contributing to OpenWISP have a lot more chances of being accepted**: in order to get started contributing refer to the OpenWISP Contributing Guidelines
- **Get trained in the projects you want to apply for**: once applicants have completed some basic training by contributing to OpenWISP we highly suggest to start working on some aspects of the project they are interested in applying: all projects listed this year are improvements of existing modules so these modules already have a list of open issues which can be solved as part of your advanced training. It will also be possible to complete some of the tasks listed in the project idea right now before GSoC starts. We will list some easy tasks in the project idea for this purpose.

Project Ideas

Improve resiliency and packaging of OpenWISP Monitoring on OpenWrt

Important

Languages and technologies used: Mostly **Lua**, **OpenWrt**, **Makefile** but also a bit of **Python** and **Django**.

Mentors: *Federico Capoano*.

OpenWISP Monitoring depends on specific lua code to be deployed on the OpenWrt devices, this code collects monitoring information and sends it to the OpenWISP server in NetJSON format (see Monitoring Scripts).

At the moment, this code is deployed using a configuration template which is created with a database migration when the monitoring module is installed, but we need to convert this existing code in a new OpenWrt package, well tested, documented and with a key improvement regarding its resiliency.

Prerequisites to work on this project:

The student should be familiar with OpenWISP Templates, OpenWrt, OpenWISP Monitoring and should have a basic knowledge of NetJSON format.

Measurable outcomes:

- Convert lua-monitoring into two OpenWrt packages:

    1. One package for the netjson-monitoring utility, which aims to simply return NetJSON DeviceMonitoring output
    2. One package which provides the OpenWISP Monitoring daemon, which depends on netjson-monitoring and openwisp-config (it takes the server information from the openwisp config file)
- The `netjson-monitoring.lua` file is becoming too big, we have to split it over multiple files
- Each lua function used in the package shall be unit tested, the main cases should be covered, mocks should be used to simulate the different cases
- Achieve at least 93% test coverage
- The daemon shall ran by default every 5 minutes, but this interval shall be configurable
- If for some reason the daemon cannot communicate with the server (e.g.: internet connection is down), the daemon shall:

    - check if there's enough RAM available, if not, stop, otherwise continue to the next point
    - save the data in a new file stored in a subdirectory of `/tmp/` (which is stored in memory), the file should contain the date/time and the data (e.g.: the filename could be the datetime and its contents the data)
- When the daemon sends data to the server, if the HTTP request is successful, it shall check if any stored data is present, if any stored data is present, it shall send it to the server (including the datetime when the measurement was taken) and if the request is successful it shall delete the stored file and proceed with the next file, until every stored data file is sent and deleted
- Write a README like the one of openwisp-config which explains the features of the module, how to install it/compile it
- The OpenWISP Monitoring module needs to be patched to allow the device metrics API to receive measurements that were taken while a device was offline. By default the server will keep assuming implicitly that the datetime of new measurements is the current time, but it will allow the datetime to be passed explicitly

New General Navigation Menu and UX improvements

Important

Languages and technologies used: Mostly **HTML**, **CSS** and **Javascript**, but also a bit of **Python** and **Django**.

Mentors: *Ajay Tripathi*, *Federico Capoano*.

The OpenWISP Admin site has become the most important web interface of OpenWISP, but its usability has not improved much in the last versions, in this project we aim to fix this.

Prerequisites to work on this project:

The student should have installed a full OpenWISP instance running different modules (controller, monitoring and radius) and should be familiar with openwisp-utils.

Measurable outcomes:

- Create a navigation menu with one level nesting which allows to navigate the whole OpenWISP administration site easily and quickly:
    - the menu should look good on major browsers and mobile devices (Chrome/Chromium, Firefox, Microsoft Edge, Safari, Android default browser, IOS default browser)
    - the menu should be responsive and look good on mobile phones
    - on wide screens, the menu will be always visible and on the left side
    - on narrow screens, the menu will appear only when the menu button is clicked, the second levels will also be expanded
- Add the possibility to register menu groups, as well as to specify the order at which the level should be added and an optional icon (needs also tests and documentation)
- Add the possibility to register menu items in levels/groups and specify their order (needs also tests and documentation)
- Ensure the old register_menu_items function keeps working in a backward compatible way: we could add all the items on their own level and log a warning message in the python code which encourages developers to upgrade
- Register the menu items of all the django modules of OpenWISP, by opening a pull request in each respective module:
    - Controller
    - Monitoring
    - RADIUS
    - Network Topology
    - Firmware Upgrader
    - IPAM
- Improve the general theme of the OpenWISP application to be more similar to openwisp.org, we should use a lighter color for, the header, a bigger font, more spacing between elements and we should use bigger buttons and more similar in style to the ones used in the website
- Restyle filters in the django admin list pages: on wide screens, find a way to show filters on top instead of showing them in the lateral sidebars
- **Add basic frontend tests with selenium:**
    - Log in to the admin and ensure the menu is visible
    - Click on an element of the menu
    - Go to a list page and check the filters

OpenWISP REST API

Important

Languages and technologies used: **Python**, **Django**, **Django REST Framework**.

Mentors: *Ajay Tripahi*, *Noumbissi Valere*, *Federico Capoano*.

The goal of this project is to add the much needed missing REST API endpoints for some of the django models of the oldest OpenWISP modules which do not ship a complete REST API.

Prerequisites to work on this project:

The student should have installed a full OpenWISP instance running different modules (controller, network topology) and should be familiar with openwisp-controller, openwisp-users and openwisp-network-topology.

Measurable outcomes:

- Create API endpoints for openwisp-controller:
  - REST API for main controller features
  - *pki* app models CRUD operations
  - *geo* app models CRUD operations
  - *connection* app models CRUD operations
- Create API endpoints for openwisp-users:
  - users (include possibility of changing/updating permissions, groups, organization-users)
  - endpoint to manage email addresses (e.g.: add/remove/change email address, make/unmake primary)
  - organizations CRUD
- Create API endpoints for network-topology: CRUD of all models (Topology, Node, Link)
- Each list endpoint shall be paginated
- Each endpoint should be available only to authenticated users who must either be organization managers and/or superusers, please read the entire **Django REST Framework Permission Classes** section and its subsections Mixins in the openwisp-users documentation
- Each endpoint which is writable and generates a form in the Django REST Framework browsable API shall respect multi-tenancy when showing objects that are related to organizations, please see Multi-tenant serializers for the browsable web UI in the openwisp-users documentation
- Include basic tests for each endpoint, test coverage must not decrease
- Add a basic REST API documentation like the one we have in firmware-upgrader
- Ensure the package DRF YASG is included in the test project of each module touched in this project, as in the Firmware Upgrader and RADIUS modules

Revamp Netengine and add its SNMP capability to OpenWISP Monitoring

Important

Languages and technologies used: **Python**, **Django**.

Mentors: *Gagan Deep*, *Federico Capoano*.

The goal of this project is to add support for SNMP (Simple Network Management Protocol) to OpenWISP Monitoring by using netengine a python library which aims to make easy to access monitoring information via different protocols.

We do not need to maintain backward compatibility at this stage, we have the freedom to change the library how we think is best.

Prerequisites to work on this project:

The student should be familiar with OpenWISP Monitoring and should have a basic knowledge of NetJSON format and SNMP.

Measurable outcomes:

- Revamp the OpenWrt backend of netengine, making it compliant with NetJSON DeviceMonitoring specification

- Revamp the backend for Ubiquiti making it compliant with *NetJSON DeviceMonitoring* as well (we will either buy one hardware model for the student or leave one connected to a VPN)
- Update the unit tests to reflect the changes, ensure all tests pass
- Change tests to use mocks (`unittest.mock`): the tests right now require the physical devices to be run, this is bad: we need to create mocks that allow us to run the tests without the physical devices
- Port code to python >= 3.7
- Create a test build on github actions
- Update docs to reflect the changes introduced in this project
- Remove any code not being used anymore by the new implementation
- Ensure the test coverage stays above 95%
- Modify OpenWISP Controller to allow setting the management IP from the web UI
- Add an SNMP check in OpenWISP Monitoring that pulls the monitoring information and creates the device status and charts

Bring professional efficiency to OpenWISP WiFi Login Pages

Important

Languages and technologies used: **Javascript**, **React JS**, **NodeJS**, **HTML**, **CSS**.

Mentors: *Noumbissi Valere*, *Federico Capoano*.

The goal of this project is to improve OpenWISP WiFi Login Pages by reducing boilerplate code, reduce the amount of configuration lines in the configuration files, improve test coverage and make the code more robust.

Prerequisites to work on this project:

The student should be familiar with OpenWISP WiFi Login Pages, OpenWISP RADIUS and should be proficient with Javascript, React JS, NodeJS, HTML and CSS.

Measurable outcomes:

- Implement gettext like translations: right now translations have to be defined in the configuration file of each organization, repeating the same text over and over, we should avoid this and store the translations in a central place;

  However, being able to customize the text for each organization is a great feature and should still be possible if needed
- Avoid having to repeat the whole configuration options: right now the configuration of each organization contains a lot of boilerplate. We shall introduce default configurations and ensure the application works also when the configuration file of a specific organization misses a piece of configuration.

  When the ability of removing specific sections or fields is needed, right now we resorted to deleting the specific part of the configuration, but once we introduce this change we will have to ensure the configuration options that would have been removed can be set to `null` to obtain the same result
- Rename the directory `org-configurations` to `config`, rename `{slug}-configuration.yml` to `{slug}.yml`, ensure backward compatibility is maintained
- Implement server side logging with a standard logger
- Implement reusable token validation logic
- Increase test coverage to 95%
- Implement basic browser testing with selenium for the following features:

- sign up success
- sign up failure (validation error)
- login success
- login failure
- status

Improve netjsongraph.js for its new release

Important

Languages and technologies used: **Javascript**, **NodeJS**, **HTML**, **CSS**

Mentors: *Federico Capoano*.

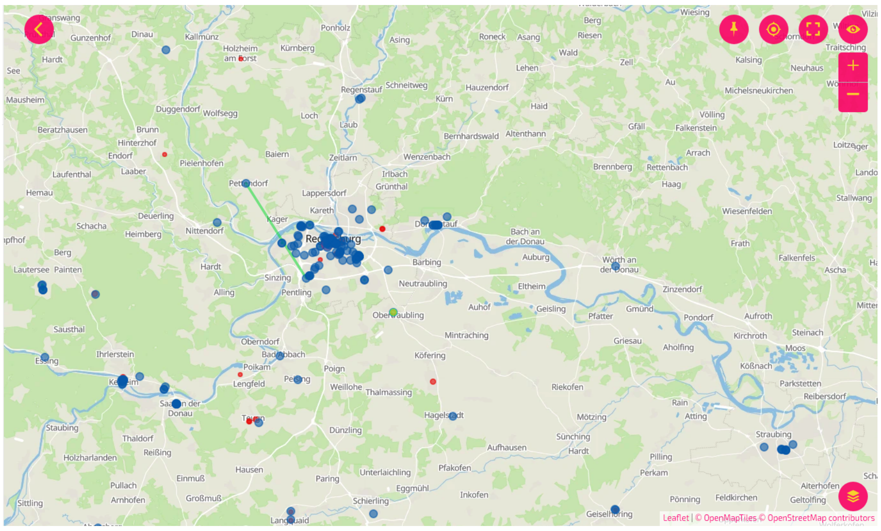
The goal of this project is to improve the new version of the netjsongraph.js visualization library, which is has not been released yet and is available in the gsoc2019 branch of netjsongraph.js on github.

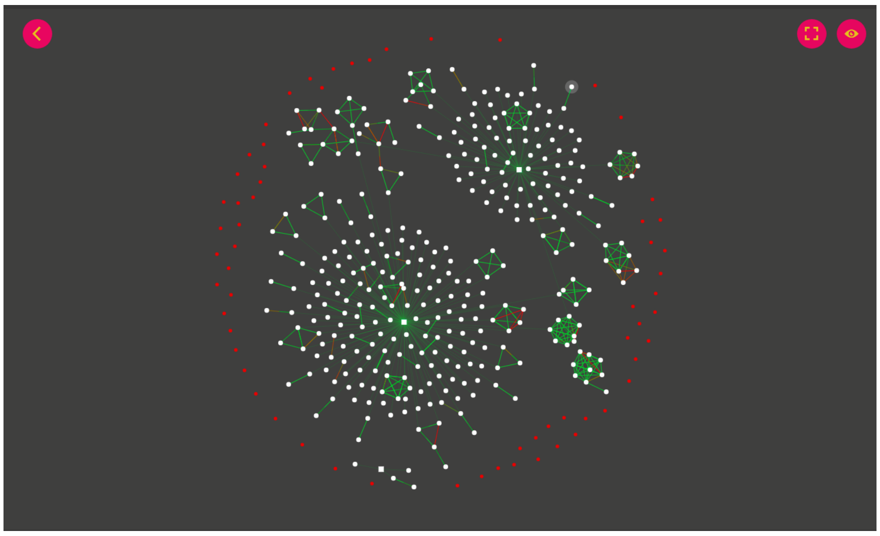
Prerequisites to work on this project:

The student should be familiar with OpenWISP Network Topology and should be proficient with Javascript, React JS, NodeJS, HTML and CSS.

Measurable outcomes:

- We want to make the geographic map feature and the logical map feature more similar to MeshViewer, see the screenshots below for reference, you can find a demo of this application in the repository just linked.





- Fix zoom animation: when the map is zoomed, there's a delay between the zoom of the map and the repositioning of the elements which looks pretty weird
- Add a clustering feature to the geographic map: when there are multiple overlapping elements group them as one cluster:

    - the cluster shall expand when it's hovered with the mouse
    - the cluster shall expand when the map zoom increases
    - the cluster may behave differently if the nodes have links to other nodes, a solution which works well aesthetically should be found
- Test the library on narrow screens and ensure quirks are fixed
- Add support for loading map data using GeoJSON
- Allow loading more than 1000 devices by using pagination, load max 10K points by default (e.g.: `maxPointsFetched`), make this max value configurable
- When more points are present than the configured `maxPointsFetched` value, if the map is zoomed more than a specific level (which shall also be configurable and have a good default), load more data from the API by specifying geographic extent, implement a mocking server for this feature on the server side
- Update OpenWISP Network Topology to use the new version of this library
- Modify OpenWISP Network Topology to provide real time updates
- Change the code of OpenWISP Monitoring so that the map dashboard is implemented using this library instead of using its own custom implementation

Keep in mind the underlying visualization library can be changed if needed.

Second release of OpenWISP Monitoring

Important

Languages and technologies used: **Python**, **Django**.

Mentors: *Gagan Deep*, *Federico Capoano*.

The goal of this project is to improve OpenWISP Monitoring by working on features and changes that have been noted down during the last year of usage of this module.

Prerequisites to work on this project:

The student should be familiar with OpenWISP Templates, OpenWrt, OpenWISP Monitoring and should have a basic knowledge of NetJSON format.

Measurable outcomes:

See the OpenWISP Monitoring 0.2 Release Milestone on Github.

GSoC Project Ideas 2022

Tip

Do you want to apply with us?

We have a page that describes how to increase your chances of success. **Please read it carefully.**

Read our Google Summer of Code guidelines.

Table of Contents:

General suggestions and warnings

- **Project ideas describe the goals we want to achieve but may miss details that have to be defined during the project**: we expect students to do their own research, propose solutions and be ready to deal with uncertainty and solve challenges that will come up during the project
- **Code and prototypes are preferred over detailed documents and unreliable estimates**: rather than using your time to write a very long application document, we suggest to invest in writing a prototype (which means the code may be thrown out entirely) which will help you understand the challenges of the project you want to work on; your application should refer to the prototype or other Github contributions you made to OpenWISP that show you have the capability to succeed in the project idea you are applying for.
- **Students who have either shown to have or have shown to be fast learners for the required hard and soft skills by contributing to OpenWISP have a lot more chances of being accepted**: in order to get started contributing refer to the OpenWISP Contributing Guidelines
- **Get trained in the projects you want to apply for**: once applicants have completed some basic training by contributing to OpenWISP we highly suggest to start working on some aspects of the project they are interested in applying: all projects listed this year are improvements of existing modules so these modules

already have a list of open issues which can be solved as part of your advanced training. It will also be possible to complete some of the tasks listed in the project idea right now before GSoC starts. We will list some easy tasks in the project idea for this purpose.

Project Ideas

Adding support for automatic management of ZeroTier Tunnels

Important

Languages and technologies used: Mostly **OpenWrt**, **Python**, **Django**, **ZeroTier**.

Mentors: *Gagan Deep* (pandafy), *Federico Capoano*.

Project size: 350 hours.

Difficulty rate: hard.

OpenWISP Controller already supports configuring **OpenVPN**, **WireGuard** and **VXLAN over WireGuard** tunnels. The goal of this project is to add support for another VPN backend: ZeroTier.

Prerequisites to work on this project

The contributor must demonstrate good understanding of the following OpenWISP modules:

- netjsonconfig
- OpenWISP Controller
- OpenWISP Network Topology

Any merged patches on any of those modules is considered an important plus point.

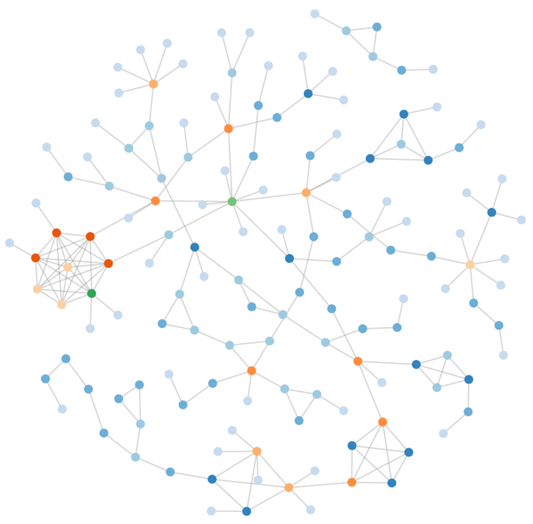
The contributor must also demonstrate familiarity with ZeroTier, and OpenWrt, moreover, they should be willing to increase their experience with these technologies and show enthusiasm toward learning and implementing IT network automation.

Expected outcomes

- Add support for ZeroTier in netjsonconfig:
  - Add capability for generating ZeroTier configuration in OpenWrt backend.
  - Add a ZeroTier backend that generates network configuration accepted by REST API endpoints of the ZeroTier Controller.
  - Write documentation for generating configuration for OpenWrt and ZeroTier Controller using netjsonconfig.

- GitHub Issues:
    - netjsonconfig #207: [feature] Add support for ZeroTier tunnels to OpenWrt backend
    - netjsonconfig #208: [feature] Add ZeroTier backend
- Add ZeroTier as a VPN backend in OpenWISP Controller.
    - Add automatic generation of templates for ZeroTier VPN backend similar to OpenVPN and WireGuard VPN backends.
    - Integrate ZeroTier Controller APIs in OpenWISP Controller to allow managing networks directly from OpenWISP.
    - Write a step by step documentation which explains how to set up and use the new ZeroTier VPN backend with a device.
    - GitHub Issues:
        - openwisp-controller #604 : [feature] Add support for ZeroTier VPN backend
        - openwisp-controller #606 : [feature] Authorize member in ZeroTier network when a new device is added
        - openwisp-controller #605 : [feature] Allow managing ZeroTier networks from OpenWISP
- Add a parser in OpenWISP Network Topology that can parse ZeroTier peer information.
    - Write documentation for using this parser to generate topology from data received from multiple devices.
    - GitHub Issues:
        - openwisp-network-topology #135: [feature] Add a parser for ZeroTier
- Achieve at least 99% test coverage for the code added for this feature.

Improve netjsongraph.js for its new release



Important

Languages and technologies used: **Javascript**, **NodeJS**, **HTML**, **CSS**

Mentors: *Federico Capoano* (more mentors TBA).

Project size: 350 hours.

Difficulty rate: medium/hard.

The goal of this project is to improve the new version of the netjsongraph.js visualization library, which is has not been released yet and is available in the gsoc2019 branch of netjsongraph.js on github.

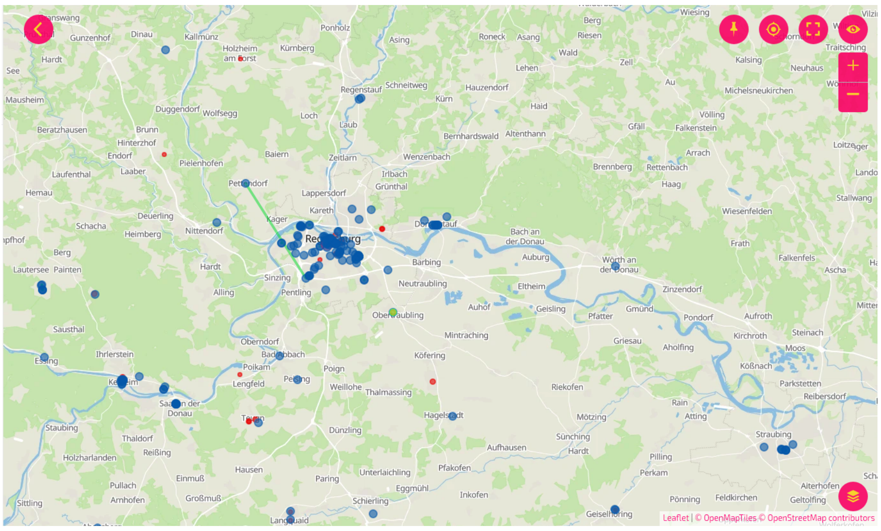
Prerequisites to work on this project

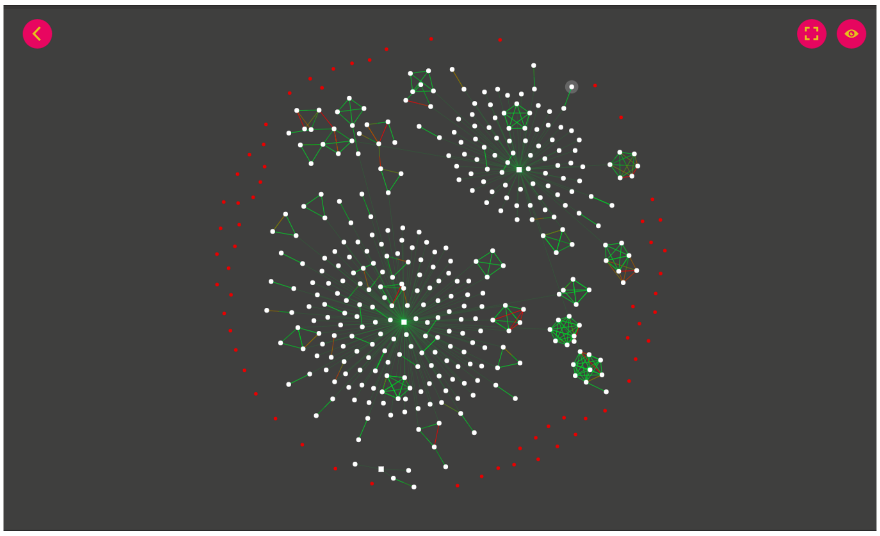
The contributor should have a proven track record and experience with Javascript, React JS, NodeJS, HTML and CSS.

Familiarity with OpenWISP Network Topology and OpenWISP Monitoring is a plus.

Expected outcomes

- We want to make the geographic map feature and the logical map feature more similar to MeshViewer, see the screenshots below for reference, you can find a demo of this application in the repository just linked.



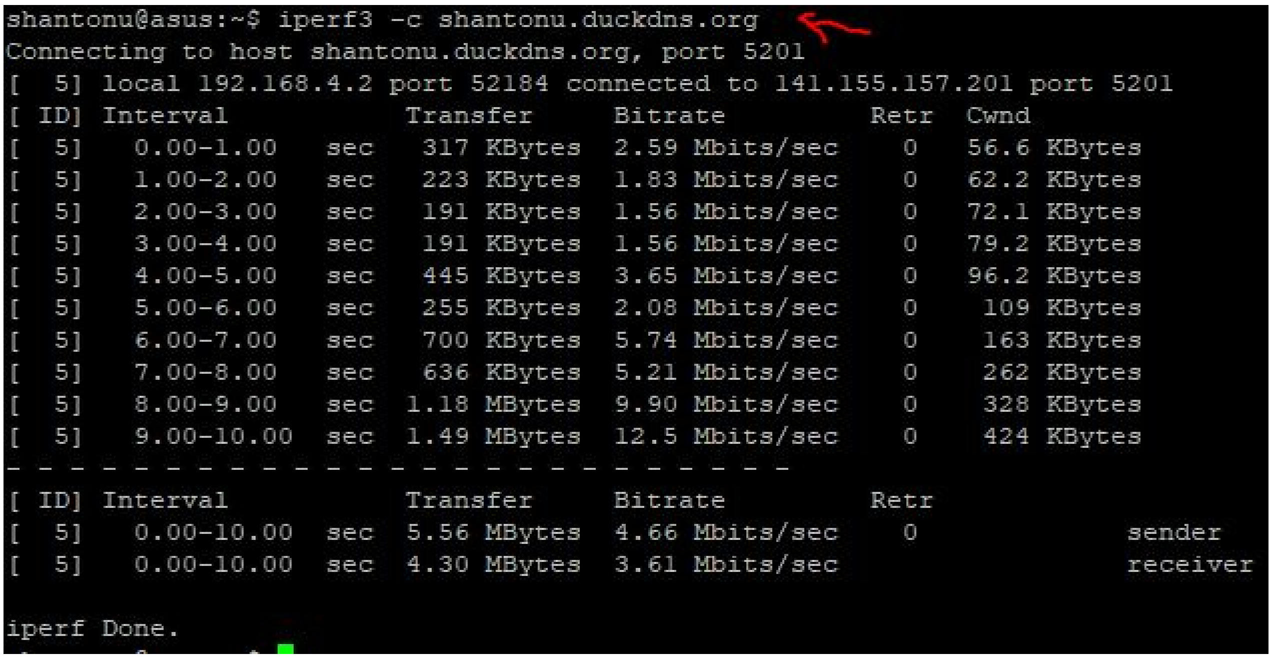


- Fix zoom animation: when the map is zoomed, there's a delay between the zoom of the map and the repositioning of the elements which looks pretty weird
- Add a clustering feature to the geographic map: when there are multiple overlapping elements group them as one cluster:
    - the cluster shall expand when it's hovered with the mouse
    - the cluster shall expand when the map zoom increases
    - the cluster may behave differently if the nodes have links to other nodes, a solution which works well aesthetically should be found

- Test the library on narrow screens and ensure quirks are fixed
- Add support for loading map data using GeoJSON
- Allow loading more than 1000 devices by using pagination, load max 10K points by default (e.g.: `maxPointsFetched`), make this max value configurable
- When more points are present than the configured `maxPointsFetched` value, if the map is zoomed more than a specific level (which shall also be configurable and have a good default), load more data from the API by specifying geographic extent, implement a mocking server for this feature on the server side
- Update OpenWISP Network Topology to use the new version of this library
- Modify OpenWISP Network Topology to provide real time updates
- Change the code of OpenWISP Monitoring so that the map dashboard is implemented using this library instead of using its own custom implementation

Keep in mind the underlying visualization library can be changed if needed.

Add iperf bandwidth monitoring check to OpenWISP Monitoring



```
shantonu@asus:~$ iperf3 -c shantonu.duckdns.org
Connecting to host shantonu.duckdns.org, port 5201
[  5] local 192.168.4.2 port 52184 connected to 141.155.157.201 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   317 KBytes  2.59 Mbits/sec    0   56.6 KBytes
[  5]   1.00-2.00   sec   223 KBytes  1.83 Mbits/sec    0   62.2 KBytes
[  5]   2.00-3.00   sec   191 KBytes  1.56 Mbits/sec    0   72.1 KBytes
[  5]   3.00-4.00   sec   191 KBytes  1.56 Mbits/sec    0   79.2 KBytes
[  5]   4.00-5.00   sec   445 KBytes  3.65 Mbits/sec    0   96.2 KBytes
[  5]   5.00-6.00   sec   255 KBytes  2.08 Mbits/sec    0    109 KBytes
[  5]   6.00-7.00   sec   700 KBytes  5.74 Mbits/sec    0    163 KBytes
[  5]   7.00-8.00   sec   636 KBytes  5.21 Mbits/sec    0    262 KBytes
[  5]   8.00-9.00   sec  1.18 MBytes  9.90 Mbits/sec    0    328 KBytes
[  5]   9.00-10.00  sec  1.49 MBytes  12.5 Mbits/sec    0    424 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  5.56 MBytes  4.66 Mbits/sec    0             sender
[  5]   0.00-10.00  sec  4.30 MBytes  3.61 Mbits/sec                  receiver

iperf Done.
```

Important

Languages and technologies used: **Python**, **Django**, **iperf3**.

Mentors: *Federico Capoano*, *Gagan Deep* (more mentors TBA).

Project size: 175 hours.

Difficulty rate: easy/medium.

The goal of this project is to add a bandwidth test using `iperf3`, using the active check mechanism of OpenWISP Monitoring.

The use case is to perform periodic bandwidth test to measure the max bandwidth available (TCP test) and jitter (UDP).

On a macro level, the check would work this way:

1. OpenWISP connects to the device (only 1 check per device at time) via SSH and launches iperf3 as a client, first in TCP mode, then in UDP mode, iperf is launched with the `-j` flag to obtain json output

2. The collected data is parsed and stored as a metric (bandwidth information and jitter)
3. SSH connection is closed

Prerequisites to work on this project

The student must demonstrate good understanding of OpenWISP Monitoring, and familiarity with Linux and iperf3.

Expected outcomes

The outcomes we expect from this project are the following:

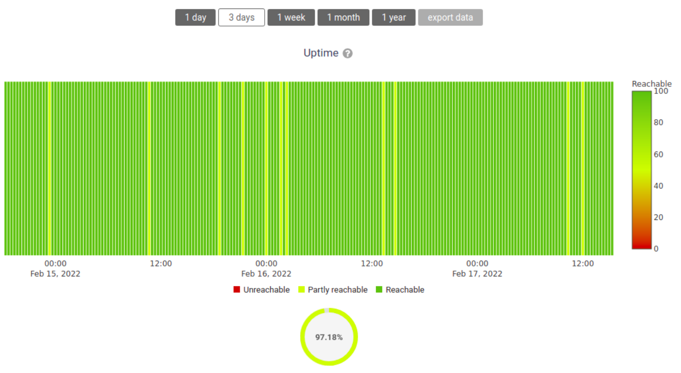
- Create iperf check class, the check must use the connection module of openwisp-controller to connect to devices using SSH
- If a device has no active Connection the check will be skipped and a warning logged
- This check should be optional and disabled by default
- We can run it by default every night
- Allow configuring the iperf server globally and by organization with a setting, e.g.:

```
OPENWISP_MONITORING_IPERF_SERVERS = {
    "": ["<DEFAULT_IPERF_SERVER_HERE>"],
    "<org-pk>": ["<ORG_IPERF_SERVER>"],
}
```

- It shall be possible to specify a list of iperf servers, this is important because on larger systems 1 server will not be enough
- We have to implement a lock to allow only 1 iperf check per server at time that is: for every server available, only 1 check can be performed at any one time, so the lock has to take this account when calculating the cache-key
- SSH into device, launch iperf TCP client, repeat for UDP, collect data of both tests in a data structure
- Handle failures, if server is down, we can store 0, which would trigger an alert (investigate the alert settings functionality)
- Implement logic which creates the metric, chart and alert settings objects
- Save data (tcp max bandwidth, UDP jitter)
- Document how this check works
- Document how to set up and use the check step by step (explain also how to set up a new iperf server)
- Achieve at least 99% test coverage for the code added for this feature.

Github issue: [monitoring/checks] Add iperf check.

Improve UX of OpenWISP Monitoring



Important

Languages and technologies used: **Python**, **Django**.

Mentors: *Ajay Tripathi*, *Federico Capoano*.

Project size: 175 hours.

Difficulty rate: easy.

The goal of this project is to improve OpenWISP Monitoring by working on features and changes that have been noted down during the last 2 years of usage of this module and have the aim of improving the user experience in analyzing the collected monitoring data, as well as the developer user experience in extracting data from the system.

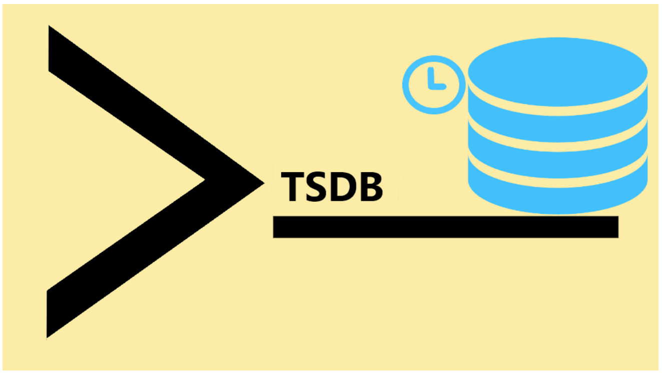
Prerequisites to work on this project

The student must demonstrate good understanding of OpenWISP Monitoring, and should have a basic knowledge of NetJSON format.

Expected outcomes

- [change] Reachable bar chart: show different color for barely reachable #301
- [feature] Charts: allow specifying range of dates #26
- [ux] Show size in (KB, MB or GB) adaptively in charts #87
- [feature] Zooming graphs: reload data in order to provide a detailed view #27
- [feature] Add REST API endpoints for device which include monitoring info #290
- [docs] Add quick start tutorial to README #285
- [feature] Add possibility to connect to *InfluxDB* on unix domain socket #312

Add more timeseries database clients to OpenWISP Monitoring



Important

Languages and technologies used: **Python**, **Django**, **InfluxDB**, **Elasticsearch**.

Mentors: *Federico Capoano*, *Gagan Deep* (more mentors TBA).

Project size: 175 hours.

Difficulty rate: medium.

The goal of this project is to add more Time Series DB options to OpenWISP while keeping good maintainability.

Prerequisites to work on this project

The student must demonstrate good understanding of OpenWISP Monitoring, and demonstrate basic knowledge of NetJSON format, **InfluxDB** and **Elasticsearch**.

Expected outcomes

- Complete the support to Elasticsearch. Support to Elasticsearch was added in 2020 but was not completed.
  - The old pull request has to be updated on the current code base
  - The merge conflicts have to be resolved
  - All the tests must pass, new tests for new charts and metrics added to *InfluxDB* must be added (see [feature] Chart mobile (LTE/5G/UMTS/GSM) signal strength #270)
  - The usage shall be documented, we must make sure there's at least one dedicated CI build for **Elasticsearch**
  - We must allow to install and use **Elasticsearch** instead of **InfluxDB** from ansible-openwisp2 and docker-openwisp
  - The requests to Elasticsearch shall be optimized as described in [timeseries] Optimize elasticsearch #168.
- Add support for InfluxDB 2.0 as a new timeseries backend, this way we can support both `InfluxDB <= 1.8` and `InfluxDB >= 2.0`.
  - All the automated tests for **InfluxDB 1.8** must be replicated and must pass
  - The usage and setup shall be documented
  - We must make sure there's at least one dedicated CI build for Elasticsearch
  - We must allow choosing between **InfluxDB 1.8** and **InfluxDB 2.0** from ansible-openwisp2 and docker-openwisp.

Google Code-in

Note

Google Code-in was a contest that introduced preuniversity students (ages 13-17) to open source software development. OpenWISP has been a mentoring organization in this program for 3 years:

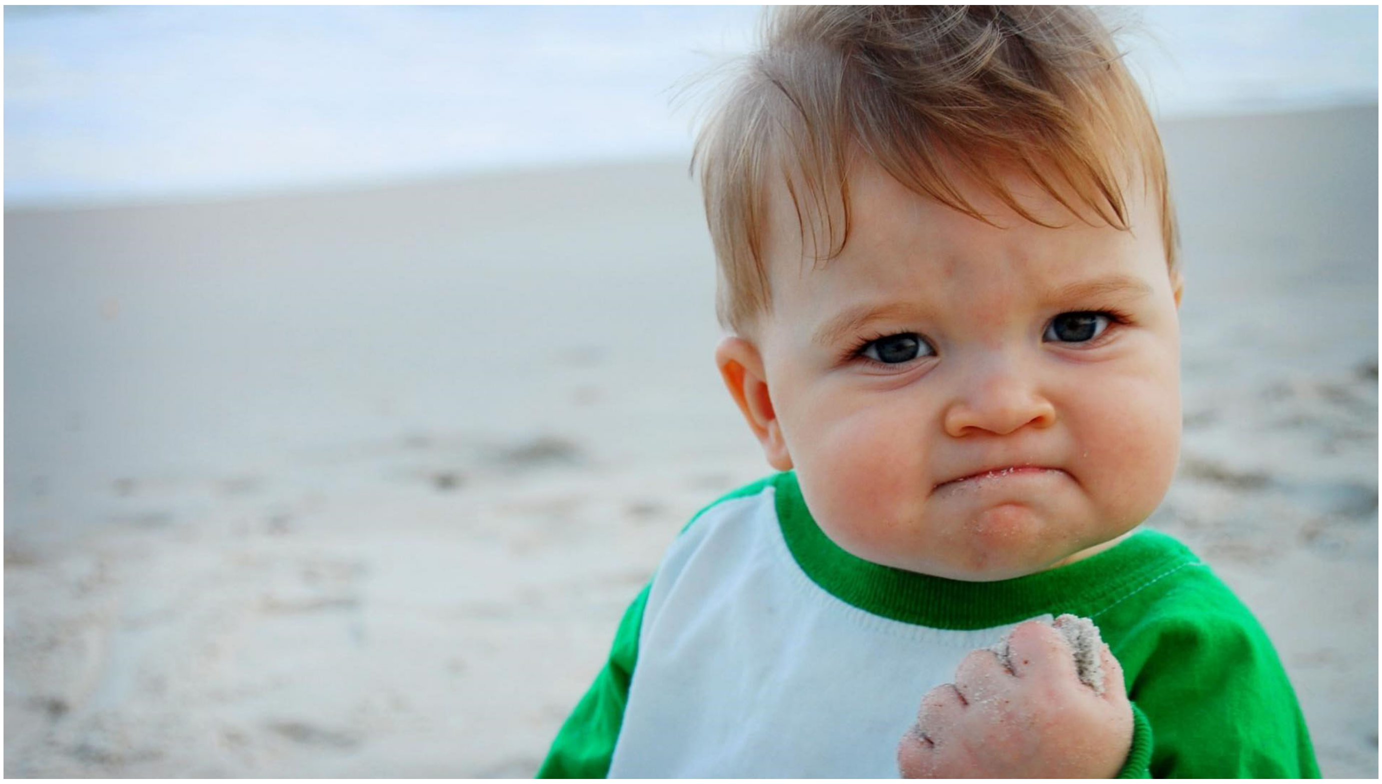
- 2017
- 2018
- 2019

The program was discontinued by Google in early 2020. This page is kept for archiving reasons.

If you are reading this page you are probably considering OpenWISP as a possible mentoring organization for the Google Code-In, that's great!

If you are looking for a **friendly community** where **your contribution will have a very tangible positive effect from the first day of your participation** and where **you can grow your tech skills at 360°**, then **CONGRATULATIONS!** OpenWISP is the right organization for you.

How to run a successful GCI



Have you read the contest rules yet? If not, please do!

!DANGER!

Students: please do not share your personal information (full name, age, location) during the program for safety reasons. This is also a new rule established since 2018.

The most important thing to keep in mind is that you may claim only one task at time and if you decide you don't want to work on it anymore is totally fine but **please** communicate it to your mentors and remove yourself from the task on the Google Code-In dashboard so that someone else will be able to work on it.

Communication with the rest of the community is vital for a successful Google Code-In, please join our communication channels, presenting yourself on the mailing list and on chat, tell us who you are, what your values are, what is attracting to OpenWISP and don't be cold like a robot! Stay human :-).

How to become an OpenWISP star

Here's a few quick tricks you can use to become a star in our community:

- read the founding values and goals of OpenWISP, are you on our side?
- study and follow closely the contributing guidelines
- be patient in the interaction with your mentors, we are all volunteers, we are taking our time to mentor you from our free time which we usually spend with family and loved ones
- we know our documentation is incomplete and fragmented, we are working hard to improve it; if you find a passage that is not clear or you have an idea about how to improve it, **please let us know!**
- the same happens with the software, if you see something which looks like a bug, reach out, even if it's not a bug your feedback will help us to improve
- if we ask you to open an issue in one of our github repository, please take at least 5 minutes of time to write a proper bug report
- watch the OpenWISP 2 presentation at the OpenWrt Summit 2017 and read the slides of this more technical OpenWISP 2 talk
- try using OpenWISP in real use case scenarios (find out if there's a free wifi community near your area), spend time reading its code, ask questions
- try to participate in the community, if a fellow member is in need of help and you know how to help him, please do so, we will reward you

Evaluation criteria

These are the evaluation criteria we will use to select our finalists and vote for the winners.

Be patient

Mentors are volunteers and they have their own obligations to attend.

Avoid asking them continuously to review your task. You are in a queue and you need to wait at least 24 hours as indicated in the GCI guidelines

Hint: optimize your time!

1. ensure the work you submit is of high quality before submitting it
2. if it takes some time for your task to be reviewed, don't sit idle, read our documentation, try OpenWISP, start working on other tasks which allow multiple instances

Progression of Skills

Students should start with simple tasks and gradually progress to more difficult tasks.

Hint: push yourself gradually to harder tasks once you become confident. Leave easy tasks for beginners.

Quality over Quantity

We care more about the quality and impact of your work rather than the quantity of completed tasks.

How we define quality?

Strict adherence to our contributing guidelines, clean readable code, simplicity, elegance, good commit messages.

How we define impact?

Adding a new feature that was highly requested by the community, improving the UX, improve the documentation to help newcomers, anything that facilitates the life of our users has a positive impact on the community.

Hint: find out where the highest impact can be made. Some tasks are more important than others.

Community

Open source is not only about producing code, being active in the community (mailing list, chat, github), helping out fellow students and helping out new users who ask beginner questions is also very important to maintain a healthy community.

Hint: fully embrace the open source community, be helpful to one another. This is the true spirit of open source development.

Help us to grow

Caring for the community also means helping it to grow.

Growing is important because it will allow us to have more mentors in the future so we will able to help out more students.

Hint: try to do some of the easiest actions described in Help us to grow.

Gradual Independence

Over time we expect you to improve and need less micro-managing from mentors, we expect you to become more independent and learn to solve problems on your own.

Hint: do your own research before asking obvious questions; search in the mailing list, in the documentation, on github, on google. Send tasks for review only when you consider your work of good quality. You don't need to rush, keep in mind we value more quality and impact rather than number of completed tasks.

Learn to use OpenWISP

The best contributors are those who actively use the software; students may not have a specific need to use OpenWISP but they can simulate it in order to learn.

Hint: we will appreciate students who will demonstrate good knowledge of how OpenWISP can be used and will help us to write more documentation and tutorials on how to use it.

Learn to use OpenWrt

OpenWrt is one of the most important technologies in OpenWISP, therefore we consider important that students learn the basics of how it works and how OpenWISP can control it.

Hint: start with using a virtual instance of OpenWrt in virtualbox, then when you feel ready get a cheap OpenWrt compatible device that you can use for testing and development. A full list of the OpenWrt compatible hardware is available in the official OpenWrt Website.

Full stack knowledge

OpenWISP is really a full-stack software project, there's everything: python, django, javascript, openwrt, lua, shell scripting, openvpn, freeradius, ansible.

The best contributors are not afraid to learn new technologies and contribute on different fronts.

Hint: we will appreciate students who will spend effort in improving their skills on multiple fronts, rather than focusing exclusively on one specific technology or programming language.

Time to start hacking

If you are not familiar with the following concepts and technologies yet, take the time to read these resources, it will help you to speed up and raise to the top!

Programming languages and frameworks:

- Python (book)
- Django (official documentation)
- Lua (video tutorial)
- Shell (video tutorial)
- Javascript (tutorial)

Networking concepts:

- Introduction to networking terminology

Configuration management:

- Introduction to configuration management
- Writing Ansible playbooks
- Creating Ansible roles from scratch

FAQs

Please refer to the Google Code-in FAQs before participating.

Note

You can ask for help whenever needed, but please don't copy someone else's work. Google Code-in has zero tolerance policy regarding cheating and plagiarism. There are some tasks which require a creative mind like designing logos and T-shirts, which you need to do by yourself. Remember, learning is more important than winning.

Communication of sensitive issues

If you noticed something that you think is not right, for example: a student cheating, a mentor behaving inappropriately or any other issue you don't feel comfortable discussing in public, please get in touch with an organization administrator, the organization admins for the 2019 edition are:

- `2stacks`
- `hispanico`
- `atb00ker`
- `cappe87`
- `nemesisdesign`

How can I apply as mentor?

Thank you a lot for wanting to be a GCI mentor!

OpenWISP Mentors need to be able to guide students, hence they need to have at least a basic knowledge of how OpenWISP works and having contributed actively to the codebase is highly recommended.

If you want to apply, introduce yourself in our general chat, let us know how you are using OpenWISP and how you contributed to it.

If you haven't contributed yet, we highly suggest you to get started now.

Suspension of mentors

Once a mentor has been accepted we assume that the mentor will contribute, according to their available free time, until the conclusion of the program.

The contribution shall be in good faith, always prioritizing the interests of the students and the goals of our organization.

A mentor account may be revoked if these general principles are not followed, more specifically:

- if the mentor disappears without justification for more than 2 weeks; in this case the account can be resumed once the mentor comes back into activity
- if it's felt the participation of the mentor is not in good faith and or not helpful for the students (for example, it's ascertained that the mentors are not putting effort in reviewing and sending feedback to the students, preferring to accept tasks with very shallow or non-existent reviews, just with the goal of scoring mentored tasks)

Hacktoberfest

Note

OpenWISP participates in Hacktoberfest 2021!

If you are reading this page you are probably considering OpenWISP as a possible organization to contribute for Hacktoberfest, welcome!

If you are looking for a **friendly community** where **your contribution will have a very tangible positive effect from the first day of your participation** and where **you can grow your tech skills at 360°**, then **CONGRATULATIONS!** OpenWISP is the right organization for you.

How to get started

1. Read the contributing guidelines

Avoid common pitfalls by reading our Contributing guidelines.

This will result in **your pull requests being merged faster** and less overhead for maintainers.

2. Project Board

Look for issues labeled **hacktoberfest** in the OpenWISP Hacktoberfest Contributor's Board..

Feel free to ask question regarding points which are not clear, but please **ensure your questions are specific**.

3. Announce you're working on something

When you are working on an issue you think you are going to solve, please **let everyone know** by leaving a comment on the Github issue so we can **avoid wasted efforts** from multiple contributors working on the same patch.

However, **if you stop working on it, please also let us know**.

If you find somebody else has announced they're working on an issue you would like to do, you may want to double check they're still working on it by leaving a comment on the issue.

4. Join the general chat

Join our general chat to better coordinate with the community.

5. Help us to grow

Caring for Open Source also means helping its communities grow.

Growing is important because it will allow us to have more mentors in the future so **we will able to help out more contributors to advance their skills**.

Hint: try to do some of the easiest actions described in Help us to grow.

Main Rules

1. Stay on topic

The aim of this program is to help participants learn to contribute to open source meaningfully, which for us means contributing to our mission and end goals. Since new contributors are not suited to work on critical tasks due to their inherent complexity, we prepared a list of easier and well defined issues that can be used to get started, please refer to our OpenWISP Hacktoberfest Contributor's Board.

2. Spammy pull requests won't be accepted

Spammy pull requests containing minor changes (fixing typos, grammars, etc.) that are aimed simply at increasing your Hacktoberfest score will be flagged as invalid.

This behavior is not compatible with the spirit of the program, if you are doing this you are missing the point of Hacktoberfest, Open Source and you are just wasting everyone's time (including yours, because you could be learning something new instead of trying to naively trick the system).

3. Be constructive

Please try to be constructive and patient when interacting with maintainers and contributors.

If you feel you're not treated fairly, please refer to the instructions for reporting unacceptable behavior in our Code of Conduct.

2021
Hacktoberfest
OpenWisP
Presented by DigitalOcean and intel. appwrite deepsource