Licence

This software is licenced under the GNU Lesser General Public License v3.0.

Zabbix API PHP Client with session caching and SSL support

There are quite a lot of Zabbix API clients out that work really well with Zabbix. However most of them do not support `session caching` or need tweaks to work with self signed certificates.

This library aims to solve those problems. It supports the following features:

- Session caching.
- HTTPS connections with official- and self-signed certificates.
- Works with Linux and Windows PHP implementations.
- Multiple concurrent connections with different user accounts and/or different servers.
- Supports Zabbix 3.0, Zabbix 3.2, Zabbix 3.4 and Zabbix 4.0.
- No installation required.

It is commercially backed up and maintained by IntelliTrend GmbH, an official Zabbix Partner and Zabbix Training company.

Why Session caching?

How authentication via API usally works

Each time an application uses `user.login`, the Zabbix API creates a so called `AuthKey`. (See user.login). This key is then passed via each request through the `auth` request property.

In most cases, the API libary does this transparently for the user. However, if the script is called later on again (i.e by a cron job), a new `user.login` is performed, thus creating a new `AuthKey` for that new session.

Zabbix keeps those session, until a session expired or is logged out. You can check your actual settings through the Zabbix frontend: `Administration/General/Housekeeper - Section Sessions`. The default value is 365days.

This means, any created session (if there is no logout) will be kept for 365 days.

Assume we have a set of 10 API scripts that run every hour. This means we will create `10 x 24 = 240` sessions per day. Using more scripts or a smaller interval will of cause increase this number.

The problem with many sessions in the session table

Everytime a request hits the Zabbix frontend, either per webbrowser or as a JSON RPC API request, Zabbix has to verify the request against the existing sessions. The more sessions to verify, the longer this will take.

Note: We have seen installations with millions of sessions, where frontend access slowed down considerable by 15sec+ per request. Keep in mind that for example the dashboard not only performs one request, but multiple requests depending on the number of widgets used.

So the best is to `reuse` a session, until it expires. This is where session caching steps in.

Using session caching with the Zabbix API

When a new session is created, the AuthKey is saved encrypted to disk. This is similar to using a cookie in a classic webbrowser. If a new request is performed, the existing session is read once and the AuthKey is reused.

So to follow up the example given before: Calling a script using session-caching, even over a month, will create just `1` session.

If the AuthKey becomes invalid, the libary automatically performs a new `user.login`, re-executes the failed request and updates the stored session. All of this happens in the background, the user of the libary has not to deal with it.

How does session encryption work, what about multiple sessions?

Each session has a unique name based on a hash using the `zabbixUserName` and the `zabbixUrl`. The session itself is encrypted using the `zabbixUserName` and the `zabbixPassword`.

This way, the libary can be used with different useraccounts and also different zabbix server instances.

Where are sessions stored?

Sessions are stored by default in the users `tmp` directory. However there is a config option `sessionDir` that allows to override this setting. See the detailed description below.

Installation

The is no installation required. Simply copy the file `Zabbixapi.php` and use the class.

```
require_once "Zabbixapi.php";
$zbx = new Zabbixapi();
```

Usage

Error handling

The library makes use of `Exceptions`. There is no need to check each response value. Simply wrap the calls in `try/catch blocks`.

Depending on where an error occurs, different error codes are passed to the exception object, together with the message property.

- `Zabbix API errors`: The original API error code and message is passed.

- `Connection and SSL errors`: The original CURL error code and message is passed.
- `Library errors`: A constant error code, as defined in the class constant EXCEPTION_CLASS_CODE, and a useful message is passed. Default=1000.

Configuration

The class is configured when calling the `login` method. Any further Zabbix API call is performed by the `call` method.

Note: One can run multiple instances at the same time, connecting with different user accounts to the same zabbix server or to another zabbix server.

Basic usage

Lets start with a very simple example:

```
require_once "Zabbixapi.php";
$zbx = new Zabbixapi();
try {
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword');
    $result = $zbx->call('host.get', array("countOutput" => true));
    print "Number of hosts:$result\n";
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Basically this is all needed. The `call` method is transparent to the Zabbix API definition. It takes 2 parameter: `$method` and `$params` as specified for the particular Zabbix API method.

For example to retrieve all 'Host Groups' with all properties, we can do this:

```
require_once "Zabbixapi.php";
$zbx = new Zabbixapi();
$zabUrl = 'https://my.zabbixurl.com/zabbix';
$zabUser = 'myusername';
$zabPassword = 'mypassword';
try {
    $reUsedSession = $zbx->login($zabUrl, $zabUser, $zabPassword);
    if ($reUsedSession) {
        print "Existing Session reused\n";
    }
    $result = $zbx->call('hostgroup.get', array("output" => 'extend'));
    foreach ($result as $hostGroup) {
        $hostGroupId = $hostGroup['groupid'];
        $hostGroupName = $hostGroup['name'];
        print "groupid:$hostGroupId, hostGroupName:$hostGroupName\n";
    }
} catch (Exception $e) {
```

```
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Note: This second example would not create a new session when calling `login` again after the first example. It would reuse the session from the previous example. `login` returns true when an existing session was found.

Advanced usage and SSL Options

The basic example works fine, even with HTTPS, given there is a valid certificate the php installation is aware of. But what todo when using self-signed certificates?

Here we can use the optional `options` argument when calling `login` to setup the SSL options.

Example - Turn off SSL verification:

```
require_once "Zabbixapi.php";
$zbx = new Zabbixapi();
$options = array('sslVerifyPeer' => false, 'sslVerifyHost' => false);
try {
    $zbx->login('https://my.zabbixurl.com/zabbix', 'myusername', 'mypassword', $options);
    $result = $zbx->call('host.get', array("countOutput" => true));
    print "Number of hosts:$result\n";
} catch (Exception $e) {
    print "==== Exception ===\n";
    print 'Errorcode: '.$e->getCode()."\n";
    print 'ErrorMessage: '.$e->getMessage()."\n";
    exit;
}
```

Functions reference

Basic functions

login($zabUrl, $zabUser, $zabPassword, $options)

Initial login. Configures the class, loads a cached session if it exists and executes a request to the remote server to test the credentials or session.

- `return` boolean $reusedSession. True if an existing session was reused.
- `throws` Exception $e. Invalid options, session issues or connection problems.
- `param` string $zabUrl
- `param` string $zabUser
- `param` string $zabPassword

- `param` array $options - optional settings. Example: `array('sslVerifyPeer' => false, 'sslVerifyHost' => false);`
    - `debug`: boolean - default=false. Show debug information. Also setDebug() can be used. Default is false
    - `sessionDir`: string - default=user `tmp` directory. Directory where to store the sessions.
    - `sslCaFile`: string - default=use php.ini settings. Filename of external CACertBundle. Useful when using self signed certificates from internal CA. See the CURL or Mozilla websites for those bundles.
    - `sslVerifyPeer`: boolean - default=true. Verify certificate. Throws Exception on failure. When false, ignore any verfication errors.
    - `sslVerifyHost`: boolean - default=true. Verify Hostname against CN in certificate. Only works if certificate can be validated. Note: If sslVerifyPeer=false but the certificate itself is valid and the hostname does not match, then sslVerifyHost=true will raise an exception.
    - `useGzip`: boolean - default=true. Use gzip compression for requests.
    - `connectTimeout`: integer - default=10. Max. time in seconds to connect to server.
    - `timeout`: default=30. Max. time in seconds to process request.

call($method, $params)

Execute Zabbix API call. Will automatically re-login and retry if the call failed using the current authKey read from session.

Note: Can only be called after login() was called once before at any time.

- `return` mixed $reusedSession. Decoded Json response from API call or a scalar. See Zabbix API documentation for details.
- `throws` Exception $e. API Error, Session issues or connection problems.
- `param` string $method. Zabbix API method i.e. 'host.get'
- `param` mixed $params. Params as defined in the Zabbix API for that particular method.

logout()

Logout from Zabbix Server and also delete the authKey from filesystem.

Only use this method if its really needed, because you cannot reuse the session later on.

- `return` void
- `throws` Exception $e. API Error, Session issues or connection problems

setDebug($state)

Enable / Disable debug output. Can be used any time.

- `return` void
- `param` boolean $state. True enables debug output.

Convenient functions

getVersion()

Get version of this library.

- `return` string $version.

getApiVersion()

Get Zabbix API version from Zabbix Server.

- `return` string $version. Uses API method 'apiinfo.version'.
- `throws` Exception $e. API Error, Session issues or connection problems

Utility functions

getAuthKey()

Get authKey used for API communication

- `return` string $authKey.

getSessionDir()

Get session directory.

- `return` string $directory.

getSessionFileName()

Get session FileName storing the encrypted authKey without path.

- `return` string $fileName.

getSessionFile()

Get full FileName with path.

- `return` string $fileName.