HTTP Notification Service

Features:

- Guaranteed delivery of http notifications
    - No losses even in case of a server's emergency shutdown
    - Repeated delivery in case of unavailability of the recipient or his error
    - Optimized delivery system with configurable attempt intervals and their duration
- Extended support of HTTP recipient notification (url)
    - basic authorization
    - support of POST, GET, PUT, JSON-RPC 2.0 methods of HTTP
    - Correct work with HTTPS recipients
    - support of custom port
- All settings and optimizations in one file (config.yml)
- Verification of domain or page ownership to protect the recipient
    - via robots.txt
    - via meta tags within the html page or the main page of the website
    - configuring exceptions for debugging
- Correct error processing
    - Generates error codes with a short message.
    - the service does not break down when submitting incorrect data
- Logging of sending events to InfluxDB (for import to Grafana, etc.)
- Built-in freeze protection that stops the application in case of inactivity
- In the future, the use of more databases (PostgreSQL, H2, SQLite, Oracle, SQL Server) due to the utilization of the ORM framework

Presentation and demonstration of work on YouTube: **https://youtu.be/XkrFFRWj_UA**

Repository address: https://github.com/chain-action/http-notification/tree/contest
I recommend continuing reading on Github as the PDF may display inaccuracies on various clients.

Information on the instructions https://github.com/freeton-org/readme will be placed in the repository, since the application number is not known before publication

Contact:

- Telegram: @webcounters
- Surf: 0:a9ef47b6bec35e001d1f295b34b9ec9abc0ca5c8623de4f414b4fd0b0dc6ca08

Documentation

Testing

To test the fact of receiving http notifications, you can use the following publicly accessible services:

- https://pipedream.com/ (recommended, has a shutdown support)
- https://requestinspector.com/ (the simplest one)
- https://mocklab.io/ (beautiful)

Important: When testing other URLs (the above URLs are added to debug exceptions), you must verify ownership of the domain or page.

Testing with decryption support

A special test server with protocol decryption support was developed to receive notifications

Installation

Fast launch with docker

Required applications

- Docker (install)
- Docker Compose (install)

Install

- Download and unpack repository `https://github.com/chain-action/http-notification`
- Go to directory ./docker `cd ./docker`
- Create Docker Volumes for Mysql Data Persistent Storage

```
docker volume create --driver local --opt type=none \
    --opt device=/home/user/notify_db \
    --opt o=bind notify_db
#docker volume create --name=notify_db  # For Windows or build via Docker Desktop
```

`*/home/user/notify_db` replace for full path on your system

- Start the docker application (change ports in the `.env` file if they are busy)

```
docker-compose up --build -d
```

Manual installation

Required applications

- MariaDB or Mysql (install)
- Java JRE (OpenJDK JRE at least 15 version)
- Redis (install)
- InfluxDB 2.0 (install) (optional)

Install

- Create application directory and go to it

```
mkdir app
cd app
```

- Download or compile the application into an executable JAR file

```
wget -O HttpNotification.main.jar
https://github.com/chain-action/http-notification/releases/download/v1.0.0/
HttpNotification.main.jar
```

- Create file `config.yml` based on `example.config.yml` and edit its parameters (mysql, kafka, influxdb sections). More on config.yml parameters

```
cp example.config.yml config.yml
vim config.yml
```

- Application initialization (creating tables in the database)

```
java -Dfile.encoding=UTF-8 -jar HttpNotification.main.jar --createtable
```

- Application launch

```
java -Dfile.encoding=UTF-8 -jar HttpNotification.main.jar
```

- Configure a backend server (nginx, apache, etc, ...) for API accessibility via https protocol

Examples `location` for Nginx

```
    location ~ ^/(v1|v0){
      proxy_pass http://127.0.0.1:8010;
    }
```

Usage application

Misc for developers

```
# Running helper services (mysql,) in docker
docker-compose -f docker-compose.dev.yml -p http_notify_dev up --build --force-recreate
-d

# Application initialization (creating tables in the database)
java -Dfile.encoding=UTF-8 -jar HttpNotification.main.jar --createtable
# Application launch
java -Dfile.encoding=UTF-8 -jar HttpNotification.main.jar
```

Usage

Attention: we should replace `localhost` with Ip or domain of your service

Adding notification entry

api v0

Send by method POST to `http://localhost:8010/v0/jsonrpc` with hash (unique entry hash) and data (base64 encoded URL) parameters

```
curl 'http://localhost:8010/v0/jsonrpc' -X POST -H 'Content-Type: application/x-www-
form-urlencoded' --data-raw
'hash=0a0b0c0d&data=aHR0cDovLzEyNy4wLjAuMTo4MTgxL3JlcXVlc3Q='
```

api v1 (beta)

JSON-RPC request to `http://localhost:8010/v1/jsonrpc`

Minimalistic example

```
curl --header "Content-Type: application/json" --request POST \
  --data '{"jsonrpc": "2.0","id":1,"method":"add", "params": {"hash":"0a0b0c0d",
"url":"http://127.0.0.1:8181/request"}}' \
  http://localhost:8010/v1/jsonrpc
```

Advanced example

```
curl --header "Content-Type: application/json" --request POST \
  --data '{"jsonrpc":"2.0","id":1,"method":"add","params": {"hash":"0a0b0c0d",
"method":"POST", "url":"http://127.0.0.1:8181/request", auth={"user":"user1",
"pass":"password"}}}' \
  http://localhost:8010/v1/jsonrpc
```

Decoding json object parameters API v1

Name	Description
hash	unique hash of the entry
method	http methods: POST, PUT, JSONRPC, GET. default POST
url	url address: https://domain/request
auth	Authorization object. default `null`, Basic authorization: `{"user":"user1", "pass":"password"}`, Bearer token: `{"bearer":"secret-token"}`

You can get all parameters by API

Protecting the recipient page

Allowing access through robots.txt

Add an entry to the `robots.txt` file entry where `FtPro-Notify-Bot` is a parameter from the `config.yml` config or via api.

```
User-agent: FtPro-Notify-Bot
Allow: /
```

Security code for the page

You need to add the `<meta name="ftpro-notify-verification" content="...">` entry inside the head tag on the page

```
<head>
...
  <meta name="ftpro-notify-verification"
content="69aad49845d88aee43e5b696a5b249abb91a69b18dee85eaf5c76d31970b04fc">
</head>
```

Getting code via API

Post a request to address `http://localhost:8010/v0/approve-code` with the `url` parameter equal to the computed URL

```
curl -X POST "http://localhost:8010/v0/approve-code" -F "url=https://testn1.free-
ton.online/test_push.php"
```

You will get a JSON object

```
{
  "result": {
    "metaName": "ftpro-notify-verification",
    "success": true,
    "robotsTxt": "FtPro-Notify-Bot",
    "metaUrl": "69aad49845d88aee43e5b696a5b249abb91a69b18dee85eaf5c76d31970b04fc",
    "metaMain": "69e9029faf0644ed77a6ea92e9195764b1c662d642b4861b221ebffb102d506d"
  },
  "id": "1",
  "jsonrpc": "2.0"
}
```

Name	Description
metaName	parameter `name` for meta tag
metaUrl	parameter `content` for the page meta tag
metaMain	parameter `content` for the main page meta tag

**metaMain for Main URL including /*

Offline code computation

SHA 256 string

Pages: Web{URL}Notify, main page: Web{Main URL including /}Notify

Examples

```
echo -n Webhttps://testn1.free-ton.online/test_push.phpNotify | sha256sum
echo -n Webhttps://testn1.free-ton.online/Notify | sha256sum
```

Parameter structure

Required to create a notification entity via the API. Available at `http://localhost:8010/v0/structure`, but only practical in v1 `http://localhost:8010/v1/structure`


```
{
  "result": {
    "library": {
      "types": ["string", "int", "list", "object"],
      "list": {
        "method": ["GET", "PUT", "POST", "JSONRPC"]
      }
    },
    "success": true,
    "list": [
      {
        "descr": "Unique hash of the event ",
        "name": "hash",
        "type": "string",
        "required": true
      },
      {
        "descr": "url address",
        "name": "url",
        "type": "string",
        "required": true
      },
      {
        "descr": "List of available http methods",
        "default": "POST",
        "name": "method",
        "type": "list",
        "required": false
      },
      {
        "descr": "Parameter name",
        "default": "param",
        "name": "query",
        "type": "list",
        "required": false
      },
      {
        "descr": "Authorization object",
        "name": "auth",
        "type": "object",
        "required": false
      }
    ]
  },
  "id": "1",
  "jsonrpc": "2.0"
}
```

This API can be used to automatically generate a user interface form. For example, you can get a list of options for the `method` parameter at `library.list.method`

Service Information

`http://localhost:8010/v0/info`


```
{
  "result": {
    "success": true,
    "info": {
      "descr": "service description",
      "logo": "https://service.domain/logo.svg",
      "title": "Service name",
      "support_surf":
"0:a9ef47b6bec35e001d1f295b34b9ec9abc0ca5c8623de4f414b4fd0b0dc6ca08"
    }
  },
  "id": "1",
  "jsonrpc": "2.0"
}
```