How to Review & Test the SCAIFE Beta v2 VM

Contents

This document describes how to review and test the SCAIFE (Source Code Analysis Integrated Framework Environment) beta version 2 virtual machine (VM) release.

Differences from SCAIFE Beta Version 1 Prototype VM

SCAIFE beta version 2 prototype differs from version 1 prototype in the following ways: More of the API-defined functions have been implemented between SCALe (acting as the SCAIFE UI server) and other SCAIFE servers. This includes implemented API calls between SCALe and the Prioritization server (now storing some of the prioritization schemes remotely), between SCALe and the Registration server (logging out of SCAIFE as well as logging in), between SCALe and the Statistics server (SCALe receives and displays classifier options provided by the Statistics server, and allows the user to select options). We have added more adaptive heuristic options (now there are 3) to the Statistics server, for use with the 3 types of classifier type options that the prototype starts with. (However, dataflows have not yet been implemented between SCALe, the DataHub, and the Statistics servers that will be required for the adaptive heuristics to be used.) We also modified the code to make adding classifiers and adaptive heuristics more modular. Additional functionality has been implemented within the servers as well, which will be used after API calls between the SCAIFE servers by forthcoming versions of SCAIFE. Also, we included .YAML files for the SCAIFE API version 0.0.4.

Pre-Requisites

The VM requires space and memory:

The OVA files are ~6.4 GB.
This VM is set to 24GB memory and expand dynamically up to 200GB disk space.
While running in VMWare, this beta VM uses ~40 GB (but we haven't loaded any big test suites or big tool outputs onto this VM. We expect SCAIFE v1 to use a lot more space, closer to 200GB.)

How to Test (and Further Review) the Non-SCALe Servers:

Open a terminal from the icon on the desktop. Then run the tox tests for each server (`tox -e`), ensuring the tests all pass. You can also start each server (by running `python3 -m` swagger_server within each server's respective directory) after running tox tests on them. Note that the Datahub server should be started before running the tox tests on the Statistics server. The following commands can be executed in order to view the tox test outputs and start each server properly:

1. Move to the DataHub server directory, test the server, and then start the server.
   ```
   cd ~/model_api/datahub_server_stub
   tox -e python35
   python3 -m swagger_server --mode test
   ```

2. Move to the Statistics server directory, and test the server.
   ```
   cd ~/model_api/stats_server_stub
   tox -e python35
   ```

3. Move to the Registration server directory, and test the server.
   ```
   cd ~/model_api/registration_server_stub
   tox -e python35
   ```

4. Move to the Priority server directory, test the server, and then start the server.
   ```
   cd ~/model_api/priority_server_stub$
   tox -e python35
   python3 -m swagger_server --mode test
   ```

5. Move to the Statistics server directory, and then start the server.
   ```
   cd ~/model_api/stats_server_stub
   python3 -m swagger_server --mode test
   ```

6. Move to the Registration server directory and start the server.
   ```
   cd ~/model_api/registration_server_stub
   python3 -m swagger_server --mode test
   ```

The names of the tox tests (which scroll down as they are run) are clues to what they test: functionalities that have been implemented and are necessary for the modular alert classification-enabling architecture to work. Your team can examine the code for the tests and for the full servers, as all code we've implemented is included in this distribution as source code. The bottom section of this document provides information about differences between the code and the swagger .yaml files (both code and .yaml files are provided in this VM). The APIs provided in the .yaml can be used by some tools (like swagger-codegen, a tool we use) to automatically generate stubs for the server controller functions in various coding languages. The idea is that any tool developer could use them to integrate their own tools with the SCAIFE system. For those of you who develop tools, your feedback on the API will be especially valuable.

The SCAIFE beta version 0.0.4 architecture as defined by the .yaml files within the 4 SCAIFE server directories can be reviewed. We recommend using an API viewer like the Swagger-Editor[1] or Swagger-

[1] Swagger pinned repositories. GitHub Website. May 23, 2019 [accessed]. github.com/swagger-api

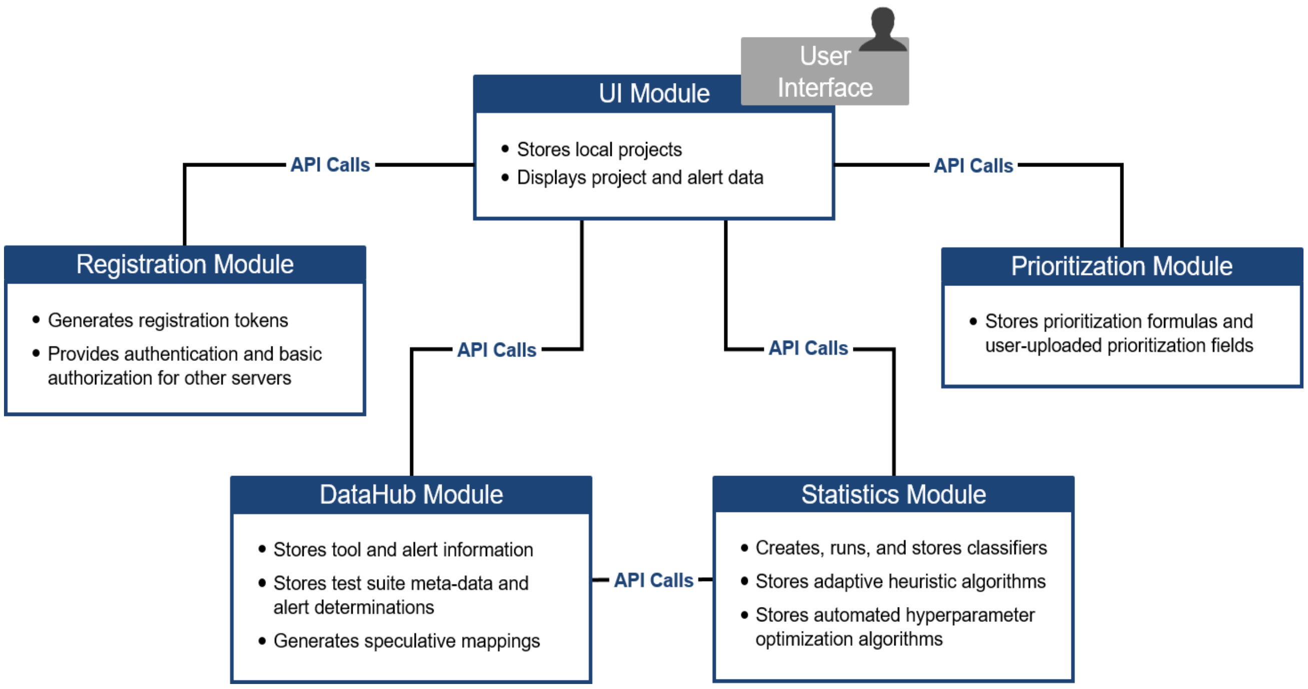
User Interface
UI Module
Stores local projects
Displays project and alert data
API Calls
API Calls
Registration Module
Generates registration tokens
Provides authentication and basic authorization for other servers
Prioritization Module
Stores prioritization formulas and user-uploaded prioritization fields
API Calls
API Calls
DataHub Module
Stores tool and alert information
Stores test suite meta-data and alert determinations
Generates speculative mappings
API Calls
Statistics Module
Creates, runs, and stores classifiers
Stores adaptive heuristic algorithms
Stores automated hyperparameter optimization algorithms

Figure 1: The Five SCAIFE Servers

UI[1]. Note that there are many "TODO"s in the code, many with a comment including "RC-###", which is the ID of a JIRA issue for a development task which we've detailed but haven't yet completed.

How to test and otherwise examine the SCALe server:

Start the SCAIFE registration server (see command #6 above).
Then open the Firefox browser from the icon on the desktop, and go to the URL

```
localhost:8083
```

Use the credentials saved in Firefox (also sent to you by DoD SAFE, in August 2019) to login to SCALe.
Select the blue button 'Connect to SCAIFE' at the top of the page, then select the 'Sign up' button. Enter login credentials you can use to login to SCAIFE. Click the 'Register' button.

Open a new tab in Firefox (select the '+' icon just above the URL bar) and go to the SCALe manual by entering this address: file:///home/vagrant/scale.app/public/doc/index.html

- The section file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html has useful info relevant to the new features added to SCALe for SCAIFE integration. This includes definitions (e.g., for alertCondition, meta-alert, condition, alert, and fused alertConditions) used throughout SCAIFE, how to audit alertConditions, how to filter results, and how to set prioritization schemes, classification schemes, and how users can upload their own fields and use them in prioritization formulas.
- Follow this quick-start demo and create a basic dos2unix demo project: file:///home/vagrant/scale.app/public/doc/scale2/SCALe-Quick-Start-Demo-for-Auditors.html
- Create local and remote classifiers and prioritization schemes. Use instructions in
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#selecting-a-prioritization-scheme
        - Create a local-only, global, and both types of remote prioritization schemes.

    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#selecting-a-classification-scheme
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#running-the-classifier
- Check as follows:
    - In a bash terminal in the VM, view mongo database entries and when done quit viewing them, with the following commands:
      ```
      mongo
      use registration_db
      db.user.find().pretty()
      use prioritization_db
      db.prioritization_scheme.find().pretty()
      use stats_db
      db.classifier.find().pretty()
      quit()
      ```
    - Note that:
        - The global scheme is saved in the prioritization server's database, but the remote schemes are not yet saved there (that is implemented in a SCAIFE release to be published in October)
        - The classifier types are stored in the stats server's database, but not the data for the classifier specified by the SCALe user (with project data to use plus whether to use adaptive heuristic and AHPOs). Storing that in the Statistics server's database will be implemented in a later SCAIFE release.
    - In a bash terminal in the VM, view SCALe sqlite3 database entries and when done quit viewing them, with the following commands:
      ```
      cd /home/vagrant/scale.app/db
      cp development.sqlite development2.sqlite3
      sqlite3 development2.sqlite3
      select * from priority_schemes;
      select * from classifier_schemes;
      .quit
      ```
- Disconnect from SCAIFE either by selecting the blue 'Disconnect from SCAIFE' button in the black header bar, or else select SCAIFE mode: 'Demo' (see middle button, just above the list of alertConditions).
- Randomly make primary and secondary audit determinations and notes and flags, on many alert conditions. Change some of them, and notice that a count is kept of determination changes (the database actually tracks each of the previous determinations, along with times of changes).
- Next, in 'Demo' mode, create a classifier and a prioritization scheme, and run both. Use instructions in
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#selecting-a-prioritization-scheme
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#uploading-additional-fields
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#selecting-a-classification-scheme
    - file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#running-the-classifier

- Next: edit, export (to a SQLite database), sanitize, and at the end delete the dos2unix project, using instructions in file:///home/vagrant/scale.app/public/doc/scale2/The-SCALe-Web-App.html#editing-exporting-sanitizing-or-deleting-an-existing-project
- The Sanitizer section file:///home/vagrant/scale.app/public/doc/scale2/Sanitizer.html explains how sanitization is done on exported SCALe databases and also how the sanitizer is updated as SCALe code evolves.

The Five Server Types in the SCAIFE Architecture and this VM

This beta SCAIFE release VM includes all 5 types of servers (a.k.a. modules) in the SCAIFE architecture:

1. DataHub server: stores SCAIFE packages (codebase, flaw-finding static analysis (FFSA) tools' alert data, code metrics tool data, data about codebase functions and files, test suite data including NIST SARD-style manifest, and SCAIFE projects (an auditing construct with manually labeled meta-alerts for non-test-suite data and automatically-labeled meta-alerts for test suite data) It has functions that upload test suite codebases and metadata, then automatically label test suite meta-alerts.
2. Statistics server: This server stores types of classifiers, adaptive heuristics, and automated hyper-parameter optimization options that users can select, along with default values to enable users without statistics or AI backgrounds to quickly start using classifiers. This server creates classifiers as specified by the UI module and runs a classifier on a project and returns a set of confidence values (confidence that it is a true positive code flaw) for meta-alerts that haven't been manually adjudicated. It uses both test suite data and non-test suite data. When the UI module specifies, an adaptive heuristic reweights the classifier confidence values as new relevant data comes in to the SCAIFE Datahub server, and under certain conditions the classifier is recreated from the larger set of relevant data and run again on the project's non-adjudicated meta-alerts.
   a. implemented: classification, adaptive heuristic, API communications between other SCAIFE servers and the Statistics server
   b. not implemented: API calls between SCALe and the Statistics server
3. UI (user interface) server: We use SCALe here, with new features and bugfixes required for use in SCAIFE. (Other tools could be used in the architecture instead of SCALe.) From SCALe, the SCALe user can create a SCAIFE registration account and login to the SCAIFE registration server. Some of the SCALe API interactions with the other SCAIFE servers are not yet implemented. Fields and modals for making classifier and prioritization scheme selections and seeing results have been implemented in the GUI and in the databases on the SCALe back-end, and SCALe interacts via API calls with the Prioritization, Statistics, and Registration servers.
4. Prioritization server: This server stores prioritization schemes, that can include classifier confidence values and other fields in a mathematical formula that can be used to prioritize meta-alerts according to an organization's weighted priorities. Prioritization calculation is done locally at the UI module, using the prioritization formula selected/created.
5. Registration server: This server is used to register users in the SCAIFE system. Users register and login through this server to receive an access token that allows the users to communicate securely with the Statistics, DataHub and Prioritization servers

Differences between the code and the Swagger YAML files in this VM

The registration and prioritization servers are fully implemented in accordance with their respective swagger YAMLs.

Information about the difference between the swagger YAML and the Python code will take the form:

- **API call**; code method name, description and/or additional steps to complete.

DataHub Module:

Partially implemented:

- **PUT /projects/{project_id}**; edit_project, used by the UI module to update and existing package.

Not yet implemented:

- * **POST /projects/{project_id}/classifier/hold**; hold_data_for_classifier, a new method used by the Stats module to place adaptive heuristic data at the DataHub on hold, usually when a UI module becomes unresponsive.
- * **PUT /projects/{project_id}/classifier/hold**; release_data_hold_for_classifier, a new method used by the Stats module to release the data hold at the DataHub on the adaptive heuristic data.
- **POST /taxonomies**; create_taxonomy, used by the UI module to create a new taxonomy object in the DataHub. The method provides taxonomy information, including conditions and languages for each condition in the taxonomy.
- **PUT /tools/{tool_id}**; edit_tool, used by the UI module to add new checker mappings for a tool object in the DataHub.

* Methods are placeholders for internal functionality that may be replaced by the implementation of another server-client model.