Table of Contents

1 Introduction 2
1.1 Purpose and scope 2
1.2 Out of Scope 5
1.3 Background 6
2 Test Strategy 8
2.1 Software Development Lifecycle Model 8
2.2 Test Coverage Strategy 8
2.3 Test Level Coverage 8
2.4 Test Level Characteristics 8
2.5 Test Scope 9
2.6 Prioritization 9
2.6.1 Prioritization for Test Execution 9
2.6.2 Prioritization for Bugs 9
2.7 Testing tools 10
3 Roles and responsibilities 10
4 test environment and Resources 12
4.1 System test Environment 12
4.1.1 Hardware 12
4.1.2 Software 12
4.2 Test Data Acquisition 13
5 Test Assumptions and Risks 13
5.1 Assumptions 13
5.2 Risks 13
6 Test Schedule 14
7 Test Reporting 14
8 References 15
8.1 External References 15
9 Glossary 15

1 INTRODUCTION

This document provides the overall testing strategy and approach required to ensure that the requirements of the rolling upgrade OpenStack Mitaka – Newton (master) for Cinder, Swift and Nova projects – For now on referred as "The Projects" – are tested adequately, and that the required level of quality and reliability of the software deliverables is attained.

Master Test Plan is initiated in the Planning phase, however, this document could be updated throughout the project.

1.1 PURPOSE AND SCOPE

The purpose of this document is to communicate activities related to the planning, staffing, managing and execution of testing activities for the rolling upgrade joint deliverable.

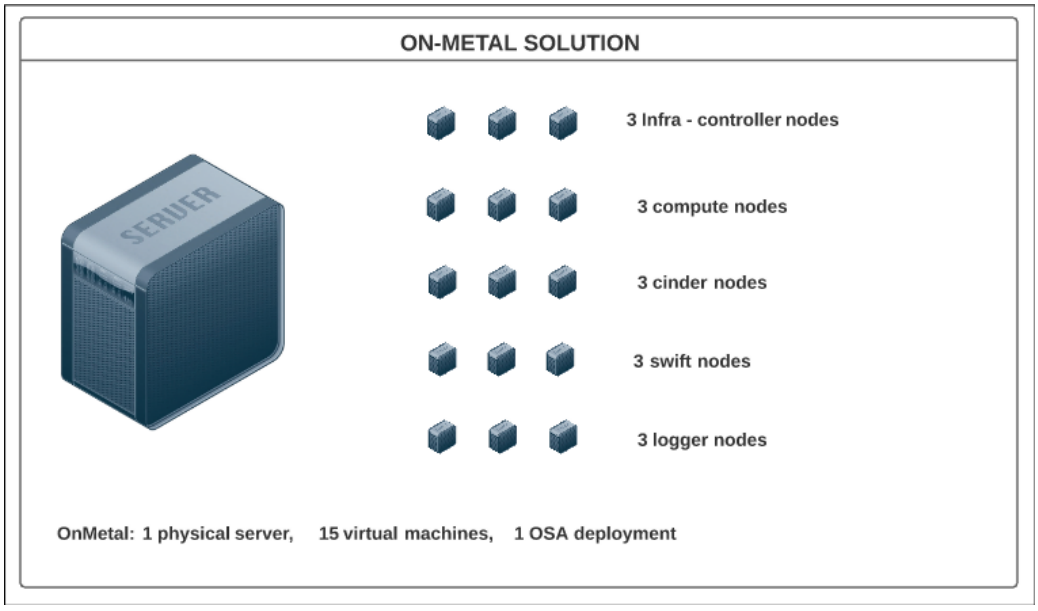
This document focuses on:

- Overall testing strategy
- Levels of testing to be performed
- Entry and Exit criteria for each test level
- Supporting testing tools
- Roles and Responsibilities of supporting testing resources
- System resources
- Assumptions and risks

During the OpenStack (OS) Newton release cycle. QE team will test upgradability of OpenStack (Cinder, Swift and Nova "The projects") and will measure API downtime – Control Plane.

To accomplish this goal, team will create a third party multi-node rolling upgrade CI (See maturity stages and flows below).

All-in-one onMetal means – 1 physical server, multiple VMs on it, create multi-node OS deployment using those VMs.



1.2 CI MATURITY STAGES

See available upgrade approaches on section 1.6 Background

This CI should be capable of performing pre-upgrade and post-upgrade testing

1.2.1 CI ALL-IN-ONE ONMETAL – LIBERTY TO MITAKA SIMPLE UPGRADE (NOT ROLLING).

Goal: Have baseline to prove that simple upgrade flow within the CI works and can be tested.

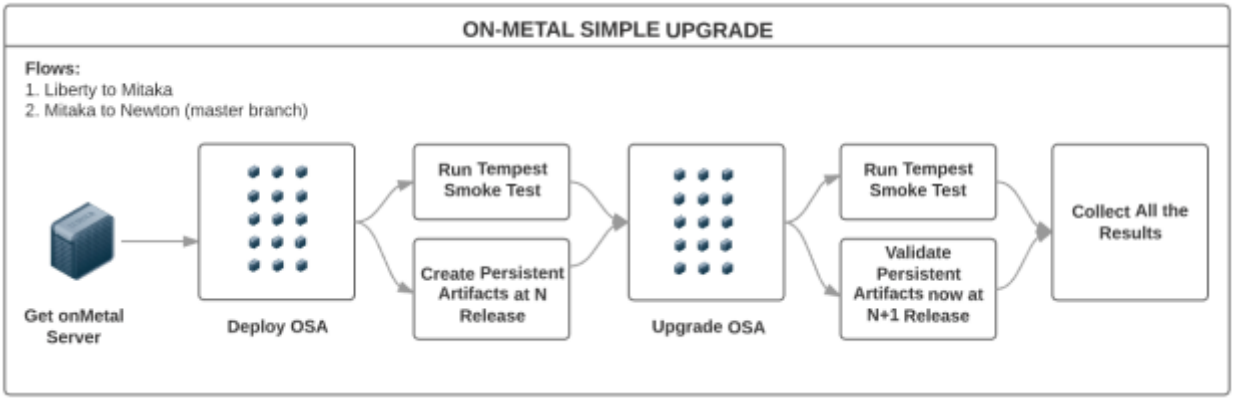
Expected to run without any major incident. OSA playbooks exists already in the community.

1. Deploy Liberty release on an all-in-one onMetal.
2. Run smoke test (Tempest)
3. Create persistent objects at Liberty
    - VM
    - Object container
    - Upload object
    - Create and attach volume to a second VM.
4. Upgrade deployment from Liberty to latest stable release – Mitaka – simple upgrade.
5. Run smoke test (Tempest)
6. Verify sanity of persistent objects created now that system is at Mitaka release.
    - ssh to existing VM
    - Add new object to existing container
    - Download existing object
    - Write a file into existing volume (second VM).

1.2.2 CI ALL-IN-ONE ONMETAL – MITAKA TO MASTER-NEWTON SIMPLE UPGRADE (NOT ROLLING).

Goal: Baseline from latest stable release – Mitaka – to daily branch Newton (Master)

Same flow as above (a-f) changing OS versions – Expect several issues TBD Document known problems.



1.2.3 CI ALL-IN-ONE ONMETAL – MITAKA TO MASTER-NEWTON ROLLING UPGRADE.

Goal: Prove rolling upgrade, measure downtime on the control plane at different stages of the upgrade, measure time on each upgrade stage, prove stability of the environment.

Depends on deployment team deliverables – upgrade process and stages.

Runs on a daily basis.

1. Deploy Mitaka release on an all-in-one onMetal.
2. Run smoke test (Tempest)
3. Create persistent objects at Mitaka
    - VM
    - Object container
    - Upload object
    - Create and attach volume to a second VM.
4. Run rolling upgrades (See below)
5. Verify sanity of persistent objects now that system is at Newton release.
    - ssh to existing VM
    - Add new object to existing container
    - Download existing object
    - Write a file into existing volume (second VM).
6. Store all results from the upgrade and testing process

Project upgrade order might change

NOVA

1. Fire Nova project rolling upgrade from Mitaka to Newton
2. Start Nova during-upgrade testing (To be run during the whole process – See Matrix HERE.
3. On each defined upgrade step
   Depends on project upgrade steps and upgrade permutation matrix
    - Measure time to perform the step
    - Run selected scenarios, might direct scenarios for a specific node/release.
4. Finish Nova project rolling upgrade (additional steps)
5. Stop Nova during-upgrade testing
6. Run smoke test (Tempest)
7. Run post-upgrade tests – TBD

CINDER

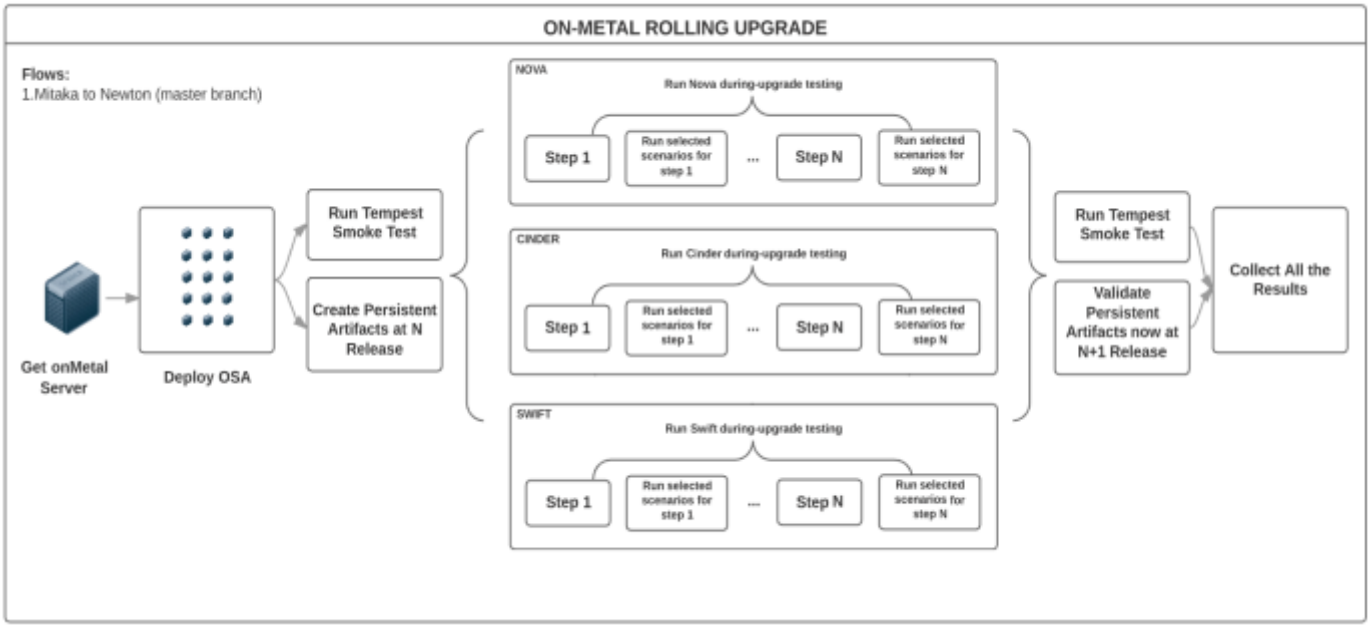
1. Fire Cinder project rolling upgrade from Mitaka to Newton
2. Start Cinder during-upgrade testing (To be run during the whole process – See Matrix HERE.
3. On each defined upgrade step
   Depends on project upgrade steps and upgrade permutation matrix
    - Measure time to perform the step
    - Run selected scenarios, might direct scenarios for a specific node/release.
4. Finish Cinder project rolling upgrade
5. Stop Cinder during-upgrade testing
6. Run smoke test (Tempest)
7. Run Cinder post-upgrade tests

SWIFT

1. Fire Swift project rolling upgrade from Mitaka to Newton
2. Start Swift during-upgrade testing (To be run during the whole process – See Matrix HERE)
3. On each defined upgrade step
   Depends on project upgrade steps and upgrade permutation matrix
    - Measure time to perform the step

- Run selected scenarios, might direct scenarios for a specific node/release.

4. Finish Swift project rolling upgrade
5. Stop Swift during-upgrade testing
6. Run smoke test (Tempest)
7. Run post-upgrade tests



1.3 DEPENDENCIES WITH OTHER TEAMS:

Nova, Cinder and Swift teams: Provide rolling upgrade steps. Provide test scenarios for the "during-upgrade" testing and "post-upgrade" test.

Deployment team: Provide Deployment and Upgrade mechanisms (scripts, playbooks, etc).

Deployment team will assist on the stabilization of the CI flow (troubleshooting, script changes).

1.4 IN SCOPE

QA team to provide: CI infrastructure, configuration and workflows for all maturity CI phases. Integration of deployment and upgrade mechanisms into the CI. Integration of automated test scenarios into the CI. Collect metrics and test results. Normalization of results into elastic-search. Presentation of results via Kibana reporter.

For the first two CI maturity stages OpenStackAnsible will be the underlying deployment mechanism. Last maturity level technology is to be confirmed with deployment team TBD.

Additional details on Trello Epic Cards: https://trello.com/c/7bwNwAQr

TEST CASE MATRIX - TBD

1.5 OUT OF SCOPE

Hardware provisioning

Any special or custom OpenStack configuration

Manual test cases or scenarios

Data plane testing

New features availability (TBD)

Deprecated features (TBD)

1.6 BACKGROUND

What are the current approaches to upgrade OpenStack?

For simplicity:

1. Simple Upgrade
    - Procedure: Turn off all services, run upgrade tools (commonly DB migration which time is often proportional to its size), turn on new services.
    - Notes: No data plane downtime. Control plane downtime is expected. Flags: Supports-upgrade and Follows-standard-deprecation. Supports-upgrade cover details like supporting previous release configuration and run same procedures across releases. Follows-standard-deprecation covers non-deleting features without deprecation window and warnings the users.
2. Online DB Migrations
    - Procedure: While old services are running, prepare for the upgrade (i.e. Expand DB schema). Then, turn off all old services, (if needed, do something with all the services turned off, but ideally nothing), then quickly start up the new version of all the services. Do some further work once the new services are running (i.e. online data migrations).
    - Notes: Doing DB migration outside the maintenance window help to reduce the downtime (for large DBs). Aim is to reduce the API downtime for users, and reduce the maintenance window, even though it might take longer overall.
3. Rolling upgrade
    - Procedure (Variable, Common one): Prepare for the upgrade with old services running (expand DB schema). Leave workers running old versions, but turn off all old control node services (API, etc.) and then turn on the new control node services. Graceful shutdown of old worker (i.e. Try not to interrupt any operations that are in progress by the worker – new work is queued), and start back up the new version of the worker. Do some further work once all new services are running (i.e. SIG_UP all services so they all know there are no old services around anymore, and complete data migrations).
    - Notes: Clearly less relevant if you only have API nodes in your service. Helps limit the number of services that need to be shutdown then restarted. Graceful shutdown allows workers with long running tasks to complete their existing work before they are upgraded. Sometimes you need to set a configuration (i.e. upgrade_levels.compute="auto") to allow the new services to support the rolling upgrade of the workers, rather than the non-rolling upgrade.
4. Zero downtime upgrade
    - Procedure (Not yet implemented by any project): split the system into: API, control nodes, and workers. Upgrade the control nodes, doing a graceful shutdown, then starting up the new version. Upgrade workers as with rolling upgrade. Old and new APIs are run side by side, with the load balancer sending new connections to the new API nodes. Once old API nodes have no connections, they are turned off.

- Notes: Community needs to consider if this is worth the complexity.

Upgrade from what, to what?

Currently, OpenStack only support to upgrade from N to N+1 release.

Many projects aim to support deploy from any commit on Master (within in the same release) but gets tricky. Recommendation is to upgrade only to stable releases.

Project Status

- Nova: Supports approaches 1, 2 and 3.
- Swift: TBD – Uses a different methodology.
- Cinder: Supports approaches 1 and 3 (approach 2 under review).

For the CI, consider upgrade approach 1 (simple upgrade) and ultimately approach 3 (Rolling upgrade).

For Testing Purposes

Rolling upgrade consists of upgrading progressively the servers of a distributed system to reduce service downtime. It does not require complete elimination of downtime during upgrade, but rather reducing the scope from "all services" to "some services at a time." In other words, "restarting all API services together" is a reasonable restriction.

Rolling upgrades imply that during some interval of time there will be services or components of a service running and interacting at different code versions in the same cloud. It puts multiple constraints onto the software.

- older services should be able to talk with newer services
- older services should not require the database to have older schema (otherwise newer services that require the newer schema would not work).
- Zero data plane downtime
- Minimal control plane downtime
- System is functional during the "rolling" phases of the upgrade.

Testing rolling upgrades may include several combinations, permutations, scenarios and areas of focus. Hence a priority or risk matrix is good way to select which scenarios and test cases will be executed at each upgrade stage.

Additional test cases and implementation details will be provided by each of the teams.

About Grenade

Focuses on control plane with old workers, i.e. Having multi-node deployment with one old and one upgraded worker node.

Grenade only covers a small subset of what could be tested.

Doesn't cover running old API nodes with new API nodes, nor with new control plane, nor mixed workers. Hence several issues are expected as other combinations are tested.

2 TEST STRATEGY

This section addresses test level selection, characteristics and testing tools.

2.1 SOFTWARE DEVELOPMENT LIFECYCLE MODEL

Rolling Upgrade CI and all QA team activities will follow agile practices. Team will have sprints of two weeks' duration, daily standups, backlog grooming, and sprint planning's.

2.2 TEST COVERAGE STRATEGY

Coverage Strategy	Choose One (x)
100% Upgrade Scenarios Covered	
Upgrade Scenarios selected via Risk Based Analysis	X

2.3 TEST LEVEL COVERAGE

This section contains specific information relating to the selection of the test levels. Refer to Apendix 1 – test argon for details of each test level.

Test Level	Applicable?	Rationale for omitting test level
Unit Testing (UT)	☐ *Yes* ☒ *No*	Taken care by the projects – Projects unit test
Integration Testing (IT)	☐ *Yes* ☒ *No*	*Taken care by the projects – Tempest gate testing*
System Testing (ST)	☒ *Yes* ☐ *No*	
Acceptance Testing (UAT)	☐ *Yes* ☒ *No*	*Out of Scope*

2.4 TEST LEVEL CHARACTERISTICS

Test Level	Owner	Entry Criteria	Exit Criteria
ST – Simple Upgrade & Rolling Upgrade	QE Team And Focal Points	• *Projects have unit testing, and integration testing passing with all the versions involved.* • *The projects are in compliance with the OpenStack upgrades guidelines HERE* • *System test environment is established (CI)* • *Adequate Test data is available (Test Cases, DBs, etc)* • *Completed and reviewed*	• 100% of planned test specifications (test cases/scripts/scenarios) for system test level must be executed and/or dispositioned with an agreement of the testing stakeholders. • *TBD - Defects were documented and reported in Launchpad* • *TBD - All severity 1 (critical) and 2 (major) defects are ??.*

		test cases / test scripts • *All scenarios to be tested are identified, and automated.* • *Test scenarios are included on the CI*	

2.5 PRIORITIZATION

This section describes the methodology that will be used prioritize test execution and bugs.

2.5.1 PRIORITIZATION FOR TEST EXECUTION

CI will go through the mentioned maturity levels mentioned on Section 1.
All exiting automated test cases and scenarios will be run.
Non-automated scenarios will be prioritized based on a test case risk assessment. The test case risk will be a combination of likelihood of failure and impact if it fails. All test cases will receive an overall score that will be grouped into a high, medium, or low category. These categories will be used to determine the order that test cases will be automated and then executed.
As CI matures more and more Test scenarios will be automated and included into the execution.
The order of test case execution will be based on each project upgrade stages (if any) and in the order in which each project gets upgraded.

Score will be done using:

Impact
1 = None – No noticeable impact to features
2 = Little – Low impact to features
3 = Moderate – Medium impact to features
4 = Severe – High impact to features
5 = Extreme – Critical impact to features

Likelihood of feature failure
1 = Somewhat Likely – Liittle chance that the feature will fail
2 = Likely – Feature will probably fail, but not certain
3 = Very Likely – Very high probability that the feature will fail

Overall score is the product of the impact value times the likelihood value
High = 9-15
Medium = 5-8
Low = 1-4

2.5.2 PRIORITIZATION FOR BUGS

Bug priority will be suggested and documented on Launchpad, following OpenStack community guidelines http://docs.openstack.org/contributor-guide/doc-bugs.html#doc-bug-triaging-guidelines

Assistance of each of the project will be required for bugs that:
- *Causes all upgrade, CI or testing activities to be halted.*
- *Severely affects the functionality.*

2.6 TESTING TOOLS

Tool(s)	Description
Jenkins 2.0	*Jenkins pipelines will be used to automate the complete flow including OS deployment, upgrade and testing activities.*
Tempest	*OS integration test suite – To be used for the sanity of the environment and potentially persistent testing during the upgrade*
Python Scripts	*Additional automated test suites and scripts.*
Groovy Scripts	*Language used by Jenkins pipelines*
OSA	*OpenStack-Ansible OS deployer to perform environment deployment and upgrade.* *Uses Ansible tool.*
Ironic	*OpenStack baremetal project to provision operating system into physical nodes.* *Potentially replaced by cobbler if going on virtual instead of physical hardware.*
Elastic-Search	*Non SQL DB to store results information*
Kibana	*Reporter server to display results*
Rally	*Benchmarking tool for System Integration Test level. TBD*

3 ROLES AND RESPONSIBILITIES

Role/ Group	Responsibilities	Name(s)
QA Test Lead	*Provides testing management oversight.* *Responsibilities:* • *provide technical direction* • *acquire appropriate resources* • *provide management reporting*	Daryl Walleck

Product Owner	*Represents customer's interest and represents the product to the outside world (Customer).* *Responsibilities:* • *Responsible for market, business case, and competitive analysis* • *Responsible for long and short term product vision* • *Prioritizes features for releases based upon expected ROI* • *Writes Acceptance Criteria* • *Writes user stories* • *Makes trade-off decisions between scope and schedule*	Kenny Krish Sonu
QA Team	*Responsible for qualification of product.* *Responsibilities:* • *decide on the scope of the testing in agreement with Project Manager* • *Configure CI infrastructure* • *Create CI flows* • *Integrate test cases into the CI* • *Automates additional test cases and scenarios* • *log results* • *open and verify bugs* • *help troubleshooting error*	OSIC QA
Deployment Team	*Responsible for creation of OpenStack deployment and rolling upgrade automated mechanism.* *Responsibilities:* • *Get and configure infrastructure(baremetal 22 nodes)* • *Provide OpenStack architecture* • *Create automated way to deploy OpenStack.* • *Create automated way to rolling upgrade the selected projects.* • *Help with integration and stabilization of the CI flow*	OSIC Deployment

Nova, Cinder, and swift focal points	*Responsible to provide the rolling upgrade information and test plan for their own projects:* • *Provide deployment team with the steps and knowledge about "How to live upgrade the project" - Documentation, links, release notes, locate any other relevant information.* • *Identify the scenarios that should run during and after the rolling upgrade (either automated or manual)* • *If needed help with the automation of the manual identified test scenarios*	Shashi Pushkar Shiva Szimon

4 TEST ENVIRONMENT AND RESOURCES

The following tables are used to identify the system resources (hardware, software etc. required for the test environment.

4.1 SYSTEM TEST ENVIRONMENT

4.1.1 HARDWARE

Component	Description	Server name (Optional)	Network Information (Optional)	Notes
Bare metal Server	1 Rackspace onMetal I/O V2 server	Variable	Variable	Need credential to spin it up
Jenkins master	Principal Jenkins component	Cloud1	172.99.106.115	
Jenkins agents	Jenkins slaves – perform actual work	Variable – automated	Private-net	

4.1.2 SOFTWARE

Environment	Component	Product/Application	Versions
System under test	*Platform*	*OpenStack*	*Liberty* *Mitaka* *Newton (Master)*
CI	*Ansible*	*Configuration management* *OS deployment*	*Latest via pip install*
CI	*Web Server*	*Jenkins*	*2.0*
CI	Programming languages	Python Groovy Shell	

4.2 TEST DATA ACQUISITION

The following table is used to identify the approach for acquiring and securing the test data to be used for each test level.

Source of Test Data	Extraction approach	Type of test data (input or pre-existing)	Security controls
TBD – Might be OSIC Cloud1 DBs but not confirmed	*TBD*	*input*	*TBD*

5 TEST ASSUMPTIONS AND RISKS

5.1 ASSUMPTIONS

This section lists assumptions that are specific to the test planning.

#	Assumption
1	*A stable mechanism to deploy OpenStack (all-in-one onMetal or any other) is owned and provided by the deployment team.*
2	*A stable mechanism to upgrade OpenStack is owned and provided by the deployment team.*
3	*A stable mechanism to rolling upgrade OpenStack is owned and provided by the deployment team.*
4	Deployment team will assist on the stabilization of the CI flow
5	Deployment team will help troubleshooting and find root cause analysis of issues.
6	Issues won't be fixed unless caused by the deployment tools (deployment team) or the CI (QA team)
7	Nova, Cinder and Swift projects met the OpenStack community requirements to perform upgrades and rolling upgrades efficiently
8	Upgrade steps/ stages are provided by the projects and agreed for implementation with the deployment team.
9	Test Plans are provided by the projects assisted by QA team

5.2 RISKS

The following risks to the testing plan have been identified and the supporting contingency plans included to mitigate their impact on the project. The impact (or severity) of the risk is based on how the project would be affected if the risk was triggered. The trigger is the milestone or event that would cause the risk to become an issue to be addressed.

#	Risk	Impact	Trigger	Mitigation/ Contingency Plan
1	*Fail to deliver rolling*	*Unable to complete the*		*Have CI with simple upgrade*

	upgrade mechanism.	*CI flow*		*(Maturity stages 1 and 2)*
2	Unstable OpenStack upgrade and rolling upgrade	*Unable to complete the CI flow* *Delay testing* *Untrusty results*	*Unstable branches* *Bugs on the projects*	*TBD*
3	*CI stability*	*Delay testing* *Untrusty results*	*Unstable branches* *Issues on the deployment tools* *Issues on the CI*	*TBD*
4	Selected scenarios not reflecting critical areas	Untrusty results	Blind spots Lack of knowledge	Working with technical leaders to validate the scenarios
5				

6 TEST SCHEDULE

Testing Level	Test Activity	Timeframe
System Integration Test	*Sprint 1 – Id test scenarios, test cases and upgrade procedures.*	*Sep 2*
	Sprint 1 – Automation of an all-In-one OnMetal maturity level 1 – Liberty to Mitaka	Sep 2
	Sprint 2 – Automation of test cases and integration into Jenkins CI flows	Sep 16
	Sprint 2 – Automation of an all-In-one OnMetal maturity level 1 – Mitaka to Newton Master branch	Sep 16
	Sprint X – One server with multi node VMs – rolling upgrade DEPENDS ON DEPLOYMENT PLAYBOOKS (specifically rolling upgrade steps)	TBD
	Sprint X – Multi node bare metal – rolling upgrade DEPENDS ON DEPLOYMENT PLAYBOOKS	TBD

7 TEST REPORTING

Following measurements will be collected and reported.

#	Metrics	Measurement Data	Frequency	Responsible	Reported To
	API downtime	Time End to End - Trending	Daily	CI	
	Playbooks elapsed times	Time End to End - Trending			
	Test suites failure ratio	Trending			

8 REFERENCES

Main repository https://github.com/osic/osic-upgrade-test
http://docs.openstack.org/contributor-guide/doc-bugs.html
http://docs.openstack.org/index.html#install-guides
http://docs.openstack.org/developer/grenade/readme.html
http://docs.openstack.org/developer/grenade/readme.html#basic-flow
http://www.danplanet.com/blog/2015/06/26/upgrading-nova-to-kilo-with-minimal-downtime/
http://docs.openstack.org/ops-guide/ops-upgrades.html
http://docs.openstack.org/developer/neutron/devref/upgrade.html
https://governance.openstack.org/reference/tags/assert_follows-standard-deprecation.html
https://governance.openstack.org/reference/tags/assert_supports-rolling-upgrade.html

8.1 EXTERNAL REFERENCES

This section lists references to the relevant policies or laws that give rise to the need for this plan.
NA

9 GLOSSARY

Item	Description
Black box testing	Focus is on the external attributes and behavior of the software. Such testing examines the software from the user perspective. UAT is the classical example of this type of testing
Bug	A bug is a flaw, error or omission identified during the testing process. Bugs are typically classified by level of severity ranging from non-critical to "show stopper"
Negative Testing (destructive)	Testing attempts to prove that the software can be broken using invalid or erroneous input conditions. Both defined and undefined error conditions should be generated.
Positive Testing	Testing attempts to prove that the software satisfies the requirements
S&P testing	Stress and Performance testing
Test Case	A test case is a specific test designed to verify a particular condition or requirement. It identifies input data with predicted results and describes the testing objective.
Test Script	Provide the step by step procedures comprising the actions to be taken and the verification of the results
White-box testing	It tests software with knowledge of internal data structures, logical flow at the source code level. Unit testing is the classical example of this type of testing.

10 APENDIX 1 – TEST ARGON

10.1 TEST LEVELS

A test level is a group of test activities that are organized and managed together in order to reach a goal.

Item	Description
Unit testing	Verifies code flows of software components. For instance statements, decisions, branches, menus, processes, inputs and outputs.
Integration testing	Verifies the interfaces between components.

System testing	Concerned with the behavior of the whole product.
Acceptance testing	Regarding user needs and business processes.

10.2 TEST TYPES

A test type is focused on a particular test objective. A test type can be executed at any test level.

Item	Description
Functional	A Specific function performed by the software.
Non functional	Test required to measure other aspects or characteristics of the system. Examples: accessibility, performance, and upgradability testing.
Structural	Relates to the architecture of the software or system.
Change-Related	Re-test to confirm original defect has been removed and no new defects are injected.