GROUNDWORK

TEST REPORT

for

fuel-plugin-groundwork-monitor 7.1-7.1.0-1

Mirantis OpenStack 8.0

Contents

Revision history

Version	***Revision date***	***Editor***	***Comment***
1.0	2015-01-23	Irina Povolotskaya (ipovolotskaya@mirantis.com)	Created the template structure.
1.1	2015-03-14	Irina Povolotskaya (ipovolotskaya@mirantis.com)	Updated the template structure and contents.
2.0	2016-07-13	GroundWork, Inc.	Cleaned up the formatting. Adapted to the fuel-plugin-groundwork-monitor plugin. Included that plugin's test results.
2.1	2016-08-02	GroundWork, Inc.	Fixed mistakenly duplicate test-case IDs. Added results for test cases that are peripheral to operation of GroundWork Monitor and instead have to do with changes to the broader OpenStack deployment. *fuel_create_mirror_update_core_repos* *apply_maintenance_update*

Document purpose

This document provides test run results for all the tests of the fuel-plugin-groundwork-monitor Fuel Plugin 7.1-7.1.0-1 on Mirantis OpenStack 8.0 that are listed in the corresponding Test Plan.

Test environment

An overview of the required test environment is presented in the Test Plan. That material will not be duplicated here.

All testing was done using the Mirantis liberty-8.0 release, and the plugin release and GroundWork installer noted in the tables below.

For the actual tests run for this Test Report, we used an environment with 6 nodes available; usually only 5 nodes were actively used. The discovered nodes generally had 4x2.1GHz CPUs, 8GB or 16GB memory, 80GB disk space, and 2 network connections. The specific numbers and the assignments of roles to particular nodes are not critical for this testing, as none of it stresses the available resources.

Plugin's RPM

Name	***md5 checksum***
`fuel-plugin-groundwork-monitor-7.1-7.1.0-1.noarch.rpm`	`34aa947e4eb84b16411251ed9bfc69e9` `(to be updated)`

Interoperability with other plugins

This plugin does not depend on facilities provided by other OpenStack plugins. It does, however, depend on a companion GroundWork installer being available when the Fuel plugin is installed.

Name	***md5 checksum***
`groundworkenterprise-7.1.0-br391-gw2842-linux-64-installer.run`	`5526a5ea28a1da2098b6a206117f5c74` *The size of this GroundWork installer is 834734027 bytes, which is a prime number.*

Test coverage and metrics

92% of the possible documented test cases were attempted in this testing; only the underresourced-node test was skipped because of available test conditions. Inasmuch as none of the tests are automated, there are no particular test metrics to report.

Test results summary

Parameter	*Value*
Total quantity of executed test cases	12 (?)
Total quantity of not executed test cases	2
Quantity of automated test cases	0
Quantity of not automated test cases	16

Install/deploy test cases

Coverage of features

All tests documented in this category of the Test Plan are listed below.

Detailed testrun results

There is a general issue with running these tests, namely that deploying even the small cluster we used (5 nodes maximum) takes an inordinate amount of time. 75 minutes is not unusual; in some cases, the cluster deploy action could take over two hours. We have no experience with normal OpenStack behavior, nor how to interpret whatever information might be available in the logs which are accessible through the Fuel UI. So we did not collect any data on either why the deploy actions were taking so long or why there could be so much variance in the timing.

Item	*Test case ID*	*Passed*	*Failed*	*Skipped*	*Comment [known issue item number]*
1	*install_no_groundwork_installer*	✓			
2	*install_plugin_deploy_env*	✓			*Bad node status [1].*
3	*install_symlinked_plugin_deploy_env*	✓			*Extra node-state sequencing [2].*
4	*deploy_env_install_plugin*	✓	✓		*The first attempt to enable the plugin failed, for some unknown reason. A second attempt succeeded. [3]*
5	*install_corrupted_groundwork_installer*	✓			
6	*install_symlinked_corrupted_groundwork_installer*	✓			
7	*install_plugin_deploy_no_groundwork_installer*	✓			*Strange node count in Dashboard [4].*
8	*install_plugin_deploy_corrupted_groundwork_installer*	✓			*All nodes end up in ERROR state [1, 5].*
9	*install_plugin_deploy_on_deployed_node*			✓	*The Fuel UI does not allow the conditions for this test to arise.*

Item	*Test case ID*	*Passed*	*Failed*	*Skipped*	*Comment [known issue item number]*
10	*install_plugin_deploy_underresourced_node*			✓	*This test was not run, because we had no easy way to configure an appropriately underresourced node (a machine using only 2GB of memory, instead of the 8GB or 16GB of memory seen on the discovered nodes available in the test environment).*
11	*fuel_create_mirror_update_core_repos*				
12	*uninstall_plugin_with_deployed_env*	✓			
13	*uninstall_plugin*	✓			
14	*uninstall_plugin_after_disabling*		✓		*Plugin is still evident after deletion [6].*
15	*uninstall_plugin_when_never_enabled*		✓		*Plugin is still evident after deletion [6].*
16	*apply_maintenance_update*				
Total	Counts of observed test results.	10 (?)	3	2	*Sum of totals is greater than the number of tests because of unreliable results.*
Total, %	Percentages of observed test results compared to the overall test count.	71 (?)	21 (?)	14 (?)	*Sum of percentages is greater than 100% because of unreliable results.*

Known issues

Item	*Description*	*Severity*	*Status*
1	During the cluster deployment process, the current status of individual nodes is shown in the Fuel UI in the Nodes screen. However, the states shown there (and reflected in the Dashboard screen as well) can be very misleading and confusing. Several particular problems arise: ● The status of the GroundWork Monitor node changes in distinct stages. Once the base OS is installed, the status changes to READY, and it stays that way for a very long time while the other nodes, such as the Controller, are having OpenStack installed. Only once that activity on the other nodes is done is the GroundWork plugin deployment actually run on the GroundWork node. (Checking the target node before this, while the node already appears as READY, shows that the plugin has not yet been deployed.) But that means that the READY state (which is the	Medium	

	terminal state for all nodes during a cluster deployment) was falsely declared during most of the deployment. In fact, the node was not READY; it was actually PENDING PLUGIN DEPLOYMENT, which perhaps ought to be a state label in its own right. ● Once the actual plugin deployment begins, there is no change in the node's state to indicate that this activity is occurring on the node. This is problematic partly because running the GroundWork installer can take several minutes, and the administrator needs to know what is happening during that time. The only indication that the deployment is not fully complete is that the Dashboard screen shows the deployment progress stuck at 100% for a long time, which is itself confusing. ● We had one early plugin deployment testing run where the GroundWork installer failed sha256 checksum validation on the target node, but in spite of this the node was still marked as READY anyway. I have not tried to replicate the exact circumstances of this occurrence, because the plugin was then modified in response to the checksum validation failure. But it does suggest that the on-screen status might not reliably reflect what has happened with the plugin deployment once it is complete. ● At multiple times during a Deploy Changes action, various nodes have their status marked as INSTALLING OPENSTACK. But that claim seems bizarre, because that state can apparently be triggered by changes on some other node, long after OpenStack was first installed on the node so marked. There ought to be a separate state label used for INSTALLING OPENSTACK the first time, and any later UPDATING OPENSTACK action on the node. ● When there is a failed deployment of the GroundWork plugin, I fully expect that one node to enter an ERROR state. But what I do not expect is for that condition to be propagated to every other node in the cluster, such that the failure on just one node marks all nodes in the cluster with an ERROR state. Yet that has happened in our testing.		
2	The following was observed between tests, as part of resetting the system for a following test. Deletion of just the GroundWork node triggers an INSTALLING OPENSTACK state on the telemetry node, then on the controller node, then on the combined compute and storage nodes, in succession. This is disturbing on two counts. One, as noted in the previous item, if such state changes make any sense, they ought to be labeled as UPDATING OPENSTACK instead. But I have to question why such this state sequencing is appropriate in the first place. Why, for instance,	Low	

	would an unrelated compute node, never directly associated with the GroundWork node, need to have its OpenStack setup modified simply because the GroundWork node got deleted?		
3	Given the occasional non-repeatability of a test, one needs to question the stability of the OpenStack implementation. More specifically there has to be a documented process for exactly, not just generally, how to troubleshoot deployment failures when they unexpectedly occur. What logs are important to look at, in what order? The fact is, looking at the logs in general through the Fuel UI reveals a huge amount of output, so it's very hard to tell what is important and how to find it. Are there other sources of information that should be investigated as well?	Medium	
4	While testing (probably in between formal tests, when we were taking shortcuts to reconfigure the nodes without fully deploying the entire environment from scratch), we once observed the following set of node states listed in the Fuel UI Dashboard screen: Pending deletion 1 Ready 4 Installing OpenStack 1 This set of states, totaling 6 nodes, was displayed in spite of the fact that there were only 5 active nodes in the cluster!	Low	
5	When attempting the deployment of a corrupted GroundWork installer, we expect the node deployment to fail, and that node to be marked in some sort of ERROR state. But in fact, **all** nodes in the cluster ended up marked in ERROR state, not just the GroundWork node. This seems like an extreme reaction to a localized error, especially for a hot-pluggable plugin that might be added to an active cluster. Why should the entire cluster be taken down just because one new node has trouble deploying? I can't imagine that this behavior would be acceptable in a production cloud.	High	
6	Evidence of the Fuel plugin remains in the environment, as seen in the Fuel UI, after a plugin is completely removed from the Fuel Master. See the test definitions for how this may be observed. Even worse, a completely missing plugin may be Enabled in such an environment! Again, see the test definitions for how to observe this. I suppose one must be careful in this regard as to what happens during a plugin upgrade. If there is some transitory state when the old version of a plugin is gone before the new version of the same plugin is made available, you don't necessarily want to delete the enablement and configuration of the plugin in various environments simply because of a brief, controlled disappearance of the plugin. But for a permanent deletion of the plugin from the Fuel Master, keeping	Low	

	around state information related to the plugin seems inappropriate.		

Logs

No log information was permanently recorded as part of these tests. Certain specific expected log content is occasionally referred to in the Test Plan test cases.

Upgrade/update test cases

Coverage of features

No tests were attempted in this category. See the Test Plan for explanation.