Test Plan

|  |  |
| --- | --- |
| **Name** | Data SDET Exercise |
| **Description** | Test Plan |
| **Document Owner** | Komal Kumar |
| **Baseline version link** | In Progress |
| **Baseline approved date** |  |
| **Attestation date** |  |
| **Workstream** | Data |

**Table of contents:**

Introduction

Overview

Scope

Components to be tested

Testing Types

Level of testing

Testing Task

Tech Stack and Tooling

Test Deliverables

Risk and Contingencies

References

**Introduction:** This test plan document is for the Kafka-Exercise-SDET and will cover the granular details of testing of Kafka producer and consumer communication. This Test Plan will set-out and document the details for the testing effort in terms of scope, approach, resources, schedule, and associated risk.

**Overview:** Kafka-Exercise-SDET is a Java project that create a messaging system build on Kafka in which a producer publishes the message in the format of {id, message} and consumer reads it. Messages persist in the Postgres DB in 'exercise' table in 'test\_db' database. Secondly, to verify the communication, BDD test framework is written that verify the completeness and correctness of the message flow from producer to consumer and then DB. Kafka system and test framework is dockerized using the docker-composer.

**Scope:** The scope includes the Producer, Consumer and Postgress DB storage and Communication. Validation will focus on data consistency, data completeness, data duplication and Message-Key consistency validation between Producer and Consumer.

**Components to be tested:** Producer, Consumer, Broker, Topics, Postgress DB

**Testing Types:** Following are the testing types

* Producer Testing – Verify the message-Key acknowledgement and existence from the kafka broker by producing a record to a topic.
* Consumer Testing – We can verify the record(s) fetched from the topics by consuming from the topics. Here we can validate/assert the message-Key against the producer message-key.
* Data Testing – Validate the data in the DB table against the produced message by validating the correctness, completeness and duplication.
* Volume Testing - To verify the behaviour under large number of data sets.

**Level of testing:**

Unit Testing – This to check if software components are fulfilling functionalities or not.

Integration - This to checks the data flow from one module to other modules.

System Testing - To evaluates both functional and non-functional needs for the testing.

Volume Testing - To verify the behaviour under large number of data sets.

Acceptance Testing - To checks the requirements of a specification or contract are met as per its delivery.

**Testing Task:** Following are the testing tasks

* Feature Files creation
* Test Scenarios creation
* Create Steps classes
* Create BDD Assertion
* Create Data Quality step definition
* Create the test data

**Tech Stack and Tooling:** Following are the tech stack will be used to validate the system.

* **Java –** Programming language to write the steps classes.
* **Maven –** Build automation and project management tool.
* **Cucumber –** Cucumber is a software tool that supports behavior-driven development.
* **Gherkin –** Gherkin uses a set of special keywords to give structure and meaning to executable specifications.
* **Junit –** Junit will be used to create the runner class to start the test execution.
* **Postgres –** Postgress databases is used to store the data that created during testing of producer and consumer and for validation.
* **Docker –** Docker will be used to run the application and bdd in containers.
* **Docker-Compose –** Will use docker-compose to compose the activity steps of the system in the a package.
* **Intellij –** An IDE for system development and to write the java code.

**Test Deliverables:** Deliverable are

* BDD Feature File
* Java step definition and assertions
* Test Case Report
* Traceability Report
* Defect Report
* Test Summary and coverage Report

**Risk and Contingencies:** Test Env is not available, Producer and Consumer API are not working, Test Data is not available.

**References:**