Authenticating scripted clients

- Groovy script using cdancy/jenkins-rest
- Groovy script using commons-httpclient
- Perl LWP example for a scripted client
- Java example with httpclient 4.1.2
- Java example with httpclient 4.3.x

To make scripted clients (such as wget) invoke operations that require authorization (such as scheduling a build), use HTTP BASIC authentication to specify the user name and the API token. This is often more convenient than emulating the form-based authentication.

API token is new since 1.426

Earlier versions of Jenkins require you to specify your real password, and it is only available when your security realm is password-based (for example, OpenID, Crowd and CAS plugins authenticate you without a password, so you simply don’t have any password!) Specifying the real password is still supported after 1.426, but it is not recommended because the risk of revealing password, and the human tendency to reuse the same password in different places.

The API token is available in your personal configuration page. Click your name on the top right corner on every page, then click “Configure” to see your API token. (The URL $root/me/configure is a good shortcut.) You can also change your API token from here.

Note that Jenkins does not do any authorization negotiation. i.e. it immediately returns a 403 (Forbidden) response instead of a 401 (Unauthorized) response, so make sure to send the authentication information from the first request (aka “preemptive authentication”).

Groovy script using cdancy/jenkins-rest

The cdancy/jenkins-rest client greatly simplifies REST API access. The following Groovy code shows how to authenticate to Jenkins and get some system info:

```
@Grab(group='com.cdancy', module='jenkins-rest', version='0.0.18')

import com.cdancy.jenkins.rest.JenkinsClient

JenkinsClient client = JenkinsClient.builder()
    .endPoint("http://127.0.0.1:8080") // Optional. Defaults to http://127.0.0.1:8080
    .credentials("user:apiToken") // Optional.
    .build()

println(client.api().systemApi().systemInfo())
```

For additional information, see the cdancy/jenkins-rest wiki.

Groovy script using commons-httpclient

In a groovy script, this could look something like this (using commons-httpclient):

```
API token is new since 1.426

Earlier versions of Jenkins require you to specify your real password, and it is only available when your security realm is password-based (for example, OpenID, Crowd and CAS plugins authenticate you without a password, so you simply don’t have any password!) Specifying the real password is still supported after 1.426, but it is not recommended because the risk of revealing password, and the human tendency to reuse the same password in different places.

The API token is available in your personal configuration page. Click your name on the top right corner on every page, then click “Configure” to see your API token. (The URL $root/me/configure is a good shortcut.) You can also change your API token from here.

Note that Jenkins does not do any authorization negotiation. i.e. it immediately returns a 403 (Forbidden) response instead of a 401 (Unauthorized) response, so make sure to send the authentication information from the first request (aka “preemptive authentication”).

Groovy script using cdancy/jenkins-rest

The cdancy/jenkins-rest client greatly simplifies REST API access. The following Groovy code shows how to authenticate to Jenkins and get some system info:

```
@Grab(group='com.cdancy', module='jenkins-rest', version='0.0.18')

import com.cdancy.jenkins.rest.JenkinsClient

JenkinsClient client = JenkinsClient.builder()
    .endPoint("http://127.0.0.1:8080") // Optional. Defaults to http://127.0.0.1:8080
    .credentials("user:apiToken") // Optional.
    .build()

println(client.api().systemApi().systemInfo())
```

For additional information, see the cdancy/jenkins-rest wiki.

Groovy script using commons-httpclient

In a groovy script, this could look something like this (using commons-httpclient):

```
API token is new since 1.426

Earlier versions of Jenkins require you to specify your real password, and it is only available when your security realm is password-based (for example, OpenID, Crowd and CAS plugins authenticate you without a password, so you simply don’t have any password!) Specifying the real password is still supported after 1.426, but it is not recommended because the risk of revealing password, and the human tendency to reuse the same password in different places.

The API token is available in your personal configuration page. Click your name on the top right corner on every page, then click “Configure” to see your API token. (The URL $root/me/configure is a good shortcut.) You can also change your API token from here.

Note that Jenkins does not do any authorization negotiation. i.e. it immediately returns a 403 (Forbidden) response instead of a 401 (Unauthorized) response, so make sure to send the authentication information from the first request (aka “preemptive authentication”).

Groovy script using cdancy/jenkins-rest

The cdancy/jenkins-rest client greatly simplifies REST API access. The following Groovy code shows how to authenticate to Jenkins and get some system info:

```
@Grab(group='com.cdancy', module='jenkins-rest', version='0.0.18')

import com.cdancy.jenkins.rest.JenkinsClient

JenkinsClient client = JenkinsClient.builder()
    .endPoint("http://127.0.0.1:8080") // Optional. Defaults to http://127.0.0.1:8080
    .credentials("user:apiToken") // Optional.
    .build()

println(client.api().systemApi().systemInfo())
```

For additional information, see the cdancy/jenkins-rest wiki.

Groovy script using commons-httpclient

In a groovy script, this could look something like this (using commons-httpclient):

```
API token is new since 1.426

Earlier versions of Jenkins require you to specify your real password, and it is only available when your security realm is password-based (for example, OpenID, Crowd and CAS plugins authenticate you without a password, so you simply don’t have any password!) Specifying the real password is still supported after 1.426, but it is not recommended because the risk of revealing password, and the human tendency to reuse the same password in different places.

The API token is available in your personal configuration page. Click your name on the top right corner on every page, then click “Configure” to see your API token. (The URL $root/me/configure is a good shortcut.) You can also change your API token from here.

Note that Jenkins does not do any authorization negotiation. i.e. it immediately returns a 403 (Forbidden) response instead of a 401 (Unauthorized) response, so make sure to send the authentication information from the first request (aka “preemptive authentication”).

Groovy script using cdancy/jenkins-rest

The cdancy/jenkins-rest client greatly simplifies REST API access. The following Groovy code shows how to authenticate to Jenkins and get some system info:

```
@Grab(group='com.cdancy', module='jenkins-rest', version='0.0.18')

import com.cdancy.jenkins.rest.JenkinsClient

JenkinsClient client = JenkinsClient.builder()
    .endPoint("http://127.0.0.1:8080") // Optional. Defaults to http://127.0.0.1:8080
    .credentials("user:apiToken") // Optional.
    .build()

println(client.api().systemApi().systemInfo())
```

For additional information, see the cdancy/jenkins-rest wiki.

Groovy script using commons-httpclient

In a groovy script, this could look something like this (using commons-httpclient):
import org.apache.commons.httpclient.*
import org.apache.commons.httpclient.auth.*
import org.apache.commons.httpclient.methods.*

@Grab(group='commons-httpclient', module='commons-httpclient', version='3.1')
void createNewJenkinsProject() {

def server = "server"
def jenkinsHost = "https://${server}/jenkins/"
def projectName = "TEST"
def configurationFile = "config.xml"

def username = "username"
def apiToken = "apiToken"

def client = new HttpClient()
client.state.setCredentials(  
    new AuthScope( server, 443, "realm"),  
    new UsernamePasswordCredentials( username, apiToken )  
)

// Jenkins does not do any authentication negotiation,  
// i.e. it does not return a 401 (Unauthorized)  
// but immediately a 403 (Forbidden)  
client.params.authenticationPreemptive = true

def post = new PostMethod( "${jenkinsHost}/createItem?name=${projectName}" )
post.doAuthentication = true
File input = new File(configurationFile);
RequestEntity entity = new FileRequestEntity(input, "text/xml; charset=UTF-8");
post.setRequestEntity(entity);
try {
    int result = client.executeMethod(post);
    println "Return code: ${result}";
    post.responseHeaders.each{ println it.toString().trim() }
    println post.getResponseBodyAsString()
} finally {
    post.releaseConnection()
}
}
createNewJenkinsProject()
Perl LWP example for a scripted client

The following Perl example uses the LWP module to start a Job via a "Trigger builds remotely" token:

```perl
# Use LWP to run a Jenkins job
# set authorization_basic on the request object
# to make use of BASIC HTTP authorization, apparently
# already handling the preemptive part correctly this
# way.
# use strict;
use warnings;
use LWP;

my $server = 'srvname';
my $srvurl = "http://$server/jenkins";
my $uagent = LWP::UserAgent->new;
my $req = HTTP::Request->new(
  GET => "$srvurl/job/test/build?token=theTokenConfiguredForThisJob&cause=LWP+Test"
);
$req->authorization_basic('username@mydomain.com', 'apiToken');
my $res = $uagent->request($req);

# Check the outcome of the response
print "Result: " . $res->status_line . "\n";
print $res->headers->as_string;
print "\n";
if (!$res->is_success) {
  print "Failed\n";
} else {
  print "Success!\n";
  # print $res->content, "\n";
}
```

Java example with httpclient 4.1.2

Note: If you are using wget 1.11 against Jenkins version 1.586 and above with the JENKINS-25169 fix, you might need to use the following options:

```bash
wget --auth-no-challenge --http-user=user --http-password=apiToken --secure-protocol=TLSv1
http://jenkins.yourcompany.com/job/your_job/build?token=TOKEN
```

Note: If you are using wget 1.11, you might need to use the following options:

```bash
wget --auth-no-challenge --http-user=user --http-password=apiToken
http://jenkins.yourcompany.com/job/your_job/build?token=TOKEN
```

With wget 1.10.x the following is enough (but will not work with 1.11.x):

```bash
wget http://user:apiToken@jenkins.yourcompany.com/job/your_job/build?token=TOKEN
```

See this RedHat bug report for more detailed explanations: https://bugzilla.redhat.com/show_bug.cgi?id=446949 (this also affect other distributions)
(Report indicates that wget 1.0 and 1.1 don't appear to have a --auth-no-challenge option.)
This will authenticate you on your jenkins and launch the defined build. Be careful on security issues since this sample is based on username/password authentication.

```java
import java.io.IOException;
import org.apache.http.HttpException;
import org.apache.http.HttpHost;
import org.apache.http.HttpRequest;
import org.apache.http.HttpRequestInterceptor;
import org.apache.http.auth.AuthScheme;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.AuthState;
import org.apache.http.auth.Credentials;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.auth.BasicScheme;
import org.apache.http.util.EntityUtils;

/**
 * Simple class to launch a jenkins build on run@Cloud platform, should also work on every jenkins instance (not tested)
 */

public class TestPreemptive {

    public static void main(String[] args) {

        // Credentials
        String username = "YOUR_USERNAME";
        String password = "YOUR_PASSWORD";

        // Jenkins url
        String jenkinsUrl = "JENKINS_URL";

        // Build name
        String jobName = "JOB";

        // Build token
        String buildToken = "BUILD_TOKEN";

        // Create your httpClient
        DefaultHttpClient client = new DefaultHttpClient();

        // Then provide the right credentials
        client.getCredentialsProvider().setCredentials(new AuthScope(AuthScope.ANY_HOST, AuthScope.
                ANY_PORT),
                new UsernamePasswordCredentials(username, password));

        // Generate BASIC scheme object and stick it to the execution context
        BasicScheme basicAuth = new BasicScheme();
        BasicHttpContext context = new BasicHttpContext();
        context.setAttribute("preemptive-auth", basicAuth);

        // Add as the first (because of the zero) request interceptor
        // It will first intercept the request and preemptively initialize the authentication scheme if there is not
        client.addRequestInterceptor(new PreemptiveAuth(), 0);

        // You get request that will start the build
        String getUrl = jenkinsUrl + "/job/" + jobName + "/build?token=" + buildToken;
        HttpGet get = new HttpGet(getUrl);
```
try {
    // Execute your request with the given context
    HttpResponse response = client.execute(get, context);
    HttpEntity entity = response.getEntity();
    EntityUtils.consume(entity);
}
catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

/**
 * Preemptive authentication interceptor
 *
 */
static class PreemptiveAuth implements HttpRequestInterceptor {

    /*
     * (non-Javadoc)
     *
     * @see org.apache.http.HttpRequestInterceptor#process(org.apache.http.HttpRequest,
     */
    public void process(HttpRequest request, HttpContext context) throws HttpException, IOException {
        AuthState authState = (AuthState) context.getAttribute(ClientContext.TARGET_AUTH_STATE);
        if (authState.getAuthScheme() == null) {
            AuthScheme authScheme = (AuthScheme) context.getAttribute("preemptive-auth");
            CredentialsProvider credsProvider = (CredentialsProvider) context
              .getAttribute(ClientContext.CREDS_PROVIDER);
            HttpHost targetHost = (HttpHost) context.getAttribute(ExecutionContext.HTTP_TARGET_HOST);
            if (authScheme != null) {
                if (authScheme[result-1] != null) {
                    Credentials creds = credsProvider.getCredentials(new AuthScope(targetHost.getHostName(), targetHost
                        .getPort()));
                    if (creds == null) {
                        throw new HttpException("No credentials for preemptive authentication");
                    }
                    authState.setAuthScheme(authScheme);
                    authState.setCredentials(creds);
                }
            }
        }
    }
}

Java example with httpclient 4.3.x

This will cause httpclient 4.3 to issue authentication preemptively:
import java.io.IOException;
import java.net.URI;
import org.apache.http.HttpHost;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.CredentialsProvider;
import org.apache.http.impl.auth.BasicScheme;
import org.apache.http.impl.client.BasicCredentialsProvider;
import org.apache.http.util.EntityUtils;

public class JenkinsScraper {

    public String scrape(String urlString, String username, String password) throws ClientProtocolException, IOException {
        URI uri = URI.create(urlString);
        HttpHost host = new HttpHost(uri.getHost(), uri.getPort(), uri.getScheme());
        CredentialsProvider credsProvider = new BasicCredentialsProvider();
        credsProvider.setCredentials(new AuthScope(uri.getHost(), uri.getPort()), new UsernamePasswordCredentials(username, password));

        // Create AuthCache instance
        AuthCache authCache = new BasicAuthCache();
        // Generate BASIC scheme object and add it to the local auth cache
        BasicScheme basicAuth = new BasicScheme();
        authCache.put(host, basicAuth);

        CloseableHttpClient httpClient = HttpClients.custom().setDefaultCredentialsProvider(credsProvider).build();
        HttpGet httpGet = new HttpGet(uri);
        HttpClientContext localContext = HttpClientContext.create();
        localContext.setAuthCache(authCache);
        HttpResponse response = httpClient.execute(host, httpGet, localContext);
        return EntityUtils.toString(response.getEntity());
    }
}