Remote access API

Jenkins provides machine-consumable remote access API to its functionalities. Currently it comes in three flavors:

1. XML
2. JSON with JSONP support
3. Python

Remote access API is offered in a REST-like style. That is, there is no single entry point for all features, and instead they are available under the "`.../api/`" URL where "`...`" portion is the data that it acts on.

For example, if your Jenkins installation sits at http://ci.jruby.org/, visiting http://ci.jruby.org/api/ will show just the top-level API features available – primarily a listing of the configured jobs for this Jenkins instance.
Or if you want to access information about a particular build, e.g. http://ci.jruby.org/job/jruby-base/lastSuccessfulBuild/, then go to http://ci.jruby.org/job/jruby-base/lastSuccessfulBuild/api/ and you'll see the list of functionalities for that build.

The work on this front is ongoing, so if you find missing features, please file an issue.

What can you do with it?

Remote API can be used to do things like these:

1. retrieve information from Jenkins for programmatic consumption.
2. trigger a new build
3. create/copy jobs

Submitting jobs

Jobs without parameters

You merely need to perform an HTTP POST on `JENKINS_URL/job/JOBNAME/build`.

Jobs with parameters

Also see Parameterized Build.

Simple example - sending "String Parameters":

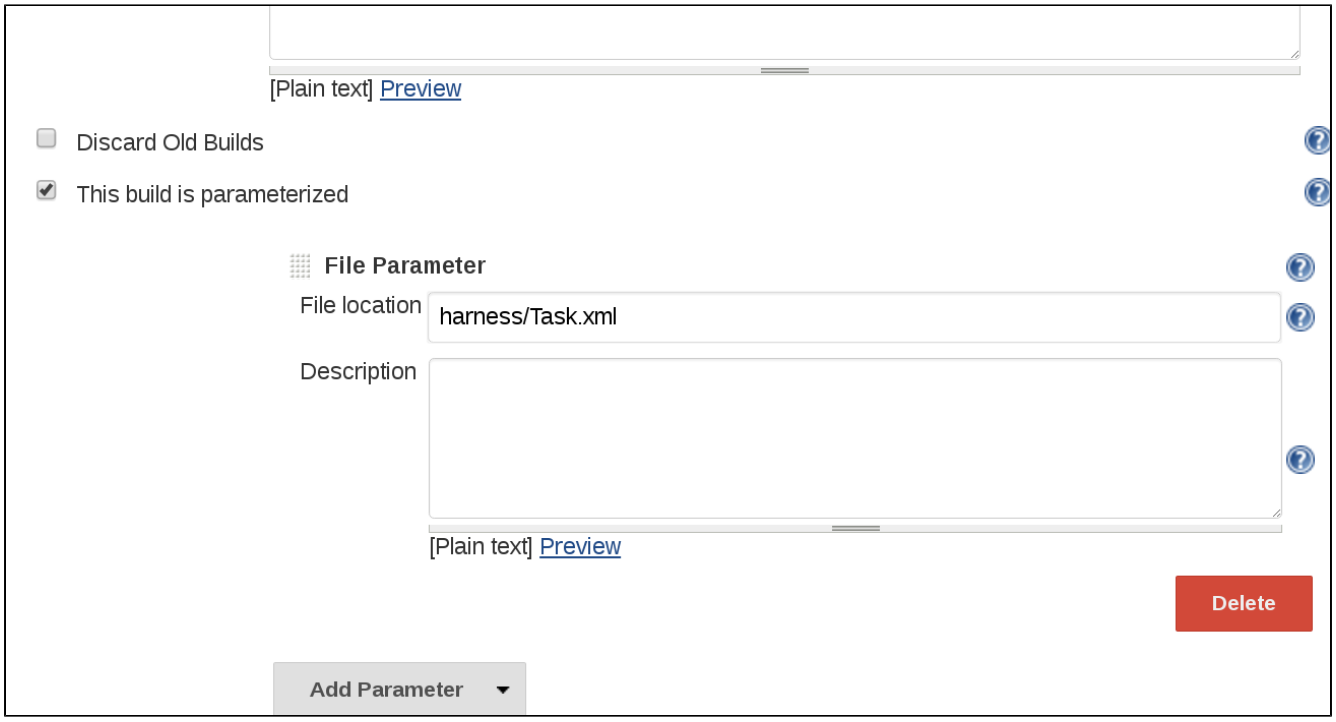
```
curl -X POST JENKINS_URL/job/JOB_NAME/build \
  --user USER:TOKEN \
  --data-urlencode json='{"parameter": [{"name":"id", "value":"123"},
{"name":"verbosity", "value":"high"}]}'
```

Another example - sending a "File Parameter":

```
curl -X POST JENKINS_URL/job/JOB_NAME/build \
  --user USER:PASSWORD \
  --form file0=@PATH_TO_FILE \
  --form json='{"parameter": [{"name":"
FILE_LOCATION_AS_SET_IN_JENKINS", "file":"file0"}]}'

E.g.curl -X POST http://JENKINS_URL/job/JOB_NAME/build  --form file0=@
/home/user/Desktop/sample.xml --form json='{"parameter": [{"name":"
harness/Task.xml", "file":"file0"}]}'
Please note, in this example, the symbol '@' is important to mention.
Also, the path to the file is absolute path.
In order to make this command work, you need to configure your Jenkins
job to take a file parameter and 'name' in this command corresponds to
'file location' field in the Jenkins job configuration.

In above example, 'harness' is just a name of folder that may not exist
in your workspace, you can write "name":"Task.xml" and it will place
the Task.xml at root of your workspace.
Remember that name in this command should match with File location in
file parameter in job configuration.
```



Remote API and security

When your Jenkins is secured, you can use HTTP BASIC authentication to authenticate remote API requests. See Authenticating scripted clients for more details.

CSRF Protection

If your Jenkins uses the "Prevent Cross Site Request Forgery exploits" security option (which it should), when you make a `POST` request, you have to send a CSRF protection token as an HTTP request header.
For curl/wget you can obtain the header needed in the request from the URL `JENKINS_URL/crumbIssuer/api/xml` (or `.../api/json`). Something like this:

```
wget -q --auth-no-challenge --user USERNAME --password PASSWORD --
output-document - \
'JENKINS_URL/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",
//crumb)'
```

This will print something like ".crumb:1234abcd", which you should add to the subsequent request.

Sample code

A simple client is available to demonstrate how you can invoke the XML from Java (Java source)

XPath selection

The XML API supports a selection by XPath by using the query parameter 'xpath'. This is convenient for extracting information in environments where XML manipulation is tedious (such as shell script.) See issue #626 for an example of how to use this.
See `.../api/` on your Jenkins server for more up-to-date details.

XPath exclusion

Similar to the 'xpath' query parameter above, you can use (possibly multiple) 'exclude' query patterns to exclude nodes from the resulting XML. All the nodes that match the specified XPath will be removed from the XML.
See .../api/ on your Jenkins server for more up-to-date details.

Depth control

Sometimes the remote API doesn't give you enough information in one call. For example, if you'd like to find out all the last successful build of a given view, you'd realize that the invocation to the remote API of the view won't give you this, and you'd have to recursively call the remote API of each project to find this out. The depth control, introduced in 1.167, solves this problem. To understand this feature, it's good to start with how the remote API works.

The data model that Jenkins maintains internally can be thought of as a big tree structure, and when you make a remote API call, you are getting a small subtree of it. The subtree is rooted at the object for which you made a remote API call, and the sub-tree is cut beyond certain depth to avoid returning too much data. You can adjust this cut-off behavior by specifying the depth query parameter. When you specify a positive depth value, the subtree cut-off happens that much later.

So the net result is, if you specify a bigger depth value, you'll see that the remote API will now return more data. Because of the algorithm, this works in such a way that the data returned by a bigger depth value includes all the data returned by smaller depth value.

See `.../api/` on your Jenkins server for more up-to-date details.

Python API wrappers

JenkinsAPI and Python-Jenkins are object-oriented python wrappers for the Python REST API which aims to provide a more conventionally pythonic way of controlling a Jenkins server. It provides a higher-level API containing a number of convenience functions. Services offered currently include:

- Query the test-results of a completed build
- Get objects representing the latest builds of a job
- Search for artifacts by simple criteria
- Block until jobs are complete
- Install artifacts to custom-specified directory structures
- username/password auth support for jenkins instances with auth turned on
- Ability to search for builds by subversion revision
- Ability to add/remove/query jenkins slaves

Ruby API wrappers

Jenkins API Client is an object oriented ruby wrapper project that consumes Jenkins's JSON API and aims at providing access to all remote API Jenkins provides. It is available as a Rubygem and can be useful to interact with the Job, Node, View, BuildQueue, and System related functionalities. Services currently offered include:

- Creating jobs by sending xml file or by specifying params as options with more customization options including source control, notifications, etc.
- Building jobs (with params), stopping builds, querying details of recent builds, obtaining build params, etc.

- Listing jobs available in Jenkins with job name filter, job status filter.
- Adding/removing downstream projects.
- Chaining jobs i.e given a list of projects each project is added as a downstream project to the previous one.
- Obtaining progressive console output.
- Username/password based authentication.
- Command Line Interface with a lot of options provided in the libraries.
- Creating, listing views.
- Adding jobs to views and removing jobs from views.
- Adding/removing jenkins slaves, querying details of slaves.
- Obtaining the tasks in build queue, and their age, cause, reason, ETA, ID, params and much more.
- Quiet down, cancel quiet down, safe restart, force restart, and wait till Jenkins becomes available after a restart.
- Ability to list installed/available plugins, obtain information about plugins, install/uninstall plugins and much more with plugins.

This project is in rapid development and new features are getting added every day. Watch the progress here.

Java API wrappers

The jenkins-rest library is an object oriented java project that provides access to the Jenkins REST API programmatically to some remote API Jenkins provides. It is build using the amazing jclouds toolkit and can easily be extended to support more REST endpoints. Its feature set evolves and users are invited to contribute new endpoints via pull-requests. In its current state it is possible with this library to submit a job, track its progress through the queue, and during its execution until its completion, and obtain the build status. Services currently offered include:

- Endpoint definition (property or environment variable)
- Authentication (basic and API token via property or environment variable)
- Crumbs Issuer support (auto-detect crumbs)
- Folder support
- Jobs API (build, buildInfo, buildWithParameters, config, create, delete, description, disable, enable, jobInfo, lastBuildNumber, lastBuidTimestamp and progressiveText)
- Plugin manager API (installNecessaryPlugins, list current plugins)
- Queue API (cancel, list queue items, query queue item)
- Statistics API (overall load)
- Systems API (systemInfo)

The project can evolve rapidly, this list is accurate only as of the date of writing.

Detecting Jenkins version

To check the version of Jenkins, load the top page (or, as of 1.483, any `.../api/*` page too) and check for the `X-Jenkins` response header. This contains the version number of Jenkins, like "1.404" This is also a good way to check if an URL is a Jenkins URL.

Discovering Jenkins on the network

Jenkins instances listen on UDP port 33848. Whenever a UDP packet is received, it will respond with a short XML fragment that shows the connection information. This XML has the following format:

```
<hudson>
  <version>1.380</version>            <!-- version of Jenkins -->
  <url>http://somewhere/jenkins/</url> <!-- HTTP URL. Not available if
not configured -->
  <slave-port>12345</slave-port>      <!-- if TCP slave listener port
is configured, its number -->
</hudson>
```

By using this, a client can use a UDP broadcast to try to discover nearby Jenkins instances. This is primarily useful for Swarm Plugin.