KIP-25 - System test improvements

Status

Current state: *Accepted*

Discussion thread: here

JIRA: JIRA for adding initial ducktape tests, JIRA for tracking porting of existing tests

Released: 0.8.3

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

The existing system tests have been around for a few years, and technical debt has accrued. As Kafka matures, our system tests need to become first-class citizens so we can be more confident in the stability and performance of the overall system as well as compatibility from release to release.

From “things we can test” perspective, we need convenient ways to do things like:

- Sanity checks for normal conditions and operations (e.g. soft bouncing of services)
- Failures (e.g. bounce/hard bounce/chaos tests)
- High load
- Network errors or failures (partitions, loss, or high latencies)
- Run against different combinations of versions to test version compatibility.

From a usability perspective, we need:

- Possible to run ‘in the cloud’, locally on a single machine, or locally on a cluster of machines without too many tweaky config changes.
- Can run all tests with a single command (i.e. reasonable test discovery).
- Possible to run multiple tests in parallel on different machines.
- Nice reporting, with regression checks against performance statistics.
- Readable, well structured code to encourage community members to develop tests.

Current system tests are:

- Unstable - relatively little maintenance effort has gone in, and many are now redundant or simply too unstable to run successfully.
- Hard to configure and run, and therefore rarely run.
- Incomplete - there are a number of gaps in coverage to fill.
- Difficult to write new tests - overall community feedback has been that the existing framework presents a significant barrier to contribution of new tests.

Public Interfaces

N/A

Proposed Changes

What do we want out of system tests going forward?

The basic proposal is to

1. Develop new tests against the ducktape testing framework/library with the goal of filling out existing gaps in coverage.
2. Gradually replace existing system_tests with tests using ducktape with the goal of removing the old system_test directory.

So why ducktape? To some extent, this is arbitrary, but there aren’t many existing options for running distributed system tests in a flexible way. We are developing ducktape itself as well as system tests against it at Confluent, and based on our experience with it so far, we think it represents a change in the right direction:

- Using the built-in support for Vagrant, tests are runnable locally and on aws
- Has test discovery, decent reporting, and reasonably easy to automate (see http://testing.confluent.io/confluent_platform/latest for example)
- Makes it simple to dynamically setup and tear down a variety of services on a cluster.
- Automatically collects service logs for post-mortem analysis.
- Individual tests can be fairly small since much of the boilerplate involved in dealing with service clusters is taken care of by Service classes and ducktape itself

Some examples of tests are here

Compatibility, Deprecation, and Migration Plan

What impact (if any) will there be on existing users?

- Only impacts users interested in writing or running system tests.

If we are changing behavior how will we phase out the older behavior?

- Begin with a patch that adds new system test directory as a sibling of the old one. This directory will have bootstrap scripts for Vagrant, a few specific 'services' such as KafkaService and ZookeeperService, and a few new system tests.
- Make sure provisioning and setup scripts are in a state where everyone in the community can reasonably easily run the full test suite on their own testbed.
- Once test coverage with new tests is roughly a superset of old system test coverage, old system_tests directory will be removed. Roadmap for this is here.
- Existing system_tests roughly cover the following, and new tests should cover these categories before deleting the existing directory:
  - Producer performance (producer_perf, which has wrappers for kafka.tools.ProducerPerformance)
  - Various correctness tests in the face of broker failover (replication_testsuite)
  - Consumer offset management in the face of consumer failure (offset_management)
  - Mirror maker tests which propagate data across broker clusters in the face of various types of failure.

Rejected Alternatives

The other testing tool we considered was Zopkio. A viable option would have been to contribute the features we need to Zopkio, but ultimately we didn't think it was the best choice for a few reasons, including:

- Hard to parallelize tests runs
- Unclear ultimate direction of the project

A table comparing features is here.