instaclustr

Apache Cassandra Technology Test Plan

Document Identifier: ICF_CTP
Document Version: 1.1
Issue Date: 15 NOV 2021

Table of Contents

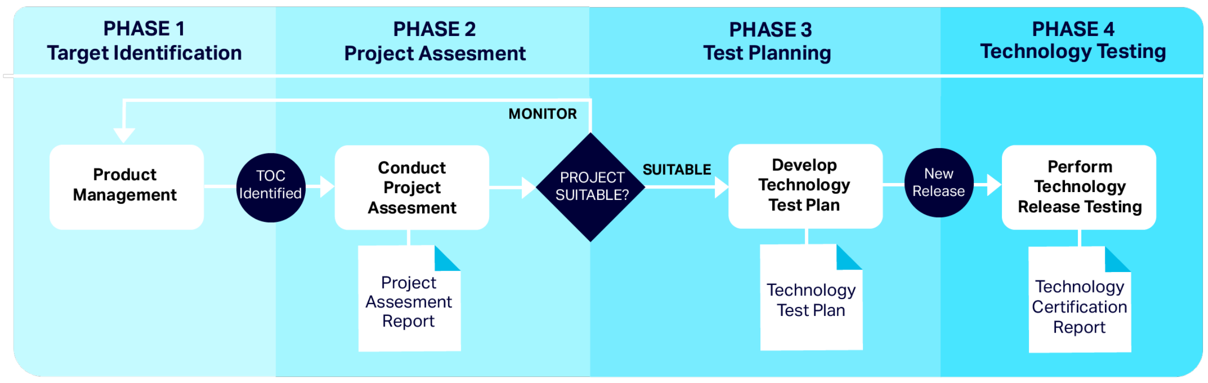
1 Introduction

1.1 Overview

The **Instaclustr Certification Framework for Open Source Software (ICF-OSS)** provides an analytical basis for selecting and recommending various open source technologies for production usage. This is achieved by:

a) assessing various open source projects to gain a level of assurance that the project has the foundation and capability to build and sustain production-grade software, and

b) testing specific versions of open source technologies for function, performance, and interoperability.



1.2 Documentation Identification Information

Identifier	ICF_CTP – Apache Cassandra
Description	This document is the Technology Test Plan (TTP) for Apache Cassandra as part of the Instaclustr Certification Framework for Open Source Software (ICF-OSS).
Target of Certification	Apache Cassandra http://cassandra.apache.org/

1.3 Copyright Statement

2 Introduction

2.1 Overview

This document is a Technology Test Plan that has been developed as Phase 3 of the certification process for Apache Cassandra.

Apache Cassandra is a distributed, wide column store, NoSQL database management system. The database is open source and is managed and governed by the Apache Software Foundation under its license agreements.

The database is designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. The software provides support for clusters that can span multiple data centers with asynchronous masterless replication.

2.2 Purpose

The purpose of these testing efforts is to provide a level of assurance that a specific release of Apache Cassandra is suitable for inclusion in the Instaclustr Managed Platform and is suitable for production integration and deployment.

The testing efforts will result in a certification report that provides the results of the testing effort and identify any performance regression, or potential caveats, as well as some further guidance on how to use the release.

2.3 Scope

The scope of any testing effort is for a specific release or version of Apache Cassandra software. Any limitations or recommendations will be specified in the certification report for specific releases of Apache Cassandra.

3 Functional Testing

3.1 Overview

The aim of functional testing for Apache Cassandra is to ensure that the version under test has had all the features and components tested in accordance with the developers.

Functional testing is achieved using the following test methods:

- The unit tests that are included for each component of the software.
- The distributed tests that test the distributed functionality of the software.

For any failures that are identified during the functional testing process a detailed analysis of the risks associated with these functions will be provided.

3.2 Unit Tests

Unit tests are maintained as part of the Apache Cassandra project. These tests are focused on the functionality of the various components of the Apache Cassandra codebase within a specific instance.

The unit tests will be performed using the following testing procedures:

1. All unit tests provided with the release of Apache Cassandra that is under test will be run by the certification team.
2. Each unit test will have a result of either PASS or FAIL.
3. If a unit test has a result of FAIL then further explanation and relevant details will be provided.
4. Both the “All” and the “long” tests are to be performed.

3.3 Distributed Tests

The distributed tests (also known as DTests) are maintained as part of the Apache Cassandra project. These tests are specifically focused on the functionality of the software in a multi-node cluster. For our certification process we run the distributed tests branch that corresponds to the relevant release of Apache Cassandra.

The distributed tests will be performed using the following testing procedures:

1. All distributed tests that correspond to a release of Apache Cassandra will be run by the certification team.
2. Each distributed test will have a result of either PASS or FAIL.
3. If a distributed test has a result of FAIL then further explanation and relevant details will be provided.

4 Performance Testing

4.1 Overview

The aim of performance testing for Apache Cassandra is to determine that the version being tested delivers suitable performance under stress and with production-grade workloads.

Performance testing is delivered through the following specific methods:

- Stress testing aims to test the performance of the software on various infrastructure under increasing workloads.
- Soak testing aims to test the performance of the software under sustained production-grade workloads.

4.2 Stress Tests

The aim of the stress testing is to benchmark the read and write latency of various infrastructure types running Apache Cassandra under a range of relevant workloads.

4.2.1 Test Configurations

The baseline infrastructure for the testing efforts is performed on AWS and are as specified in the following table.

Identifier	Node Type	Cluster Size	Node Specifications
CS01	AWS I3EN.XL	3 Nodes	• 4 CPU cores • 32 GB RAM • 2328 GB SSD
CS02	AWS I3.2XL-V2	3 Nodes	• 8 CPU cores • 61 GB RAM • 1769 GB SSD
CS03	AWS I3EN.XL	9 Nodes	• 4 CPU cores • 32 GB RAM • 2328 GB SSD

4.2.2 Test Payload

The test payload is created by providing a stress `Spec yaml`, and the list of newly created nodes, to `cassandra-stress`. From here, we run `cassandra-stress` with the required parameters to perform the current test run.

We defined the column specifications for the test run inside a table named **typestest**. For the full YAML file, see 5.2.3Annex A- Performance Testing Definitions.

The following table describes the operations that are used. Each operation type is performed for each of the infrastructure tests and the following WRITE operation sizes are used:

- **SMALL:** Approx. 0.6kb per row
- **MEDIUM:** Approx. 12kb per row

WRITES	Writes	partitions: fixed(1) batchtype: UNLOGGED select: uniform(1..10)/10
READS	A combination of the two queries at a rate of 10:1	Simple: select * from typestest where name = ? and choice = ? LIMIT 1 Range: select name, choice, uid from typestest where name = ? and choice = ? and date >= ? LIMIT 10
MIXED	A combination of writes, simple reads, and range reads at a ratio of 10:10:1	Writes: 10 Simple: 10 Range: 1

4.2.3 Test Procedure

The stress tests will be performed using the following procedures:

1. Fill the disk to approximately 5 times the maximum Java heap size for a cluster using a combination of “Small” and “Medium” size writes. This is done to ensure that the entire data model is not sitting in memory, and to give a better representation of a production cluster.

2. Wait for the cluster to finish all compactions. This is done to ensure that all clusters are given the same starting point for each test run, to make test runs comparable and verifiable.

3. For each both **SMALL** and **MEDIUM** writes, and waiting for compactions to finish between each step:

    a. Perform a series of stress tests with fixed operations per second (70% of the maximum throughput for a 3.11.8 cluster) to assess latency:

        i. Perform writes for 1 hour at a given OPS/Second to determine write latency.

        ii. Perform reads for 1 hour at a given OPS/Second to determine if read latency has changed.

        iii. Perform a combination of reads and writes at a given OPS/Second to determine mixed latency.

Instance	# of Nodes	Write OPS/Sec		Reads OPS/Sec		Mixed OPS/Sec	
		Small	Medium	Small	Medium	Small	Medium
i3en.xl	3	14974	238	9437	8373	7878	387
i3.2xl-v2	3	22085	230	13524	12050	10842	371
i3en.xl	9	34031	779	27226	22248	23988	1849

a. Perform multiple 30-minute stress tests on the cluster to determine the maximum number of operations per second before a cluster is overloaded. The cluster will be considered overloaded if any of three conditions are met:

   i. The latency of the operation is above the threshold of 5 ms median latency for writes or 20 ms median latency for reads and mixed loads.

   ii. If the average OS load for a cluster, normalised to the number of CPU cores available, over the last 15 minutes of the test is greater than 4.

   iii. If the test is using a medium-sized payload and the average slope of the pending compactions over the last 15 minutes of the test is greater than 0.004 (which indicates that the cluster is not maintaining currency with compactions and the load is not sustainable.)

b. The following tests are then run to measure maximum throughput in an overloaded state:

   i. Perform a write operation in 30-minute test runs, increasing threads until a median write latency of 5ms is reached.

   ii. Perform a read operation in 30-minute test runs, increasing threads until a median read latency of 20ms is reached.

   iii. Perform a combination of reads & write operation in 30-minute test runs, increasing threads until a read median latency of 20ms is reached.

4.3 Soak Tests

The aim of soak testing is to test how a cluster will perform under sustained production like conditions.

This test is performed by running fixed operations per second with a mixed workload for 24 hours, while simulating node outages and maintenance operations (repairs) to confirm that Apache Cassandra operates as expected under typical production conditions.

The CS03 configuration is used for infrastructure and the following testing procedure is used:

Each disk of each node is filled to approximately 5 times the available JVM max
heap size.

1. Cluster compactions are performed.
2. Once compactions are completed, run a mixed load performance test for 24 hours with the following activities:
   a. Remove node 1 in the first rack every hour for 5 minutes.
   b. Remove an additional node in the first rack every 2 hours for 5 minutes.
   c. Remove a third node in the first rack every 3 hours for 5 minutes
   d. Run a full repair after 12 hours.
3. Collect and record latency, operations per second, failures, and Garbage Collection metrics.

5 Integrations Testing

5.1 Overview

The purpose of our integration testing is to confirm that the version of Cassandra under test has not broken any compatibility with the major Cassandra drivers. In this way, we are able to indicate that your application should work with minimal changes.

5.2 Driver Tests

Driver testing aims to confirm interoperability between major Cassandra language drivers and the new release of Apache Cassandra under testing.

5.2.1 Test Configuration

We create a 3-node cluster which runs the version of Cassandra under test. We then open the Cassandra firewall port to our test box and attempt to perform operations on the cluster from various Programming Languages.

5.2.2 Test Scope

The following drivers are to be tested:

1. CQLSH
2. The DataStax Java Driver
3. The DataStax Python Driver
4. The DataStax Ruby Driver
5. The Gocql Go Driver
6. The DataStax Scala Driver
7. The ALIA Clojure Driver

5.2.3 Test Procedures

For each of the drivers to be tested the following test procedures are used:

1. Connect to the new 3 node cluster which is running the Cassandra Version under test, and the language and driver we are testing
2. Create a basic keyspace with RF 3
3. Create a basic table with two text columns, and a decimal column. The first text column is the primary key
4. Insert a small row into the table which contains two text columns, and a decimal column
5. Query the written data by its primary key
6. Confirm that the queried data matches the written data.

If the driver version is able to successfully complete all operations, then the test result will be PASS. If the connection does not work as expected it will be marked as FAIL and details will be provided in the certification report.

6 Security Vulnerability Assessment

6.1 Overview

The purpose of this assessment is to determine whether the Java libraries used by the version of Cassandra under test contain any known security risks.

6.2 Methodology

Scanning is performed using the open source tool Dependency-Check (https://github.com/jeremylong/DependencyCheck) which compares the dependencies in the project with CVE (Common Vulnerabilities and Exposures) identified in the National Vulnerability Database.

CVEs found using the tool will be classified as follows:

- False positive if the tooling has identified a CVE for a different version or incorrect dependency
- Not vulnerable if the dependency is correct but the CVE does not apply base on the usage of the dependency
- Partially vulnerable if the CVE applies based on the usage but Cassandra can be configured to avoid the CVE
- Vulnerable if Cassandra is vulnerable to the CVE

Along with the classification we will provide justification for each of the classifications.

Annex A: Performance Testing Definitions

```
keyspace: stresscql2<Small|Medium>

keyspace_definition: |
  CREATE KEYSPACE stresscql2<Small|Medium> WITH replication =
       {'class': 'NetworkTopologyStrategy', <datacentre>': 3};

table: typestest

table_definition: |
  CREATE TABLE typestest (
       name text,
       choice boolean,
       date timestamp,
       address inet,
       dbl double,
       lval bigint,
              ival int,
       uid timeuuid,
       value blob,
       PRIMARY KEY((name,choice), date, address, dbl, lval, ival, uid)
  ) WITH compaction = { 'class':'LeveledCompactionStrategy' }
    AND comment='A table of many types to test wide rows'

columnspec:
  - name: name
   size: fixed(48)
   population: uniform(1..<Number of Writes>)
  - name: date
   cluster: uniform(20..1000)
  - name: lval
   population: gaussian(1..1000)
   cluster: uniform(1..4)
  - name: value
   size: fixed(<Operation Size in Bytes>)

insert:
  partitions: fixed(1)
  batchtype: UNLOGGED
  select: uniform(1..10)/10
queries:
   simple1:
       cql: select * from typestest where name = ? and choice = ? LIMIT 1
       fields: samerow
        range1:
       cql: select name, choice, uid  from typestest where name = ? and choice = ?
and date >= ? LIMIT 10
       fields: multirow
```