OVSDB:CSIT

Introduction

Continuous System Integration Test (CSIT) area has been created to develop automated system test executed at distribution build time.

This page will presents the CSIT implemented for OVSDB project. It will also explain how they have been implemented, and how they can be tested either locally (using Vagrant) or remotely (in the Sandbox).

General information related to system test creation can be find here CrossProject:Integration_Group:System_Test:Step_by_Step_Guide#System_Test_Job , it is a step by step guide to implement system test.

Test Descriptions

Job Title	Notes
ovsdb-csit-1node-netvirt-all-beryllium	All features are loaded.[1]
ovsdb-csit-1node-netvirt-stable-lithium	All features are loaded.[1]
ovsdb-csit-1node-netvirt-only-beryllium	Only odl-ovsdb-openstack is loaded.[1]
ovsdb-csit-1node-netvirt-only-stable-lithium	Only odl-ovsdb-openstack is loaded.[1]
ovsdb-csit-verify-1node-netvirt	
ovsdb-csit-1node-periodic-scalability-daily-only-master	Load odl-ovsdb-openstack.[2]
ovsdb-csit-1node-periodic-scalability-daily-only-stable-lithium	Load odl-ovsdb-openstack.[2]
ovsdb-csit-1node-southbound-all-beryllium	All features are loaded.[1]
ovsdb-csit-1node-southbound-all-stable-lithium	All features are loaded.[1]
ovsdb-csit-1node-southbound-only-beryllium	Only odl-ovsdb-openstack is loaded.[1]
ovsdb-csit-1node-southbound-only-stable-lithium	Only odl-ovsdb-openstack is loaded.[1]
ovsdb-csit-verify-1node-southbound	[1]

ovsdb-daily-beryllium	
ovsdb-daily-stable-lithium	
ovsdb-daily-clustering-netvirt-lithium	
ovsdb-daily-clustering-netvirt-master	
ovsdb-sonar	Scheduled randomly once a day. Runs all integration tests so all ovsdb features are covered.
ovsdb-daily-full-integration-beryllium	Runs all integration tests against multiple OVS versions.[2]
ovsdb-daily-full-integration-stable-lithium	Runs all integration tests against multiple OVS versions.[2]
ovsdb-daily-openstack-beryllium	Runs odl-ovsdb-openstack against tempest tests.[2]
ovsdb-daily-neutron-yang-migration	Used for neutron yang migration. Need to deprecate since no longer used.
ovsdb-distribution-beryllium	
ovsdb-integration-beryllium	
ovsdb-openstack-gerrit	Triggered from upstream openstack gerrits to run tempest tests against odl-ovsdb-openstack.

[1]Triggered when changes are made to controller, openflowjava, openflowplugin, ovsdb and yangtools.
[2]Scheduled randomly once a day.

Test Execution

We use Jenkins to trigger and execute the system test.

System Test runs continuously in Linux Foundation Lab

Test Work Flow

- Setup System Test Environment
- Download System Test Code
- Write Automation in Robot Framework
- Upload Test files to GIT repo using this instruction

Prerequisites

- Java

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

- Python

https://www.python.org/downloads/

- Robot framework

https://github.com/robotframework/robotframework#installation
You can also install some Robot tools for instance:
- the Eclipse plugin: https://github.com/NitorCreations/RobotFramework-EclipseIDE/wiki
- the IntelliJ pluggin: http://plugins.jetbrains.com/plugin/7430

- ODL Integration project

```
odluser@odl-vm:~\> git clone https://git.opendaylight.org/gerrit/integration
```

- ODL RelEng Builder project

```
odluser@odl-vm:~\> git clone https://git.opendaylight.org/gerrit/releng/builder
```

OVSDB functionalities

netvirt

Create a test folder in Integration project

Navigate to system test suites folder:

```
odluser@odl-vm:~\> cd integration/test/csit/suites
```

System test suites are organized in 2 level folder structure:

- First level folder is the project folder, here you can see projects in OpenDaylight.
- Second level folder is the robot suites folder to verify specific project functionality.

As there is already a folder for OVSDB project, get into this folder. There, create the folder for the functionality you want to test: in our case Openstack_Neutron folder has been created.

```
odluser@odl-vm:~\> mkdir integration/test/csit/suites/OVSDB/Openstack_Neutron
```

This folder will contain all the test cases dealing with the functionality under testing.

Create Robot test cases

Now is the time when we want to write Robot test cases.
At this time, two files were created:

- 001__connection_manager.robot
- 010__ovsdb_flow.robot

001__connection_manager.robot: This robot file defines, as the name can let you guess, test cases regarding the connection between OVS and ODL through the netvirt bundle. *odl-ovsdb-openstack* provides OVSDB netvirt bundle, and also gathered Neutron Northbound bundles, in order to have end to end functionality.

Let's talk about the tests implemented here:
First of all, we clean the OVS instance to make sure nothing is present in the OVSDB.
Then we set the manager using:

```
mininet@mininet-vm:~\> sudo ovs-vsctl set-manager tcp:192.168.1.107:6640
```

Once connected, the OVSDB netvirt bundle pushes flows to the OVS instance, creating a basic topology, containing a bridge, a port, and an interface and set up the pipeline.
The created topology can be seen using:

```
mininet@mininet-vm:~\> sudo ovs-vsctl show
1d31bf6b-ec1e-4f6c-958d-23a2b36892db
    Manager "tcp:192.168.1.107:6640"
        is_connected: true
    Bridge br-int
        Controller "tcp:192.168.1.107:6653"
            is_connected: true
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
    ovs_version: "2.3.1"
```

Finally we also verify the pipeline in OVS is created:

```
mininet@mininet-vm:~\> sudo ovs-ofctl -O OpenFlow13 dump-flows br-int
OFPST_FLOW reply (OF1.3) (xid=0x2):
 table=0, n_packets=0, n_bytes=0, priority=0 actions=goto_table:20
 table=0, n_packets=0, n_bytes=0, dl_type=0x88cc actions=CONTROLLER:65535
 table=20, n_packets=0, n_bytes=0, priority=0 actions=goto_table:30
 table=30, n_packets=0, n_bytes=0, priority=0 actions=goto_table:40
 table=40, n_packets=0, n_bytes=0, priority=0 actions=goto_table:50
 table=50, n_packets=0, n_bytes=0, priority=0 actions=goto_table:60
 table=60, n_packets=0, n_bytes=0, priority=0 actions=goto_table:70
 table=70, n_packets=0, n_bytes=0, priority=0 actions=goto_table:80
 table=80, n_packets=0, n_bytes=0, priority=0 actions=goto_table:90
 table=90, n_packets=0, n_bytes=0, priority=0 actions=goto_table:100
 table=100, n_packets=0, n_bytes=0, priority=0 actions=goto_table:110
 table=110, n_packets=0, n_bytes=0, priority=0 actions=drop
```

To sum up, this first robot file make sure the topology is created, the pipeline is present and the Manager/Controller has the flag *is_connected: true*.

010__ovsdb_flow.robot: This robot file creates a much more sophisticated topology. Basically, it replicates what has been implemented in this script: createFloat.sh
This script make REST queries. So the robot test cases implement those REST queries and analyze their return HTTP status code. If the HTTP status code is compliant with what is expected, the test PASS, else it FAIL.

Create a test plan in Integration project

Basically a test plan is a .txt file containing the path to the folder created in the step before.

It is declared as follow: $yourproject-$functionality.txt

Navigate to system test plan fodler:

```
odluser@odl-vm:~\> cd integration/test/csit/testplans/
```

Create the test plan file

```
odluser@odl-vm:~\> vim test/csit/testplans/ovsdb-netvirt.txt
```

At the end, the test plan file should look like:

```
# Place the suites in run order:
integration/test/csit/suites/ovsdb/Openstack_Neutron
```

Create the CSIT JJB file in the RelEng Builder project

Copy the integration System Test (CSIT) JJB file to your project folder chaging the project and the functionality under test in the filename.

```
odluser@odl-vm:~\> cp jjb/integration/integration-csit-basic.yaml jjb/ovsdb/ovsdb-csit-netvirt.yaml
```

Then we must edit this file to be specific for the OVSDB netvirt functionality:

```
odluser@odl-vm:~\> vim jjb/ovsdb/ovsdb-csit-netvirt.yaml
```

At the end the file looks like:

```
- project:
    name: ovsdb-csit-netvirt
    jobs:
        - '{project}-csit-1node-cds-{functionality}-{install}-{stream}'
        - '{project}-csit-verify-{functionality}-{stream}'

    # The project name
    project: 'ovsdb'

    # The functionality under test
    functionality: 'netvirt'

    # Project branches
    stream:
        - master:
            branch: 'master'
        - stable-lithium:
            branch: 'stable/lithium'

    install:
        - only:
            scope: 'only'
        - all:
            scope: 'all'

    # Mininet configuration
    mininet-image: 'rk-c-el6-mininet'
    mininet-vms: 2

    # Features to install
    install-features: 'odl-ovsdb-openstack'

    # Robot custom options
    robot-options: ''

    # Trigger jobs (upstream dependencies)
    trigger-jobs: 'yangtools-distribution-{stream},controller-distribution-{stream},openflowjava-distribution-
{stream},openflowplugin-distribution-{stream},ovsdb-distribution-{stream}'
```

The following is what this file declared:
- project name: ovsdb-csit-netvirt
- project: ovsdb
- functionality: netvirt
- stream: List the project branches you are going to generate system test
- You can set the mininet VMs you need and the mininet image, here we're using 2 VMs rk-c-el6-mininet (old mininet with ovs 2.0)
- feature-install: with features you want to install in controller separated by comma, here we only want to install odl-ovsdb-openstack
- robot-options: any robot option that need to be specify, here we don't have any.
- trigger-jobs: all the dependencies of the project.

southbound

TBD

How to test

netvirt

Locally

All you need is define in the following repository: https://github.com/opendaylight/ovsdb/tree/master/resources/robot

Once the repo is cloned, all you have to do is:

```
odluser@odl-vm:~\> robot/vagrant up
```

This will bring up 2 OVS instances ready to run Robot tests.

Purpose

1) Creates VMs running CentOS 7.0 x64 with OpenVSwitch. Please use the Vagrant environment variable OVS_NODES to set the number of VMs that would be created. Default value is 2 (ovs1 and ovs2).

2) Sets up Robot framework in the first VM (ovs1). Subsequent VMs are will only have OVS

About the included OVS rpm

To improve provisioning time, "openvswitch-2.3.1-1.x86_64.rpm" is pulled from dropbox. You can add rpm files for other OVS version if desired. Default ovs version is 2.3.1.

To build ovs for the VMs from source, open the vagrant file and make changes to :

Line 19: ovsversion = "" Line 50: puppet.manifest_file = "ovsnode_build.pp"

Running integration tests for OVSDB netvirt

After the VMs are provisioned. ssh into ovs1 to run integration tests for OVSDB

```
odluser@odl-vm:~\> vagrant ssh ovs1
vagrant@ovs1:~\> sh run_robot_tests.sh
```

OpenDaylight Controller

The controller should be running on the host machine before you run the integration tests. The VMs are setup with environmental variable $CONTROLLER with the default IP: 192.168.100.1

Output and log from each test

The output and logs for each test will be left in ovs1 home directory. For convinience of accessing the test results at a later time from the host machine, check "robot/scripts/results" for the result of the current and previous tests. Those are timestamped and cumulated over time. You are responsible for cleaning up this cache.

To run specific patches

This script will automatically download the latest version of the master branch of the integration project. If you need to test a specific patch, open run_robot_tests.sh on the home directory of ovs1

```
vagrant@ovs1:~\> vim run_robot_tests.sh
```

Edit the git clone url as desired. For instance, instead of

```
...
else
   echo "downloading integration..."
   sudo git clone https://git.opendaylight.org/gerrit/integration
fi
...
```

You could have

```
...
else
   echo "downloading patch..."
   sudo git clone https://git.opendaylight.org/gerrit/#/c/{PATCH_ID}/{PATCH_SET_#}
fi
...
```

To run integration tests for other projects

If this is temporary, edit line 17 of run_robot_tests.sh in the home directory of ovs1.

```
vagrant@ovs1:~\> export test_suite_dir="$HOME/integration/test/csit/suites/ovsdb/"
```

For a permanent change, make the edits described above in the version of this file in robot/scripts and re-provision your VM.

Remotely

In order to test remotely we can use the sandbox, as described here: RelEng/Builder/Jenkins#Using_the_Sandbox.

Please refer to this page to setup properly your environment RelEng/Builder/Jenkins#Jenkins_Sandbox.

southbound

TBD

References

CrossProject:Integration_Group:System_Test:Step_by_Step_Guide
CrossProject:Integration_Group:Using_Robot_Framework
CrossProject:Integration_Group:CSIT
CrossProject:Integration_Group:Download_and_Run_System_Test
CrossProject:Integration_Group:Create_System_Test_Environment#Mininet_Environment_Adjustment
RelEng/Builder/Jenkins#Jenkins_Sandbox

Category: OVSDB