KVSaaS Master Test Plan

Version 0.8

Revision history

Date	Version	Description	Author
Apr 18, 2014	0.8	Initial draft version created	M. Usichenko

Table of Contents

1 References and Abbreviations

2 Introduction

This document represents Master Test Plan (MTP) for the Symantec MagnetoDB (SMDB) project.

This Plan created for the complete life-cycle of the project and is intended to:

- Provide a central artifact to govern the planning and control of the test effort. It defines the general approach that will be employed to test the software and to evaluate the results of that testing, and is the top-level plan that will be used by team leads and managers to govern and direct the detailed testing work
- Provide visibility to stakeholders in the testing effort that adequate consideration has been given to various aspects of governing the testing effort, and, where appropriate, to have those stakeholders approve the plan.

3 Governing Evaluation Mission

Primary objectives of testing effort are to assure that the system:

- Meets the requirements, including the non-functional ones specified for SMDB and outlined by the OpenStack community
- Satisfies the Use Cases / User Stories and community standards, maintains the quality of the product.

The ***secondary objective*** is to:

- Identify and expose all issues and associated risks
- Communicate all known issues to the project team and the community, and ensure that all issues are addressed in an appropriate matter before release. As an objective, this requires careful and methodical testing of the system to ensure that all areas of the system are scrutinized and, consequently, all the found issues (bugs) are dealt with appropriately.

4 Target Test Items

There are several major parts of the system to be tested. From the point of view of their semi-independence and taking into account iteration- and phase-oriented design and development, testing is to be divided into the appropriate several directions depending on the chosen ranking:

1. DDL and DML operations:
    a. Save/retrieve an item using primary key
    b. Lookup using dictionary of keys
    c. Lookup using conditions
2. Manage Quotas
3. Auto-scale
4. Failover
5. Geo Affinity
6. Monitoring

7. Multi data center replication

In the following part of the document we:

A. Initially focus on CPE implementation part (Phases 1 to 3) as the major one for reaching the target goal
B. Then proceed with OpenStack adoption as a final community-oriented part (Phase 4) – that requires specific efforts to be described later in the appropriate sections.

The list of phases looks like:

1. CPE implementation:
    a. Phase 1: MagnetoDB PoC
    b. Phase 2: MagnetoDB Sandbox
    c. Phase 3: MagnetoDB production/operations ready
2. OpenStack adoption:
    a. Phase 4: Complete Community Requirements

In reality though, it means that both just mentioned processes (implementation and adoption) should go in parallel to preserve progressive development flow and to keep tight integration with OpenStack community.

As SMDB takes sufficient time for design and development, its validation is a subject for further detalization during several iterations.

5 Risk Issues (Test-related)

Both User Stories set and the Requirements as well as the type of background DB itself are subjects to changes. So, for each sprint a set of goals and User Stories is to be formed for the following design and development of the specific part(s) of the system. In view of this, a number of visible **initial** risks should be taken into consideration.

1. Lack of (or weak) synchronization between the customer and the community: difference between customer needs and timing, and the community's ones will lengthen the development cycle
2. Gradual but essential extending of system functionality (as a result of point 1) may require additional efforts for validation
3. Depending on the results of R&D tasks and various PoC activities, project scope, plan, amount of tasks may considerably vary and get out of the initial limitations.

6 Overview of Planned Tests

The majority of testing will consist of functional and performance test activities. Except that, there will be usability, reliability tests that are to be grouped into sets for running regularly or on demand during the whole project development process. Ordinary (by default) tests runs will be performed by the CI system on a QA environment.

Taking into account complexity of actions, variety of target configurations and other factors, it is planned to maximize test automation approach. The final goal is to automate 70+% of all the API-level functional tests to run them using an automated test framework based on MagnetoDB client.

On each sprint, a number of newly introduced tests will be added complementing the areas of functional, reliability, performance and other types of tests which are described below.

6.1 Functional Testing

Will be performed against the following features of MagnetoDB:

1. DDL and DML Operations: the most massive group of tests representing operations performed by statements:
    a. Managing Tables
    b. Data reading/modifying
2. Data bulk load
3. Manage Quotas
4. Auto-scale
5. Failover
6. Geo Affinity
7. Monitoring
8. Multi data center replication

The appropriate tests cases will be designed and developed for both manual (on demand) and automatic runs.

6.2 Usability Testing

TBD

6.3 Performance/Load Testing

Will be applied to all the performance-critical and throughput-dependable operations/functionality such as:

- Indexing
- BatchRead and BatchWrite statements
- BulkRead and BulkWrite statements
- Massive data export and import processes

Additionally, there are specific operations and processes (see sprint backlogs) that are to be tested for performance.

The same test approach is planned to be used for performance profiling when the project team will be eliminating ineffective mechanisms and performance bottlenecks using various typical real-life scenarios of system usage.

6.4 Reliability Testing

Is one of key parts of testing procedure and consists of the following steps:

1. Long-run tests under different typical workloads to confirm system's stability
2. Heavy load and critical-mode tests to assure that the system automatically recovers after various critical modes, fall-downs and then keeps functioning properly

See section Performance/Load Testing for the list of operations and functions to check:

- Lack of resources:
    - RAM

    - CPU performance
    - HDD space
    - Network throughput
- Connection breaks and timeouts
- Overall low performance
- Falling down, unresponsive state
- Failover/recovery
- Long-run

6.5 Scalability Testing

Is applied in both ways:

- Implicitly, when the system performs data export/import operations
- Explicitly, when checking SMDB performance in different configurations

The mentioned checks imply running tests that simulate different intensities of work loads

7 Test Approach

To accelerate the process of tests implementation, it was decided to:

- Apply an approach when each developer creates the automated test scenarios for the functionality he works on during this iteration
- Test planning and design will be performed by a dedicated QA Lead

By executing manual and automated tests as soon as feature is ready, test engineer will provide minimal “bugs detection time”. It means that we start to test feature without waiting when all of them are done. It requires accurate planning of tests implementation based on feature implementation plan. At the beginning of a sprint, QA Lead synchronizes a test plan with the Dev team and prioritizes tests.

Tests are run as a part of CI system based on Jenkins. The appropriate job is performed on each successful commit of code into the branch. Each test scenario run is logged, its result is accumulated to be a part of Test Report document. After Test Report is ready, the results are analysed for fails and mistakes to shape the overall status of testing and to perform the required changes into the source code.

On reaching “Feature freeze” sprint milestone, QA engineers are able to run all the tests including system-level and non-functional ones. “Code freeze” declares that making changes to the code is not allowed and this code is ready for final testing and, in most cases, delivery to a customer.

7.1 Test Strategy

QA Lead is responsible for the test processes on the project including test management, monitoring, control, reporting, consultancy, issues lifecycle, etc.

The overall strategy for functional tests is to run tests as soon as possible. It requires that:

- Test planning and design (on all levels) are performed by a dedicated professional QA Lead

- Functional test case design on component and integration levels should be ready (if possible) by the time when the appropriate feature development is completed
- QA engineer (in our approach it is a dev engineer, the author of the feature) starts working on creating the appropriate automated scenarios (using already existing conceptual or detailed test case descriptions) right after completion with the feature implementation
- QA engineer stores all the created scenarios into the test repository dividing all the tests into several folders (stable, in_progress, not_ready) to define voting status for each test according to its current status. See the detailed description of this concept and its purpose in "Organization of tests in CI" document.
- The created test sets are executed on each commit

This strategy works well for component and integration levels of testing. However, for other test types (performance, reliability, supportability, etc) and for system level testing the strategy is different.

First Sprints

As first several projects sprints are dedicated to design and development of basic functionality for SMDB (see Target Test Items), the majority of tests to design, implementation and running will be of functional type. In view of this, vast test automation of this functionality will provide us with a good verification mechanism and then with a strong basis for regression testing.

Performance, load and other test types are applied only for PoC tasks and on a limited basis. The goal is to increase the functional basis of the system.

Further Sprints

All the mentioned test types are applied (see Overview of Planned Tests and Test Levels). Regression testing includes regular (but not for each SMDB build) running of Performance and Reliability tests. The final goal is to prepare the system for functioning in production mode.

7.2 Test Levels

The testing for the SMDB project will undergo several levels of testing:

1. Unit
2. Component
3. Integration
4. System/Acceptance

which are divided in sets depending on characteristics of a specific phase/iteration and its goals. The details of each level are addressed below.

Unit testing will be performed by the developers during implementation of functionality. *Component, Integration, and System testing* is responsibility of the QA team (dev engineers performing test-related tasks).

Below is the generalized sequence of test actions performed on each iteration.

Unit Testing will be done by the developers and will be approved by the development team leader. Proof of unit testing (test case list, sample output, data printouts, defect information) must be

provided by a developer to the team leader before unit testing will be accepted and passed on to the test person. All unit test information will also be provided to the test person.

Integration and System Testing will be performed by the QA team. Build wiil undergo Integration and System testing only after all critical defects, found on the unit test level, were corrected. A build may have some major defects as long as they do not impede testing of the program (i.e. there is a work around for the error).

Acceptance Testing will be performed by either actual end users with the assistance of the test manager and development team leader or the project QA team (as a part of internal SDP). The acceptance test will be done as the next step after completion of the Integration and System testing. Build will enter into Acceptance testing phase after all critical and major defects have been corrected. A build may have one major defect as long as it does not impede testing of the program (i.e. there is a work around for the error). Prior to final completion of acceptance testing all open critical and major defects MUST be corrected and verified.

7.3 Test Cycles Structure

On each sprint, contractor's team delivers a specified set of features described in the sprint scope/backlog.

A sample of sprint-long schedule for QA activities looks like:

1. Tests planning and design
2. Implementation of test scripts and tools. Manual (if applicable) testing of new features
3. Manual and automated regressions test runs for smoke, validation and regression testing
4. Demo preparation

In the same time, during one sprint, there will be several test cycles. Each new SMDB build should sequentially pass (completely or partly) the following cycle of testing:

1. Unit automated tests
2. Automated ***smoke*** tests
3. Manual and automated integration (***feature validation***) tests
4. Automated ***regression*** tests

Unit tests are managed and exclusively supported by Dev Team. They verify code integrity and run continuously as a part of build process. Smoke tests are intended to identify broken build before functional integration tests will start. Integration and regression tests are to verify existing and new functionality.

Each test cycle has its own goal (depending on the situation) like:

- Check that the functionality was implemented and passes positive flows
- Find as many defects as possible
- Verify that all major and medium level defects fixed successfully
- Confirm readiness of the product features to demo
- etc

7.4 Test Tools and Automation

Continuous integration (CI) server is based on Jenkins managed by the development team. It performs several tasks:

1. Creating target builds on commit
2. Running Unit tests
3. Deploying the applications on a lab
4. Executing Component/Integration/System tests
5. Test reporting

Dev team actively introduces automation into validation process. By implementing autotests in parallel with feature completion time, the team sufficiently reduces validation time. However, this approach requires additional efforts from Dev team, such as wide usage of mocks, providing interfaces to new functionality, announcing changes made into existing ones (if needed), etc.

Automation tests include functional, performance/load, reliability, other tests. According to the chosen automation strategy, continuous execution is used to gain maximum benefits.

7.5 Configuration Management and Change Control

<environments> and flow

7.6 Test Coverage

Dev team should keep unit test coverage at reasonable level (see Exit Criteria).

Coverage of integration tests is calculated by building Traceability Matrix (see Test_Cases document on the wiki)

	Test 1	Test 2
Requirement 1		+
Requirement 2	+	+

Traceability matrix is stored in version control system and kept up to date by the QA team.
All requirements should be covered by integration tests.

7.7 Test Scenarios

Test scenarios for validation process are specific steps of executing and evaluating the expected results. Test scenarios and test cases need to be up to date during all the project lifetime. Therefore it should be stored in a version control system or repository.

8 Entry and Exit Criteria

8.1 Entry Criteria

1. Source code of a component/subsystem/system is ready for testing
2. There is a specific environment for performing tests, validations
3. Other test resources are available
4. Requirements (User Stories, Use Cases) are clear, complete, and non-controversial

8.2 Exit Criteria

1. All *critical* and *major* bugs were fixed

2. Performance of Deployment procedure is sufficient
3. Coverage of unit tests is 70+ %
4. Coverage of automated integration tests is 70+ %

8.3 Suspension Criteria and Resumption Requirements

Testing process may be **suspended** in following cases:

1. Test resources (test labs, build environment, client stations) are or become unavailable
2. Amount of bugs found exceed the reasonable limit. Test process becomes ineffective.

9 Deliverables

1. System's source code
2. Documentation
3. etc

10 Schedule

See sprints schedule in the appropriate project wiki section