Table of Contents

[1 Introduction 2](#_Toc461020374)

[1.1 Purpose and scope 2](#_Toc461020375)

[1.2 Ci Maturity Stages 3](#_Toc461020376)

[1.2.1 CI all-in-one onMetal – Liberty to Mitaka simple upgrade (not rolling). 3](#_Toc461020377)

[1.2.2 CI all-in-one onMetal – Mitaka to Master-Newton simple upgrade (not rolling). 4](#_Toc461020378)

[1.2.3 CI all-in-one onMetal – Mitaka to Newton-master rolling upgrade. 4](#_Toc461020379)

[1.2.4 CI BareMetal – Mitaka to Newton-master rolling upgrade. 5](#_Toc461020380)

[1.3 Ci Test Suites 6](#_Toc461020381)

[1.3.1 Tempest Smoke Test Suite 6](#_Toc461020382)

[1.3.2 Persistent Resources Test Suite 6](#_Toc461020383)

[1.3.3 During Upgrade Test Suite 7](#_Toc461020384)

[1.3.4 Upgrade Steps Test Suites 7](#_Toc461020385)

[1.4 Dependencies with Other Teams 7](#_Toc461020386)

[1.5 In Scope 7](#_Toc461020387)

[1.6 Out of Scope 7](#_Toc461020388)

[1.7 Background 8](#_Toc461020389)

[2 Test Strategy 9](#_Toc461020390)

[2.1 Software Development Lifecycle Model 10](#_Toc461020391)

[2.2 Test Coverage Strategy 10](#_Toc461020392)

[2.3 Test Level Coverage 10](#_Toc461020393)

[2.4 Test Level Characteristics 10](#_Toc461020394)

[2.5 Prioritization 11](#_Toc461020395)

[2.5.1 Prioritization for Test Execution 11](#_Toc461020396)

[2.5.2 Prioritization for Bugs 11](#_Toc461020397)

[2.6 Testing tools 11](#_Toc461020398)

[3 Roles and responsibilities 12](#_Toc461020399)

[4 test environment and Resources 14](#_Toc461020400)

[4.1 System test Environment 14](#_Toc461020401)

[4.1.1 Hardware 14](#_Toc461020402)

[4.1.2 Software 14](#_Toc461020403)

[4.2 Test Data Acquisition 15](#_Toc461020404)

[5 Test Assumptions and Risks 15](#_Toc461020405)

[5.1 Assumptions 15](#_Toc461020406)

[5.2 Risks 15](#_Toc461020407)

[6 Test Schedule 16](#_Toc461020408)

[7 Test Reporting 17](#_Toc461020409)

[8 References 17](#_Toc461020410)

[8.1 External References 17](#_Toc461020411)

[9 Glossary 17](#_Toc461020412)

[10 Apendix 1 – test argon 18](#_Toc461020413)

[10.1 TEST LEVELS 18](#_Toc461020414)

[10.2 TEST TYPES 18](#_Toc461020415)

DOCUMENT HISTORY

| **Document Version** | **Date** | **Changes** |
| --- | --- | --- |
| 0.0.1 | Aug 29 2016 | Initial Version |
| 0.0.2 | Aug 30 2016 | Added comments from stakeholders reviews |
| 0.0.3 | Sep 07 2016 | Added timeline and initial version of test matrixes |

# Introduction

This document provides the overall testing strategy and approach required to ensure that requirements of rolling upgrade OpenStack Mitaka to Newton (master) for Cinder, Swift and Nova projects are tested adequately, and 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.

## 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) 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 OSA deployment using those VMs.

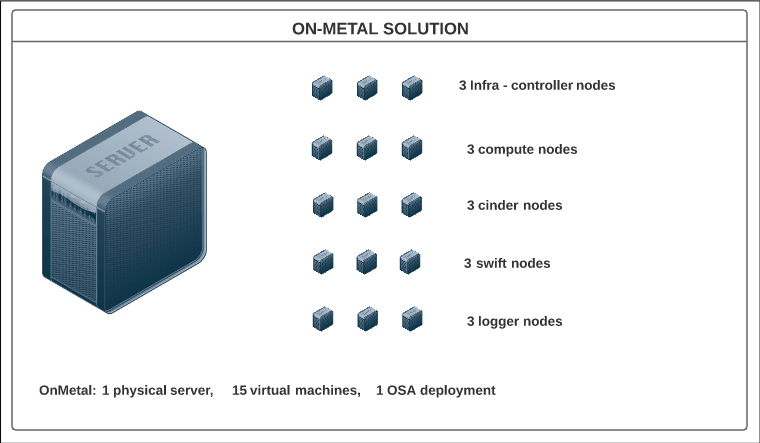
Physical Host Specs known to work well:

CPU CORES: 20

MEMORY: 124GB

DISK SPACE: 1.3TB

These specs are covered by the Rackspace OnMetal-IO v2 Server product and CI will include the provisioning of the physical server via Rackspace CLI. And the creation of the 15 VMs on top of the physical server via Ansible playbooks. Implementation details [HERE](https://github.com/osic/qa-jenkins-onmetal).



Bare-Metal means – 22 node OSA deployment on physical hardware (servers).

Each Host Specs:

TBD – **Deployment Team to provide**

\*\* See OpenStack Reference Architecture used HERE

## Ci Maturity Stages

See available upgrade approaches on section [1.7 Background](#_Background)

See all Test Suites and Matrixes at [1.3 CI Test Suites](#_Ci_Test_Suites)

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

### CI all-in-one onMetal – Liberty to Mitaka simple upgrade (not rolling).

Goal: Have baseline to prove simple upgrade (Sec1.6) within the CI works and can be tested.

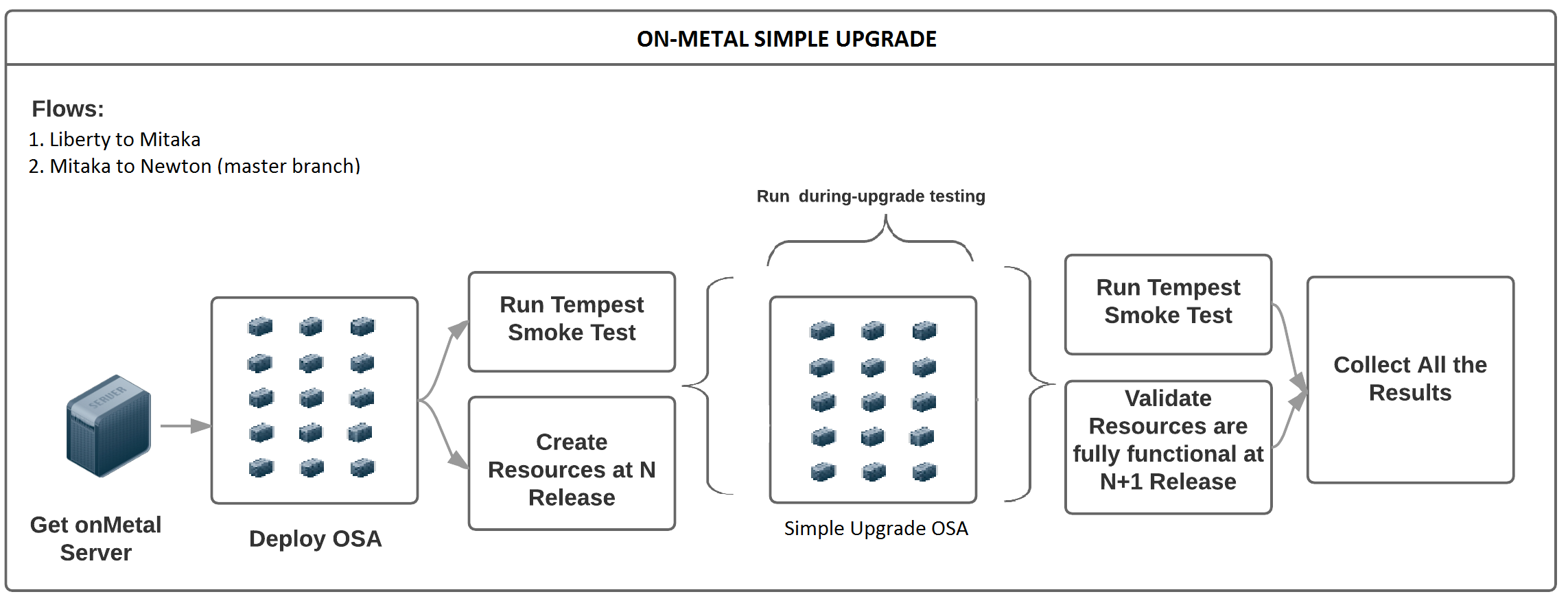
Expected to run without any major incident. OSA playbooks exists already in the community.

1. Get All-in-one onMetal Server
2. Deploy Liberty release on the All-in-one onMetal.
3. Run Tempest smoke test suite.
4. Run create Persistent resources test suite (resources lives during and after the upgrade).
5. Upgrade deployment from Liberty to latest stable release – Mitaka – simple upgrade.
6. Run Tempest smoke test suite.
7. Run the validations from Persistent resources test suite (resources created at Liberty validated at Mitaka) - verifies full functionality of existing resources.

### 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 issues do to instability of master branch.



### CI all-in-one onMetal – Mitaka to Newton-master 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.

* Get All-in-one onMetal Server.
* Deploy Mitaka release on an all-in-one onMetal.
* Run Tempest smoke test suite.
* Run create Persistent resources test suite (resources lives during and after the upgrade).
* Start During upgrade test suite.
* Start rolling upgrades (See below).
* Stop During upgrade test suite.
* Run Tempest smoke test suite.
* Run the validations from Persistent resources test suite (resources created at Mitaka validated at Newton master) verifies full functionality of existing resources.
* Colect 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. On each defined upgrade step

Depends on OSA upgrade steps and upgrade permutation matrix

* Measure time to perform the step
* Run selected scenarios. TBD- Add Matrix

1. Finish Nova project rolling upgrade (additional steps)

**CINDER**

1. Fire Cinder project rolling upgrade from Mitaka to Newton
2. On each defined upgrade step

Depends on OSA upgrade steps and upgrade permutation matrix

* Measure time to perform the step
* Run selected scenarios. TBD- Add Matrix

1. Finish Cinder project rolling upgrade (additional steps)

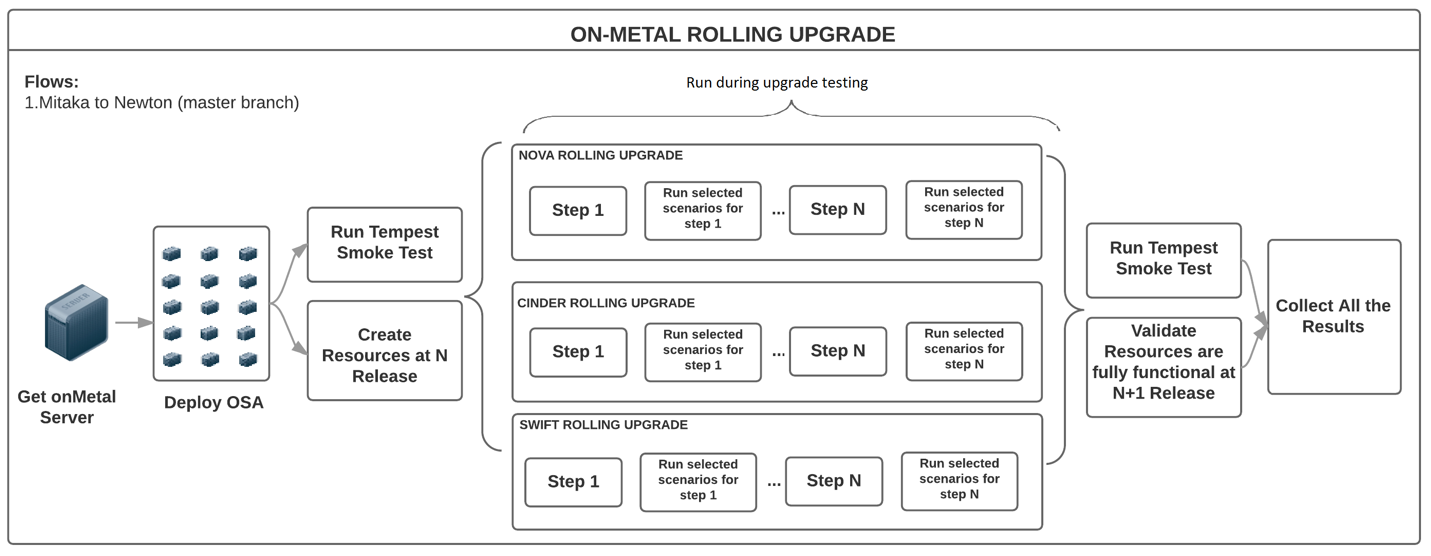
**SWIFT**

1. Fire Swift project rolling upgrade from Mitaka to Newton
2. On each defined upgrade step

Depends on OSA upgrade steps and upgrade permutation matrix

* Measure time to perform the step
* Run selected scenarios. TBD- Add Matrix

1. Finish Swift project rolling upgrade (additional steps)



### CI BareMetal – Mitaka to Newton-master rolling upgrade.

Goal: Prove rolling upgrade on a Production like environment.

Depends on deployment team deliverables:

* OpenStack reference design architecture implemented on physical environment.
* Hardware provisioning.
* Operating system provisioning on physical severs.
* OpenStack deployment playbooks/scripts
* OpenStack rolling upgrade process, stages and scripts

Run on a daily basis.

CI steps are the same as in 1.2.3 – Several changes are required:

* Preparing physical environment
* Integrate new OpenStack deployment playbooks.
* Integrate new OpenStack rolling upgrade playbooks.

## Ci Test Suites

### Tempest Smoke Test Suite

OpenStack integration test suite - <http://git.openstack.org/openstack/tempest>

Smoke test is a subset of the community test suite which intends to be a sanity test for a live OpenStack cluster.

### Persistent Resources Test Suite

| **Create Resources** | **Steps** | **Verification Points** |
| --- | --- | --- |
| Create VM | Create keypair  (Evaluating rally.tasks.nova.create-and-list-keypairs.yaml) |  |
| Create secgroup (Evaluating rally.tasks.nova.create-and-list-secgroups.yaml) |  |
| Create VM  (Evaluating: rally.tasks.nova.boot.yaml using created keypair and secgroup)  (Evaluating rally.plugins.openstack. scenarios.vm.utils.\_boot\_server\_with\_fip) |  |
| ssh to the VM (Evaluating rally.plugins.openstack. scenarios.vm.utils.\_run\_command\_over\_ssh) | Able to ssh to the VM |
| tempest.api.compute.servers.test\_server\_actions.ServerActionsTestJSON.test\_pause\_unpause\_server | Able to pause\_unpause the VM |
| tempest.api.compute.servers.test\_server\_actions.ServerActionsTestJSON.test\_suspend\_resume\_server | Able to suspend\_resume the VM |
| tempest.api.compute.servers.test\_server\_actions.ServerActionsTestJSON.test\_shelve\_unshelve\_server | Able to shelve\_unshelve the VM |
| tempest.api.compute.servers.test\_server\_rescue.ServerRescueTestJSON.test\_rescue\_unrescue\_instance | Able to rescue\_unrescue the VM |
| tempest.api.compute.servers.test\_server\_rescue.ServerRescueTestJSON.test\_rescued\_vm\_add\_remove\_security\_group | Able to add security group to a rescued VM |
| Create and attach volume to a second VM | TBD |  |
|  |  |
|  |  |
| Volume is available on the vm | Able to write to the volume |
| Create Container and upload object | TBD |  |
|  |  |
|  | Able to download the object |
|  | Able to upload a second object |
|  | Able to delete second uploaded object |

### During Upgrade Test Suite

| **Test Scenario** | **Expected Output** | **Comments** |
| --- | --- | --- |
| nova.servers.list | No downtime |  |
| cinder.volume.list | No downtime |  |
| swift.container.list | No downtime |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Upgrade Steps Test Suites

Depends on upgrade flow delivered by deployment team, still under discussion

| **Test Scenario** | **Expected Output** | **Comments** |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 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).

For maturity level 1.2.4 – bare metal: Deployment team to provide the physical reference architecture used. Handle hardware provisioning and configuration. Provide OpenStack deployment automated mechanism. Provide OpenStack rolling upgrade automated mechanism.

## 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.

CI maturity stages will use OpenStackAnsible as the underlying deployment mechanism.

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

## Out of Scope

Hardware provisioning

Any special or custom OpenStack configuration

Manual test cases or scenarios

Data plane testing

New features availability

Deprecated features

## 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.

# Test Strategy

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

## 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.

## Test Coverage Strategy

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

## Test Level Coverage

This section contains specific information relating to the selection of the test levels. Refer to [Apendix 1 – test argon](#_Apendix_1_–) 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* |

## 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*](http://docs.openstack.org/ops-guide/ops-upgrades.html) * *System test environment is established (CI)* * *Adequate Test data is available (Test Cases, DBs, etc)* * *Completed and reviewed test cases / test scripts* * *All scenarios to be tested are identified, and automated.* * *Test scenarios are included on the CI* | * 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 ??.* |

## Prioritization

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

### 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*

### 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*](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.*

## 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* |

# 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 |

# test environment and Resources

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

## System test Environment

### 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 |
| 22 physical servers | TBD – Access to the nodes |
| Jenkins master | Principal Jenkins component | Cloud1 | 172.99.106.115 |  |
| Jenkins agents | Jenkins slaves – perform actual work | Variable – automated | Private-net |  |

### 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 |  |

## 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* |
|  |  |  |  |

# Test Assumptions and Risks

## 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 |
|  |  |

## 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 upgrade mechanism.* | *Unable to complete the CI flow* |  | *Have CI with simple upgrade (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** | Fail to deliver CI rolling upgrade on bare-metal | *Unable to complete the CI flow on baremetal physical servers* | External dependencies | Have CI with rolling upgrade on top of a single physical server but multi-vms OSA deployment (Maturity stage 3) |

# 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 HARDWARE PROVISIONING, HARDWARE CONFIGURATION, REFERENCE ARCHITECTURE, OSA DEPLOYMENT AND ROLLING UPGRADE PLAYBOOKS | TBD |

# 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 |  |

# References

*Main repository https://github.com/osic/osic-upgrade-test*

[*http://docs.openstack.org/contributor-guide/doc-bugs.html*](http://docs.openstack.org/contributor-guide/doc-bugs.html)

[*http://docs.openstack.org/index.html#install-guides*](http://docs.openstack.org/index.html#install-guides)

[*http://docs.openstack.org/developer/grenade/readme.html*](http://docs.openstack.org/developer/grenade/readme.html)

[*http://docs.openstack.org/developer/grenade/readme.html#basic-flow*](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*](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_follows-standard-deprecation.html)

*https://governance.openstack.org/reference/tags/assert\_supports-rolling-upgrade.html*

## External References

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

NA

# 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. |

# Apendix 1 – test argon

## 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. |

## 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. |