Archived: ISP Go Content Filtering Platform (Sales Discontinued)

Documentation and description of the ISP Go Filtering Server API methods.

1. Architecture and system requirements

ISP Go solution for Internet Service Providers consists of three components
• Filtering DNS Server
• Web application ‘Block page’
• Web application ‘ISP Go API’ to control and manage user’s settings and integrate the whole ISP Go solution with ISP’s systems.

There are dependencies on the external software packages exist:
• `Nginx` is used as a reverse proxy for the block page and ISP Go API
• `Redis` is used as storage
• `Rsync` is used to download updates for domains database from SafeDNS

When a user enables a content filtering service, an ISP should set the address of the ISP Go server as the default DNS on the user’s computers or devices (via DHCP, connection scripts, and so on) or forcefully redirect all DNS requests from the such user to the ISP Go server.

Using the ISP-Go API (special HTTP requests executed from the provider's scripts) this DNS server is informed about which categories of sites should not be shown to the user. The API also supports individual custom black and white lists and their global variants, which are valid for all users.

If a user types in the browser address of a prohibited site, the filtering DNS server will respond to a DNS packet from the IP address of the block page, then the browser will load the block page. The block page shows a reason why access to this domain is blocked. The block page design is customizable.

Minimal system requirements:

The server on which the ISP-Go service can be deployed and tested with up to 100 users must meet the following minimal requirements:

- Architecture x86-64
- Debian 8, 9 or 10 amd64 installed
- 2GB RAM
- 1.6 GHz 1-core CPU
- 20 GB of free disk space
- 100 Mbit\s network card

Recommended system requirements:

Average usage of the ISP-Go filtering service covers a few thouthands requests per second. To ensure this performance should meet recommended system requirements.

- Architecture x86-64
- Debian 8, 9 or 10 amd64 installed
- 2GB RAM
- 2 GHz 4-cores CPU
- 80 GB of free disk space
- 1 Gbit\s network card

High performance system requirements:

High performance usage of the ISP-Go filtering service covers ~0.5 million requests per second or more. To ensure this performance should meet recommended system requirements.

- Architecture x86-64
- Debian 8, 9 or 10 amd64 installed
- 8 or more GB RAM
- 2 GHz 8-cores CPU or more
- 160 GB or more of free disk space
- 2.5 Gbit\s network card or two 1 Gbit\s network cards

Currently, ISP Go is distributed as a deb-package for amd64 architecture.

For the functioning of the filtering server, an ISP should already have a standard recursive caching DNS server. **Bind9** or **Unbound** with the default settings on the other server is well suited, or you can put one of these DNS servers on the same server as ISP Go and configure it to listen only to the address 127.0.0.1.

The filtering DNS server (`isp-go-dnsproxy`) transfers all non-blocked queries to the caching server and does not cache any data itself. It is possible to pass queries of all ISP’s users (even those for which the filtering service is not enabled) through the filtering DNS server without filtering or with filtering of some default categories.

The "Block page" and "ISP-Go API" web applications are designed as separate daemons, each of them listens to its own port at 127.0.0.1. To transfer requests from the external networks to these web applications, **Nginx** is used, **Nginx** listens to port 80 on the external network interface. Distribution of requests across these web applications is based on the header "Host:" in HTTP requests. All requests with a dedicated hostname for ISP Go API are transferred to the ISP Go API, while all other requests are transferred to the block page.

A provider should have any entity (billing, authentication system), which knows the correspondence between internal IP addresses and end users. A prerequisite for the implementation of the ISP Go solution for ISP is the possibility in the existing ISP’s systems to run a

script when an IP address is issued to a user and (preferably) when the user logs off. Also, ISP should have a specialist who can write such scripts that will invoke SafeDNS ISP Go API on these events (user's login and log off).

A server on which `isp-go-dnsproxy` is installed should be able to send HTTP POST requests to licensing control server on the address www.safedns.com. The absence of such requests is a violation of the license.

> ATTENTION!!! `isp-go-dnsproxy` always rejects non-recursive DNS queries. This is done to eliminate the possibility of an attack like "DNS loop" when isp-go-dnsproxy and Bind transmit requests from one to another without end. Since Bind and other DNS servers always generate non-recursive DNS queries, browsers, and other DNS clients generate only recursive ones, then the loop will be broken. This, however, means that the configuration like "Bind with multiple zones on 127.0.0.1, isp-go-dnsproxy on the external address, and this external address is in NS records in these zones" is unworkable and strongly discouraged. It is not a bug but deliberately introduced limited functionality.

2. Before installation

> The ISP-Go server can be installed on Debian Jessie/Stretch/Buster, choose the amd64 packet. 32bit computers are not supported, arm64, i386, and other packets are not supported.

> Please make sure that the standard repositories are added to the /etc/apt/sources.list, as this will be required for installing additional ISP Go components (nginx, redis, etc.).

After you installed the Debian amd64 OS please run the following commands. All commands must be executed by the `root` user.

- Install `SSH` and `iptables` if they are not installed yet

```
apt-get install ssh iptables
```

- Using `iptables`, block access to port 53 (both UDP and TCP) from networks that do not belong to the provider. A minimal example for an ISP with a *192.168.5.0/24* network:

```
iptables -A INPUT -p udp --dport 53 ! --src 192.168.5.0/24 -j DROP
iptables -A INPUT -p tcp --dport 53 ! --src 192.168.5.0/24 -j DROP
```

> It is impossible to automate this step, because the provider may have some ready-made script for configuring the firewall, and it is a bad idea to add rules blindly on top of it.

- You need to make `iptables` rules automatically apply after a server reboot. One way to achieve this is to install the package `iptables-persistent`:

```
apt-get install iptables-persistent
```

- Save the rules:

```
service netfilter-persistent save
```

3. Installation

Installation Script

To install **ISP Go**, save, grant execution rights, and then run the script below.

Or execute the commands given in the script in the same sequence in the command line manually.

```
#! /bin/bash

LOGIN=$1

PASS=$2

wget https://$LOGIN:$PASS@mirror.safedns.com/repo/ispgo/isp-go-frontend_0.11_all.deb
https://$LOGIN:$PASS@mirror.safedns.com/repo/ispgo/isp-go_1.2.1_amd64.deb

apt update

apt install -y curl daemon dbconfig-common rsync redis-server nginx postgresql nodejs

dpkg -i isp-go_1.2.1_amd64.deb isp-go-frontend_0.11_all.deb
```

Creating an installation script

Open a text editor that is convenient for you (Nano will be used in the example):

```
nano install.sh
```

Copy the commands from the example above into it and save it.

Grant the right of execution for the created script, and then run it using the username and password provided to you by the manager.

```
chmod +x install.sh
./install.sh login password
```

In the process of installation the following configuration files will be created:

- `/etc/isp-go/config.ini` – components configuration

- `/etc/nginx/conf.d/log_format.conf` – configuration of the block page logs format
- `/etc/nginx/sites-available/isp-go-api` – virtual host for handling API requests
- `/etc/nginx/sites-available/isp-go-blocked` – blockpage virtual host

All files should be manually configured to finish the ISP-Go installation.

4. Configuration

Redis configuration

Set the maximum size of memory allocated for data storage and policy of work when this limit will be achieved in `/etc/redis/redis.conf` the file. To prevent loss of data you should use these settings:

```
maxmemory 2GB
maxmemory-policy noeviction
```

To apply the settings restart **Redis server** service:

```
service redis-server restart
```

ISP Go modules configuration

All the modules read the same configuration file `/etc/isp-go/config.ini`
The file consists of several sections.

[dnscache] section

[dnscache] section

This section is used by `isp-go-dnsproxy` and contains only one key - forward. As a value, you should set an IP address and a port of caching DNS server which will be used to resolve all non-blocked DNS requests.

Example:

```
[dnscache]
forward = 8.8.8.8:53
```

[proxy] section

[proxy] section

This section is used by `isp-go-dnsproxy` and contains listen ***blockpage-ip***, ***log***, and ***PID*** keys. The 'Listen' key contains an IP address and a port on which `isp-go-dnsproxy` should get DNS requests from users. It can be duplicated to accept requests on several addresses at once, for example, IPv4 and IPv6 at the same time. The number of keys, as well as the number of listening addresses, is unlimited

`blockpage-ip` key contains an IP address of a block page, which should be passed to users in answers to blocked requests. This is the IP address where **Nginx** accepts requests to the block page. In general, case when filtering DNS server and **Nginx** are on the same server you should set for this key the same IP address as for the listening key. The key can be duplicated to specify an alternative address for the block pages, for example, for IPv6. It is allowed to use at least one key only for A redirects or only AAAA packets, but no more than two to support both IP versions.

> `log, pid` keys contain the absolute path to the log file and PID file accordingly. To prevent loss of compatibility with init scripts from the ISP Go package you should not change the path to the PID file.

Example:

```
[proxy]
listen = 192.168.5.1:53 ; IPv4
listen = [4321:a:bcde:1::2020]:53 ; IPv6
blockpage-ip = 192.168.5.1 ; IPv4 to forwarding A
blockpage-ip = abcd:1234:zyxw::9876 ; IPv6 to forwarding AAAA
log = /var/log/isp-go/isp-go-dnsproxy.log
pid = /var/log/isp-go/isp-go-dnsproxy.pid
```

[datafiles] section

[datafiles] section

This section contains '**path', 'file', 'cats**' keys. **'file'** key can be used several times in this section.

The **'path'** key contains the absolute path to the directory with the SafeDNS domain database. Files, that database is made of, should be enumerated in the **'file'** keys. The order of these keys is important for correct work. Each top-listed file is processed as a correction to all bottom-listed files. To maintain the correct work of the database we prohibit the change of default values for these keys.

The **‘cats’** key contains the absolute path to the JSON file with the list of supported categories. You cannot change this file, because all changes made to it, will be lost on the ISP Go package update. If you need to hide some categories or translate them to some other language you should create a copy of the **catgroups.json** file and make all changes in the copy. The numbers of the categories should not be changed because they are linked to the content of the master database of SafeDNS.

Example:

```
[datafiles]
path = /var/lib/isp-go/filter/
file = host2cat-fast.dat file = host2cat.dat
cats = /usr/share/isp-go/config/
```

[blockpage] section

[blockpage] section

This section is used by the `isp-go-blockpage` application and contains '**listen**', '**templates**', '**log**', and '**pid**' keys.
‘listen’ key contains an IP address (usually 127.0.0.1) and a port that `isp-go-blockpage` daemon listens on for HTTP requests to block page.

> Daemon isp-go-blockpage does not accept requests from users. All requests should be passed through **Nginx**.

The **‘templates’** key contains the absolute path to the directory with templates of block page. Do not change templates installed with the package, because all changes made will be lost on the ISP Go package update. We recommend copying the whole directory `/usr/share/isp-go/templates` and changing templates in this copy.

‘log’ and **‘pid’** keys contain the absolute path to log and PID files accordingly. To prevent loss of compatibility with init scripts from the ISP Go package you should not change the path to pidfile.

Example:

```
[blockpage]
listen = 127.0.0.1:8081
```

```
templates = /usr/share/isp-go/templates/
log = /var/log/isp-go/isp-go-blockpage.log
pid = /var/run/isp-go/isp-go-blockpage.pid
```

[api] section

[api] section

This section is used by the web application `isp-go-api` and contains '**listen**', '**log**', and '**pid**' keys.
'listen' key contains an IP address (usually 127.0.0.1) and a port that `isp-go-api` daemon listens on for HTTP requests to API. An IP address or a port should be different from the set in the **[blockpage]** section

> Listening on an externally accessible IP address will be a security problem. The isp-go-api daemon does not contain any authorization mechanisms, so anyone who can send a request can make any changes to user settings (including someone else's). To prevent such a situation, it is recommended to use the IP address 127.0.0.1 here and to implement external access (with authorization) at the Nginx level.

Example:

```
[api]
listen = 127.0.0.1:8080
log = /var/log/isp-go/isp-go-api.log
pid = /var/run/isp-go/isp-go-api.pid
```

[common] section

[common] section

The section is used by all three daemons and contains the keys '**redis-ip**' and '**redis-port**'.

The '**redis-ip**' and '**redis-port**' keys specify which **Redis** server the daemons that are part of ISP Go should connect to. For performance reasons, it is recommended to run the **Redis** server on the same machine where ISP Go is installed.

Example:

```
[common]
redis-ip = 127.0.0.1
redis-port = 6379
```

Enabling ISP Go services

To enable ISP Go services automatic startup please run the following commands:

```
systemctl enable isp-go-dnsproxy
systemctl enable isp-go-blockpage
systemctl enable isp-go-api
```

Applying settings

After changing the configuration file you should restart all ISP Go services:

```
service isp-go-dnsproxy restart
service isp-go-blockpage restart
service isp-go-api restart
```

Nginx configuration

Nginx is used in ISP Go for the following tasks:
• separation of API requests from requests to the block page and to the administrative web interface
• proxying requests to corresponding web applications
• restricting access to the API and to the administrative web interface

For correct separation of requests, you need to register the domain name used to manage filtering through the API in the `server_name` directive of the file `/etc/nginx/sites-available/isp-go-api` instead of the value 'api.ispgo'.

All other requests will go to the virtual host configured in the file `/etc/nginx/sites-available/isp-go-blocked` due to the presence of the `default_server` modifier in the `listen` directive.

You can create additional virtual hosts with the following exceptions:
• each additional virtual host should contain the `server_name` directive
• `default_server` option cannot be used

• HTTPS usage is not recommended, because users who are requesting blocked websites via HTTPS will get browser warnings of an invalid SSL certificate.

IP addresses and ports in the `proxy_pass` directive in **Nginx** configuration files should correspond with IP addresses and ports on which web applications were launched (see **listen** key in `/etc/isp-go/config.ini` file).

API access is restricted using the `allow` and `deny` directives. Directives are processed in turn from top to bottom until the first match. The default configuration allows access only from the address **127.0.0.1**. You must allow access from the server where the billing system is installed.

> In no case, access should be allowed to the API from untrusted (including user) systems, because if access to the API is provided an attacker can change any filtering settings for any users.

To apply changes reload **Nginx**:

```
service nginx reload
```

Database update

In the installation process demo version of the database will be copied to `/var/lib/isp-go/filter/` folder. For production deployment, you should replace the demo version with the full one and configure automatic updates.
The domain database is updated by **cron** using **rsync**. You need an **ssh key** to authorize access to the **safedns.com** server. To get auto-update to perform the following steps:

Generate an ssh key that will be used to download domain database updates:

```
mkdir safedns-key
cd safedns-key
ssh-keygen -t rsa -N "" -f id_rsa
```

As a result, the files `id_rsa` (private key, which must be kept strictly secret and not lost) and `id_rsa.pub` (public key) will be created.

Send the created `id_rsa.pub` file to the technical support team at support@safedns.com Don't need to send `id_rsa` antwhere!

The technical support team will notify you when the ssh key will be authorized on the SafeDNS server.

Copy the `id_rsa` and `id_rsa.pub` files to the directory where the update script is looking for them:

```
mkdir -p -m 0755 /var/lib/isp-go/.ssh
cd safedns-key
cp id_rsa id_rsa.pub /var/lib/isp-go/.ssh/
chown -R isp-go:isp-go /var/lib/isp-go/.ssh
```

Wait for 1 hour and make sure that `host2cat.dat` and `host2cat-fast.dat` files in the directory `/var/lib/isp-go/filter` have been updated.

You can also update the data-files manually, just run the following commands:

```
su isp-go -c 'rsync -rtv --progress safedns-isp@safedns.com:host2cat.dat ~/filter/'
su isp-go -c 'rsync -rtv --progress safedns-isp@safedns.com:host2cat-fast.dat ~/filter/'
```

> The server should have access to www.safedns.com on TCP port 443.

Setting up sending the statistics

To do this, you need to allow outgoing connections from the isp-go server to www.safedns.com on **TCP** port **443** and perform all the steps from Setting up automatic updating of the domain database (previous part).

To check the correctness of sending statistics on the isp-go server, please perform:

```
curl -f -X POST --data "key=`cut -d ' ' -f 2 /var/lib/isp-go/.ssh/id_rsa.pub | base64 -d | md5sum | cut -d ' ' -f1`&count=`redis-cli --raw hlen ip`" https://www.safedns.com/isp-kit-dog/
```

OK is the correct answer to this command.

Block page design

By default, ISP Go comes with a minimal, strict, and ascetic design of the lock page. To change this design, you need to edit the **HTML** templates that are located at `/usr/share/isp-go/templates/`, where `base.html` – is the main template file, and the rest are inherited from it. The syntax of templates is described in the Go language guide:

- https://golang.org/pkg/text/template/
- https://golang.org/pkg/html/template/

The following variables are available:

- Domain: requested hostname in the Host field of HTTP request header
- Cats: array with category names of the blocked websites. This variable is allowed to use with the `blocked_by_category.html`

template file only.

To add images to a block page we recommend setting a separate virtual host for image storing and using an absolute **URL** to an image with `<img>` tag (example: `<img>http://img.isp.com/block-img.png</img>`)

To apply changes restart the `isp-go-blockpage` service:

```
service isp-go-blockpage restart
```

Peculiarities of listening on specific IP addresses

It is possible to use the **0.0.0.0** IP address in the listen key of a `[proxy]` section of the configuration file if you use caching DNS server installed on another server. In this case, `isp-go-dnsproxy` will process requests on all network interfaces.
If you have caching DNS server installed on the same server (and listening on **127.0.0.1** IP address), you should designate a specific IP address from one of the network interfaces.

User activity statistics

Statistics are recorded to a **CSV** file and imported into the **PostgreSQL** database every 5 minutes. Statistics are stored in the database in "raw" and aggregated form. "Raw" statistics are stored for 1 day. Aggregated statistics are stored for 3 months.
The statistics are recalculated every five minutes.

If a user has more than one IP address, then statistics for all addresses are summarized. Likewise for anonymous users.
The following reports are available:
• number of requests by hours and days for the period;
• number of requests to the top 100 domains;
• detailed statistics by domains and days;
• "Raw" statistics for the last hour.
Statistics are accessed through requests to the **ISP Go REST API.**

5. Examples of API requests

After the initial installation, you need to write a script that will change user settings. The settings are changed using **HTTP** requests to the **ISP Go API**. The examples below use **curl** from the command line to generate requests.

```
# Adding and removing user IP
curl -X PUT -d "[\"192.168.5.1\"]" http://api.isp.com/users/aep/ip/
curl http://api.isp.com/users/aep/ip/
curl -X DELETE http://api.isp.com/users/aep/ip/
curl http://api.isp.com/users/aep/ip/
curl -X PUT -d "[\"192.168.5.1\"]" http://api.isp.com/users/aep/ip/

# Categories list in json format.
curl http://api.isp.com/categories/
curl http://api.isp.com/categorygroups/

# Global white list # Replacing the entire list
curl -X PUT -d '["magazine.com", "feed.com"]' http://api.isp.com/whitelist/
curl http://api.isp.com/whitelist/

# Add and remove one record at a time
curl -X POST -d '["magazine.com"]' http://api.isp.com/whitelist/
curl -X DELETE http://api.isp.com/whitelist/feed.com
curl http://api.isp.com/whitelist/

# Global black list works similarly
curl http://api.isp.com/blacklist/

# Filter configuring:
# Complete replacement of the list of blocked categories:
curl -X PUT -d '[3, 4, 5, 11]' http://api.isp.com/users/aep/filter/

# Add and remove one category at a time:
curl -X POST -d '[6]' http://api.isp.com/users/aep/filter/
curl -X DELETE http://api.isp.com/users/aep/filter/5
curl http://api.isp.com/users/aep/filter/
```

```
# black and white lists of users:
# complete replacement:
curl -X PUT -d '["dark.com", "orange.com", "red.com", "cyan.com"]'
http://api.isp.com/users/aep/blacklist/

# Add and remove one domain at a time:
curl -X POST -d '["antigreen.com"]' http://api.isp.com/users/aep/blacklist/
curl -X DELETE http://api.isp.com/users/aep/blacklist/orange.com
curl http://api.isp.com/users/aep/blacklist/

# White list works similarly
curl http://api.isp.com/users/aep/whitelist/
```

To enable the mode "Use the white list only" for a user you need to add a root domain to the black list:

```
curl -X POST -d '["-"]' http://api.isp.com/users/aep/blacklist/
```

To remove the root domain, use:

```
curl -X DELETE http://api.isp.com/users/aep/blacklist/-
```

> There is a non-removable and non-editable whitelist with the highest priority, which is not accessible through any administration interface, which includes safedns.com addresses, antivirus update sites, Windows update sites, etc.

An example of a PHP script for accessing the API

```
<?php
// The examples file requires PHP curl extension installed
// Address of the API server
$server = 'http://api.isp.com';
function api_request($url, $method = 'GET', $data = null) {
	$ch = curl_init($url);
	curl_setopt ($ch, CURLOPT_HEADER, 0);
	curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
```

```
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
if (!is_null($data)) curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
$out = curl_exec($ch);

  $return_code = curl_getinfo($ch, CURLINFO_HTTP_CODE);

  switch ($return_code) {
      case 200:
        if (strlen($out)) return json_decode($out);
        else return "OKrn";
        break;
      case 400:
        throw new Exception('Bad request');
        break;
      default: throw new Exception('Error');
    }
}
/*
User record is created automatically when trying
To record any information about the user
*/

// Add to a user with an identifier aep ip-address 192.168.5.1
print api_request($server."/users/aep/ip/", 'POST', '["192.168.5.1"]');

// Replace the list of ip-addresses of the user aep to 192.168.5.1, 192.168.5.2
print api_request($server."/users/aep/ip/", 'PUT', '["192.168.5.1", "192.168.5.2"]');

// Delete ip-addresses 192.168.5.1 from the list of addresses of the user aep
print api_request($server."/users/aep/ip/192.168.5.1", 'DELETE');

// Return a list of ip-addresses for the user aep
$result = api_request($server."/users/aep/ip/", 'GET');
var_dump($result);

// Delete the whole list of ip-addresses of the user aep
print api_request($server."/users/aep/ip/", 'DELETE');
?>
```

6. Backup and restore

Set periodic backup of directories:
`/etc/isp-go`
`/var/lib/isp-go/.ssh`

Set periodic backup of **Redis** database (you can get file name and path from the dbfilename and dir parameters of **/etc/redis/redis.conf** configuration file).

Restore process:

• Install **ISP-Go**. During the installation process, specify **127.0.0.1** as the IP from which access to the API is allowed in the `/etc/nginx/sites-available/isp-go-api` file. Do not make API requests until recovery is complete;
• Restore **Redis** database:
  1. Stop Redis service on the new instance - sudo service redis-server stop
  2. Copy the dump.rdb file to /var/lib/redis/ on the new instance
  3. Start Redis service on the new instance - sudo service redis-server start
• Restore files to `/etc/isp-go` and `/var/lib/isp-go/.ssh` from backup directories;
• Restart `redis-server`, `isp-go-dnsproxy`, `isp-go-api` and `isp-go-blockpage` ;

7. ISP-Go slave server

Install second (slave) **ISP-Go** server. Make sure that `isp-go-api` is launched on the master server only. All requests to the API should be sent to the master server.

To prevent `isp-go-api` launch on the slave server and block requests proxying, execute commands:

```
service isp-go-api stop
update-rc.d isp-go-api disable
rm -f /etc/nginx/sites-enabled/isp-go-api
service nginx restart
```

Configure replication between **Redis** instances. To do this, you need to allow the main server to listen at network addresses other than **127.0.0.1** by setting the `bind` parameter in `/etc/redis/redis.conf` to **0.0.0.0**.

```
bind 0.0.0.0
```

or, if needed, you can set it to an IP or IPs of the server

```
bind 127.0.0.1 192.168.5.100 10.0.0.100
```

To apply new settings restart **Redis** on the master server:

```
service redis-server restart
```

> You should restrict access to port **TCP/6379** on the master server which is used by **Redis** to listen. The port should be closed using **iptables** and stay accessible only for the loopback interface (needed for `isp-go-dnsproxy` and `isp-go-blockpage`) and the slave server. An attacker, having access to the **Redis** server via **TCP**, could change any setting of any user, or even worse, force **Redis** to take up all available memory.

Consider an example where the master server has an IP address of **192.168.5.100** and the slave server has an IP address of **192.168.5.200**. In this case, security on the main server is ensured by this **iptables** rule:

```
iptables -A INPUT ! -s 192.168.5.200 -p tcp --dport 6379 ! -i lo -j DROP
```

To save this rule, so that it recovers after a reboot, run the command:

```
service netfilter-persistent save
```

Add to the configuration file `/etc/redis/redis.conf` on the slave server following parameters at the end of the document:

```
slaveof <masterip> <masterport>
```

Then restart **Redis** on the slave server to apply the settings:

```
service redis-server restart
```

8. Migrating from previous versions

There are two utilities for migrating from previous versions:

`isp-go-dumpdb`

`isp-go-loaddb`

The first one creates a dump of the user base through the API, and the second one downloads this dump.

Example:

```
isp-go-dumpdb -file=./dump.json -server=oldapi.myserver.com
isp-go-loaddb -file=./dump.json -server=newapi.myserver.com
```

Where **file** is the path to the file where the database dump will be written, and **server** is the name of the host where the API is located.
API availability can be checked by command:

```
curl -v http://api.myserver.com/active_users/
```

If the error **E403 Forbidden** occurs, open access in following the configuration files:

`/etc/nginx/sites-available/isp-go-api` for ISP-Go

`/etc/apache2/sites-available/isp.comf` for ISP-Kit

> Don't forget to restart services after any chages made in configuration files.

9. ISP Go API

Users

`[user_id]` is the ISP's user ID. It can contain the characters A-Z, a-z, 0-9 underscore and dash.

The maximum length is 32 characters.

Creating a user

A new user is created when trying to create a property for him: specify categories, assign ip.

```
POST /users/[user_id]/ip/[“ip_address”]
```

Deleting a user

```
DELETE /users/[user_id]
```

Check a user existence:

```
HEAD /users/[user_id]
```

If a user is not found returns HTTP response code 404.

Getting a list of active users

```
GET /active_users/
```

Response format:

```
["geek", "bugtest", "hammer"]
```

Active users are users which have at least one IP address.

Getting a list of all users

```
GET /users
```

Response format:

```
[{
  "name": "geek",
  "safesearch": "off",
  "safeyoutube": "off",
  "status": "enabled",
  "filter": [1, 2],
  "ip": ["192.168.5.6", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"],
  "whitelist": ["facebook.com", "google.com"],
  "blacklist": []
},{
  "name": "bugtest",
  "safesearch": "off",
  "safeyoutube": "on",
  "status": "enabled",
  "filter": [1, 2, 3],
  "ip": ["192.168.5.8"],
  "whitelist": ["reddit.com", "google.com"],
  "blacklist": []
}]
```


By default, returns first 100 users.

```
GET /users?start=100&stop=200
```

Start and stop parameters indicate that should be returned users with `user_id` from 100 to 200.

Search a user

```
GET /search/user1
```

Response format:


```
[{
  "name": "user1",
  "safesearch": "off",
  "safeyoutube": "off",
  "status": "enabled",
  "filter": [],
  "ip": [],
  "whitelist": [],
```

```
  "blacklist": []
}]
```

The search is possible by IP or username. Wildcards or masks can be used in the search.

```
GET /search/192.168.0.*
```

Returns all users which IP address starts with `192.168.0.`

```
GET /search/user*
```

Returns all users which name starts with `user`

GET user's information

```
GET /users/[user_id]
```

Response format:

```
{
  "name": "[user_id]",
  "safesearch": "off",
  "safeyoutube": "off",
  "status": "enabled",
  "filter": [3, 4, 5, 6],
  "ip": ["192.168.5.6", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"],
  "whitelist": ["reddit.com", "google.com"],
  "blacklist": []
}
```

Update user's information

```
PUT /users/[user_id]

Content-type: application/json

{
  "name": "[user_id]"
  "safesearch": "on",
  "safeyoutube": "off",
```

```
  "status": "enabled",
  "filter": [2, 3, 4, 5],
  "ip": ["192.168.5.8", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"],
  "whitelist": ["docs.google.com", "atlassian.net"],
  "blacklist" ["youtube.com", "pornhub.com"]
}
```

Disable filtering

This feature allows users to suspend filtering without settings reset.

```
POST /users/[user_id]/status/disabled

Content-type: application/json
```

Enable filtering

```
POST /users/[user_id]/status/enabled

Content-type: application/json
```

SafeSearch

Enabling safe search by default for a user

```
PUT /config/

Content-type: application/json

{ "safesearch": true }
```

Disabling safe search by default for a user

```
PUT /config/

Content-type: application/json

{ "safesearch": false }
```

Enabling safe search by default for users

```
PUT /userconfig/

Content-type: application/json

{ "safesearch": true }
```

Disabling safe search by default for users

```
PUT /userconfig/

Content-type: application/json

{ "safesearch": false }
```

Enabling safe Search for `[user_id]`

```
POST /users/[user_id]/safesearch/on

Content-type: application/json
```

Disabling safe Search for `[user_id]`

```
POST /users/[user_id]/safesearch/off

Content-type: application/json
```

```
PUT /userconfig/

Content-type: application/json

{ "safeyoutube": true }
```

Safe Youtube

If this option is enabled, the user will be automatically redirected to the safe version of YouTube.

Enabling safe YouTube by default

```
PUT /config/

Content-type: application/json

{ "safeyoutube": true }
```

Disabling safe YouTube by default

```
PUT /config/

Content-type: application/json

{ "safeyoutube": false }
```

Enabling safe YouTube by default for users

```
PUT /userconfig/

Content-type: application/json

{ "safeyoutube": true }
```

Disabling safe YouTube by default for users

```
PUT /userconfig/

Content-type: application/json

{ "safeyoutube": false }
```

User addresses

Get the user’s IP address

```
GET /users/[user_id]/ip/
```

Response format:

```
["192.168.5.6", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"]
```

Add user’s IP address (will be added to existing)

```
POST /users/[user_id]/ip/

Content-type: application/json

["127.0.0.1"]
```

or a list of IP addresses:

```
POST /users/[user_id]/ip/

Content-type: application/json

["127.0.0.1", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"]
```

Update user’s IP address (existing IP address will be replaced)

```
PUT /users/[user_id]/ip/

Content-type: application/json

["127.0.0.1"]
```

or a list of IP addresses:

```
PUT /users/[user_id]/ip/

Content-type: application/json

["127.0.0.1", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"]
```

Delete all user’s IP addresses (disables filtering for the user)

```
DELETE /users/[user_id]/ip/
```

Deleting one of the user's IP addresses

```
DELETE /users/[user_id]/ip/[127.0.0.1]
```

General information

Get the grouped list of categories with the names and identifiers of the categories

In order to get categories in the language other than the default one, you should add the HTTP header Accept-Language to the request with language indication (for example pt_BR).

```
GET /categorygroups/
```

Response format:

```
[{
  "group": "Security",
  "categories": {
    "3": "Virus Propagation",
    "4": "Phishing",
    ...
  }
}, {
  "group": "Illegal Activity",
  "categories": {
    "6" : "Drugs",
    ...
  }
},
...
]
```

Get a list of categories with names and identifiers of categories

```
GET /categories/
```

Response format:

```
{
  "3":"Virus Propagation",
  "4":"Phishing",
  …
}
```

Check a website

```
GET /site/facebook.com
```

Response format:

```
{
  "domain": "facebook.com",
  "categories": [29],
}
```

Categories for blocking

Adding a category to the list of blocked by default

```
PUT /config/

Content-type: application/json

{ "filter": [1, 2] }
```

Removing a category from the blocked by default

```
PUT /config/

Content-type: application/json

{ "filter": [] }
```

Adding a category to the list of blocked by default for users

```
PUT /userconfig/

Content-type: application/json

{ "filter": [1, 2] }
```

Removing a category to the list of blocked by default for users

```
PUT /userconfig/

Content-type: application/json

{ "filter": [] }
```

Get a list of user’s categories

```
GET /users/[user_id]/filter/
```

Response format:

```
[1, 2, 3]
```

Add category to user’s list

```
POST /users/[user_id]/filter/

Content-type: application/json

[1, 2]
```

Save new user’s list of categories

```
PUT /users/[user_id]/filter/

Content-type: application/json

[1, 2, 3]
```

Disable one category for a user

```
DELETE /users/[user_id]/filter/2
```

Disable all categories for a user

```
DELETE /users/[user_id]/filter/
```

Black list

Get the user’s black list

```
GET /users/[user_id]/blacklist/
```

Response format:

```
["google.com", "facebook.com"]
```

Add a domain to the user’s black list

```
POST /users/[user_id]/blacklist/

Content-type: application/json

["www.facebook.com"]
```

Replace to new black list

```
PUT /users/[user_id]/blacklist/

Content-type: application/json

["www.facebook.com"]
```

Remove a domain from the user’s black list

```
DELETE /users/[user_id]/blacklist/facebook.com
```

Delete the user’s black list

```
DELETE /users/[user_id]/blacklist/
```

Setting the "White list only" option

For the user to work in the "everything is prohibited except what is explicitly whitelisted" mode, you need to blacklist the root domain:

```
POST /users/[user_id]/blacklist/

Content-type: application/json

["-"]
```

To delete the root domain from the blacklist:

```
DELETE /users/[user_id]/blacklist/-
```

White list

Get the user’s white list

```
GET /user/[user_id]/whitelist/
```

Response format:

```
["google.com"]
```

Add domain to a user’s white list

```
POST /users/[user_id]/whitelist/

Content-type: application/json

["google.com"]
```

Replace the user’s white list

```
PUT /users/[user_id]/whitelist/

Content-type: application/json

["google.com"]
```

Remove a domain from the user’s white list

```
DELETE /users/[user_id]/whitelist/google.com
```

Delete the user’s white list

```
DELETE /users/[user_id]/whitelist
```

Global blacklist (takes precedence over user lists)

Get the global black list

```
GET /blacklist
```

Response format:

```
["google.com"]
```

Add domain to global black list

```
POST /blacklist/

Content-type: application/json

["www.google.com"]
```

Replace to new global black list

```
PUT /blacklist/

Content-type: application/json
```

```
["www.google.com"]
```

Remove a domain from the global black list

```
DELETE /blacklist/google.com
```

Delete the global black list

```
DELETE /blacklist/
```

Setting the option "The global whitelist only"

For the user to work in the "everything is prohibited except what is explicitly whitelisted" mode, you need to blacklist the root domain:

```
POST /blacklist/

Content-type: application/json

["-"]
```

To delete the root domain from the blacklist:

```
DELETE /blacklist/-
```

Global white list

Get the global white list

```
GET /whitelist/
```

Response format:

```
["google.com"]
```

Add a domain to the global white list

```
POST /blacklist/

Content-type: application/json

["www.google.com"]
```

Replace to new global white list

```
PUT /blacklist/

Content-type: application/json

["www.google.com"]
```

Remove a domain from the global white list

```
DELETE /whitelist/google.com
```

Delete the global white list

```
DELETE /whitelist/
```

Reports

User activity by hour

Where `[date_from]`, `[date_to]` are dates in **YYYY-MM-DD** format.

```
GET /users/[user_id]/stat/activity/hour?from=[date_from]&to=[date_to]


{
  "labels": ["2022-06-29 14:00:00", "2022-06-29 15:00:00"],
  "datasets": [{
    "label": "Requests",
    "data": [375, 275]
  },{
    "label": "Blocks",
```

```
    "data": [13, 0]
  }]
}
```

Where:
`labels` - a list of timestamps,
`datasets` - a list of objects containing data for charts and legends,
`data` - a list of values corresponding to timestamps,
`label` - chart name (Requests - number of requests, Blocks - number of blocks).

User activity by day

Where `[date_from]`, `[date_to]` are dates in **YYYY-MM-DD** format.

```
GET /users/[user_id]/stat/activity/day?from=[date_from]&to=[date_to]


{
  "labels": ["2022-06-29", "2022-06-30"],
  "datasets": [{
    "label": "Requests",
    "data": [5770, 3456]
  },{
    "label": "Blocks",
    "data": [8, 9]
  }]
}
```

Where:
`labels` - a list of timestamps,
`datasets` - a list of objects containing data for charts and legends,
`data` - a list of values corresponding to timestamps,
`label` - chart name (Requests - number of requests, Blocks - number of blocks).

Domains

```
GET /users/[user_id]/stat/domains/[filter]?from=[date_from]&to=[date_to]


{
  "labels": ["example.com", "google.com", "asdfg.com"],
```

```
  "datasets": [{
    "label": "Requests",
    "data": [630, 474, 290]
  },{
    "label": "NXdomain",
    "data": [0, 0, 290]
  },{
    "label": "Blocks",
    "data": [630, 0, 0]
  }]
}
```

Where:
`[date_from]`, `[date_to]` - dates in **YYYY-MM-DD** format,
`[filter]` - can be set to all, www
`all` - returns all domains,
`www` - returns only domains start with www
`labels` - a list of domains,
`datasets` - a list of objects containing data for charts and legends,
`data` - a list of values corresponding to timestamps,
`label` - chart name (**Requests** - number of requests, **Blocks** - number of blocks, **NXdomain** - number of requests to non-existing domains).

Detailed

```
GET /users/[user_id]/stat/detail?from=[date_from]&to=[date_to]


{
  "timestamps": ["2022-06-29 15:00:00", "2022-06-29 16:00:00"],
  "reports": [{
    "labels": ["clients4.google.com", "www.google.ru"],
    "cats": [[48, 251], [48]],
    "datasets": [{
      "label": "Requests",
      "data": [29, 20]
    },{
      "label": "NXdomain",
      "data": [0, 0]
    },{
      "label": "Blocks",
```

```
      "data": [0, 0]
    }]
  },{
    "labels": ["live.github.com", "lb._dns-sd._udp.0.0.200.10.in-addr.arpa",
"ssl.gstatic.com"],
    "cats": [[2], [2], [2]],
    "datasets": [{
      "label": "Requests",
      "data": [73, 48, 48]
    },{
      "label": "NXdomain",
      "data": [0, 48, 0]
    },{
      "label": "Blocks",
      "data": [0, 0, 0]
    }]
  }]
}
```


Where:
`[date_from]`, `[date_to]` - dates in **YYYY-MM-DD** format,
`timestamps` - a list of timestamps,
`reports` - a list of appropriate intervals reports,
`labels` - a list of domains,
`cats` - lists of categories for domains,
`datasets` - a list of objects containing data for charts and legends,
`data` - a list of values associated with each domain,
`label` - chart name (**Requests** - number of requests, **Blocks** - number of blocks, **NXdomain** - number of requests to non-existing domains).

Raw statistics

Full user statistics for the last hour. Updates every 5 minutes.


```
GET /users/[user_id]/stat/raw


{
  "fields": ["created", "nxdomain", "address", "host", "blockedhost", "reasom", "cats",
"blockedcats"],
  "data": [
    ["2016-07-11 17:50:26", "0", "10.200.1.88", "tpc.googlesyndication.com",
```

```
"tpc.googlesyndication.com", "4", [2], [0]],
    ["2016-07-11 17:50:26", "0", "10.200.1.88", "tpc.googlesyndication.com",
"tpc.googlesyndication.com", "4", [2], [0]]
  ]
}
```


Response codes

`200` OK - Successful request

`201` Created - Successful element creation request

`204` No Content - Successful request with an empty response

`400` Bad Request - Invalid request

`404` Resource is not found - The resource does not exist

`422` Unprocessable Entity - The request does not contain the

`500` Internal server error