Creating a CSIT Test

Overview

ONAP CSIT (Continuous System and Integration Testing) tests are defined as what we call "test plans". A test plan is meant to test a specific feature or functionality comprehensively, and each test plan runs one or more test suites that are written using the Robot Framework. A particular test suite (e.g. to test a core module) can be included by multiple test plans.

Test plans and test suites are source controlled in the integration repo, under its own GIT repository located in integration/csit.git (https://gerrit.onap.org/r/gitweb?p=integration/csit.git;a=summary). To get started and follow along, please clone the integration repo to your local environment. If you want to run the CSIT test suites locally, you will also need to have have docker and python installed.

The key contents of integration/test/csit/ are:

- run-csit.sh: the shell script that executes a particular CSIT test suite
- plans/: contains the definitions of what is invoked by each test plan
- tests/: contains the test suites written using the Robot Framework
- scripts/: contains various shared shell scripts that support the test plans

Video Tutorial

System Prerequisites

Make sure that your environment has the following packages installed:

```
sudo apt install python-pip virtualenv unzip sshuttle netcat libffi-dev libssl-dev docker-compose
sudo pip install robotframework
sudo pip install -U requests
sudo pip install -U robotframework-requests
sudo pip install -U robotframework-httplibrary
```

Setting Up the Test

Test plans are defined under the plans/ directory.

The directory layout under the plans/ directory is <project>/<functionality>/. A basic sample CSIT test plan is provided in plans/integration/functionality1/. To execute it, run the command:

```
./run-csit.sh plans/integration/functionality1/
```

The run-csit.sh will automatically set up a Robot environment for you, and execute the test plan defined in the plans/integration/functionality1/ directory.

If you look at the contents of the plans/integration/functionality1/ you will see the following:

- setup.sh: the shell script that starts all the necessary docker containers required by this test plan, as well as passing the necessary environment variables to Robot.
- testplan.txt: the text file listing, in order, the test suites that should be executed by Robot for this particular test plan. This allows you to refactor test suites to be reused by multiple test plans as necessary.
- teardown.sh: the shell script that kills all the docker containers that were started by this test plan.

As an example, the sample plans/integration/functionality1/testplan.txt contains the following:

```
# Test suites are relative paths under [integration.git]/test/csit/tests/.
# Place the suites in run order.
integration/suite1
integration/suite2
```

This means that this test plan will run those two Robot test suites in the sequence specified above.

When run-csit.sh runs a test plan, it will first execute setup.sh to start any docker containers and set up environment variables. Then, it will run the test suites listed in the testplan.txt in sequence. Finally, it will run teardown.sh to terminate all the docker containers.

Defining Test Suites

The Robot test suites are placed under the tests/ directory.

The directory layout under the tests/ directory is <project>/<suite>/. So, the sample test suites can be found in tests/integration/suite1/ and tests/integration/suite2/, respectively. The test suites are written using the Robot Framework.

We won't go into details here about how to write Robot test suites. You can take a look at the sample test suites in tests/integration/suite1/ and tests/integration/suite2/ to see some sample test cases implemented; for example, it demonstrates the retrieval of a URL to test for its return code.

Refactor Shared Shell Scripts

The scripts/ directory is where you place any shell scripts that may be useful for multiple test plans. For example, the run-instance.sh and get-instance-ip.sh shell scripts are placed here so that they can be used by all test plans. Feel free to place any commonly useful shell scripts that you need across multiple test plans here.

Setting Up a Jenkins CSIT Job

Once you have your test plan created, ensure that it can be run successfully locally with run-csit.sh, and then check it into the integration repo.

After this, it is time to set up a Jenkins CSIT Job so that your test plan will be automatically run by Jenkins daily.

The CSIT job defined for the sample test plan above is at https://jenkins.onap.org/view/CSIT/job/integration-master-csit-functionality1/.

Please refer to Jenkins -> Configuring Jenkins Jobs on the background information on setting up Jenkins jobs. Roughly speaking, you need to check in a JJB YAML definition for your Jenkins job into the ci-management repo under ci-management/jjb/.

The easiest thing to do is to copy and modify the sample JJB definition of the integration-csit-functionality1 job, found at ci-management/jjb/integration/integration-csit.yaml, which contains the following:

integration-csit.yaml

```
---
- project:
    name: integration-csit
    jobs:
      - '{project-name}-{stream}-verify-csit-{functionality}'
      - '{project-name}-{stream}-csit-{functionality}'
    project-name: 'integration'
    stream: 'master'
    functionality:
      - 'functionality1':
          trigger_jobs:
    robot-options: ''
    branch: 'master'
```

And update the project name (line 7) and functionality name (line 10) as appropriate.

The entries under trigger_jobs lists the jobs whose completion will automatically trigger the execution of this particular CSIT job. You should change this list to include the list of merge jobs that may potentially affect the CSIT test results of this particular test plan.