Project Proposal

Developers need to know what to expect in terms of responses for each REST API endpoint, so external tools like jenkins-rest can be developed with confidence, and possibly with the help of some REST specification automation. For example, this could be done using OpenAPI. The two main contents users of the REST API are looking for are:

- HTTP responses and codes
- Body of message and its format (e.g. JSON, html, etc.)

Jenkins does not have automated REST API documentation at all at this time. A lot of APIs are contributed from extensions, so there are multiple REST APIs (core and plugins) of varying versions. The goal of this project is to find and implement the extraction of the REST APIs from the sources and generate and publish the REST APIs respective documentation.

Generating the expected HTTP responses is a difficult task. The student is expected to study Jenkins core to identify ways to extract them. For example, they could be extracted from Javadocs and annotations.

As part of the community bonding and student project proposal phase, the student is expected to make a few proposals on how to specify and generate the REST API for the Jenkins core and for the plugins. In the case the student finds that it is not possible to generate the REST API from a specification, the student should identify why this is not possible. We also ask the student to explore and propose a way to have REST API of plugins be generated from a REST API specification. For example, some auto-code could populate what the javadoc would look like in an empty-plugin used by the maven plugin generator.

The student is also expected to study and propose how the REST API documentation generation could be part of the REST API generator.

It might be helpful to automatically generate some code for the REST API when the plugin developer creates a plugin for the first time using the plugin skeleton generator. Any methodology created to handle the REST API should be built into the skeleton generator.

The jenkins core REST API and the plugins own REST API need to be versioned separately. It is suggested to first focus first on generating the specification, then later look at the versioning of the REST API. Nested objects make versioning challenging.

Jenkins users should be easily able to see the REST APIs available for their installed Jenkins. For Jenkins core, this could be done with a URL like: http://localhost/rest/api/1.0, http://localhost/plugin/rest/api/1.0, and the plugins would have their own REST API path with a version number. Plugins and the core would thus have their own version number, and an additional REST API version number. Automated API documentation using the OpenAPI 3.0 specification is part of identifying the API specs.

Links

There are many examples of such documentation on the web:

- Bitbucket REST API (link to Bitbucket documentation)
- Artifactory REST API (link to Artifactory documentation)

- OpenAPI3 description of the Jenkins REST API: https://github.com/cliffano/swaggy-jenkins (note: the swaggy-jenkins author now recommends OpenAPI (explanation).
- This stackoverflow question talks briefly about generating a REST API spec in WADL format: https://stackoverflow.com/questions/12405911/how-can-i-generate-wadl-for-rest-services
- This blog post talks about swagger and WADL: https://swagger.io/blog/api-development/getting-started-with-swagger-i-what-is-swagger/
- Making Stapler more declarative discussion leading to a comment on swagger.
- Seeking help on clarifying this proposal: https://groups.google.com/forum/#!topic/jenkinsci-dev/mYeM5qA6tGM
- Plugin skeleton generator: https://github.com/jenkinsci/maven-hpi-plugin
- Jira ticket on generating spec for Jenkins REST API from 2016 https://issues.jenkins-ci.org/browse/JENKINS-35808
-

Newbie-friendly issues

- Not necessarily a project JIRA, but they maybe should become familiar with Swagger.
- Newbie friendly issues
- Javadoc Landing Page — While Javadoc isn't Rest API, user experience concerns described in INFRA-1717 for Javadoc Landing Page could be re-considered when determining where to publish Rest API accordingly

Skills to improve/study

- Java
- REST API
- OpenAPI
- Stapler http://stapler.kohsuke.org/ (more info:https://github.com/jenkins-infra/jenkins.io/pull/2157#issuecomment-471230609)

Back to the master Project Proposal Table.

Discussions

Private email with Cliffano contains important information, copied here:

> *Hi Martin,*
>
> *My apologies for the late reply, I recently returned from end of year travel*
>
> *The scope can still be improved. Primarily, I am interested on accessing any Jenkins master instance from the pipeline DSL, via the Jenkins REST API. Instead of using the http request plugin (which does the job), it would be easier for the user to work at a higher level. The idea is to create a plugin that provides easy REST API access in the Pipeline DSL. I did not know about swaggy Jenkins until*

recently. This plugin I have in mind could be a new generated client of swaggy-jenkins. Swaggy Jenkins could generate this new client and leverage the Apache jclouds library ComputeService, but it could also use any other http library under the hood (I just find jclouds to be very "declarative" and succinct).

For your information, I also want the same facility to access artifactory and bitbucket from the Jenkins DSL Pipeline.

Of course the proposal is open for improvements and clarifications

+1 on the Jenkins Pipeline DSL being an OpenAPI generator client, but I guess this means that the spec generator from the other proposal will be a dependency in order to make this one happens.

Not sure if there's a better place to put this technical note, I'd recommend writing a generator against openapi-generator instead of swagger-codegen , some reasons are captured on this Q&A https://github.com/OpenAPITools/openapi-generator/blob/master/docs/qna.md .

...

Stapler knows about the payload model, so ideally the OpenAPI spec model could come from Stapler. The area that I would be able to contribute on is the OpenAPI spec and API clients side.

What I did for Swaggy-Jenkins was to reverse engineer the response model definition from the HTTP response payload.
Part of the reason why I did it this way was because I was hearing from James Dumay back in early 2017 that there could be an effort to move to GraphQL instead of REST.

Update from Cliffano: I dug up some info via Michael Neale, he said any GraphQL effort has been shelved for now. So this means REST is not going anywhere.I haven't had the chance to check with Vivek Pandey about the future direction of Jenkins API.

Cliff

Comment by Cliffano Subagio:
Has anyone dived into Stapler yet? I haven't revisited since the early days of Hudson. One problem I thought might come up is on the fact that Jenkins (via Stapler) can produce response payload with nested properties which structure seems to be dynamic IIRC. I'm not sure how this can be represented in OpenAPI spec.