OCP Plugin: Boron: System Test Report

Contents

Feature name

The OCP Plugin project has two top level karaf features, **odl-ocpplugin-all** and **odl-ocpjava-all**, which contain the following sub-features

- odl-ocpplugin-southbound
- odl-ocpplugin-app-ocp-service
- odl-ocpjava-protocol

Description

The OCP service (odl-ocpplugin-app-ocp-service), together with the OCP southbound (odl-ocpplugin-southbound) and OCP protocol library (odl-ocpjava-protocol), provides the ODL controller with basic OCP v4.1.1 functionality:

- OCP-capable radio heads connection
- OCP inventory: radio heads
- OCP elementary functions: device management, config management, object lifecycle, object state management, fault management
- OCP indication message processing

For more information please visit main wiki page OCP_Plugin:Main

Enabling The Feature

Make sure the following prerequisite features are installed beforehand.

```
feature:install odl-restconf odl-l2switch-switch
```

Then install the odl-ocpplugin-all feature which includes the odl-ocpplugin-southbound and odl-ocpplugin-app-ocp-service features. Note that the odl-ocpjava-all feature will be installed automatically as the odl-ocpplugin-southbound feature is dependent on the odl-ocpjava-protocol feature.

```
feature:install odl-ocpplugin-all
```

After all required features are installed, use following command from karaf console to check and make sure features are correctly installed and initialized.

```
feature:list | grep ocp
```

Using The Feature

Currently there are two ways to interact with OCP service: one is via RESTCONF (programmatic) and the other is using DLUX web interface (manual).

Information on how to use the feature is available in OCP_Plugin:Main#Installation

Incompatibilities

OCP service only interacts with OCP-capable radio heads and therefore it does not show incompatibilities with other plugins.

Feature Pro-activeness

OCP Plugin uses TCP port 1033, on which it listens for connection requests from radio heads. The connection established between the radio head and controller (ocpplugin) is for transmission of OCP request/response/indication messages.

Based on the OCP specification, OCP service will perform the alignment procedure against a newly connected radio head upon connection establishment. After that, with the exception of periodic health check, OCP service does not send any request to the radio head unless you program it through NBI.

How to test

There is an OCP service system test suite running in CI:

- https://jenkins.opendaylight.org/releng/view/CSIT-Jobs/job/ocpplugin-csit-1node-get-only-boron/
- https://jenkins.opendaylight.org/releng/view/CSIT-Jobs/job/ocpplugin-csit-1node-get-all-boron/
- https://jenkins.opendaylight.org/releng/view/ocpplugin/job/ocpplugin-csit-verify-1node-get/

The test brings a number of OCP agents representing fake radio heads, using the simple OCP agent that can be found in the ocpplugin repository, and verifies:

- Radio head connectivity
- Access to radio head configuration
- Scalability
- Clustering support

Test Case Description	Pre-conditions or Pre-requisites	Test Procedure	Expected Results
Radio head connectivity	Fresh installation of the controller, then • Install prerequisite features • Install odl-ocpjava-all • Install odl-ocpplugin-all	• Launch and connect a fake radio head to the controller • View the radio head's corresponding inventory node via REST API • Verify REST API succeeded	• GET http://<controller_ip_addr>:8181/restconf/operational/opendaylight-inventory:nodes/node/ocp:TST-1 REST response should succeed with status of 200 OK
Access to radio head configuration	Fresh installation of the controller, then • Install prerequisite features • Install odl-ocpjava-all • Install odl-ocpplugin-all	• Launch and connect a fake radio head to the controller • Read RE:0 object from the radio head's resource model via REST API • Verify REST API succeeded	• POST http://<controller_ip_addr>:8181/restconf/operations/ocp-service:get-param-nb REST response should succeed with status of 200 OK and result code of SUCCESS
Scalability	Fresh installation of the controller, then • Install prerequisite features • Install odl-ocpjava-all • Install odl-ocpplugin-all	• Launch and connect 200 fake radio heads to the controller • View the last radio head's corresponding inventory node via REST API • Verify REST API succeeded • Read the last radio head's resource model via REST API • Verify REST API succeeded	• GET http://<controller_ip_addr>:8181/restconf/operational/opendaylight-inventory:nodes/node/ocp:TST-200 REST response should succeed with status of 200 OK • GET http://<controller_ip_addr>:8181/restconf/config/ocp-resourcemodel:resourceModel/RadioHead/ocp:TST-200 REST response should succeed with status of 200 OK
Clustering support	Fresh installation of the 3-node clustered controller, then • Install prerequisite features • Install odl-ocpjava-all • Install odl-ocpplugin-all	• Launch and connect a fake radio head (radio head #1) to the master node of the controller • Launch and connect a fake radio head (radio head #2) to the first slave node of the controller • Launch and connect a fake radio head (radio head #3) to the second slave node of the controller • Read RE:0 object from the resource model of radio head #1 via REST API • Verify REST API succeeded • Read RE:0 object from the resource model of radio head #2 via REST API • Verify REST API succeeded • Read RE:0 object from the resource model of radio head #3 via REST API • Verify REST API succeeded	• All POST http://<controller_ip_addr>:8181/restconf/operations/ocp-service:get-param-nb REST responses should succeed with status of 200 OK and result code of SUCCESS

Performance/Scalability Concerns

Performance and scalability have been taken into account when designing OCP Plugin, and we address them by leveraging the software architecture of OpenFlow Plugin.

Additional Requirements To Meet Test Requirements Of A Boron Stable Feature

stable feature definition

- Demonstrate (preferably via CSIT robot tests):
  - that the feature works in conjunction with a 3-node cluster using the clustered data store
  - that the feature functions appropriately over a significant duration (days/weeks)
- CSIT jobs should:
  - Have a 100% pass rate. If not, then clearly document all failures and their associated unresolved bug and explain why it will not be resolved for the release
  - not show any unexplained regressions or failures.

Scale and performance limits will need to be available for your feature at the time of release. If there are limits already available, or CSIT jobs tracking these numbers, include them here. Some examples would be:

- openflowplugin can support 200 connected switches
- aaa can validate 10k tokens per second