How to Instantiate SCAIFE API Calls:

Using SEI SCAIFE Code, the SCAIFE API, Swagger-Editor, and Developing Your Tool with Auto-Generated Code

Lori Flynn  
Ebonie McNeil  
Joseph Yankel

**July 2020 (Updated October 2020)**

**CERT Division**

Approved for public release and unlimited distribution.

http://www.sei.cmu.edu

Copyright 2020 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:\* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use:\* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

\* These restrictions do not apply to U.S. government entities.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM20-0550

# Change Log

We expect to update this technical manual regularly since the codebase will change over time. Check the SEI website at <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=644354> to check for the latest update.

October 2020 Updates

The following updates were made to this technical manual in October 2020. They are minor changes but are important because they ensure that the instructions match the recently revised SCAIFE code.

1. We substituted the name **model\_api** with **scaife** in all instances. This name is used as a directory name and a code repository name, and it is sometimes used in commands.
2. We updated two script names:

|  |  |
| --- | --- |
| From | To |
| create\_manual\_test\_project.1.rosecheckers.py | create\_manual\_test\_project\_1\_dos2unix\_rosecheckers.py |
| create\_manual\_test\_project.1.microjuliet.py | create\_manual\_test\_project\_1\_microjuliet\_cppcheck.py |

Contents

[Change Log i](#_Toc52958043)

[Purpose of This Technical Manual 1](#_Toc52958044)

[Use This Differently, Depending on Your Access to SEI SCAIFE System Code 1](#_Toc52958045)

[Instantiating Particular SCAIFE Modules: Use Cases 2](#_Toc52958046)

[Background About SCAIFE 3](#_Toc52958047)

[Prerequisites 5](#_Toc52958048)

[Prepare SCAIFE Code, Start Containers, and Install Swagger-Editor 6](#_Toc52958049)

[Populate Initial Data in SCALe and Other SCAIFE Servers 7](#_Toc52958050)

[Start to Make API curl Calls Via the Swagger-Editor GUI 8](#_Toc52958051)

[Use the Registration Module’s API with the Swagger-Editor GUI 8](#_Toc52958052)

[Use the DataHub Module’s API with the Swagger-Editor GUI 20](#_Toc52958053)

[Using Auto-Generated Client Code to Implement SCAIFE API Calls *from* Your Tool 29](#_Toc52958054)

[Detail for Auto-Generating Code from the SCAIFE APIs Yourself 29](#_Toc52958055)

[Using Auto-Generated Server Code to Implement SCAIFE API Calls *into* Your Tool 31](#_Toc52958056)

[Useful Information and Commands for SCAIFE Containers 32](#_Toc52958057)

[How to Stop, Restart, and Check if Containers Are Running During Testing 32](#_Toc52958058)

[Command to Test that Swagger Server Is Running (Substitute $PORT for Number in Table Above) 32](#_Toc52958059)

[Commands to Refresh All SCAIFE Containers 32](#_Toc52958060)

[Commands to Refresh a Single SCAIFE Independent Container (SCALe or Swagger) 33](#_Toc52958061)

[Commands to Pause and Resume All SCAIFE Independent Containers 33](#_Toc52958062)

[How to Run Scripts, Tests, and Other Commands in SCAIFE Containers 33](#_Toc52958063)

[Using Docker-Compose to Test the Script-Based Creation of SCALe Projects and Use Them in an Independent Container 33](#_Toc52958064)

[Command to Get a Bash Command Line for a Container (e.g., and Then Inspect the Container’s Internal Filesystem) 33](#_Toc52958065)

[How to Read Log Files in SCAIFE Containers 34](#_Toc52958066)

[Command to Obtain a SCAIFE Mongodb Log 34](#_Toc52958067)

[Command to Access SCAIFE Log Files 34](#_Toc52958068)

[Command to Monitor Log Files in Containers 34](#_Toc52958069)

[Commands to Connect to SCALe in an Independent Container, Access One of its sqlite3 Databases, then Execute sqlite3 Commands 35](#_Toc52958070)

# Purpose of This Technical Manual

This technical manual provides techniques we enabled and documented that help you efficiently start to instantiate Source Code Analysis Integrated Framework Environment [(SCAIFE) API](https://github.com/cmu-sei/SCAIFE-API) methods to integrate your own tools. As a developer, you can use these techniques to enable your tool (as a server) to receive SCAIFE API calls and your tool (as a client) to make SCAIFE API calls. In both cases, you need to develop some internal SCAIFE-required logic in your tool. The modular SCAIFE system enables the substitution of any of the five types of SCAIFE modules detailed in the Background About SCAIFE section.

## Use This Differently, Depending on Your Access to SEI SCAIFE System Code

This technical manual contains useful information for you, whether you have full, partial, or no access to SCAIFE System code developed by the SEI. However, depending on the extent of your access to that SCAIFE System code, you should use the manual in different ways, as described in this section.

**Full Access:** To do *everything* described by this document, you need SCAIFE System code developed by the SEI (that fully instantiates the SCAIFE API for all five SCAIFE modules, including internal logic), and provided by the SEI. Currently, we can only send SCAIFE code to DoD organizations. If you are a DoD organization and interested in the code, please send us a request (email [lfynn@sei.cmu.edu](mailto:lfynn@sei.cmu.edu)) and we will send you the full SCAIFE code.

**Partial Access:** In the near future (possibly as soon as September 2020), we expect to publicly release our (latest) research version of [the SCALe (Secure Code Analysis Lab) tool](https://insights.sei.cmu.edu/sei_blog/2018/12/scale-v-3-automated-classification-and-advanced-prioritization-of-static-analysis-alerts.html) with many features added that enable it to work as the SCAIFE UI module as well as to function as a stand-alone tool. That release will give such developers (ones that cannot receive the full SEI-developed SCAIFE System code) a lot of help since the developers will have one of the five SCAIFE modules to do the testing with. The UI module is a complex system, so this module should help external SCAIFE developers a lot for initial testing and development, even if eventually they will replace the UI module with their own tool. The Priority and Registration modules are far simpler and will take developers much less effort to instantiate. Developers that have only partial SCAIFE System code access should perform the following variation on the instructions in the main body of this technical manual:

* In the Prepare SCAIFE Code, Start Containers, and Install Swagger-Editor section, bring up the Docker container directly using the SCALe code’s Dockerfile, since the SCALe code alone currently does not natively enable docker-compose (unless the release information says that we have enabled that for SCALe stand-alone releases, by the time of the release).
* If you did not receive the full set of five SCAIFE API (.yaml) files, one for each module, you should download the SCAIFE API .yaml files published on GitHub at <https://github.com/cmu-sei/SCAIFE-API>. Use these files when instructions say to view the API in swagger-editor.
* In the Populate Initial Data in SCALe and Other SCAIFE Servers section, you will not be able to run the create\_manual\_test\_project\_1\_microjuliet\_cppcheck.py since it requires other working SCAIFE modules. Instead, you should run create\_basic\_project.py, which creates a SCALe-local project. You cannot create (nor run) a classifier, because you do not yet have a Statistics module, Registration module, and DataHub module.
* In the Start to Make API curl Calls Via the Swagger-Editor GUI section, do the following:
  + You can examine the UI module SCAIFE API interface from swagger-editor, as described in the main body of the technical manual.
  + After that, you should start by instantiating a simple registration module. Do that using information in the Using Auto-Generated Client Code to Implement SCAIFE API Calls *from* Your Tool and Using Auto-Generated Server Code to Implement SCAIFE API Calls into Your Tool sections. After that, you should register a user in SCAIFE, using instructions in the manual.
  + Performing many of the steps in this section will not be possible until you have instantiated the other SCAIFE servers and at least mocked logic to send usable-during-initial-testing return values from the initial stub functions. Do that using information in Using Auto-Generated Client Code to Implement SCAIFE API Calls from Your Tool and Using Auto-Generated Server Code to Implement SCAIFE API Calls into Your Tool sections.

**No Access:** If you are a developer who does not have access to any of the SCAIFE System code, you can use some of the recommendations the document (that this blogpost will be about). Download the SCAIFE API .yaml files published on GitHub at <https://github.com/cmu-sei/SCAIFE-API>. Use these files when instructions say to view the API in swagger-editor. You can auto-generate SCAIFE client code and server code in whatever language your own tool(s) are written in. Do that using information in the Using Auto-Generated Client Code to Implement SCAIFE API Calls from Your Tool and Using Auto-Generated Server Code to Implement SCAIFE API Calls into Your Tool sections. However, it will take you more work than developers with partial or full SCAIFE System code access. In your initial integration testing between SCAIFE modules, you will have to instantiate an initial server and get it up and running to start your initial testing, especially for integration testing between servers. (Our DoD collaborators using the instantiated full SCAIFE System have a head start because they can test with up to five of our SCAIFE System modules running.) In the instructions in this technical manual, we tell you to start up all five SCAIFE modules and populate a project in the SCAIFE DataHub. After that, we instruct you to (as preparation to do it from your own tool) interact via the swagger-editor with the active Registration and DataHub modules to upload new static analysis tool output to the DataHub module for the SCAIFE project (previously uploaded) on the DataHub. Because you have no access to the SEI SCAIFE System code, you will instead need to instantiate at least some code mock-ups returning data from at least one server to start your testing. Eventually, you will need to instantiate each of the SCAIFE module servers.

## Instantiating Particular SCAIFE Modules: Use Cases

We expect the UI module to be the most common (by far) SCAIFE module that you will want to instantiate. We expect the following are the most common SCAIFE modules you will want to substitute (in order):

1. **UI Module**: This module is the one with the GUI interface for users; this is where the static analysis tool/framework sits. For example, we use a [research version of the SEI SCALe static analysis framework](https://insights.sei.cmu.edu/sei_blog/2018/12/scale-v-3-automated-classification-and-advanced-prioritization-of-static-analysis-alerts.html) (that we developed) for this module, in the instantiated versions (SCAIFE System) of the SCAIFE architecture that we have released so far.
2. **DataHub Module**: This module stores SCAIFE projects and packages and various data structures used as part of SCAIFE projects and packages. We are designing this module so its data may come from multiple UI modules.
3. **Statistics Module**: This module performs classification, active learning, and hyper-parameter optimization, plus it stores classification schemes related to particular SCAIFE projects (and associated particular DataHub and UI module for that), including optional adaptive heuristics (aka active learning algorithms) and hyper-parameter optimization techniques.
4. **Prioritization Module**: This module stores meta-alert prioritization schemes, with permission information related to the SCAIFE project and the uploading organization name.
5. **Registration Module**: This module handles user and inter-server authentication.

**Regarding #1)** If you are a static analysis tool/framework developer or user, the benefit of integrating your tool with the modular SCAIFE system is that you can (with relatively small effort) enable your static analysis tool to use automated machine learning classifiers for static analysis meta-alerts and use sophisticated prioritization schemes for static analysis meta-alerts while still using your own tool that you prefer.

**Regarding #2)** If your organizations wants to optimize the use of particular data structures (reasons to do this may include optimizations for performance, scalability, or to re-use the organization’s data archive), you may wish to instantiate the DataHub module.

**Regarding a mix of #1 and #2 and possibly #3)** Some static analysis framework tools (e.g., [SWAMP-in-a-box](https://github.com/mirswamp/deployment), available on GitHub) include both a GUI front end (like #1) and a database back-end for aggregating project and package data (like #2), and SWAMP also does its own registration. In this case, you can simply consider the SCAIFE DataHub module, UI module, and Registration module to be a single consolidated module, and need only to instantiate API calls into or out of the consolidated module (meaning only instantiating calls to or from the Priority and Statistics modules).

## Background About SCAIFE

SCAIFE is an architecture that supports static analysis meta-alert classification and prioritization for a wide variety of static analysis tools (and other tools). The SCAIFE architecture shown in the figure includes five servers. The system is modular, designed so that each module can be instantiated by different tools/software while the overall system should maintain the same functionality. The UI Module has a GUI front-end that enables display of flaw-finding static analysis (FFSA) alerts and stores local projects.

The SCAIFE architecture is intended to enable a wide variety of FFSA tools and alert aggregator tools to obtain classification and prioritization functionality by interacting as a UI module with the rest of the SCAIFE system. These tools must instantiate UI module API calls to the other servers, to do so. The Datahub module stores data (e.g., tool, alert, project, test suite meta-data, adjudications True or False) from one or more UI modules and adjudicates some meta-alerts. The Statistics module creates, runs, and stores classifiers and adaptive heuristic (active learning) algorithms and automated hyper-parameter algorithms. The Prioritization module stores prioritization formulas and user-uploaded prioritization fields. The Registration module is used for authentication and access control. It generates registration tokens, plus it provides authentication and basic authorization for other servers.

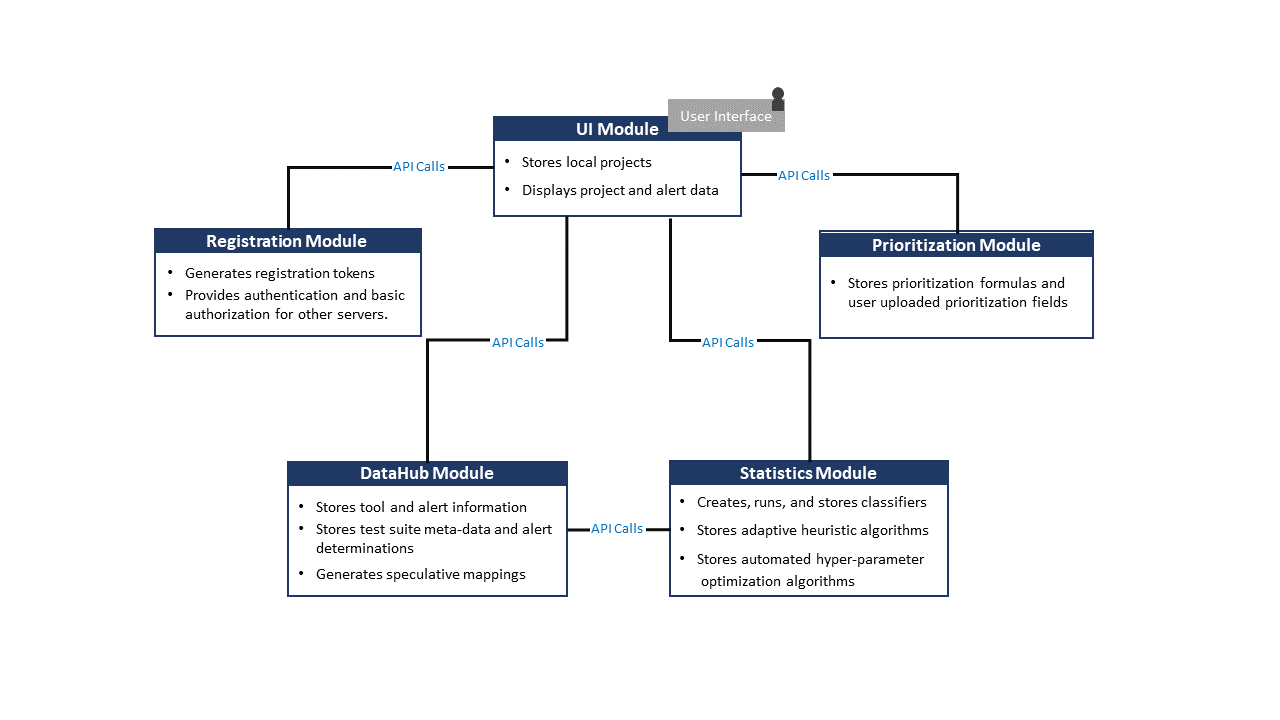


Figure : SCAIFE Architecture Diagram

# Prerequisites

|  |
| --- |
| **How To View screenshots** |
| The screenshots in this document are large. To see them, view this document in “Web mode” (not “Print mode”). Print mode will cut off much of the screenshots.  To switch to Web view in Microsoft Word, select the View tab, and then the “Web Layout” icon.  Alternatively, click the Web Layout view icon. (The only view icons in the bottom-right corner of the Microsoft Word 2019 document window are Read Mode, Print Layout, and Web Layout.) |

As described in the Use This Differently, Depending on Your Access to SEI SCAIFE System Code section, to do *everything* described by this document, you need the full SCAIFE System code developed by the SEI. Currently, we can send SCAIFE code only to DoD organizations. If you are a DoD organization and are interested in the code, please send us a request and we will get you the full SCAIFE code. For full SCAIFE System code users to do everything described in this document, you also need a Linux machine (or VM) that has the following:

* around 10 GB of extra space (2.5 GB for the expanded files, and 6.5 GB virtual space for the containers, but prior to any activity, they use less than 1 GB on our test machine, per docker ps --size)
* docker installed from the Docker website (<https://www.docker.com/resources/what-container>) (Docker Community Edition—Docker version 19.03.8, build afacb8b7f0—is a tested and working version installed on an Ubuntu version 20.0 LTS. It’s best to download the latest stable version for your machine directly from Docker’s own website, not to use the default installed docker version on the Linux operating system.)

For the Using Auto-Generated Client Code to Implement SCAIFE API Calls from Your Tool section, some of the instructions refer to using code auto-generated by the SEI. However, you can alternatively auto-generate that code, as described in that section.

# Prepare SCAIFE Code, Start Containers, and Install Swagger-Editor

On the above Linux machine, do the following:

1. Unzip and decrypt the file using the decryption passphrase provided by SEI.
2. Start up the docker containers for SCAIFE by doing the following:
   1. change the directory to the scaife directory (base directory of the unzipped download).
   2. In the scaife directory, run the following command to start the SCAIFE containers:

docker-compose -f docker-compose.yml up --build

1. Install swagger-editor. There are docker containers for swagger-editor. The explanation below of how to install and use was copied from the GitHub swagger-editor README (with slight edits to point you to the directory with the swagger.yaml files): <https://github.com/swagger-api/swagger-editor>

There is a docker image published in [DockerHub](https://hub.docker.com/r/swaggerapi/swagger-editor/).

To use this, run the following:

docker pull swaggerapi/swagger-editor

docker run -d -p 80:8080 swaggerapi/swagger-editor

This will run swagger-editor (in detached mode) on port 80 on your machine, so you can open it by navigating to http://localhost in your browser.

Open one of the five swagger.yaml definition files on your host, from the folder (substitute a server name datahub, priority, ui, registration, or stats for /$SERVER\_NAME), from a filepath that ends $BEGINNING\_OF\_FILEPATH/scaife/$SERVER\_NAME\_server\_stub/swagger\_server/swagger/swagger.yaml

as follows (edit to substitute the filepath in purple font where you actually placed the scaife directory for the SCAIFE code you unzipped).

**EXAMPLE Command for Registration Server**

docker run -d -p 80:8080 -v $(pwd):/tmp -e SWAGGER\_FILE=/home/username/scaife/registration\_server\_stub/swagger\_server/swagger/swagger.yaml swaggerapi/swagger-editor

# Populate Initial Data in SCALe and Other SCAIFE Servers

Use the following command from the scaife directory. (The purpose of this command is to prepare the system before you start testing your first API calls from your tool or the swagger-editor.) The following command uses a script in the SCALe container that creates a SCALe project, adjudicates (true or false) some meta-alerts, uploads the project to the SCAIFE datahub, then creates a classifier in the SCAIFE stats module, runs the classifier, and then receives the confidence values from the classifier in SCALe:

docker exec scale python scripts/automation/create\_manual\_test\_project\_1\_microjuliet\_cppcheck.py

Check to make sure the command completed successfully as expected:

Open your browser to view the new project at http://localhost:8083/projects/1.

Browser versions that have been tested and are known to work include the following:

* Firefox v 76.0.1 has been tested and works, other versions may also work.
* Google Chrome has not been thoroughly tested but version 81.0.4044.138 works for at least the following step.

Scroll to the right in the Meta-alerts list until you can see the column header “Confidence.” There should be values in that field. (In the blue-backed rectangle at top for selecting filtering options, you can select “Sort by:” to be “Confidence” then select the “Filter” button. If you select “Sort Direction” to be ascending (“asc”), the top Confidence value will be 0.0. If you select “Sort Direction to be descending (“desc”), select the “Filter” button again, and the top confidence value will be 0.75.

If your check above showed the expected values, then the Project is now successfully uploaded in SCAIFE to be used with SwAT and/or the swagger-editor.

# Start to Make API curl Calls Via the Swagger-Editor GUI

All of the following API call requests will be shown via curl request statements, so your machine needs curl to be installed (e.g., for machines that can install using apt, sudo apt-get install curl would install curl). The curl tool is used in command lines or scripts to transfer data. For more information about curl, see <https://curl.haxx.se/>). To begin, you should run the API calls from the swagger-editor GUI.

## Use the Registration Module’s API with the Swagger-Editor GUI

First, load the registration’s swagger.yaml file into the swagger-editor via the GUI or with the startup command example above. Your view will look like the following:

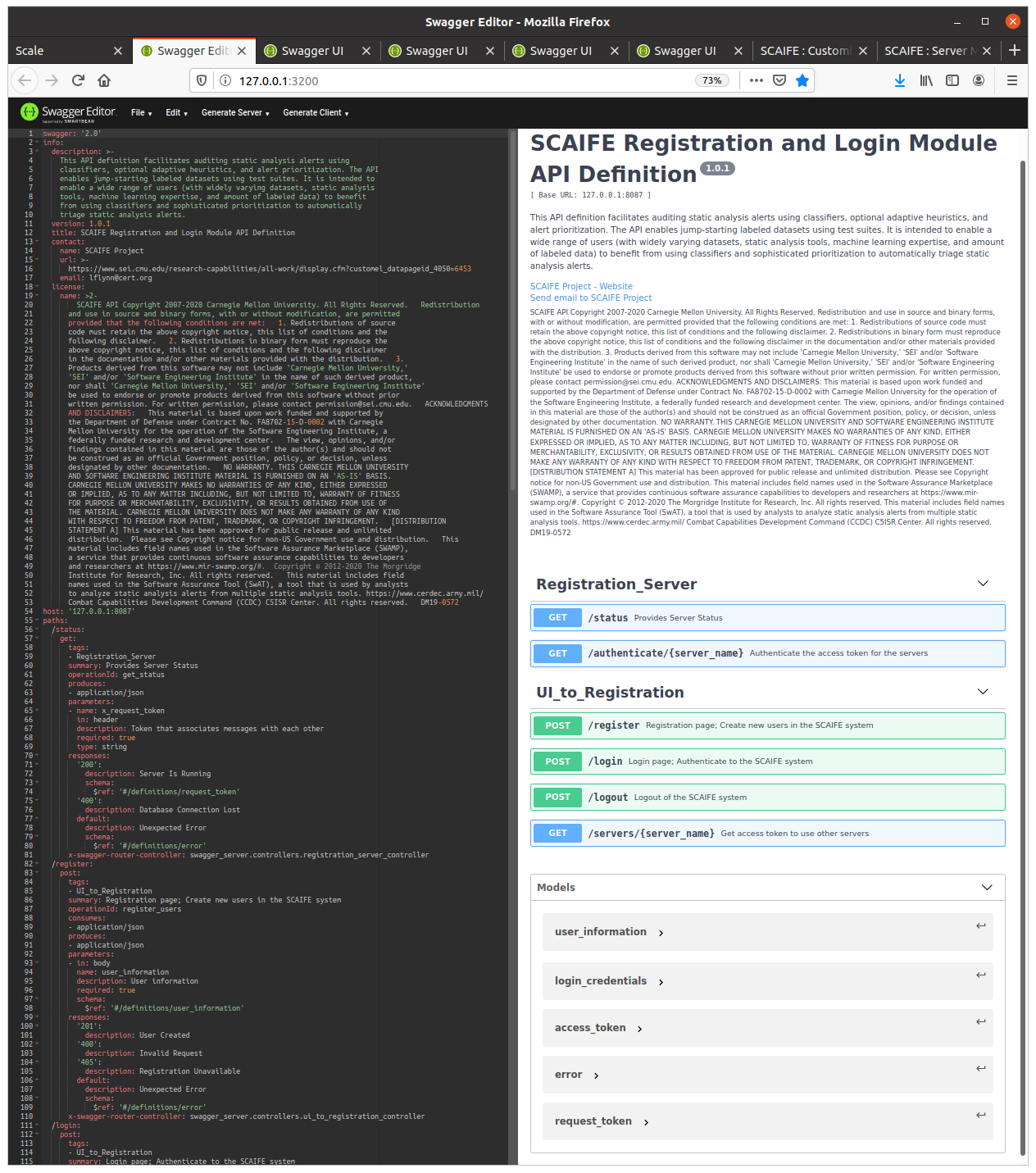


Figure : Load the Registration Module’s API into the Swagger-Editor GUI

Register a user in SCAIFE. In the swagger-editor GUI, under the section “UI\_to\_Registration,” select the POST /register API call and it will expand. Now your view (on the right) looks like the following:

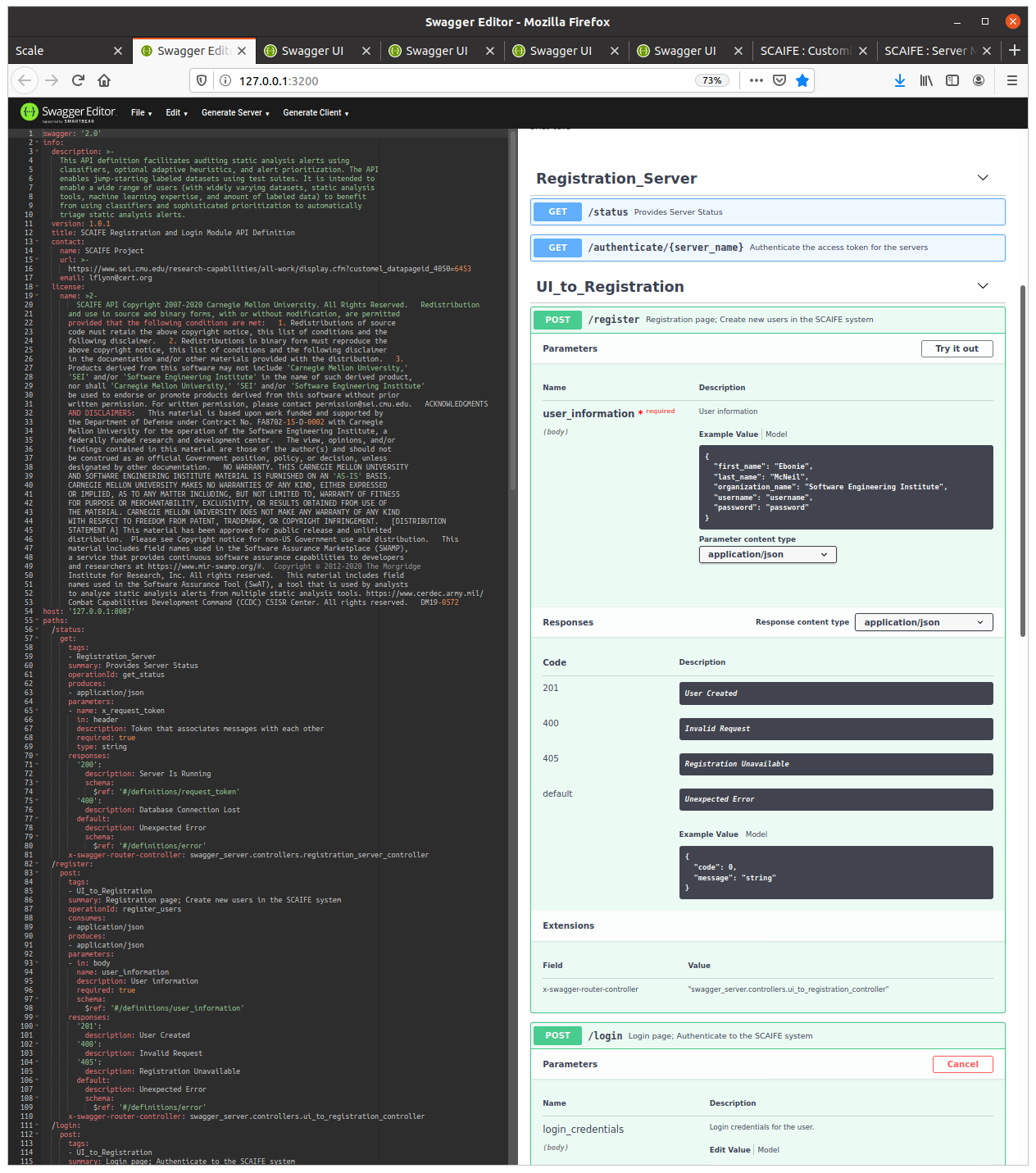


Figure : Register a SCAIFE User via the Swagger-Editor GUI

Select the button “Try it out” (on the top right). That expands the options. Keep the default values.

Next, select the blue button “Execute.”

The screenshot below shows the successful results on the right, after execution. Note that under “Server Response” it shows “201” and “User Created,” indicating success.

Also, note that the screenshot below also shows the exact Curl command that was used. (This command could be used by your code in SwAT, but likely you will want to use the Java client code that has been auto-generated for you instead.) That Curl command (which uses the default values that you kept) is:

curl -X POST "http://127.0.0.1:8087/register" -H "accept: application/json" -H "Content-Type: application/json" -d "{ \"first\_name\": \"Ebonie\", \"last\_name\": \"McNeil\", \"organization\_name\": \"Software Engineering Institute\", \"username\": \"username\", \"password\": \"password\"}"

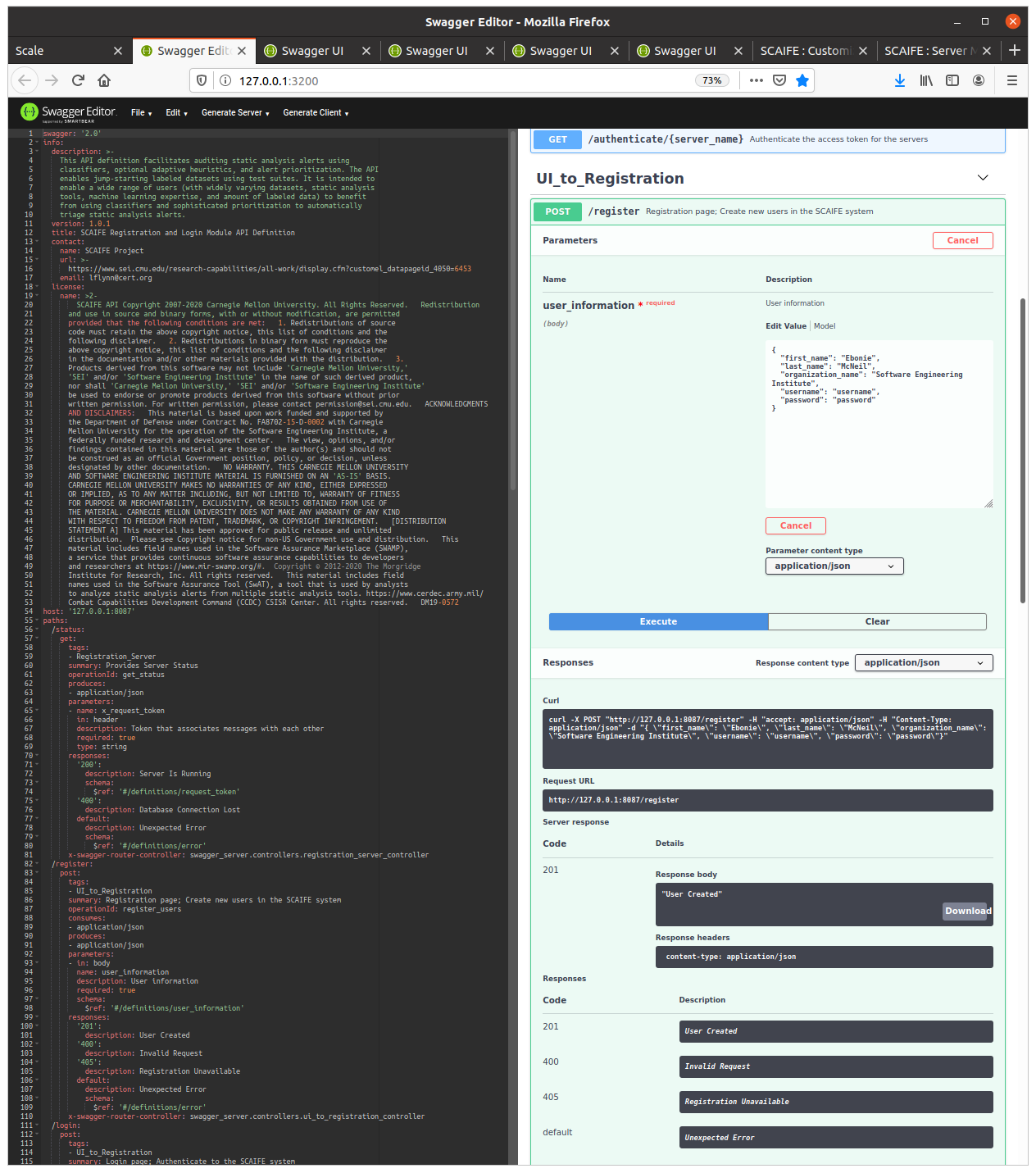


Figure : Results of the Successful Execution of SCAIFE User Registration via the Swagger-Editor

Select the POST /registration API call in the GUI again so it compresses the view to just that API name again.

To send tool output (or full SCAIFE projects, etc.) to SCAIFE, first SwAT (or the user of swagger-editor) must log into the Registration module (POST /login) to retrieve an access\_token for the DataHub. In the swagger-editor GUI, under the section “UI\_to\_Registration,” select the POST /login API call and it will expand. Now your view (on the right) looks like the following:

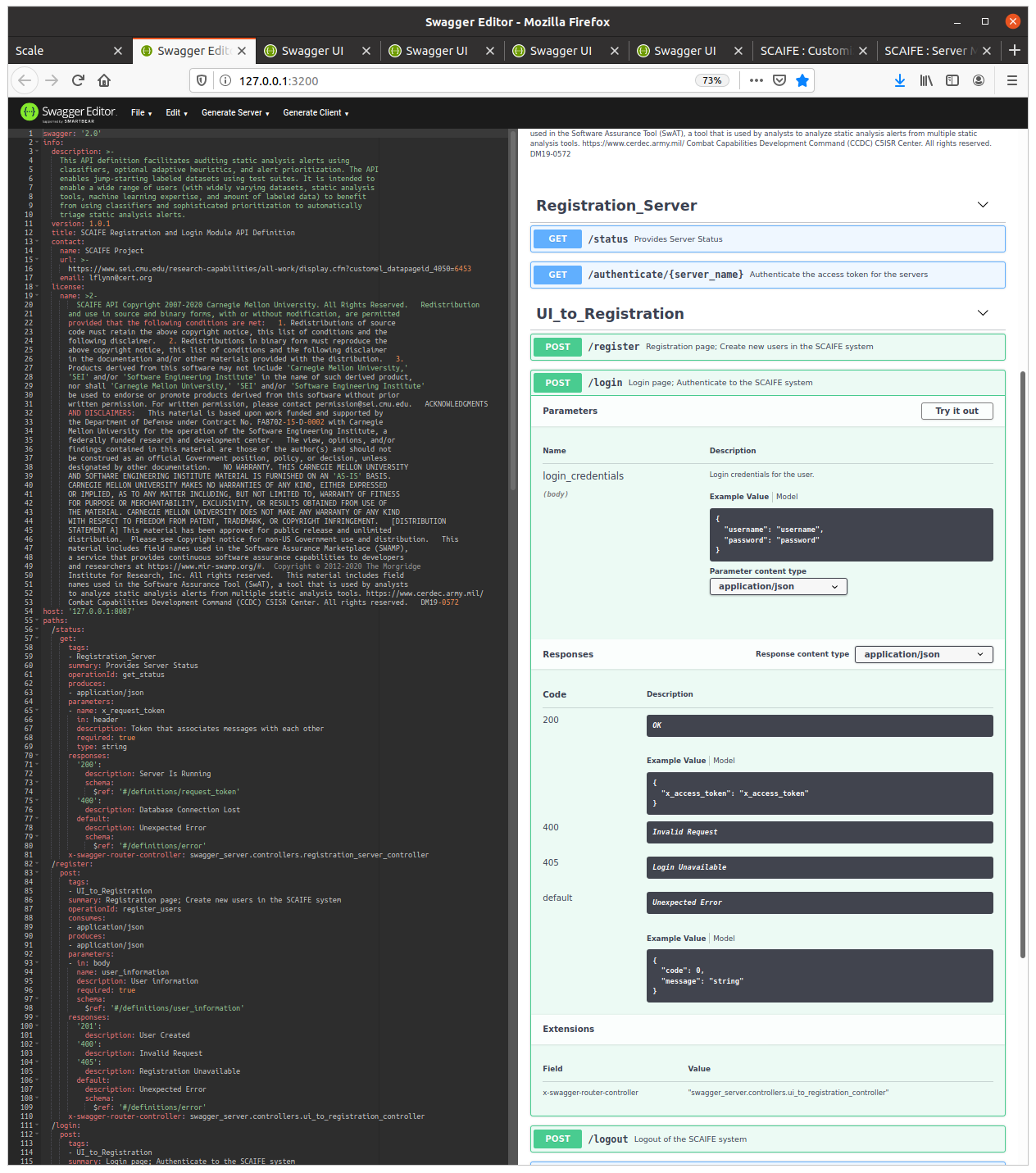


Figure : SCAIFE Login API Call Options via the Swagger-Editor GUI

Select the button “Try it out” (on the top right). That expands the options.

This time, you may need to edit from the default username and password (since the default here may be different than the default used in POST /registration). You should use the following values:

"username": "username",

"password": "password"

Next, select the blue button “Execute.”

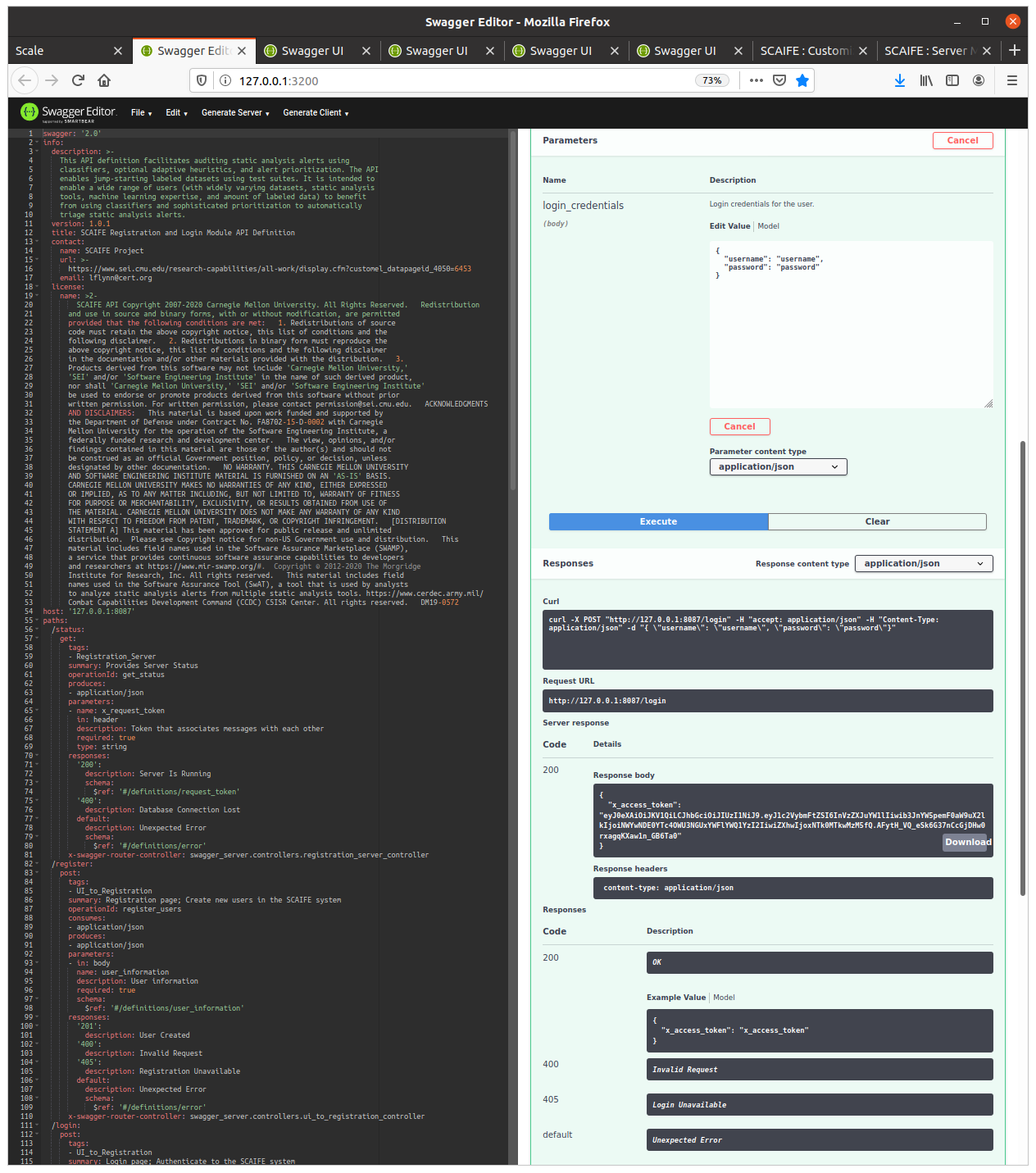


Figure : Results of the Successful Execution of SCAIFE Login via the Swagger-Editor GUI

Note in the screenshot above that the return value is successful “200” and the Curl command used is shown:

curl -X POST "http://127.0.0.1:8087/login" -H "accept: application/json" -H "Content-Type: application/json" -d "{\"username\": \"username\", \"password\": \"password\"}"

**The access token returned (i.e., user access token) will be used in the subsequent call to send tool output to the DataHub.**

Copy the access token from the swagger-editor interface to a file.

NOTE: Later, you will do this within SwAT. After making this client API call, SwAT should store the access token.

In this example, the access token value is

“x\_access\_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InVzZXJuYW1lIiwib3JnYW5pemF0aW9uX2lkIjoiNWYwNDE0YTc4OWU3NGUxYWFlYWQ1YzI2IiwiZXhwIjoxNTk0MTkwMzM5fQ.AFytH\_VQ\_eSk6G37nCcGjDHw0rxagqKXaw1n\_GB6Ta0"

Next, do the following:

* Click on GET /servers/{server\_name}.
* Click Try it out button.
* Enter the user access\_token copied in previous step.
* Enter datahub for server\_name.
* Click the Execute button.
* Copy the x\_access\_token that is returned to use it for the rest of the tests.

The swagger-editor GUI shows that the curl command used was

curl -X GET "http://127.0.0.1:8087/servers/datahub" -H "accept: application/json" -H "x\_access\_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InVzZXJuYW1lIiwib3JnYW5pemF0aW9uX2lkIjoiNWYwNDE0YTc4OWU3NGUxYWFlYWQ1YzI2IiwiZXhwIjoxNTk0MTkwMzM5fQ.AFytH\_VQ\_eSk6G37nCcGjDHw0rxagqKXaw1n\_GB6Ta0"

A screenshot of the successful call is shown below:

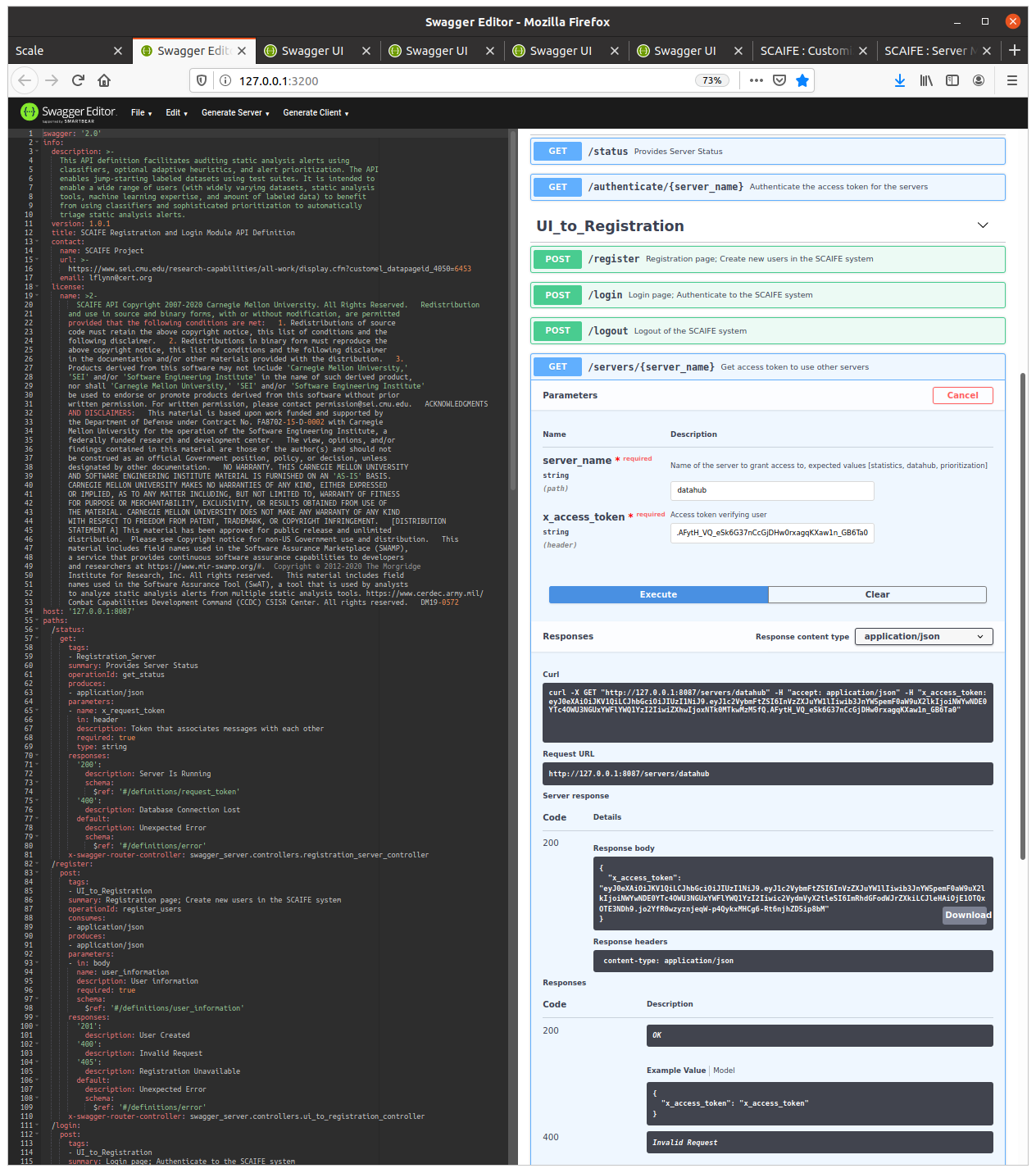


Figure : Successful Execution of Getting the SCAIFE Server Access Token via the Swagger-Editor GUI

Note in the screenshot above that the token received in the response body is

"x\_access\_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InVzZXJuYW1lIiwib3JnYW5pemF0aW9uX2lkIjoiNWYwNDE0YTc4OWU3NGUxYWFlYWQ1YzI2Iiwic2VydmVyX2tleSI6ImRhdGFodWJrZXkiLCJleHAiOjE1OTQxOTE3NDh9.jo2YfR0wzyznjeqW-p4QykxMHCg6-Rt6njhZD5ip8bM”

## Use the DataHub Module’s API with the Swagger-Editor GUI

Next, load the DataHub’s swagger.yaml file into the swagger-editor GUI. From the menubar (on the top left), select File->Import file and then navigate to scaife/datahub\_server\_stub/swagger\_server/swagger/swagger.yaml

See the initial screenshot below:

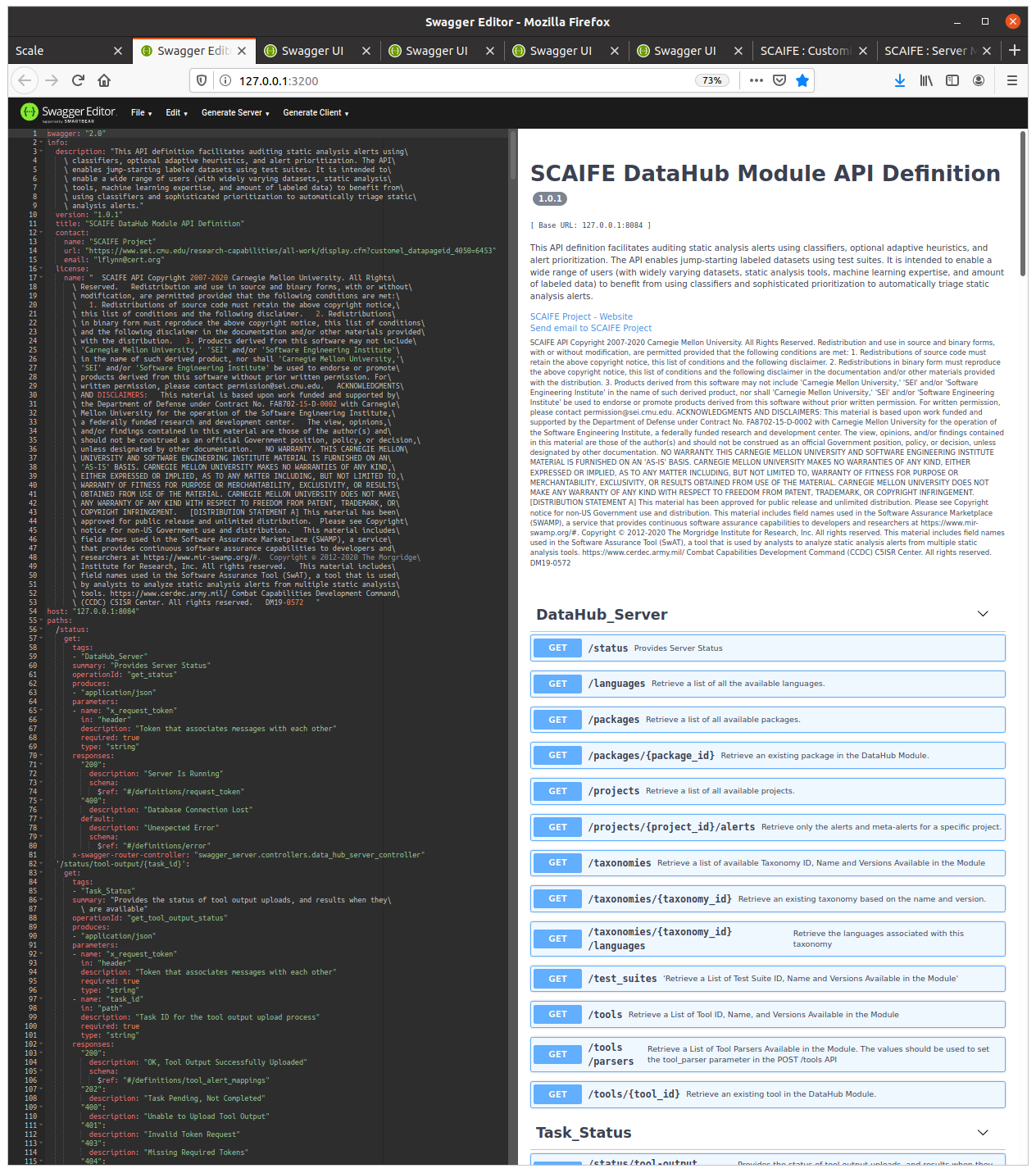


Figure : The DataHub Module’s API with the Swagger-Editor GUI

Next, get the package id by doing the following:

* Navigate to Datahub\_Server GET /packages.
* Click Try it out.
* Enter x\_access\_token and any value for x\_request\_token (e.g., for the latter, “abc”)
* Click Execute and retrieve the package\_id.

See the screenshot below after successful execution. (Code “200” and the Response body include package\_id of 5f04041a823a3994fe4a08f7. Since we’ve uploaded only one SCAIFE package [using that SCALe script at the beginning], we take the only package\_id that is in the response.)

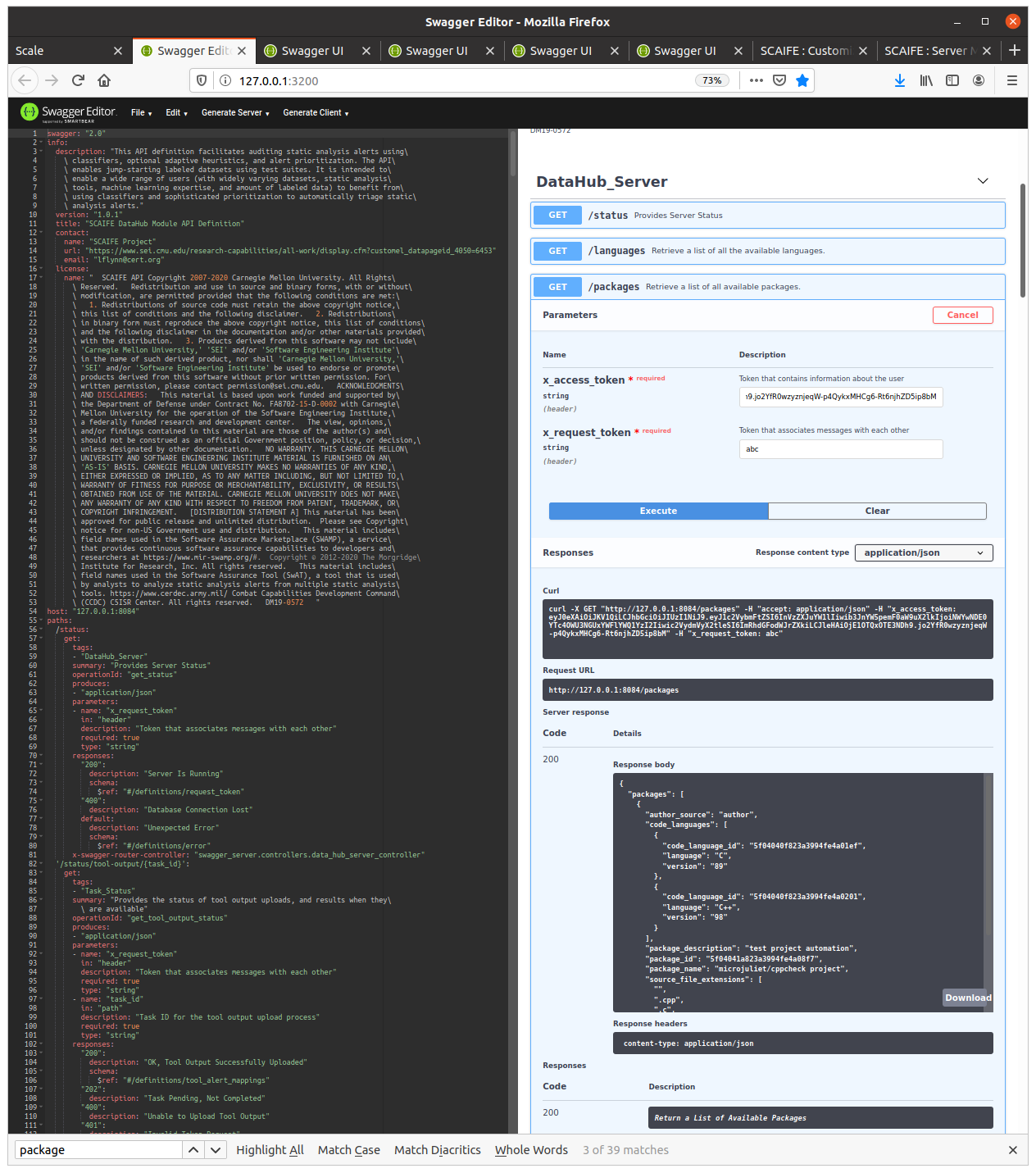


Figure : Successful Execution of Getting Packages from DataHub via the Swagger-Editor GUI

Next, get the tool id by doing the following:

* Navigate to Datahub\_Server GET /tools.
* Click Try it out.
* Enter x\_access\_token and any value for x\_request\_token (e.g., for the latter, “abc”).
* Click Execute and retrieve the tool\_id.

The results of successful execution are shown in the screenshot below, which in this case shows a tool\_id of 5f040412823a3994fe4a066d.

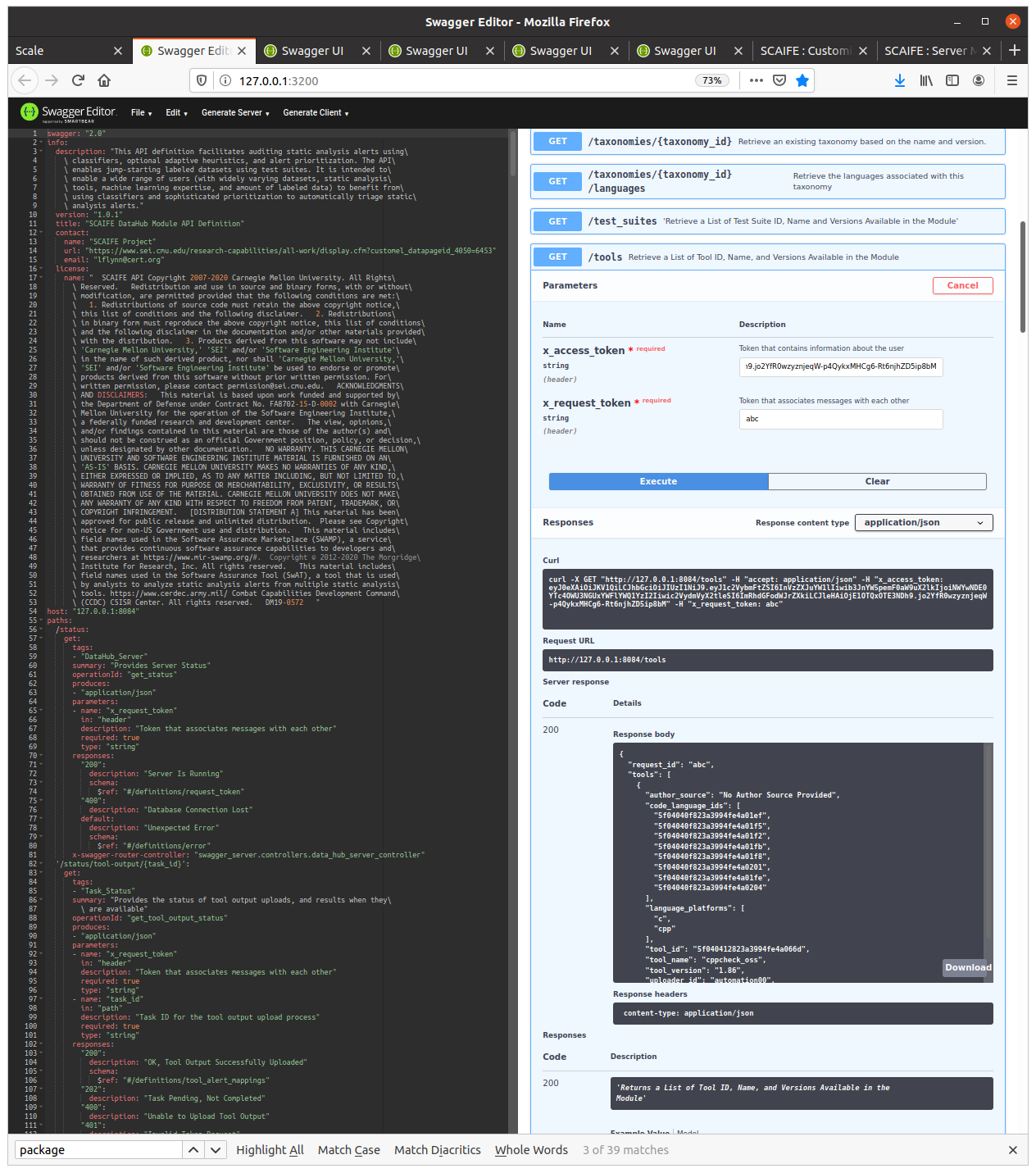


Figure : Successful Execution of Getting Tools from DataHub via the Swagger-Editor GUI

Next, you will upload a cppcheck tool output file to the DataHub.

Use the file scaife/ui\_server\_stub/scale.app/demo/micro\_juliet\_v1.2\_cppcheck\_b/micro\_juliet\_cppcheck\_tool\_output.xml that is included in the code provided to you.

* Navigate to Datahub\_Server POST /tools/{tool\_id}/packages/{package\_id}.
* Click Try it out.
* Enter x\_access\_token and any value for x\_request\_token (e.g., for the latter, “abc”).
* Enter tool\_id and package\_id.
* Select the “Browse” button for raw\_tool\_output file, and browse to select the file scaife/ui\_server\_stub/scale.app/demo/micro\_juliet\_v1.2\_cppcheck\_b/micro\_juliet\_cppcheck\_tool\_output.xml.
* Click Execute and retrieve the tool\_id.

Below is the request to send tool output. (The purple font indicates where substitutions should be made.)

curl -X POST http://127.0.0.1:8084/tools/$tool\_id/packages/$package\_id -H "accept: application/json" -H "x\_access\_token: $access-token" -H "x\_request\_token: $random\_string" -H "Content-Type: multipart/form-data" -F raw\_tool\_output=@$CPPCHECK\_XML\_FILE.xml;type=text/xml

The actual Curl command used for a particular test is shown in the screenshot below.

curl -X POST "http://127.0.0.1:8084/tools/5f040412823a3994fe4a066d/packages/5f04041a823a3994fe4a08f7" -H "accept: application/json" -H "x\_access\_token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InVzZXJuYW1lIiwib3JnYW5pemF0aW9uX2lkIjoiNWYwNDE0YTc4OWU3NGUxYWFlYWQ1YzI2Iiwic2VydmVyX2tleSI6ImRhdGFodWJrZXkiLCJleHAiOjE1OTQxOTE3NDh9.jo2YfR0wzyznjeqW-p4QykxMHCg6-Rt6njhZD5ip8bM" -H "x\_request\_token: abc" -H "Content-Type: multipart/form-data" -F "raw\_tool\_output=@micro\_juliet\_cppcheck\_tool\_output.xml;type=text/xml"

The DataHub returns the following, also shown in the screenshot below:

{

"task\_id": "e12d2d3d-8dc8-4d03-8962-07e3f3c4b68b",

"task\_url": "/status/tool-output/e12d2d3d-8dc8-4d03-8962-07e3f3c4b68b"

}

See the screenshot below after successful tool output upload:

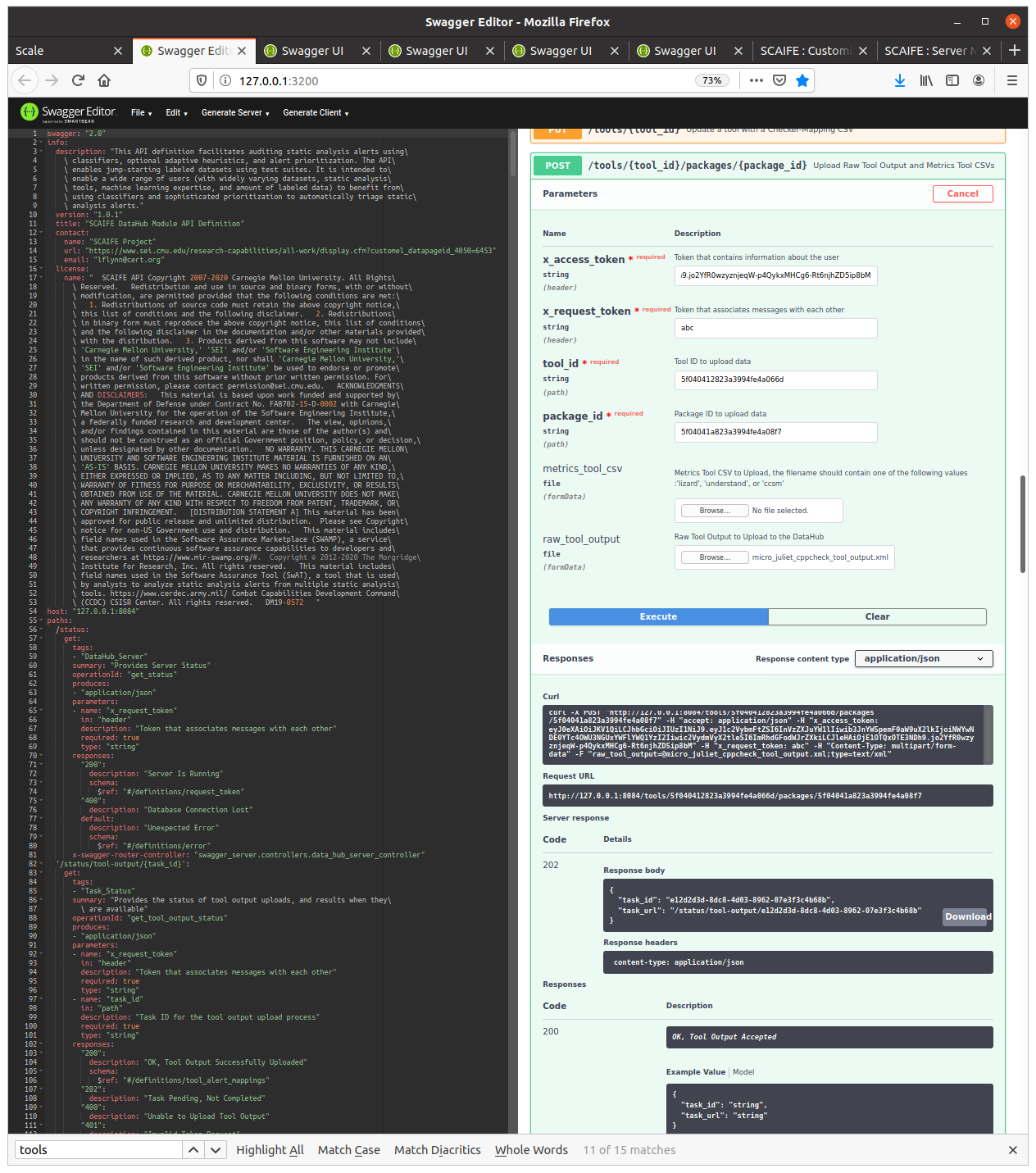


Figure : Successful Tool Output Upload to DataHub via the Swagger-Editor GUI

# Using Auto-Generated Client Code to Implement SCAIFE API Calls *from* Your Tool

The swagger-codegen tool automatically generates code from an Open API formatted YAML or JSON file. swagger-codegen can be used on each of the five SCAIFE swagger.yaml files to generate client code in a wide variety of languages (including C, Python, and Java). We auto-generated Java client code for the latest version of each of the SCAIFE API files.

This client code makes the API call, using the language generated, with the correct parameters and return values set. However, you must do additional work to make the code functional. Set the API call parameters using data values that might initially be hardcoded with test values, and later will be substituted for values from dataflows within your own tool. Automatically inserting reasonable values for the parameters may require coding new logic within your tool. Similarly, the return value(s) from the API call will need to be dealt with by your tool, which will require some development of your tool’s code.

We suggest that the first API call you automate be either 1) registering with the SCAIFE registration server or 2) making the API call to upload raw tool output. Both of these (and other options you may wish to consider) are described above in the swagger-editor example. For implementing the API call from your tool, select where in your tool’s code you will place the generated client code for that API call. Next, modify as needed to hardcode test parameters or to use your tool’s dataflow to select parameter values for your API call. Then, start up the SCAIFE containers as before (same as described at the beginning of this document, possibly using detail in the last section). Next, you test, debug, and complete the implementation of that SCAIFE API call from your tool.

We welcome feedback as you work through this process. We hope to be able to cite your tool integration (although you can be anonymous, if you prefer) as a successful example of SCAIFE integration with a variety of tools.

## Detail for Auto-Generating Code from the SCAIFE APIs Yourself

Information about swagger-codegen (free, downloadable software from GitHub) is here: <https://github.com/swagger-api/swagger-codegen>.

NOTE: The SCAIFE swagger APIs use version 2, so the master branch of swagger-codegen should be used (not the 3.0.0 branch).

The command below was used for Java code generation for the SCAIFE registration module. (Note that others should modify the command below since the swagger.yaml file location is specific to the directory structure of your own machine.)

java -jar modules/swagger-codegen-cli/target/swagger-codegen-cli.jar generate \

-i /home/lflynn/scale/epp/scaife/registration\_server\_stub/swagger\_server/swagger/swagger.yaml \

-l java \

-o /home/lflynn/temp/swagger-java-client

A similar command was used to generate Java code for the other four SCAIFE modules, using their own swagger.yaml filepaths.

Similar commands can be used to auto-generate code in many different languages. Note that “-l java” is the key part of the command that determines the generated code language.

# Using Auto-Generated Server Code to Implement SCAIFE API Calls *into* Your Tool

Similar to swagger’s auto-generation of client code in many languages, swagger-codegen can auto-generate server code in many languages. The auto-generated server code includes stub functions for SCAIFE API function calls into the server. You must develop the internal logic within these functions (turn the function stubs into full functions), and you must enable the function to return the value it should return (if any). Also, as part of coding the internal logic, you need to store whatever state the module is supposed to maintain as a result of that SCAIFE API function being called.

Instructions on how to auto-generate a server stub for various languages and frameworks is provided on this swagger-codegen GitHub page: <https://github.com/swagger-api/swagger-codegen/wiki/Server-stub-generator-HOWTO>.

# Useful Information and Commands for SCAIFE Containers

## How to Stop, Restart, and Check if Containers Are Running During Testing

NOTE: Commands provided here are slightly different than in the HTML manual, since those commands are for a SCAIFE VM distribution with dependent containers (containers that share files with the host machine), but this document describes using independent containers (that do not share files with the host machine) with a code-only distribution. (This document’s instructions are different in that way from instructions provided at scaife/ui\_server\_stub/scale.app/public/doc/scale2/SCAIFE-Server-Management.html.)

| Server | Port | Mongo Server Associated with Column 1’s Swagger Server | Mongo Server’s Port | Redis Server’s Port |
| --- | --- | --- | --- | --- |
| scale | 8083 |  |  |  |
| datahub | 8084 | mongodb\_datahub | 28084 | 28184 |
| priority | 8085 | mongodb\_priority | 28085 |  |
| stats | 8086 | mongodb\_stats | 28086 |  |
| registration | 8087 | mongodb\_registration | 28087 |  |

Figure : Reference Table with Port IDs per Server (Container)

### Command to Test that Swagger Server Is Running (Substitute $PORT for Number in Table Above)

Note that the following command requires the wget package to be installed. (On a machine that uses apt, the following command installs it: sudo apt-get install wget.)

|  |
| --- |
| wget -q -O - localhost:$PORT/status --header="x\_request\_token: deadbeef" |

#### Example Command and Response

scaife$ **wget -q -O - localhost:8084/status --header="x\_request\_token: deadbeef"**

{

"message": "DataHub Server is Running Properly",

"request\_id": "deadbeef"

}

### Commands to Refresh All SCAIFE Containers

|  |
| --- |
| docker-compose down  docker-compose -f docker-compose.yml up --build |

### Commands to Refresh a Single SCAIFE Independent Container (SCALe or Swagger)

CAUTION: In many cases, this will make the other SCAIFE containers have inconsistent states. However, this is sometimes useful when testing only one container (e.g., when only testing the SCALe container without uploading any projects to the datahub container).

|  |
| --- |
| docker-compose stop ${SERVER}  docker-compose rm ${SERVER}  docker-compose -f docker-compose.yml up ${SERVER} --build |

#### Example Commands

docker-compose stop datahub

docker-compose rm datahub

docker-compose -f docker-compose.yml up datahub

### Commands to Pause and Resume All SCAIFE Independent Containers

#### To Pause SCAIFE, Without Resetting Any Data

docker-compose stop

#### Resume SCAIFE from a Pause

docker-compose start

## How to Run Scripts, Tests, and Other Commands in SCAIFE Containers

### Using Docker-Compose to Test the Script-Based Creation of SCALe Projects and Use Them in an Independent Container

NOTE: The first two commands below require all the SCAIFE containers to be active (up).

|  |  |
| --- | --- |
|  | docker exec scale python scripts/automation/create\_manual\_test\_project\_1\_microjuliet\_cppcheck.py  docker exec scale python scripts/automation/create\_manual\_test\_project\_1\_dos2unix\_rosecheckers.py  docker exec scale python scripts/automation/create\_basic\_project.py |

### Command to Get a Bash Command Line for a Container (e.g., and Then Inspect the Container’s Internal Filesystem)

|  |
| --- |
| Access a bash command line on the container $SERVER (e.g., datahub) you specify.  docker exec -it $SERVER /bin/bash |

#### Example Commands

docker exec -it datahub /bin/bash

root@**622cbc872c78:/usr/src/app#** ls

ABOUT api\_html\_formatter.py init.sh requirements.txt swagger\_server.egg-info

Dockerfile api\_json\_and\_html\_generator.py insert\_project.py setup.py test-requirements.txt

README.md git\_push.sh nosetests.xml swagger\_server tox.ini

### How to Read Log Files in SCAIFE Containers

You can access the log of a swagger-server's associated Mongo container.

### Command to Obtain a SCAIFE Mongodb Log

|  |
| --- |
| docker logs mongodb\_${SERVER} |

#### Some Useful log Files

| Container | Path | Purpose |
| --- | --- | --- |
| SCALe | /scale/log/development.log | Rails log |

If $CONTAINER is your container’s name (specified above via the --name flag, and a value you see listed if you run the command docker container ls), and $LOG is the path of a log file in a container, you can copy the log file from container to host with this command:

### Command to Access SCAIFE Log Files

|  |
| --- |
| docker cp $CONTAINER:$LOG . |

You could also watch new output to the log file, without copying it, with this command:

### Command to Monitor Log Files in Containers

|  |
| --- |
| docker exec -it $CONTAINER tail -f $LOG |

### Commands to Connect to SCALe in an Independent Container, Access One of its sqlite3 Databases, then Execute sqlite3 Commands

|  |
| --- |
| # if SCALe is not running, start it  cd scaife  docker-compose -f docker-compose.yml up scale  # Access SCALe  docker exec -it scale /bin/bash  # Inside container  sqlite3 db/development.sqlite  # Next, commands within the sqlite3 command environment (commandline starts with ">")  # List all tables  .tables  # Show everything from the 'projects' table  SELECT \* FROM projects;  .quit  # This line outputs a dump of the SCALe internal database  echo .dump | sqlite3 db/development.sqlite |