CONFIDENTIAL

Test SAMBHA 16 vOLT POD

Edition 1.0
Date: March 2023
Filename: Test SAMBHA 16 vOLT POD.pdf

Issued by: R&D DigitalPlatforms
Checked by:
Approved by: DigitalPlatforms R&D Manager

Edition	Edit	Date	Decorr.	Approv.
1.0	First document	March 2023	March 2023	March 2023

List of revisions

Ed.	Date	Editors	Short description
1.0	March 2023		This document describes the ONF tests performed with our SAMBHA 16 vOLT, in order to obatin the ONF certification.

Test SAMBHA 16 vOLT POD

SAMBHA 16 vOLT POD

Selta
DigitalPlatforms Group's BU
Via Emilia 231
29010 Cadeo (PC)
ITALIA
Telefono: +39.0523.5016.1
Web Site: www.selta.com

Edition 1.0 – March 2023

Contents

1 INTRODUCTION

This document describes the working principles of the VOLTHA system tests with our virtual Optical Line Terminal (vOLT), named SAMBHA 16 vOLT. Firstly, we provide some preliminary details of the VOLTHA system tests and some information about our physical POD, and finally, we introduce the results of the tests ran on our physical POD. Tests were performed with ATT workflow and with TIM workflow.

1.1 VOLTHA SYSTEM TESTS

The VOLTHA system tests are written in Robot Framework and Python, and instructions on how to install prerequisites and to set up the test environment can be found in the following link:

https://docs.voltha.org/master/voltha-system-tests/README.html

It is possible to run tests with the BroadBand Simulator (BBSim), or with a physical POD. In our case, since we have a physical POD running in our laboratory with all the requested hardware and connections, we ran the tests with our physical POD.

1.2 INFORMATION ABOUT OUR PHYSICAL POD

Our physical POD is composed by ONOS v2.5.8 and VOLTHA v2.10 running in a kubernetes cluster on a physical server, an aggregation switch (AGG), our SAMBHA vOLT with 16 PON ports, an Optical Network Unit (ONU), and a Residential Gateway (RG).

Here we list the versions of our kubernetes cluster's components.

openolt@openolt-voltha:~$ kubectl get pods --all-namespaces -o jsonpath="{.items[*].spec.containers[*].image}" | tr -s '[[:space:]]' '\n' | sort | uniq -c

1 atomix/atomix:3.1.12
1 docker.elastic.co/elasticsearch/elasticsearch:7.10.1
1 docker.elastic.co/kibana/kibana:7.10.1
1 docker.io/bitnami/etcd:3.5.2-debian-10-r52
1 docker.io/bitnami/kafka:2.8.1-debian-10-r73
1 docker.io/bitnami/zookeeper:3.7.0-debian-10-r215
1 freeradius/freeradius-server:3.0.21
1 gcr.io/k8s-minikube/storage-provisioner:v4
1 jaegertracing/all-in-one:1.18
1 k8s.gcr.io/coredns:1.7.0
1 k8s.gcr.io/etcd:3.4.13-0
1 k8s.gcr.io/kube-apiserver:v1.20.2
1 k8s.gcr.io/kube-controller-manager:v1.20.2
1 k8s.gcr.io/kube-proxy:v1.20.2
1 k8s.gcr.io/kube-scheduler:v1.20.2
1 nginx:1.14.2

1 opencord/onos-classic-helm-utils:0.1.0
1 quay.io/fluentd_elasticsearch/fluentd:v3.0.4
1 voltha/voltha-ofagent-go:2.1.1
1 voltha/voltha-onos:5.1.1
1 voltha/voltha-openolt-adapter:4.2.2
1 voltha/voltha-openonu-adapter-go:2.2.2
1 voltha/voltha-rw-core:3.1.0

openolt@openolt-voltha:~$ kubectl get pods --all-namespaces -o jsonpath="{.items[*].spec.containers[*].image}" | tr -s '[[:space:]]' '\n' | sort | uniq -c

coredns-74ff55c5b-9r9n9: k8s.gcr.io/coredns:1.7.0,
elasticsearch-master-0: docker.elastic.co/elasticsearch/elasticsearch:7.10.1,
etcd-openolt-voltha: k8s.gcr.io/etcd:3.4.13-0,
kube-apiserver-openolt-voltha: k8s.gcr.io/kube-apiserver:v1.20.2,
kube-controller-manager-openolt-voltha: k8s.gcr.io/kube-controller-manager:v1.20.2,
kube-proxy-m6mwj: k8s.gcr.io/kube-proxy:v1.20.2,
kube-scheduler-openolt-voltha: k8s.gcr.io/kube-scheduler:v1.20.2,
nginx-deployment-748fbcbd75-b6q4b: nginx:1.14.2,
storage-provisioner: gcr.io/k8s-minikube/storage-provisioner:v4,
voltha1-voltha-adapter-openolt-79884847cb-f6dbl: voltha/voltha-openolt-adapter:4.2.2,
voltha1-voltha-adapter-openonu-7b99d7dd45-5c4d9: voltha/voltha-openonu-adapter-go:2.2.2,
voltha1-voltha-ofagent-6cc8f7fc6f-qsx8h: voltha/voltha-ofagent-go:2.1.1,
voltha1-voltha-rw-core-fbcc86f7b-cqsv7: voltha/voltha-rw-core:3.1.0,
voltha-infra-atomix-0: atomix/atomix:3.1.12,
voltha-infra-etcd-0: docker.io/bitnami/etcd:3.5.2-debian-10-r52,
voltha-infra-fluentd-elasticsearch-x8ws2: quay.io/fluentd_elasticsearch/fluentd:v3.0.4,
voltha-infra-freeradius-7cbcdc66f-78b5r: freeradius/freeradius-server:3.0.21,
voltha-infra-kafka-0: docker.io/bitnami/kafka:2.8.1-debian-10-r73,
voltha-infra-kibana-6cc8b8f779-wxglm: docker.elastic.co/kibana/kibana:7.10.1,
voltha-infra-onos-classic-0: voltha/voltha-onos:5.1.1,
voltha-infra-onos-classic-onos-config-loader-84597b5696-cbq2s: opencord/onos-classic-helm-utils:0.1.0,
voltha-infra-voltha-tracing-jaeger-7fffb6cdf6-sdhms: jaegertracing/all-in-one:1.18,
voltha-infra-zookeeper-0: docker.io/bitnami/zookeeper:3.7.0-debian-10-r215,

1.3 RUNNING TESTS ON A PHYSICAL POD

Running the VOLTHA system tests on a physical POD requires some preliminary steps. This section in intended to give more details about these steps.

First of all, we need to bring the *voltha-system-tests* github repository on our virtual machine where VOLTHA is running. To do so, we can use the command *git clone* as indicated below:

git clone https://github.com/opencord/voltha-system-tests

Afterwards, two configuration files are required in order to start the tests:

- a yaml file containing information about ONUs, OLT, and others
- a json file containing information about sadis

Both files need an appropriate name and need to be placed in the correct folder (*voltha-system-tests/tests/data*). In particular, our configuration files are named *selta-sambha-config-ATT.yaml* and *selta-sambha-sadis-config-ATT.json*.

To write these configuration files, we started from predefined example, that can be found at the following links:

- yaml file: *https://github.com/opencord/pod-configs/blob/master/deployment-configs/flex-ocp-cord.yaml*
- json file: *https://github.com/opencord/voltha-system-tests/blob/master/tests/data/flex-ocp-cord-sadis.json*

For our tests, we decided to validate the tests with the ATT workflow. As suggested by the ONF documentation about the VOLTHA system tests, we worked with the robot file named *Voltha_PODTests.robot* and we used the tag *test1* to run only a specific test, i.e., the sanity E2E (end-to-end) test for OLT/ONU on POD. The sanity test, defined by ONF, has the goal to prove the correctness of the workflows and the right connection of the devices. In particular, this test is used to validate the authentication, the DHCP and the ping for the tech profile that is used.

To trigger this specific test on our physical POD for the ATT workflow, we run the following command:

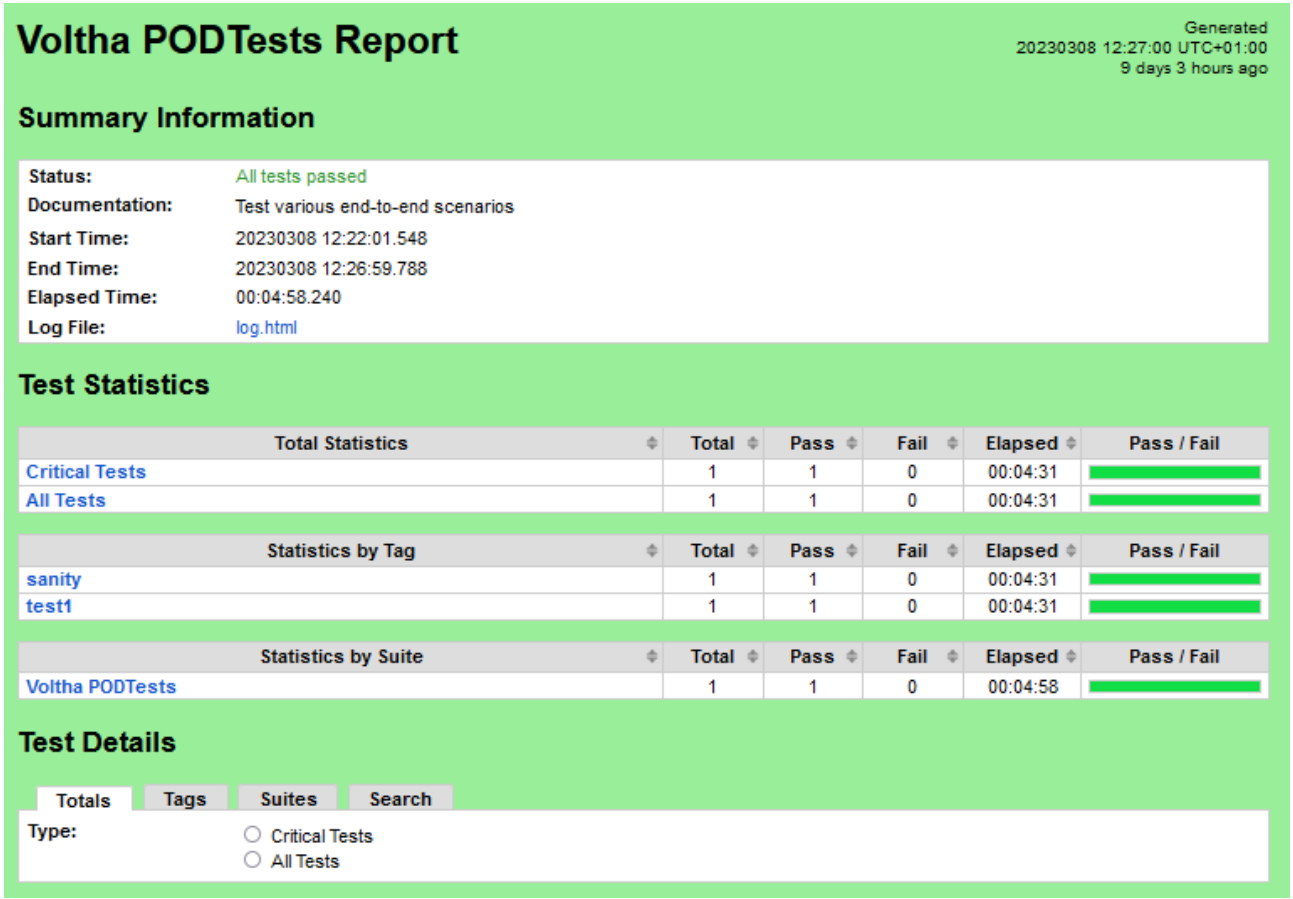
make voltha-test ROBOT_MISC_ARGS="-i test1"ROBOT_FILE="Voltha_PODTests.robot" ROBOT_CONFIG_FILE="<PATH_TO_YOUR_POD_CONFIGURATION_FILE>"

2 TEST RESULTS

In this chapter, we analyze the results of the test described in the previous section.

2.1 ANALYZING THE TEST RESULTS

The tests generate three report files: *output.xml*, *report.html* and *log.html*.
In particular, the *report.html* file informs us which tests passed and which failed. Below we show the *report.html* file for our test scenario, showing that all tests passed.



Voltha PODTests Report

Generated
20230308 12:27:00 UTC+01:00
9 days 3 hours ago

Summary Information

Status:	All tests passed
Documentation:	Test various end-to-end scenarios
Start Time:	20230308 12:22:01.548
End Time:	20230308 12:26:59.788
Elapsed Time:	00:04:58.240
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:04:31	
All Tests	1	1	0	00:04:31	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
sanity	1	1	0	00:04:31	
test1	1	1	0	00:04:31	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Voltha PODTests	1	1	0	00:04:58	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests

3 TIM WORKFLOW

As we mentioned earlier, we firstly validated the tests with the ATT workflow, and after that, we moved to the TIM workflow.
Here we briefly introduce the changes needed to run the tests with the TIM workflow.

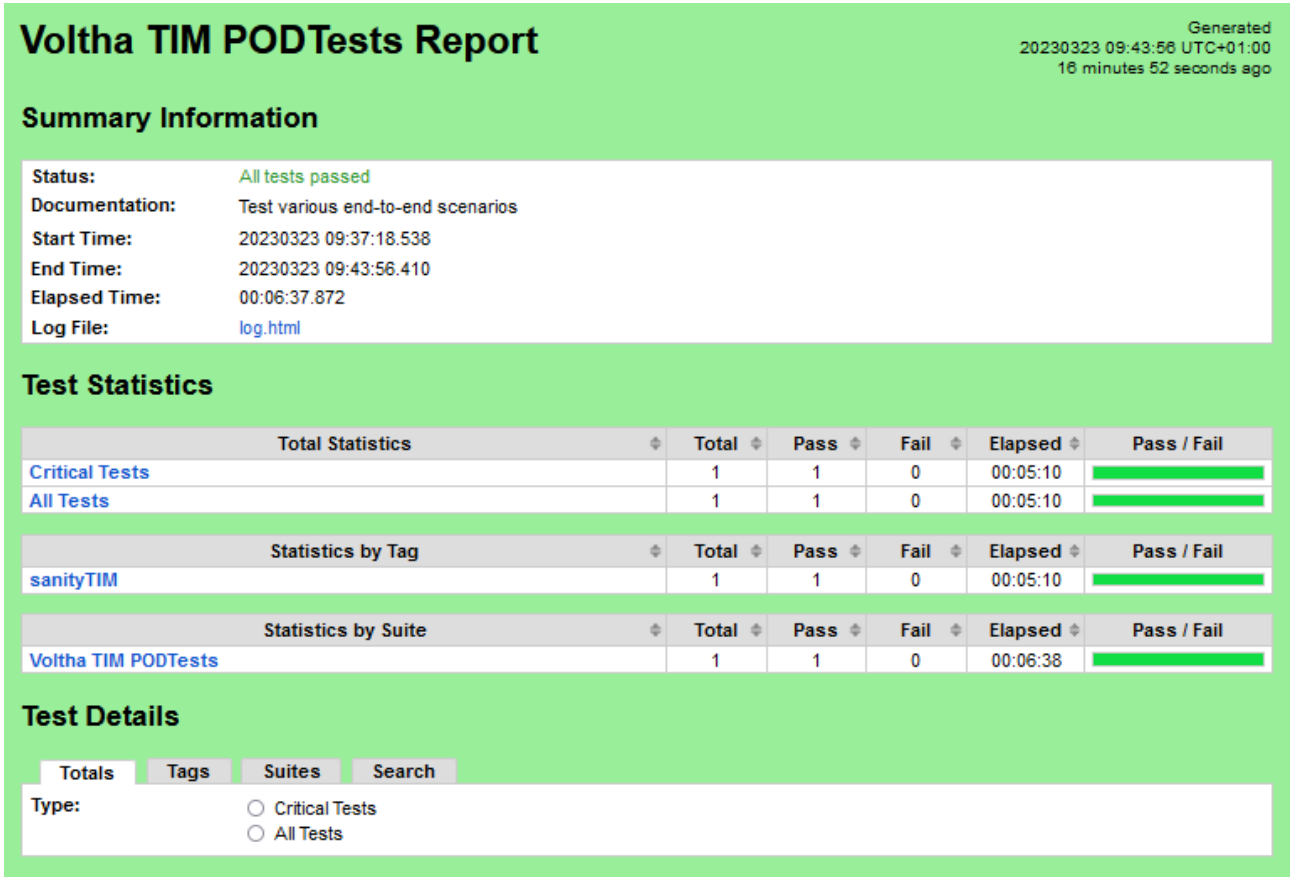
3.1 RUNNING TESTS ON A PHYSICAL POD WITH TIM WORKFLOW

In order to run the tests with the TIM workflow, we firstly need to create new configuration files and give them a specific name. In particular, we named them *selta-sambha-config-TIM.yaml* and *selta-sambha-sadis-config-TIM.json*.
Compared with the configuration files for the ATT workflow, to pass the tests with the TIM workflow, we need to change few parameters as: c-tag, s-tag, uni_tag, and service-type.
Finally, we have to run the specific test for the TIM workflow, that is *Voltha-TIM-PODTests.robot* with the tag named *sanityTIM*.

make voltha-tim-test ROBOT_MISC_ARGS=”-i sanityTIM” ROBOT_FILE=”Voltha_TIM_POSTests.robot” ROBOT_CONFIG_FILE=/home/openolt/voltha-system-tests/data/selta-sambha-config.yaml

3.2 TEST RESULTS WITH TIM WORKFLOW

As it was for the ATT workflow, the test generates three report files that are used to analyze the results. In the figure below, we show the *report.html* file, where we can see that all the tests arrived successfully at the end.



Voltha TIM PODTests Report

Generated
20230323 09:43:56 UTC+01:00
16 minutes 52 seconds ago

Summary Information

Status:	All tests passed
Documentation:	Test various end-to-end scenarios
Start Time:	20230323 09:37:18.538
End Time:	20230323 09:43:56.410
Elapsed Time:	00:06:37.872
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:05:10	
All Tests	1	1	0	00:05:10	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
sanityTIM	1	1	0	00:05:10	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Voltha TIM PODTests	1	1	0	00:06:38	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests