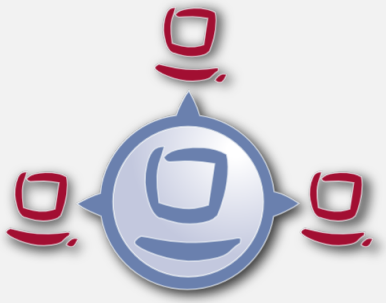

opsi Directory Connector

uib gmbh
Bonifaziusplatz 1b
55118 Mainz
Tel.:+49 6131 275610
www.uib.de
info@uib.de

Stand: June 8, 2018

Contents

Chapter 1

Copyright

Chapter 2

opsi directory connector

Introduction

The opsi directory connector is a tool to transfer data from a directory service to an opsi installation. This avoids the need of maintaining data in two separate systems.

Prerequisites for the opsi extension *opsi directory connector*

This module is currently a co-funded opsi extension.
Some preconditions are required, in order to use this module. That means that you need a suitable modules file to unlock this extension. You can get this file by purchasing the extension module. For evaluation purposes you can get a temporary modules file without charge. (→ mail us at info@uib.de).

General Requirements

The source directory service must implement the LDAP protocol. Samba 4 or Active Directory are examples for directory services that support the LDAP protocol.

The target opsi server should run at least opsi 4.0.7. Prior versions may work but have not been tested.

The machine running the connector must have access to the ones running the directory and opsi over the network.

It is possible to have everything running on the same server but this manual will assume that different machines are used.

Hardware-Requirements

These requirements are intended for a basic use in a small-sized environment with up to 500 clients. The requirements may change running the connector in larger environments so please be aware that adjustments may be necessary.

- 256 MB of free RAM
- Network connection

Software-Requirements

The installation and operation is only supported on Linux. Support for Windows is not planned.

The connector uses Python 3 which has to be available in at least version 3.2.

The connector uses standardized protocols and therefore does not require any further opsi- or directory-components to be installed.

Installation

To install the connector please add the opsi repository as described in the *Getting Started* document.

Then use your operating system's package manager to install the package `opsi-directory-connector`.

On an Debian-based machine installation can be done like this:

```
apt-get install opsi-directory-connector
```

Note
CentOS and RedHat version 6 and 7 do not have Python 3 as part of their core repositories. Therefore we currently can not support installation on these operating systems.

Configuration

The connector features a variety of settings to fit into different environments.

The configuration is made in a JSON-formatted file and must contain valid JSON! To specify any boolean value please use `true` or `false`. Text has to be entered in double quotes like `"this is text"`.

An example configuration will be provided at `/etc/opsi/opsidirectoryconnector.example.conf`. This file can be used as a template for your own configurations.

```
cp /etc/opsi/opsidirectoryconnector.example.conf /etc/opsi/opsidirectoryconnector-custom.conf
```

Directory settings

These are the settings required to access the directory and limit the search scopes to specific areas and entries.

```
{
    "directory": {
        "address": "ldap://192.168.12.34",
        "user": "DOMAIN\\opsiconnector",
        "password": "insertpasswordhere",
        "passwordFile": "",
        "search_base": "dc=testcompy,dc=local",
        "search_query_computers": "(objectClass=computer) ",
        "search_query_groups": "(objectClass=organizationalUnit)",
        "connection_options": {
            "start_tls": true,
            "paged_search_limit": 768
        }
    },
    ...
}
```

Under `address` you specify the directory-server. `user` and `password` are required for authentication at the directory. If `passwordFile` is set the value will be interpreted as path to a file that contains the password. The content of the file will be used as password. This allows for not having the password written in plaintext in the file. This will override the value set for `password` if the file can be read.

Tip
We recommend using a dedicated user account.

Note
Depending on the used directory software and it's configuration the format for the username can be different.
Besides *Down-Level Logon Name* in the format of `DOMAIN\\username` it is possible that the format uses *User Principal Name* with the format of `user@domain` or a *Distinguished Name* (DN) like `uid=opsiconnect,cn=users,dc=test,dc=intranet`.

With `search_base` you specify the location from where on the connector looks for matching entries. Through `search_query_computers` and `search_query_groups` the conditions for finding entries can be configured. Either `search_query_computers` or `search_query_groups` or both need to be configured. To disable one of these set them to `""`. The search for groups will be implemented in a future version. Until then the setting has no effect.

The parameter `connection_options` contains additional options to configure the connection. With `start_tls` it is possible the control if a secure connection should be used.

Note
Additional connection options will be implemented on demand.

If the optional parameter `paged_search_limit` is present and it's value is an integer multiple queries are made to the directory in order to read it's elements. How many elements a response contains is limited through the given value. This is supported since version 20.

Through the optional parameter `identifying_attribute` it is possible to set the attribute used for the unique identification of an client. This is possible since version 23. The default is the usage of `dn`.

Since version 14 it is possible to test the connection to the directory through the paramter `--check-directory` without connecting to the opsi server.

Configure connnection for UCS

For a connection to Univention Corporate Server a full *Distinguished Name* hast do be used as username. This has the form `uid=<username>,cn=users,dc=company,dc=mydomain`.

On UCS LDAP is reachable through ports 7389 (unsecured) resp. 7636 (secured via SSL). If Samba is installed on the Server and used as AD-compatible domain controller then it is listening on ports 389 (unsecured) resp. 636 (secured via SSL). To make use of the secured ports set the connection option `start_tls` to `true`.

The different connections also change the DN used for authentication. LDAP uses `uid=...` where are Samba works with `dn=...`.

Usually clients are found in the container `computers`. The following command shows a matching value for `search_base`:

```
echo "cn=computers,$(ucr get ldap/base)"
```

To search for Windows clients you can set `search_query_computers` to `(objectClass=univentionWindows)`.

How you can create a user with read only access is described in the Univetion wiki: Cool Solution - LDAP search user

Behaviour settings

These settings defines the behaviour of the connector.

```
{
    ...
    "behaviour": {
        "write_changes_to_opsi": true,
        "root_dir_in_opsi": "clientdirectory",
        "update_existing_clients": true,
        "prefer_location_from_directory": true
    },
    ...
}
```

If `write_changes_to_opsi` is set to `false` no data will be written to opsi. This can be used to check settings before applying them.

Via `root_dir_in_opsi` you define what group should be used as the root in opsi. You need to make sure that this group exists.

Note
The group *clientdirectory* is shown as *DIRECTORY* in configed. If clients or groups are to appear directly below *DIRECTORY* the value for `root_dir_in_opsi` has to be `clientdirectory`.

If `update_existing_clients` is set to `false` clients already existing in opsi will not be altered. If this is set to `true` clients may have any manually set data overridden with the values from the directory.

If `prefer_location_from_directory` is set to `true` clients will be moved in opsi to the same location they have in the directory. If you want to disable this set it to `false`.

Attribute-Mappings

With a system as flexible as a directory service the connector must be given information about what attributes in the directory match these of the corresponding opsi objects.

```
{
    ...
    "mapping": {
        "client": {
            "id": "name",
            "description": "description",
            "notes": "",
            "hardwareAddress": "",
            "ipAddress": "",
            "inventoryNumber": "",
            "oneTimePassword": ""
        },
        "group": {
            "id": "name",
            "description": "description",
            "notes": ""
        }
    },
    ...
}
```

There is a mapping for clients and one for groups.

The key of each mapping is the attribute in opsi and the value is the attribute from the directory. If the value (in the mapping) is empty no mapping will be done.

Note
If the value read from the directory for the client ID does not seem to be an FQDN an FQDN will be created. The domain part for this will be created from the DC of the read element.

Tip
On UCS the value for `hardwareAddress` can be set to `macAddress` if the connection is made through LDAP (ports 7389 or 7636).

Manual assignment of group names

Group names are usually used without any major adjustments. But this may lead to cases where names should be used that are invalid in opsi.

For this special cases a manual assignment of group names can be helpful.

To configure this an entry `group_name` has to be created in `mapping`. This holds the mapping from the directory to opsi. Names that are not present in this mapping aren't changed. The group names are always processed in lowercase. This can be configured since version 23.

The following example handles the group `_server` originating from the directory as `server` in opsi.

```
{
    ...
    "mapping": {
        ...
        "group": {
        ...
        },
    "group_name" {
        "_server": "server"
    }
    },
    ...
}
```

Warning
Please be careful with this feature as it may introduce undesired side effects. It should only be used for special cases!

opsi connection settings

This specifies how the connector accesses opsi.

```
{
    ...
    "opsi": {
        "address": "https://localhost:4447",
        "username": "syncuser",
        "password": "secret",
        "exit_on_error": false
        "passwordFile": "",
        "connection_options": {
            "verify_certificate": true
        }
```

```
    }
}
```

Set `address` to the address of your opsi server. Please include the port.

Note
To use a proxy for the connection use the environment variable *HTTPS_PROXY*.

`username` and `password` should be set accordingly to authenticate at the opsi server. If `passwordFile` is set the value will be interpreted as path to a file that contains the password. The content of the file will be used as password. This allows for not having the password written in plaintext in the file. This will override the value set for `password` if the file can be read.

Tip
We recommended setting up a dedicated user for this task. Refer to the document *Getting Started* on how to do this.

If the parameter `exit_on_error` is `true` then any problem that appears when updating data in opsi opsi - this could be triggered by submitting values that are invalid in opsi - results in a break. If this is `false` then problems will be logged but the run will not be stopped.

With `connection_options` the options for connecting to opsi can be set. `verify_certificate` configures the verification of the server certificate. For selfsigned certificates this can be set to `false`.

Since version 14 it is possible to test the connection to the opsi server through the paramter `--check-opsi` without connecting to the directory.

Running the connector

After installation a binary called `opsidirectoryconnector` will be present on the system.

It is required to pass an argument `--config` together with the path to the configuration.

```
opsidirectoryconnector --config /etc/opsi/opsidirectoryconnector-custom.conf
```

Note
The user running the binary does not require any access to opsi as this is all specified in the configuration.

Example: recurring runs with systemd

The connector currently does one synchronisation run when executed but the chances are good that you want to have a constant synchronisation of data.

It is easy to automate the execution of the connector to have recurring runs.

We will use systemd for this. In contrast to cronjobs systemd will avoid overlapping runs and is therefore a good choice.

The following example will set up the connector so that it is run five minutes after the machine was booted and from then on every hour.

In the directory `/etc/systemd/system/`, this is the directory for user-defined units, you need to place the two following files. One for the timer that makes the job recurring and one for the job itself.

Please put this inside `opsi-directory-connector.timer`:

```
[Unit]
Description=Start the opsi-directory-connector in regular intervals

[Timer]
OnBootSec=5min
OnUnitActiveSec=1hour

[Install]
WantedBy=timers.target
```

And this is the content of `opsi-directory-connector.service`:

```
[Unit]
Description=Sync clients from AD to opsi.
Wants=network.target

[Service]
Type=oneshot
ExecStart=/usr/bin/opsidirectoryconnector --config /etc/opsi/opsidirectoryconnector-custom.conf
```

To enable the timer and start it right away use the following commands:

```
systemctl enable opsi-directory-connector.timer
systemctl start opsi-directory-connector.timer
```

If the timer does not get started it will be first run after the next reboot of the machine.

Example: recurring runs as cronjob

It is easy to automate recurring runs through a cronjob.

Please be aware that overlapping runs may happen with cron and therefore the interval should be higher. To avoid this problem it is recommended to use **systemd** instead of **cron**

The cronjob file can usually be edited through `crontab -e`. For an synchronisation that happens every hour there can be used the following:

```
0 * * * * /usr/bin/opsidirectoryconnector --config /etc/opsi/opsidirectoryconnector-custom.conf
```