automua

Email client configuration made easy

Version 2023.0, 2023-01-03

1. Copyright

automua is a fork of automx2. Check per-file copyright notices to see the sole modifications and the new files of the fork automua that are Copyright © 2022 Gaspard d'Hautefeuille. automx2 and most of the files of the fork are Copyright © 2019-2022 Ralph Seichter. Patrick Ben Koetter has contributed to the documentation.

automua is licensed under the GNU General Public License V3 or later. The project is hosted on GitHub in the HLFH/automua [https://github.com/HLFH/automua] repository.

2. Contact

If you have questions about setting up or operating automua, or if you would like to discuss or suggest features, please use issue tracker [https://github.com/HLFH/automua/issues].

Should you be interested in supporting the project as a sponsor, you can find a contact email address in the sponsorship section.

3. Preface

This document explains how automua works, how automated mail client configuration works, and what it takes to install and configure automua. If you are already familiar with automated mailbox configuration methods you may want skip the following sections and jump right ahead to Installing automua and Configuring automua.

4. How does automua operate?

automua is a web service. It is usually located behind a web server like NGINX and waits for configuration requests. When a mail user agent (MUA), a.k.a. mail client, requests configuration it contacts the web server. The web server then acts as a proxy and forwards all requests to automua and passes answers back to the MUA.

5. How does auto config work?

Modern email clients (Mail User Agents) can look for configuration data when a user begins to create a new account. They will either send the user's mail address to a service and ask the service to reply with configuration that suits the user's profile or they will query the DNS system for advice.

Using a specialized mail account configuration service allows for individualized setups. It also allows to enforce a specific policy, which for example configures the mail client to use a specific authentication mechanism. Quering the DNS for mail service locations allows for generic instructions, but it doesn't give as much control over settings as a specialized service like automua will do.

As of today, there are four methods that help configuring a mail account. Three of them – Autoconfig, Autodiscover and Mobileconfig – have been developed by vendors to cover their products' specific needs. The fourth is an RFC standard specifying the aformentioned more general DNS SRV resource records method.

The vendor specific methods have in common that the mail client seeking configuration needs to send a request, which includes at least the user's mail address, to a configuration service. The service will use the mail address to lookup configuration data and will return that data as response to the client. Format – XML response or file – and complexity differ depending on the method.

automua implements everything necessary to configure email accounts. Functionality to configure calendar or address book settings is not included. This may change in some future version, but the focus of automua is email.

5.1. Autoconfig

Autoconfig is a proprietary method developed by the Mozilla foundation. It was designed to configure a mail account within Thunderbird, and other email suites like Evolution and KMail have adopted [https://wiki.mozilla.org/Thunderbird:Autoconfiguration:ConfigFileFormat] the mechanism.

When a user begins to create a new mail account she is asked to enter her realname and mail address, e.g. *alice@example.com*. Thunderbird will then extract the domainpart (*example.com*) from the mail address and build a list of URIs to search for a configuration web service in the following order:

```
https://autoconfig.thunderbird.net/v1.1/example.com
https://autoconfig.example.com/mail/config-v1.1.xml?emailaddress=alice@example.com
https://example.com/.well-known/autoconfig/mail/config-v1.1.xml
http://autoconfig.thunderbird.net/v1.1/example.com
http://autoconfig.example.com/mail/config-v1.1.xml?emailaddress=alice@example.com
http://example.com/.well-known/autoconfig/mail/config-v1.1.xml
```

A configuration service such as automua listening on one of the listed URIs will receive the request, process it and respond with a set of configuration instructions.

Thunderbird will use the instructions to automatically fill in the required fields in the account. The only remaining task for the user is to confirm the settings. After that she can immediately start to use her new mail account.

5.2. Autodiscover

Autodiscover is a proprietary method developed by Microsoft. The protocol version supported by automua was designed to configure a mail account within Outlook 2016. Service lookups use the

URLs shown below and, as a fallback option, DNS lookups. Please note that Microsoft uses a different autodiscover mechanism for Office 365, which is not yet supported by automua because information about the technical details are not available free of charge.

```
https://example.com/autodiscover/autodiscover.xml
https://autodiscover.example.com/autodiscover/autodiscover.xml
http://autodiscover.example.com/autodiscover/autodiscover.xml

dns: autodiscover.example.com
dns: _autodiscover._tcp.example.com
```

All HTTP(S) queries send a POST request and submit XML which contains information about the account that should be configured. The DNS queries search for a `CNAME` resource record first, which is supposed to redirect the mail client to a resource outside of the mailbox owners domain, e.g. *alice@example.com* would be redirected to `service.example-provider.com` for configuration instructions. If the first DNS query fails the client may be redirected to a configuration service using a `SRV` RR like this:

```
_autodiscover._tcp.example.com.  0   443 service.example-provider.com.
```

The `SRV` RR used in the example above would send Alice's client to `service.example-provider.com` and tell it to send the query to the configuration service on port `443`.

5.3. Mobileconfig

Requests and responses use proprietary content types with an underlying property list [https://en.wikipedia.org/wiki/Property_list] format. automua will return *unsigned* data, which means that the service must be accessed via HTTPS only. Otherwise, your users have no way of knowing if they access the correct service and will be vulnerable to man-in-the-middle attacks.

5.4. DNS SRV resource records

```
_imap._tcp.example.com          SRV  10  20  143  mail.example.com.
_imaps._tcp.example.com         SRV  0   1   993  .
_pop3._tcp.example.com          SRV  0   1   110  .
_pop3s._tcp.example.com         SRV  0   1   995  .
_smtp._tcp.example.com.         SRV  0   1   25   .
_submission._tcp.example.com.   SRV  10  20  587  mail.example.com.
```

6. Installing automua

6.1. Installing via Arch Linux AUR

automua has been released on Arch Linux AUR [https://aur.archlinux.org/packages/automua] to simplify the installation and the setup. You would require the Arch Linux distribution for that end. The AUR helper yay [https://github.com/Jguer/yay] is recommended.

```
yay -S automua
```

6.2. Installing from command line

automua requires Python version 3.7 or greater, ideally in the form of a virtual Python environment, to run. Check the python3 version like this:

```
$ python3 --version
Python 3.9.11
```

Don't run as root

If you use a port number greater than 1024 (I suggest port 4243), the application does not require super user privileges when running. Doing so would pose a security risk and is therefore strongly discouraged. I recommend creating a fresh user account called `automua`.

Prepare the virtual environment for the automua web service, adjusting the installation path to your taste (automua itself does not care). The path `/srv/http/automua` will be used as an example throughout this documentation. The BASH shell commands below should work with any modern Linux distribution.

```
# Best practice: Create a fresh user account.
sudo useradd --home-dir /srv/http/automua --create-home automua

# Alternative: If the user account already exists.
# sudo bash -c 'mkdir -p /srv/http/automua && chown automua /srv/http/automua'
```

Next, make sure to either login as the user created above, or change to this user via the 'su' command. This is important to ensure the correct file permissions. Download the script that will download and setup your automua service:

```
cd /srv/http/automua
wget https://github.com/HLFH/automua/raw/master/contrib/setupvenv.sh
chmod u+x setupvenv.sh
```

Executing the setup script will create a Python virtual environment called `.venv` in the current directory.

```
./setupvenv.sh
```

Activate the virtual environment and install the latest automua release from PyPI. Make sure to pick the correct activation for your shell from the `.venv/bin` directory. This is an example for BASH:

```
source .venv/bin/activate
pip install automua
```

Updating to a newer automua release

Change to the directory where automua has been installed previously. Activate the virtual environment as usual and use pip's `--upgrade` option:

```
cd /srv/http/automua
source .venv/bin/activate
pip install --upgrade automua
```

7. Configuring automua

automua uses a file to read runtime instructions from and a database to lookup mail account configuration data.

7.1. Placeholders

To make configuration more convenient, automua supports Mozilla-style placeholders [https://wiki.mozilla.org/Thunderbird:Autoconfiguration:ConfigFileFormat#Placeholders]. For example, the string `%EMAILADDRESS%` in database records will be replaced with the email address specified during the query. While based on a proprietary feature of Autoconfig, automua also applies placeholders to Autodiscover and Mobileconfig responses.

7.2. Runtime configuration

The configuration file defines automua runtime behaviour and it specifies the backend automua should read mailbox account configuration data from.

Running without runtime config

If you launch automua without a configuration file, it will use internal defaults. These are suitable for testing only. Launched without a config it will use an in-memory SQLite database and all data will be lost once the application terminates.

During startup automua searches for runtime configuration instructions in the following locations. The first match will determine the configuration used.

```
env  : AUTOMUA_CONF  ①
file : ~/.automua.conf
file : /etc/automua/automua.conf
file : /etc/automua.conf
```

① If present, the environment variable AUTOMUA_CONF must point to the absolute path of a configuration file.

To specify parameters and options automua uses an INI file [https://docs.python.org/3.9/library/configparser.html#supported-ini-file-structure] syntax. The example configuration [https://github.com/HLFH/automua/blob/master/contrib/automua-sample.conf] that ships with automua looks like this:

```
[automua]
# A typical production setup would use loglevel WARNING.
loglevel = DEBUG
# Echo SQL commands into log? Used for debugging.
db_echo = no
# In-memory SQLite database
#db_uri = sqlite:///:memory:

# SQLite database in a UNIX-like file system
#db_uri = sqlite:////var/lib/automua/db.sqlite

# MySQL database on a remote server. This example does not use an encrypted
# connection and is therefore *not* recommended for production use.
db_uri = mysql://automua:REPLACEPASSWORD@localhost/automua

# Number of proxy servers between automua and the client (default: 0).
# If your logs only show 127.0.0.1 or ::1 as the source IP for incoming
# connections, proxy_count probably needs to be changed.
proxy_count = 1
```

Place the content of the example configuration into one of the configuration locations automua looks for and adapt it to your needs. Then configure the database backend with data that suits your setup, as described below.

7.3. Testing standalone automua

If you want to verify a vanilla installation of automua works, you can populate it with internal test data. Start automua as described in section Running automua and send the following request to populate your database:

```
curl http://127.0.0.1:4243/initdb/
```

This example assumes you are running automua on localhost listening on TCP port 4243, which is the suggested default port.

Once you have populated the database with sample data you can test if automua works. Use curl to send an account configuration request for user@example.com:

```
curl 'http://127.0.0.1:4243/mail/config-v1.1.xml?emailaddress=user@example.com'
```

As shown in the example, make sure to quote the URL as necessary. Otherwise, your command shell might perform pattern matching for characters like the question mark ? (FISH does).

7.4. Database configuration

automua uses the SQLAlchemy toolkit to access databases. This allows a variety of databases, a.k.a. dialects [https://docs.sqlalchemy.org/en/latest/dialects/], to be used, simply by defining the appropriate connection URL.

API based configuration

I consider adding an API for configuration changes in an upcoming version but have not decided when that might happen. Feel free to contact me if you are interested in a sponsorship.

7.4.1. Database support

While you probably already have SQLite support available on your local machine, you may need to install additional Python packages for PostgreSQL, MySQL, etc. Detailed instructions to support a particular database dialect are out of scope for this document. Please search the Internet for detailed instructions on supporting a particular dialect. The SQLAlchemy documentation provides a useful starting point.

While the `contrib` directory contains example database schemas which you can use as a reference, I recommend using the built-in method to create the necessary DB structure from scratch by sending a HTTP GET request to the `/initdb/` service endpoint. This will also populate the database with some hard-coded example data. Alternatively, you can send a POST request with custom JSON data to the same endpoint, as described below.

Purging the database

Sending a HTTP DELETE request to `/initdb/` will purge all existing data. Be sure to limit access accordingly!

```
curl -X DELETE http://127.0.0.1:4243/initdb/
```

If you upgrade from an early automua release and wish to migrate your existing database, you can use the built-in Alembic support. However, this requires cloning the Git repository, modifying `alembic.ini` and invoking the migration from the command line. It is usually easier to export your existing data, create a fresh DB and import the data.

7.4.2. SQLite

This section demonstrates what you need to do to in order to use SQLite version 3 or higher as a backend database for automua.

Step 1: Set the database URI in your automua configuration. Please note that specifying an absolute path for the database requires a total of four slashes after the schema identifier:

```
[automua]
db_uri = sqlite:////var/lib/automua/db.sqlite
```

Step 2: Launch automua and access the DB initialisation URL.

```
# Method 1: Populate DB with example data
curl -X GET http://127.0.0.1:4243/initdb/
# Method 2: Populate DB based on the content of a JSON file
curl -X POST --json @mydata.json http://127.0.0.1:4243/initdb/
```

Starting with automua version 2022.0, JSON data can be used to populate the database in a simplified manner, without the need to use SQL statements. The required data format is as follows:

```
{
  "provider": "Example Inc.",
  "domains": ["example.com", "example.net", "example.org"],
  "servers": [
    {"name": "imap.example.com", "type": "imap"},
    {"name": "smtp.example.com", "type": "smtp"}
  ]
}
```

Using JSON is recommended when you are content with automua choosing some details like port numbers or socket types for you.

If you prefer SQL statements for greater control of the database content, the Git repository contains a sqlite-generate.sh helper script which demonstrates how the database can be populated programmatically. You only need to adapt a few settings according to your needs:

```
PROVIDER_NAME='Example Inc.'
PROVIDER_SHORTNAME='Example'
PROVIDER_ID=100

DOM='example'
TLD='com'
```

The script will print the SQL statements to standard output, which can be piped into `sqlite3` for processing. Make sure to match the automua.conf `db_uri` setting when specifying the database.

```
contrib/sqlite-generate.sh | sqlite3 /var/lib/automua/db.sqlite
```

Once you have populated the database automua is ready to run.

7.4.3. MySQL

Step 1: Create a database.

```
CREATE DATABASE `automua` COLLATE 'utf8mb4_general_ci';
```

Step 2: Set the database URI in your automua configuration. The following example uses *pymysql* as a DB driver, which is not included in the automua distribution.

```
[automua]
db_uri = mysql+pymysql://user:pass@dbhost/automua?charset=utf8mb4
```

Step 3: Launch automua and access the DB initialisation URL:

```
curl http://127.0.0.1:4243/initdb/
```

7.4.4. PostgreSQL

Step 1: Create a database.

```
CREATE DATABASE automua LOCALE 'en_US.utf8';
```

Step 2: Set the database URI in your automua configuration. The following example uses *psycopg2* as a DB driver, which is not included in the automua distribution.

```
[automua]
db_uri = postgresql+psycopg2://user:pass@dbhost/automua
```

Step 3: Launch automua and access the DB initialisation URL:

```
curl http://127.0.0.1:4243/initdb/
```

7.5. Alembic

As mentioned in a previous section, you can use Alembic [https://alembic.sqlalchemy.org/] to create or upgrade your database. You need to start your first run using an empty database for this to work, because Alembic stores versioning information in said database. Database upgrades are based on

this information. Follow the steps shown below, setting the `RELEASE` variable to the GitHub tag or release number of your choice.

```
export RELEASE="2021.6"
wget https://github.com/HLFH/automua/archive/refs/tags/$RELEASE.zip
unzip $RELEASE.zip
cd automua-$RELEASE/alembic
```

Next, change the value for *sqlalchemy.url* in `alembic.ini` to match your automua configuration. Create an empty database unless you are using SQLite, in which case Alembic will create the database for you. The final steps are activating the automua virtual Python environment and invoke *make*.

```
source /path/to/automua/.venv/bin/activate
make upgrade
```

You should see output similar to the following:

```
PYTHONPATH=.. FLASK_APP=automua.server:create_app flask db upgrade -d .
Running automua version 2021.6
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> f62e64b43d2f, DB schema for
automua version 2020.0
Created: 2020-01-17 22:30:05.748651
INFO  [alembic.runtime.migration] Running upgrade f62e64b43d2f -> 5334f8a8282c, Add
"prio" column to "server" table.
Created: 2020-12-15 15:04:49.371802
INFO  [alembic.runtime.migration] Running upgrade 5334f8a8282c -> 43ebb40d0578, DAV
server support
```

8. LDAP support

automua supports looking up user account data using LDAP. This is typically used to find users' login IDs for IMAP/SMTP authentication given an associated email address. Note that this is an optional configuration element commonly used by larger organisations. For smaller user bases, using placeholders may be sufficient.

The following partial LDIF snippet shows how a mail account can be defined in a widely used LDAP schema:

```
dn: uid=jdoe,ou=mailusers,dc=example,dc=com
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: posixAccount
```

```
objectClass: top
cn: John Doe
givenName: John
homeDirectory: /var/maildata/jdoe
mail: johndoe@example.com
sn: Doe
uid: jdoe
uidNumber: 4321
[... more attributes here ...]
```

In order to allow automua to connect, an entry similar to the following needs to be created in the database:

```
INSERT INTO ldapserver (
    id, name, port, use_ssl,
    search_base, search_filter, attr_uid, attr_cn,
    bind_password, bind_user
) VALUES (
    100, 'ldap.example.com', 636, 1,
    'ou=mailusers,dc=example,dc=com', '(mail={0})', 'uid', 'cn',
    'PASSWORD', 'cn=automua,ou=services,dc=example,dc=com'
);
```

An encrypted connection (LDAPS) is used and the filter and attribute names are set according to the LDIF above. It is assumed that `cn=automua,ou=services,dc=example,dc=com` with the given password is permitted read-only access to the necessary LDAP records/attributes. The search filter needs to contain the placeholder `{0}` which will be replaced with the email address used as the lookup key.

Now all that is left is to connect the example.com domain to LDAP server ID 100:

```
UPDATE domain SET ldapserver_id=100 WHERE name='example.com';
```

9. Running automua

Running automua requires to start automua as service and serve its output via a web server to the public. You should not run automua with superuser privileges. Use a dedicated user instead.

The following examples assume you have created a user and group `automua` and have granted appropriate rights to this user:

- Read permissions for the `automua.conf` configuration file.
- Read and access permissions for the virtual Python environment.
- Read and access permissions for the SQLite database.

9.1. As a OpenRC service

The following is an example for a OpenRC run script `/etc/init.d/automua` which I use for Gentoo Linux:

```
#!/sbin/openrc-run
#
# /etc/init.d/automua

: ${AUTOMUA_CONF:="/etc/${RC_SVCNAME}.conf"}
: ${AUTOMUA_USER:="automua"}
: ${AUTOMUA_ARGS:="--port 4243"}

command="/usr/bin/python"
command_args="/usr/bin/flask run ${AUTOMUA_ARGS}"
command_background="true"
command_user="${AUTOMUA_USER}"
pidfile="/run/${RC_SVCNAME}.pid"
required_files="${AUTOMUA_CONF}"

depend() {
    use logger net
    before nginx
}

start_pre() {
    export AUTOMUA_CONF
    export EPYTHON="python3.9"
    export FLASK_APP="automua.server:create_app"
    export FLASK_ENV="production"
}
```

If you wish to override any of the settings, copy the following to `/etc/conf.d/automua` and uncomment/change variables according to your needs. This is purely optional.

```
# /etc/conf.d/automua

# Additional parameters passed to Flask
#AUTOMUA_ARGS="--host 127.0.0.1 --port 4243"

# Configuration file
#AUTOMUA_CONF="/etc/automua.conf"

# Process owner (choose a non-privileged user)
#AUTOMUA_USER="automua"
```

9.2. As a systemd service

If your system uses *systemd* you may want to deploy the following automua.service unit file from the contrib section and place it in `/etc/systemd/system/automua.service`:

```
[Unit]
After=network.target
Description=Auto MUA configuration service
Documentation=https://hlfh.github.io/automua/

[Service]
Environment=FLASK_APP=automua.server:create_app
Environment=FLASK_CONFIG=production
ExecStart=/srv/http/automua/.venv/bin/python /srv/http/automua/.venv/bin/flask run
--host=127.0.0.1 --port=4243
Restart=always
User=automua

[Install]
WantedBy=multi-user.target
```

Once you have installed the service you need to tell systemd to reload its list of available services:

```
sudo systemctl daemon-reload
```

It should now be able to tell you about a service named automua:

```
sudo systemctl status automua
● automua.service - MUA configuration service
     Loaded: loaded (/etc/systemd/system/automua.service; disabled; vendor preset:
enabled)
     Active: inactive (dead)
```

Next enable and start automua using the following command:

```
sudo systemctl enable automua --now
Created symlink /etc/systemd/system/multi-user.target.wants/automua.service →
/etc/systemd/system/automua.service.
```

You should see automua enabled and running:

```
sudo systemctl status automua
● automua.service - MUA configuration service
     Loaded: loaded (/etc/systemd/system/automua.service; enabled; vendor preset:
enabled)
```

```
     Active: active (running) since Mon 2021-03-01 12:54:31 CET; 19s ago
   Main PID: 126966 (python)
      Tasks: 1 (limit: 4620)
     Memory: 46.1M
     CGroup: /system.slice/automua.service
             └─126966 /srv/www/automua/bin/flask run --host=127.0.0.1 --port=4243
[...]
Mar 01 12:54:32 mail python[126966]: Reading /etc/automua.conf
Mar 01 12:54:32 mail python[126966]: Config.get: loglevel = WARNING
Mar 01 12:54:32 mail python[126966]:  * Running on http://127.0.0.1:4243/ (Press
CTRL+C to quit)
```

You are now ready to start testing automua, as described below.

9.3. Manually from a shell

While logged in as an unprivileged user, change into the installation directory and start the `.venv/scripts/flask.sh` launch script:

```
cd /srv/web/automua
.venv/scripts/flask.sh run --host=127.0.0.1 --port=4243
```

Handling terminal output

The launch script will deliberately keep automua running in the foreground, and log data will be displayed in the terminal. If you press Ctrl-C or close the shell session, the application will terminate. To run automua in the background, you can use a window manager like GNU Screen [https://www.gnu.org/software/screen/] or tmux [https://en.wikipedia.org/wiki/Tmux].

Now that automua is up and running, you need to configure the web server proxy that will receive requests from the outside and forwards them to automua.

10. Testing automua locally

You can use *curl* in a command shell to send a GET request to your local automua-instance. The following example assumes your service runs on localhost on port 4243. The exact output depends on your database content, but should look similar.

```
curl 'http://127.0.0.1:4243/mail/config-v1.1.xml?emailaddress=user@example.com'
```

```
<clientConfig version="1.1">
    <emailProvider id="automua-100">
        <identity/>
        <domain>example.com</domain>
```

```
        <displayName>Example Inc.</displayName>
        <displayShortName>Example</displayShortName>
        <incomingServer type="imap">
            <hostname>mail.example.com</hostname>
            <port>993</port>
            <socketType>SSL</socketType>
            <username>%EMAILADDRESS%</username>
            <authentication>plain</authentication>
        </incomingServer>
        <incomingServer type="pop3">
            <hostname>mail.example.com</hostname>
            <port>110</port>
            <socketType>STARTTLS</socketType>
            <username>%EMAILADDRESS%</username>
            <authentication>plain</authentication>
        </incomingServer>
        <outgoingServer type="smtp">
            <hostname>mail.example.com</hostname>
            <port>587</port>
            <socketType>STARTTLS</socketType>
            <username>%EMAILADDRESS%</username>
            <authentication>plain</authentication>
        </outgoingServer>
        <!-- ... -->
    </emailProvider>
</clientConfig>
```

Having verified that automua returns configuration data, you should make the service available using a web server as a proxy.

11. Configuring a web server

While it is technically possible to run automua without a web server in front of it, I do not recommend doing that in a production environment. A web server can provide features automua was designed not to have. Features such as transport layer encryption for HTTPS (required for Mobileconfig) or, for example, the capability to rate-limit clients are handled very well by full-fledged web servers working as reverse proxies. It would be a waste to re-implement all this in a web service.

This section will explain how to configure a web server as a reverse proxy in front of automua. Before you set up the proxy you need to tell automua it operates behind one. Add the `proxy_count` parameter to your automua configuration file or uncomment the parameter if it is already there:

```
[automua]
# A typical production setup would use loglevel = WARNING
loglevel = WARNING

# Disable SQL command echo.  ①
db_echo = no
```

```
# SQLite database in a UNIX-like file system
db_uri = sqlite:////var/lib/automua/db.sqlite

# Number of proxy servers between automua and the client (default: 0).
# If your logs only show 127.0.0.1 or ::1 as the source IP for incoming
# connections, proxy_count probably needs to be changed.  ②
proxy_count = 1
```

① Echoing SQL commands is only meant for debugging purposes.

② Set the number to reflect the number of proxies chained in front of automua, i.e. the number of "proxy hops" a client's request must pass before it reaches automua.

11.1. NGINX

The following example defines a HTTP server, which will listen for requests to both *autoconfig.example.com* and *autodiscover.example.com*. All requests will be forwarded to automua, which listens on TCP port 4243 in this example. Requests to `/initdb` are restricted to clients connecting from the local host. The `proxy_set_header` directives will cause NGINX to pass relevant data about incoming requests' origins.

```
# NGINX example configuration snippet to forward incoming requests to automua.
# vim:ts=4:sw=4:et:ft=nginx

http {
    server {
        listen *:80;
        listen [::]:80;
        server_name autoconfig.example.com autodiscover.example.com;
        location /initdb {
            # Limit access to clients connecting from localhost
            allow 127.0.0.1;
            deny all;
        }
        location / {
            # Forward all traffic to local automua service
            proxy_pass http://127.0.0.1:4243/;
            proxy_set_header Host $host;
            # Set config parameter proxy_count=1 to have automua process these headers
            proxy_set_header X-Forwarded-Proto http;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

11.2. Apache

The following example shows an Apache configuration similar to the one above. `ProxyPreserveHost` directives will cause apache to pass relevant data about incoming requests' origins.

```
# Apache 2.4 example configuration snippet to forward incoming requests to automua.
# vim:ts=4:sw=4:et:ft=apache

<VirtualHost *:80>
    ServerName autoconfig.example.com
    ServerAlias autodiscover.example.com
    ProxyPreserveHost On
    ProxyPass "/" "http://127.0.0.1:4243/"
    ProxyPassReverse "/" "http://127.0.0.1:4243/"
    <Location /initdb>
        # Limit access to clients connecting from localhost
        Order Deny,Allow
        Deny from all
        Allow from 127.0.0.1
    </Location>
</VirtualHost>
```

12. Sponsorship

If you are interested in sponsoring a specific feature, please contact me using the email address <*contact AT hlfh DOT space*>.