EETeamJ1

QA Test Plan

Version: 3.0

|  |
| --- |
| **Revision History** |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Description | Author |
| 25/05/2016 | 1.0 | Original Draft | Dilip Dave |
| 10/06/2016 | 2.0 | Added details into test strategy for types of testing | Dilip Dave |
| 26/06/2016 | 3.0 | Modified “Functional Testing” | Dilip Dave |

**1. Introduction**

The document is a high-level overview outlining the test planning for the case study project assignment to be taken up by ***EE Team J1*** group. The objective is to set the test strategy for the particular product releases. This broadly describes the objective, testing scope, testing type, Entry & Exit criteria details in order to deliver adequate quality product.

## Document Objectives

* Define the general strategy and approach that will be incorporated to test the software and evaluate the test results.
* Define the top-level plan that will be used to govern and direct detailed testing for the following test types: Functional, Environmental, Regression and User Acceptance.
* Unit testing be covered as the development activity.
* Provide visibility to stakeholders that adequate consideration has been given to the various aspects of testing, and where appropriate have the stakeholders approve the plan.
* Identify the test environment required for each phase.
* Identify the deliverables that should be targeted by the test phases.
* Identify the modification for and the ideas behind the test areas to be covered.
* Outline the test approach that will be used.

**2. Test Strategy**

Below identification table helps to identify what are different types of testing should be performed for the release.

|  |  |  |
| --- | --- | --- |
| **Type of verification** | **Description** | **Applicable** |
| **Sanity and Smoke testing** | Essential to perform before starting full-blown testing | Y |
| **Functional testing** | Addition of new feature as well as impacted areas of application to be covered while testing in integrated environment | Y |
| **Regression testing** | Essential to perform in order to make sure that the defects are got fixed and other existing application behavior wouldn’t be affected. | Y |
| **Security Testing** | Essential to perform in order to test the security and authentication mechanism implemented in the environment | Y |
| **Non-Functional Testing (Environment Benchmarking High Availability and Failover Testing)** | Essential to perform in order to benchmark multi-node cluster environment and ensure High Availability of multi-node cluster. System will be tested once all the modules are integrated, failover testing will be performed to ensure this. | Y |

**2.1 Definitions**

**2.1.1 Sanity and Smoke testing:**

**Smoke testing** is a subset of Regression testing. Smoke Testing is performed to ascertain that the critical functionalities or components of the environment is working fine. It is executed "before" any detailed functional or regression tests are executed on the software build. The purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application. In Smoke Testing, the test cases chosen cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine. The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing. Smoke testing exercises the entire system from end to end, it’s like a general health checkup.

**Sanity Testing** is done after receiving a software build, with minor changes in code, or functionality or any change in any key component of the cluster. Sanity testing is performed to ascertain that the issues have been fixed and no further issues are introduced due to these changes. The goal is to determine that the multi node cluster environment works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing. The objective is "not" to verify thoroughly the new functionality or workflow, but to determine that the developer has applied some rationality (sanity) while producing the workflow or feature. Sanity testing exercises only the particular component of the entire system. Sanity Testing is like specialized health check up

**2.1.2 Functional Testing:**

Addition of new feature as well as impacted areas of application to be covered while testing in integrated environment. This is the trickiest phase, as functional implementation is very unique for each implementation. Here we present an approach which could be applied based on black box testing methodology and covers both flavors of Big Data system implementations (batch and real time). We would be covering the following while testing the sample application-

* Schema validation
* Verification of data ingestion from source to destination works as expected
* Verification that the data is PROCESSED correctly according to various business requirements and rules

**2.1.2.1 How the Functional testing will be performed?**

Data is stored in HDFS/Hive tables for movie lens application after processing, so native tools provided by vendors are used for checking excepted outcome against the stored data. Schema is also validated after data is ingested into the cluster.

* Verify schema of the incoming data with the schema defined within the application. The idea is to validate the existence of key columns and the data present in those columns. We will use Hive/Pig queries to verify the schema and application data.
* Checking number of processed records can also be validated easily by using queries language (Hive/Pig) provided by the native data store (wrapping up the queries in the form of scripts which could be rerun and scheduled). Any discrepancy at this level suggests incorrect application of business logic at the processing engine level.

**2.1.3 Regression Testing:**

The purpose of regression testing is to ensure that changes such as installation of new service/component, change in data nodes, commissioning/de-commissioning of any node, ingestion of bulk data or any change in cluster have not introduced new faults. One of the main reasons for regression testing is to determine whether a change in one part of the cluster or environment affects other parts of the hadoop cluster.

**2.1.3.1 How Regression Testing will be performed**

Common methods of regression testing include rerunning previously completed tests and checking whether cluster or environment or application behavior has changed and whether previously fixed issues have re-emerged. Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of test cases needed to adequately cover a particular change.

**2.1.4 Security Testing:**

The purpose of security testing is to ensure that the security mechanisms that are implemented to the multi-node cluster environment are applied successfully and are working as expected. In our assignment Kerberos security will be enabled to authenticate users and services of cluster. Kerberos is a distributed authentication service that allows a process (a client) running on behalf of a principal (a user) to prove its identity to a verifier (an application server, or just server) without sending data across the network that might allow an attacker or the verifier to subsequently impersonate the principal. Kerberos optionally provides integrity and confidentiality for data sent between the client and server. The verification of all the services , components and users will be done after applying Kerberos weather the security mechanism is working as expected or not.

**2.1.5 Non-Functional Testing:**

Non-functional testing plays a key role in ensuring the scalability of the application. Non-functional testing identifies performance bottlenecks and validates the non-functional requirements. System benchmarking, high availability and fail-over mechanism to ensure that the data is processed seamlessly across the entire system when node(s) is/ are down or switched from one to another.

* **System Benchmarking:** Without a consistent and methodical benchmarking process, engineers at times do not foresee the potential performance bottlenecks, while spending resources on testing unimportant functionalities. Having a common benchmarking process helps them to correctly test the systems and reduce the production performance issues. Hadoop comes with in-built benchmarks which can used directly to test performance of Hadoop file system. Hadoop comes with in-built benchmarks which can used directly to test performance of Hadoop file system like MRBench, TestDFSIO, etc.
* **High Availability:** High Availability is another important need of any application developed to process real-time or batch data. The clusters should be verified for high availability for data processing. In case of any failure or node downtime, it should not affect application processing and yet provide useful insights. So it is required to test the application and cluster for high availability by intentionally making a node down for some time during data processing and to ensure its proper working.
* **Failover Testing:** Hadoop installations typically have dozens or hundreds, if not thousands of nodes, each with four or more disk drives. Some disk drives or even entire nodes will fail every week. HDFS architecture is designed to detect these failures and automatically recover to proceed with processing despite these failures. Failover testing validates the recovery process and ensures data processing continues correctly when switched to other data nodes. As Hadoop involves radically different architectures from traditional information-processing systems, verification of the design and configuration is critical. The NameNode, in particular, remains a single point of failure and should be maintained in a high availability configuration and submitted to appropriate failover testing.

**4. Test Scope**

A Hadoop data lake will be created with 4 node cluster setup. The team will develop a reference use case using multiple technologies. This section describes the features which would be covered during the exercise along with features which would not be covered.



# Things needs to be covered

1. Following technologies will be used to create the environment
   1. Apache Hadoop/Yarn
   2. Ambari
   3. Pig
   4. Hive
   5. Kerberos (Security)
   6. Ranger
   7. ELK
   8. Oozie
   9. Spark
2. A sample movie lens application will be created to use the movie lens data for performing various operations. Few high level features that will be implemented as part of this activity-
   1. List all the movies and the number of ratings
   2. List all the users and the number of ratings they have done for a movie
   3. List all the Movie IDs which have been rated (Movie Id with at least one user rating it)
   4. List all the Users who have rated the movies (Users who have rated at least one movie)
   5. List of all the User with the max,min,average ratings they have given against any movie
   6. List all the Movies with the max,min,average ratings given by any user
3. Monitoring the multi-node cluster environment (Nagios/Ganglia)

**5.** **Test assumptions**

* Test environment is available.
* User stories, Acceptance Criteria is clearly mentioned to avoid disparity between Test Cases and actually developed functionality.
* Exploratory Testing would be carried out once the build is ready for testing.
* Performance testing will not be considered for this estimation.
* All the defects would come along with snapshot.
* Test case design activities will be performed by QA.
* Test environment and preparation activities will be owned by Dev Team.
* Dev team will provide Defect fix plans based on the Defect meetings during each cycle to plan. The same will be informed to Test team prior to start of Defect fix cycles.
* There is no environment downtime during test due to outages or defect fixes.
* The system will be treated as a black box; if the information shows correctly online and in the reports, it will be assumed that the database is working properly.
* Benchmark testing will start only when QA environment is in stable state. (No Blocker, Critical, Major bugs is in open state).

# 6. Risk & Mitigation

|  |  |
| --- | --- |
| **Risks** | **Mitigation** |
| Application delivery to QA may delay due to major defects found during integration testing or any other requirement changes this may impact the schedule. | User Story, Acceptance Criteria should be updated about any changes in schedule delivery of the application well in advance. |
| If User Stories / Acceptance Criteria on which test cases will be based are not clearly defined. This may result in too many defects and conflict with development team. | Any dispute in defect resolution will be sorted out in a meeting with Lead. |
| Delayed Testing Due to new Issues in Sprint related to any specific user story. | During testing, there is a good chance that some “new” defects may be identified and may become an issue that will take time to resolve.  There are defects that can be raised during testing **because of unclear document specification**. These defects can yield to an issue that will need time to be resolved.  If these issues become showstoppers, it will greatly impact on the overall project schedule.  If new defects are discovered, the defect management and issue management procedures are in place to immediately provide a resolution. |

# 7. Entry & Exit Criteria

Entry and exit criteria are flexible benchmarks. If they are not met, the test team will assess the risk, identify mitigation actions and provide a recommendation. All this is input to the project manager for a final “go-no go” decision.

* **Entry criteria of release**

|  |  |
| --- | --- |
| **Items** | **Criteria** |
| Document | * Requirements are finalized and approved * Functional Specification * Design document * Approved test approach , test plan document |
| Test case | * All expected results are documented with the User Stories * Execution result of Unit test cases from development * QA Test data and checklist completed * Environment configured and setup properly |

* **Exit criteria of release**

|  |  |
| --- | --- |
| **Items** | **Criteria** |
| Product management | * High priority & severity bug review as per the need. |
| Test completion | * 100% Test Cases executed. * Defect verification completion(All Fixed issues are verified and closed) * All remaining defects are either cancelled or documented as Change Requests for a future release * No Critical / Major bugs should be in New /Open / Fixed state. * Bugs with Status as ‘No Fix Needed’ & ‘More Information Required ‘should be closed. |