DEEP PAAS Orchestrator Test Plan

Issue: 2.0
Date: 8 November 2019

Document Change Record

Issue/Rev	**Partner**	**Reason for change**	**Date**
2/0	INFN	DEEP-2 release	11/08/2019

Table of Contents

Introduction

The present document describes the test cases defined for verifying the new functionalities implemented in the PaaS Orchestrator for the DEEP-2 (Rosetta) release:

JIRA DPD-624: PaaS integration with QCG
JIRA DPD-625: Add Marathon secrets management
JIRA DPD-678: Add HTTPS in Marathon applications
JIRA DPD-665: Add configuration endpoint
JIRA DPD-137: Extend the Orchestrator scheduling capabilities to allocate resources in sites with specialized hw
JIRA DPD-433: Monitoring data usage
JIRA DPD-399: Missing deployment information

The following features can be verified by code inspection and verifying that in any of the tests the Orchestrator complains about the submitted TOSCA templates:

- JIRA DPD-580: Integration of the modified A4C Tosca Parser

The target product version is 2.2.0-FINAL.

Test Case: PaaS Integration with QCG

Test Case ID: QCG-TC-01
Purpose: End to end testing of the deployment of a user's job that requires execution on QCG-Computing service.
Features under test: JIRA DPD-624
Prerequisites:

- Full PaaS stack is available and configured with at least one QCG-Computing service available
- The user is authenticated in IAM and can get a valid token
- The user has set up the execution script on a gist repository; the path to the script has been referenced in the TEMPL01.yml template
- The user has read/write access to a file repository to direct the output of the process, e.g. Swift container.
  Note: all occurrences of the '&' ampersand character in the repository path passed as a parameter to the orchestrator must be replaced with the sequence '%26'

Test data:

- Shell script test-script.sh
- TOSCA template TEMPL01.yml

Description: This test case is aimed at verifying the correct deployment, execution and monitoring of a job using the QCG-Computing service
Items under test: Orchestrator, SLAM, CMDB, Monitoring, CPR
Validation step: Unit testing ☐ Functional Testing ☒ Integration ☒
Pass/Fail Criteria: This TC is considered passed if:

- ❑ The deployment creation request is correctly managed by the Orchestrator;
- ❑ The response status contains a UUID for the submitted deployment;
- ❑ The final status of the deployment is CREATE_COMPLETE;
- ❑ The job has successfully run on the QCG compute service;
- ❑ The environment variables have been correctly passed;
- ❑ The requested command has been correctly executed;
- ❑ The output file was successfully written to the repository.

Test Case: Marathon secrets management

Test Case ID: SEC-TC-01
Purpose: End to end testing of the deployment of a long-term service using Marathon integrated with secrets management.
Features under test: JIRA DPD-625
Prerequisites:

- Full PaaS stack is available and configured with at least one Marathon provider with secrets support enabled
- The user is authenticated in IAM and can get a valid token

Test data:

- TOSCA template TEMPL02.yml

Description: This test case is aimed at verifying the correct deployment, execution and monitoring of a long-term service using the Marathon provider that use secrets to store sensitive data (i.e. password);
Items under test: Orchestrator, SLAM, CMDB, Monitoring, CPR
Validation step: Unit testing ☐ Functional Testing ☒ Integration ☒
Pass/Fail Criteria: This TC is considered passed if:

- ❏ The deployment creation request is correctly managed by the Orchestrator;
- ❏ The response status contains a UUID for the submitted deployment;
- ❏ The final status of the deployment is CREATE_COMPLETE;
- ❏ The service has successfully started on Marathon;
- ❏ The sensitive data are not exposed directly in the Marathon UI

Test Case: Support https for Marathon deployments

Test Case: Enable HTTPS in Marathon applications

Test Case ID: HTTPS-TC-01
Purpose: End to end testing of the deployment of a long-term service using Marathon accessible using HTTPS protocol.
Features under test: JIRA DPD-678
Prerequisites:

- Full PaaS stack is available and configured with at least one Marathon provider integrated with secrets available
- The user is authenticated in IAM and can get a valid token

Test data:

- TOSCA template TEMPL03.yml

Description: This test case is aimed at verifying the correct deployment, execution and monitoring of a long-term service using the Marathon provider enabling access via HTTPS protocol;
Items under test: Orchestrator
Validation step: Unit testing ☐ Functional Testing ☒ Integration ☒
Pass/Fail Criteria: This TC is considered passed if:

❏ The deployment creation request is correctly managed by the Orchestrator;
❏ The response status contains a UUID for the submitted deployment;
❏ The final status of the deployment is CREATE_COMPLETE;
❏ The service has successfully started on Marathon;
❏ The deployed service is accessible via HTTPS;

Test Case: Query Orchestrator configuration endpoint

Test Case ID: CONF-TC-01
Purpose: Verify the availability of configuration query endpoint.
Features under test: JIRA DPD-665
Prerequisites:

- Full PaaS stack is available
- The user is authenticated in IAM and can get a valid token

Test data:

- None

Description: This test case is aimed at verifying the correct response of the orchestrator when someone queries the /configuration endpoint;
Items under test: Orchestrator
Validation step: Unit testing ☐ Functional Testing ☒ Integration ☐
Pass/Fail Criteria: This TC is considered passed if:

❑ The response contains a json representation of the orchestrator configuration;

Test Case: Schedule VM deployment requesting GPU

Test Case ID: GPU-TC-03
Purpose: End to end testing of the deployment of VM that requires one or more GPUs.
Features under test: JIRA DPD-137, JIRA DPD-433
Prerequisites:

1. Full PaaS stack is available and configured with at least one openstack site providing GPUs
2. The user is authenticated in IAM and can get a valid token

Test data:

1. TOSCA template TEMPL04.yml

Description: This test case is aimed at verifying the correct deployment, execution and monitoring of a VM deployment requiring GPU(s)
Items under test: Orchestrator, SLAM, CMDB, Monitoring, CPR
Validation step: Unit testing ☐ Functional Testing ☒ Integration ☒
Pass/Fail Criteria: This TC is considered passed if:

1. The deployment creation request is correctly managed by the Orchestrator:
    1. The response status contains a UUID for the submitted deployment
2. The final status of the deployment is CREATE_COMPLETE
3. The VM has been successfully spawned using a flavor that provides GPU(s)

Test Procedures

This section provides the detailed procedures that can be run in order to verify the identified test cases.

Test Procedure QCG-TP-01

Verified Test Case: QCG-TC-01

This test procedure checks the correct deployment and execution of a job using the QCG-Compute service through the Orchestrator.

Component:	**PaaS Orchestrator**	**Release:**	**Procedure ID: QCG-TP-01**
Step #	**Description**	**Status**	**Notes (including JIRA Tickets)**
1.	Take note of the date at the bottom of this form		
2.	Prepare the URL where the output will be stored and use it in the next step to complete the submission command		
3.	Launch the deployment: `#orchent depcreate TEMPL01.yml '{ "output_url": "$URL"}'` Verify that the deployment request is accepted and a deployment UUID is returned. Take note of the deployment UUID.		`http://cloud.recas.ba.infn.it:8080/v1/AUTH_f41187320a504846b132582e172fa268/testcontainer/deep-qcg.out?temp_url_sig=5c163f3c9a639f4ae819db42ca139417a21dcae8%26temp_url_expires=1569669298`
4.	Monitor the status of the deployment with the command: `#watch -n 10 orchent depshow <DEP_UUID>`		
5.	Verify that the final status of the deployment is CREATE_COMPLETE		

6.	Get the output from the job: `#curl $URL >` deep-qcg.out		
7.	Verify that the output is accessible and the file is not empty		
8.	Verify the presence of the passed environment variable into the output file: `#grep SOME_VAR deep-qcg.out`		
9.	Remove the test deployment: `#orchent depdel <DEP_UUID>`		
Date:		**Test Conductor:**	

Test Procedure SEC-TP-01

Verified Test Case: SEC-TC-01

This test procedure checks the correct deployment and execution of a service (Mysql in this case) using Marathon integrated with secrets management.

Component:	**PaaS Orchestrator**	**Release:**	**Procedure ID: SEC-TP-01**
Step #	**Description**	**Status**	**Notes (including JIRA Tickets)**
1.	Take note of the date at the bottom of this form.		
2.	Launch the deployment: `#orchent depcreate TEMPL02.yml '{ "service_password": "MyPassword"}'` Verify that the deployment request is accepted and a deployment UUID is returned. Take note of the deployment UUID.		
3.	Monitor the status of the deployment with the command: `#watch -n 10 orchent depshow <DEP_UUID>`		
4.	Verify that the final status of the deployment is CREATE_COMPLETE.		
5.	Grab the endpoint of the service from retrieved status info: `---`		

	`outputs:`
	`  {`
	`       "endpoint": "xxx.xxx.xxx.xxx:nnnnn"`
	`  }`
	`---`		

6.	Connect to MySQL server using a client: #mysql -hxxx.xxx.xxx.xxx -Pnnnnn -uroot -pMyPassword		
7.	Execute mysql command: mysql>show databases;		
8.	Verify output.		
9.	Quit MySQL client		
10.	Remove test deployment: `#orchent depdel <DEP_UUID>`		
Date:		**Test Conductor:**	

Test Procedure HTTPS-TP-01

Verified Test Case: HTTPS-TC-01

This test procedure checks the correct deployment and execution of a service using Marathon and accessible via HTTPS protocol.

Component:	**PaaS Orchestrator**	**Release:**	**Procedure ID: HTS-TP-01**
Step #	**Description**	**Status**	**Notes (including JIRA Tickets)**
1.	Take note of the date at the bottom of this form.		
2.	Launch the deployment: `#orchent depcreate TEMPL03.yml '{}'` Verify that the deployment request is accepted and a deployment UUID is returned. Take note of the deployment UUID.		
3.	Monitor the status of the deployment with the command: `#watch -n 10 orchent depshow <DEP_UUID>`		
4.	Verify that the final status of the deployment is CREATE_COMPLETE.		

5.	Grab the endpoint of the service from retrieved status info:
	`---`
	`outputs:`
	` {`		

	"deepaas_endpoint": "xxx.xxx.xxx.xxx:nnnn", "jupyter_endpoint": "xxx.xxx.xxx.xxx:mmmm", "monitor_endpoint": "xxx.xxx.xxx.xxx:oooo" } ---		
6.	Using a browser, connect to the newly created server using the HTTPS protocol: https://xxx.xxx.xxx.xxx:nnnn		
7.	Verify output.		
8.	Remove test deployment: `#orchent depdel <DEP_UUID>`		
Date:		**Test Conductor:**	

Test Procedure CONF-TP-01

Verified Test Case: CONF-TC-01

This test procedure checks the correct response when someone submit a query to the orchestrator configuration endpoint.

Component:	**PaaS Orchestrator**	**Release:**	**Procedure ID: HTS-TP-01**
Step #	**Description**	**Status**	**Notes (including JIRA Tickets)**
1.	Take note of the date at the bottom of this form.		
2.	Query the orchestrator configuration endpoint: `# curl -H "Authorization: Bearer $IAM_TOKEN" http://<orchestrator_url>/configuration`		
3.	Verify output.		
Date:		**Test Conductor:**	

Test Procedure GPU-TP-03

Verified Test Case: GPU-TC-03

This test procedure checks the correct scheduling and deployment of virtual infrastructure requiring GPU resources.

Component:	**PaaS Orchestrator**	**Release:**	**Procedure ID: HTS-TP-01**
Step #	**Description**	**Status**	**Notes (including JIRA Tickets)**
1.	Take note of the date at the bottom of this form.		
2.	Launch the deployment: `#orchent depcreate TEMPL04.yml '{}'` Verify that the deployment request is accepted and a deployment UUID is returned. Take note of the deployment UUID.		
3.	Monitor the status of the deployment with the command: `#watch -n 10 orchent depshow <DEP_UUID>` Verify that the final status of the deployment is CREATE_COMPLETE		
4.	Access the deployed VM and verify the presence of the GPU(s) : `# lspci | grep NVIDIA`		
5.	Remove test deployment:		

	`#orchent depdel <DEP_UUID>`		
Date:		**Test Conductor:**	