Kafka System Tests

Introduction

System Test is a Python based regression test framework to run system testing for Kafka. This document is intended to serve as a quick start guide. If you are a Kafka developer and would like to run a sanity test before checking in your change, you may just need to read the following sections:

- Running System Test
- Specify what test cases to run
- Test Report Interpretation

Quick Start

Running System Test

```
1. Check out kafka codebase:

   a. ~ $ git clone https://git-wip-us.apache.org/repos/asf/kafka.git
   b. ~ $ cd <kafka>

2. Under <kafka>, build kafka

   a. <kafka> $ ./gradlew jar

3. Set JAVA_HOME environment variable (optional, but recommended):

   a. export JAVA_HOME=/usr/java/jdk1.7.0_67

4. Make sure that you can ssh to localhost without a password.
5. Under <kafka>/system_test, execute the following command to start System Test :

   $ python -u -B system_test_runner.py 2>&1 | tee system_test_output.log
```

Specify what test cases to run

```
System Test reads <kafka>/system_test/testcase_to_run.json for a list of test cases to run.

1. The following is the configuration of the file out of the box :

{
    "ReplicaBasicTest"   : [
        "testcase_0001"
    ]
}

2. To run testcase_0002 as well in ReplicaBasicTest, modify the file to look like the following :

{
    "ReplicaBasicTest"   : [
        "testcase_0001",
        "testcase_0002"
    ]
}

3. To run testcase_5001 as well in MirrorMakerTest, modify the file to look like the following :

{
    "ReplicaBasicTest"   : [
        "testcase_0001",
        "testcase_0002"
    ],

    "MirrorMakerTest"   : [
        "testcase_5001"
    ]
}

4. To run all test cases (100+), you may copy testcase_to_run_all.json into testcase_to_run.json as shown below
:

    $ cp <kafka>/system_test/testcase_to_run_all.json <kafka>/system_test/testcase_to_run.json
```

Test Report Interpretation

```
# =============================================
#
#         Test results interpretations
#
# =============================================

1. PASSED case - A PASSED test case should have a test result similar to the following :

_test_case_name  :  testcase_0201
_test_class_name  :  ReplicaBasicTest
arg : bounce_broker  :  true
arg : broker_type  :  controller
arg : message_producing_free_time_sec  :  15
arg : num_iteration  :  3
arg : num_messages_to_produce_per_producer_call  :  50
arg : num_partition  :  3
arg : replica_factor  :  3
arg : signal_type  :  SIGTERM
arg : sleep_seconds_between_producer_calls  :  1
validation_status  :
     No. of messages from consumer on [test_1] at simple_consumer_test_1-0_r1.log  :  711
     No. of messages from consumer on [test_1] at simple_consumer_test_1-0_r2.log  :  711
     No. of messages from consumer on [test_1] at simple_consumer_test_1-0_r3.log  :  711
     No. of messages from consumer on [test_1] at simple_consumer_test_1-1_r1.log  :  700
     No. of messages from consumer on [test_1] at simple_consumer_test_1-1_r2.log  :  700
     No. of messages from consumer on [test_1] at simple_consumer_test_1-1_r3.log  :  700
     No. of messages from consumer on [test_1] at simple_consumer_test_1-2_r1.log  :  604
     No. of messages from consumer on [test_1] at simple_consumer_test_1-2_r2.log  :  604
     No. of messages from consumer on [test_1] at simple_consumer_test_1-2_r3.log  :  604
```

```
     Unique messages from consumer on [test_1]  :  2000
     Unique messages from producer on [test_1]  :  2000
     Validate for data matched on topic [test_1]  :  PASSED                                  <----------
     Validate for data matched on topic [test_1] across replicas  :  PASSED                  <---------- All
validations
     Validate for merged log segment checksum in cluster [source]  :  PASSED                 <----------   PASSED
     Validate index log in cluster [source]  :  PASSED                                       <----------


2. FAILED case - A FAILED test case is shown below with data loss in topic test_1 :

_test_case_name  :  testcase_5005
_test_class_name  :  MirrorMakerTest
arg : bounce_leader  :  false
arg : bounce_mirror_maker  :  true
arg : bounced_entity_downtime_sec  :  30
arg : message_producing_free_time_sec  :  15
arg : num_iteration  :  1
arg : num_messages_to_produce_per_producer_call  :  50
arg : num_partition  :  2
arg : replica_factor  :  3
arg : sleep_seconds_between_producer_calls  :  1
validation_status  :
     Unique messages from consumer on [test_1]  :  1392                                     <------
     Unique messages from consumer on [test_2]  :  1400                                            |
     Unique messages from producer on [test_1]  :  1400                                            |
     Unique messages from producer on [test_2]  :  1400                                            |
     Validate for data matched on topic [test_1]  :  FAILED                                  <--------- FAILED
because of data matched issue on topic "test_1"
     Validate for data matched on topic [test_2]  :  PASSED
     Validate for merged log segment checksum in cluster [source]  :  PASSED
     Validate for merged log segment checksum in cluster [target]  :  PASSED


3. Skipped case will have result similar to the following (No validation status details) :

_test_case_name  :  testcase_0201
_test_class_name  :  ReplicaBasicTest
arg : bounce_broker  :  true
arg : broker_type  :  controller
arg : message_producing_free_time_sec  :  15
arg : num_iteration  :  3
arg : num_messages_to_produce_per_producer_call  :  50
arg : num_partition  :  3
arg : replica_factor  :  3
arg : signal_type  :  SIGTERM
arg : sleep_seconds_between_producer_calls  :  1
validation_status  :
```

Test Case Description

testcase description

Misc

Directory Structure Overview

```
<kafka>
  |- /bin
  |- /config
  |- /contrib
  |- /core
  |- /lib
  |.
  |.
  |.
  |- /system_test
       |- system_test_runner.py             # Main script to start System Test
       |
```

```
       |- /utils                              # This is a directory that contains all helper classes / util
functions for system test
       |    |- kafka_system_test_utils.py     # utilities specific to Kafka system testing   (e.g. Kafka test
cases data loss validation)
       |    |- replication_utils.py           # utilities specific to replication testing    (e.g. Leader
election log message pattern)
       |    |- setup_utils.py                 # generic helper for system test setup         (e.g. System Test
environment setup)
       |    |- system_test_utils.py           # utilities for generic testing purposes       (e.g. reading JSON
data file)
       |    |- testcase_env.py                # testcase environment setup                   (e.g. data
structure initialization such as brokers-pid mapping)
       |    |- metrics.py                     # utilities for metrics collection             (to be completed)
       |    |- pyh.py                         # from http://code.google.com/p/pyh            (open source)
       |
       |- cluster_config.json                 # this file contains the following properties:
       |                                      #   1. what entities (ZK, Broker, Mirror Maker, Migration Tool,
Producer, Consumer) should be running
       |                                      #   2. which cluster (source or target)
       |                                      #   3. where they should be running (physical nodes)
       |
       |- /replication_testsuite
-------------------------------------------------------------------------------------------------
       |    |- replica_basic_test.py                                                          |   this
block is one testsuite      |
       |    |- /config                                                                        |   for
"replication_testsuite"      |
       |         |
|-----------------------------------|
       |         |- server.properties                    # as a TEMPLATE for server.
properties                                  |
       |         |- zookeeper.properties                 # as a TEMPLATE for zookeeper.
properties                                |
       |
|
|
       |         |- (migration_consumer.properties)      # only in
migration_tool_testsuite                                          |
       |         |- (migration_producer.properties)      # only in
migration_tool_testsuite                                          |
       |         |- (mirror_maker_consumer.properties)   # only in
mirror_maker_testsuite                                            |
       |         |- (mirror_maker_producer.properties)   # only in
mirror_maker_testsuite                                            |
       |
|
|
       |         |-
/testcase_0001
|
       |         |    |- (config)                        # generated when this testcase is executed in system
test runtime              |
       |         |    |    |- (server.properties)        #   by overriding the TEMPLATE of server.properties,
zookeeper.properties      |
       |         |    |    |- (zookeeper.properties)     #   with new values from testcase_0001_properties.
json                          |
       |         |
|
|
       |         |    |- (logs)                          # generated when this testcase is executed in system
test runtime              |
       |         |
|
|
       |         |    |- testcase_0001_properties.json   # this file contains new values to override the
default settings of            |
       |         .                                      #   various entities (e.g. ZK, broker, producer,
mirror maker, ...)            |
       |         .                                      #   such as "log.segment.bytes", "num.partitions",
"broker.id", ...            |
       |
```

```
.
|
       |         |-
/testcase_NNNN
|
       |              |-
(config)
|
       |              |-
(logs)
|
       |              |- testcase_NNNN_properties.
json                                                                                    |

|
|
       |
-------------------------------------------------------------------------------------------------------------
---------------
       |
       |- /mirror_maker_testsuite
       |    |- mirror_maker_test.py
       |    |- /config
            .
            .
            .
```

How does it work

```
<kafka>
  |- /bin
  |- /config
  |- /contrib
  |- /core
  |- /lib
  |.
  |.
  |.
  |- /system_test
       |
       |- system_test_runner.py
       |
       |  # system_test_runner.py is the main script to start system test as follows :
       |  #
       |  #   1. for each test suite directory (XXXX_testsuite) under systemTestEnv.SYSTEM_TEST_BASE_DIR
(<kafka>/system_test) :
       |  #   2.     get a list of module scripts (*.py) in test suite directory
       |  #   3.     for each file in the list of module scripts :
       |  #   4.         get class name from the Python module script
       |  #   5.         retrieve corresponding suite & module names from class name
       |  #   6.         dynamically load the module and start the test class
       |  #   7.         save each test case result in systemTestEnv.systemTestResultsList
       |  #   8. for each result in systemTestEnv.systemTestResultsList :
       |  #   9.     print result
       |
       |- /utils      # This is a directory that contains all helper classes / util functions for system test
       |
       |    |- kafka_system_test_utils.py    # utilities specific to Kafka system testing   (e.g. Kafka test
cases data loss validation)
       |    |- replication_utils.py          # utilities specific to replication testing    (e.g. Leader
election log message pattern)
       |    |- setup_utils.py                # generic helper for system test setup         (e.g. System Test
environment setup)
       |    |- system_test_utils.py          # utilities for generic testing purposes       (e.g. reading JSON
data file)
       |    |- testcase_env.py               # testcase environment setup                   (e.g. data
```

```
structure initialization such as brokers-pid mapping)
       |    |- metrics.py                   # utilities for metrics collection
       |    |- pyh.py                       # from http://code.google.com/p/pyh            (open source)
       |
       |- cluster_config.json               # this file contains the following properties:
       |                                    #   1. what entities (Producer, Consumer) should be running
       |                                    #   2. which cluster (source or target)
       |                                    #   3. where they should be running (physical nodes)
       |
       |  # cluster_config.json is used to specify the logical machines configuration :
       |  #
       |  #   entity_id    : In each testcase, there may be zookeeper(s), broker(s), producer(s) and
       |  #                   consumer(s) involved. "entity_id" is used to uniquely identify each
       |  #                   component inside the system test.
       |  #   hostname     : It is used to specify the name of the machine in the distributed environment.
       |  #                   "localhost" is used by default.
       |  #   role         : The supported values are "zookeeper", "broker", "mirror_maker", "migration_tool",
       |  #                   "producer", "consumer".
       |  #   cluster_name : The supported values are "source", "target"
       |  #   kafka_home   : Specify the Kafka installation directory of each machine in a distributed
environment.
       |  #                   "default" is used by default and ../ is assumed to be "kafka_home".
       |  #   java_home    : Specify the JAVA_HOME of each machine in a distributed environment.
       |  #                   1. "default" is used by default. If JAVA_HOME is specified in the environment,
          |  #                                        this value will be used. Otherwise, System Test
executes a shell command "which java"
       |  #                      to find java bin dir and set JAVA_HOME accordingly. If no java binary can be
found,
       |  #                      it throws Exception and exit.
       |  #                   2. If a path is specified other than "default", System Test will verify java
binary.
       |  #                      Otherwise, it throws Exception and exit.
       |  #   jmx_port     : Specify a JMX_PORT for each component. It must be unique inside the cluster_config.
json.
       |  #
       |  # {
       |  #   "cluster_config": [
       |  #   {
       |  #     "entity_id"    : "0",
<-------------------------------------------------------------------------------|
       |  #     "hostname"     : "localhost",             "entity_id" in cluster_config must match the
corresponding "entity id" in       |
       |  #     "role"         : "zookeeper",             testcase_NNNN.properties as shown
below                                    |
       |  #     "cluster_name" :
"source",                                                                                    |
       |  #     "kafka_home"   :
"default",                                                                                   |
       |  #     "java_home"    :
"default",                                                                                   |
       |  #     "jmx_port"     :
"9990"                                                                                       |
       |  #
},
|
       |  #
.
|
       |  #
.
.
       |  #
.
.
       |  #
}
.
       |
       |- /XXXX_testsuite
       |
       |  # XXXX_testsuite is a directory which contains Python scripts and test case directories for a
```

```
specific group of functional testings.
       |  # For example, mirror_maker_testsuite contains mirror_maker_test.py in which it will start mirror
maker instances which is not
       |  # required for the test cases in migration_tool_testsuite.
       |  #
       |  # The following are the existing test suites:
       |  #   migration_tool_testsuite
       |  #   mirror_maker_testsuite
       |  #   replication_testsuite
       |
            |- <testsuite>.py
            |
            |  #   <testsuite>.py : This Python script may be implemented with the test logic for a group of
test scenarios.
            |  #                    For example, in the "mirror_maker_testsuite", mirror_maker_test.py is the
testsuite script
            |  #                    which is implemented with the following :
            |  #                    1. start zookeeper(s) in source cluster
            |  #                    2. start broker(s) in source cluster
            |  #                    3. start zookeeper(s) in target cluster
            |  #                    4. start broker(s) in target cluster
            |  #                    5. start mirror maker(s)
            |  #                    6. start producer
            |  #                    7. start consumer
            |  #                    8. stop all entities
            |  #                    9. validate no data loss
            |  #
            |  #                    The above test logic is implemented in a generalized fashion and read from
            |  #                    testcase_NNNN_properties.json for the following test case arguments :
            |  #                    1. replication factor (broker)
            |  #                    2. no. of partitions  (broker)
            |  #                    3. log segment bytes  (broker)
            |  #                    4. topics             (producer / consumer)
            |  #                    .
            |  #                    .
            |  #                    .
            |  #
            |  #                    Therefore, each testsuite can be thought of a functional / feature test
group
            |  #                    by varying a combination of settings for easier maintenance.
            |
            |- /config
            |
            |  #   config         : This config directory contains the TEMPLATE properties files for this
testsuite.
            |  #                    For "replication_testsuite", only server.properties and zookeeper.
properties are
            |  #                    required. However, "mirror_maker_testsuite" has mirror_consumer.properties
and
            |  #                    mirror_producer.properties as well.
            |  #
            |  #                    System Test reads the properties from each files in this directory as a
TEMPLATE
            |  #                    and override the values with those from testcase_NNNN_properties.json
accordingly.
            |
                 |- server.properties
                 |- zookeeper.properties
                 |- (migration_consumer.properties)        # only in migration_tool_testsuite
                 |- (migration_producer.properties)        # only in migration_tool_testsuite
                 |- (mirror_maker_consumer.properties)     # only in mirror_maker_testsuite
                 |- (mirror_maker_producer.properties)     # only in mirror_maker_testsuite
                 |
                 |- /testcase_NNNN
                      |
                      |  # testcase_NNNN is a directory which contains a json file to specify the arguments
                      |  # required for the <testsuite>.py script to execute.
                      |  #
                      |  # The main arguments are :
                      |  # 1. testcase_args : These are the arguments specific for that test case such as
                      |  #    "replica_factor", "num_partition", "bounce_broker", ... etc.
```

```
                      |  # 2. entities      : These are the arguments required to start running a certain
                      |  #                    entity such as a broker : port, replication factor, log file
                      |  #                    directory, ... etc.
                      |
                      |- (config)    # generated when this testcase is executed in system test runtime
                      |
                      |- (logs)      # generated when this testcase is executed in system test runtime
                      |
                      |- testcase_NNNN_properties.json     # this file contains new values to override the
default settings of
                      |                                    #   various entities (e.g. ZK, Broker, Producer,
Mirror Maker, ...)
                      |                                    #   such as "log.segment.bytes", "num.partitions",
"broker.id", ...
                      |   # {
                      |   #   "description": {"01":"Replication Basic : Base Test",
                      |   #                   "02":"Produce and consume messages to a single topic - single
partition.",
                      |   #                   "03":"This test sends messages to 3 replicas",
                      |   #                   .
                      |   #                   .
                      |   #                   .
                      |   #   },
                      |   #   "testcase_args": {
                      |   #     "broker_type"    : "leader",
                      |   #     "bounce_broker"  : "false",
                      |   #     "replica_factor" : "3",
                      |   #     "num_partition"  : "1",
                      |   #     "num_iteration"  : "1",
                      |   #
.                                                                                                    .
                      |   #
.                                                                                                    .
                      |   #
.                                                                                                    .
                      |   #
},                                                                                                     |
                      |   #   "entities":
[                                                                                                    |
                      |   #
{                                                                                                    |
                      |   #       "entity_id"       : "0",
<------------------------------------------------------------|
                      |   #       "clientPort"      : "2188",                        1. All attributes defined
in this entity must be the
                      |   #       "dataDir"         : "/tmp/zookeeper_0",               corresponding
attributes for the "role" specified
                      |   #       "log_filename"    : "zookeeper_2188.log",             by "entity_id" defined
above in cluster_config
                      |   #       "config_filename" : "zookeeper_2188.properties"    2. In this case, the
attributes are all zookeeper
                      |   #    },                                                        related as the
cluster_config has already defined
                      |   #    .                                                         entity_id 0 as role
"zookeeper" above
                      |   #    .
                      |   #    .
                      |   # }
```

How does it validate test case failure

```
1. System Test requires ProducerPerformance to print out debugging messages (with sequential message id) as
shown below for data loss validation:

    [2013-07-08 15:34:41,169] DEBUG Topic:test_1:ThreadID:0:MessageID:0000000000:xxxxxxxx
    . . .
```

When ConsoleConsumer consumes the messages, it also prints out the message to the log. The method kafka_system_test_utils.get_message_id parses the Producer / Consumer logs and get lists of message id for comparison.

2. System Test validates the test case failures in different testsuites accordingly as shown below.

```
=========================
Migration Tool test cases
=========================

The following two methods are called :
    kafka_system_test_utils.validate_data_matched(self.systemTestEnv, self.testcaseEnv, replicationUtils)
    kafka_system_test_utils.validate_broker_log_segment_checksum(self.systemTestEnv, self.testcaseEnv)


=========================
Mirror Maker test cases
=========================

The following three methods are called :
    kafka_system_test_utils.validate_data_matched(self.systemTestEnv, self.testcaseEnv, replicationUtils)
    kafka_system_test_utils.validate_broker_log_segment_checksum(self.systemTestEnv, self.testcaseEnv, "source")
    kafka_system_test_utils.validate_broker_log_segment_checksum(self.systemTestEnv, self.testcaseEnv, "target")


=========================
Replication test cases
=========================

if logRetentionTest.lower() == "true":
    kafka_system_test_utils.validate_data_matched(self.systemTestEnv, self.testcaseEnv, replicationUtils)
elif consumerMultiTopicsMode.lower() == "true":
    kafka_system_test_utils.validate_data_matched_in_multi_topics_from_single_consumer_producer(
        self.systemTestEnv, self.testcaseEnv, replicationUtils)
else:
    kafka_system_test_utils.validate_simple_consumer_data_matched_across_replicas(self.systemTestEnv, self.
testcaseEnv)
    kafka_system_test_utils.validate_broker_log_segment_checksum(self.systemTestEnv, self.testcaseEnv)
    kafka_system_test_utils.validate_data_matched(self.systemTestEnv, self.testcaseEnv, replicationUtils)
kafka_system_test_utils.validate_index_log(self.systemTestEnv, self.testcaseEnv)
```

3. The following are the details of the validation methods:

```
=========================================================
validate_data_matched
=========================================================

        1. for each topic in the test case :
        2.     get a list of all message id in producer log
        3.     get a list of all message id in consumer log
        4.     subtract the list in #3 from #2 and return the diff
        5.     if the diff in #4 is 0 and both lengths of #2 & #3 are greater than 0 :
        6.         PASSED
        7.     else if "acks == 1" :
        8.         if the % of data loss in #4 <= ackOneDataLossThresholdPercent :
        9.             PASSED
       10.         else :
       11.             FAILED
       12.     else :
       13.         FAILED


=========================================================
validate_broker_log_segment_checksum
=========================================================

        1. for each broker :
        2.     get log segment path (as shown below)
```

```
                # localLogSegmentPath :
                # .../system_test/mirror_maker_testsuite/testcase_5002/logs/broker-4/kafka_server_4_logs
                #   |- test_1-0
                #        |- 00000000000000000000.index
                #        |- 00000000000000000000.log
                #        |- 00000000000000000020.index
                #        |- 00000000000000000020.log
                #        |- . . .
                #   |- test_1-1
                #        |- 00000000000000000000.index
                #        |- 00000000000000000000.log
                #        |- 00000000000000000020.index
                #        |- 00000000000000000020.log
                #        |- . . .

        3.     for each topicPartition in localLogSegmentPath :
        4.         append each log segment file md5 hash

        5.     update the hexdigest from #3 & #4 into a dictionary such as brokerLogSegment : hexdigest (as
shown below)

                # brokerLogCksumDict will look like this:
                # {
                #   'kafka_server_1_logs:tests_1-0': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_1_logs:tests_1-1': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_1_logs:tests_2-0': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_1_logs:tests_2-1': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_2_logs:tests_1-0': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_2_logs:tests_1-1': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_2_logs:tests_2-0': 'd41d8cd98f00b204e9800998ecf8427e',
                #   'kafka_server_2_logs:tests_2-1': 'd41d8cd98f00b204e9800998ecf8427e'
                # }

        6. re-arrange the checksum of each topic-partition from different replicas as shown below
                # {
                #   'test_1-0' : ['d41d8cd98f00b204e9800998ecf8427e','d41d8cd98f00b204e9800998ecf8427e'],
                #   'test_1-1' : ['d41d8cd98f00b204e9800998ecf8427e','d41d8cd98f00b204e9800998ecf8427e'],
                #   'test_2-0' : ['d41d8cd98f00b204e9800998ecf8427e','d41d8cd98f00b204e9800998ecf8427e'],
                #   'test_2-1' : ['d41d8cd98f00b204e9800998ecf8427e','d41d8cd98f00b204e9800998ecf8427e']
                # }
        7. for each all checksum (value) inside each topic-partitin (key) :
                #   'test_1-0' : ['d41d8cd98f00b204e9800998ecf8427e','d41d8cd98f00b204e9800998ecf8427e'],
        8.     if any checksum in the list is not equal to the first checksum :
        9.         increment failure_count
       10. if failure_count > 0 :
       11.     FAILED


========================================================
validate_simple_consumer_data_matched_across_replicas
========================================================

        1. for each simple consumer :
        2.     for each topic :
        3.         get message id from consumer log and populate a list of topic-partition-messageid (as shown
below)
                    # replicaIdxMsgIdList :
                    # - This is a list of dictionaries of topic-partition (key)
                    #   mapping to list of MessageID in that topic-partition (val)
                    # - The list index is mapped to (replicaId - 1)
                    # [
                    #  // list index = 0 => replicaId = idx(0) + 1 = 1
                    #  {
                    #      "topic1-0" : [ "0000000001", "0000000002", "0000000003"],
                    #      "topic1-1" : [ "0000000004", "0000000005", "0000000006"]
                    #  },
                    #  // list index = 1 => replicaId = idx(1) + 1 = 2
                    #  {
                    #      "topic1-0" : [ "0000000001", "0000000002", "0000000003"],
                    #      "topic1-1" : [ "0000000004", "0000000005", "0000000006"]
                    #  }
```

```
                # ]
        4. take the first dictionary in replicaIdxMsgIdList (obtained from #2 & #3)
        5. for each topic-partition in the dictionary from #4 :
        6.     compare all replicas' MessageID in corresponding topic-partition
        7.     if there is any mismatch :
        8.         increment failure_count
```

Logging of Kafka Components (Broker, ZK, Producer and Consumer)

```
The various logs from Broker, ZK, Producer & Consumer can be found in the logs directory of each individual
test case after that test case is completed (regardless of the status of the test results)

<kafka>
  |- /bin
  |- /config
  |- /contrib
  |- /core
  |- /lib
  |.
  |.
  |.
  |- /system_test
       |- /XXXX_testsuite
            |- /testcase_NNNN
                 |- testcase_NNNN_properties.json
                 |
                 |  ## the directories below are generated in system test runtime ##
                 |
                 |- /config         # contains config / properties files after overriding TEMPLATE properties
files
                 |- /dashboards     # for future enhancement
                 |- /logs
                     |- /broker-1   # naming convention : <role name>-<entity id> (in this case, broker with
entity id 1)
                          |- /kafka_server_1_logs
                               |- /test_1-0                          # topic : "test_1" of partition 0
                                    |- 00000000000000000000.index    # log index
                                    |- 00000000000000000000.log      # log segment file
                               |- /test_1-1                          # topic : "test_1" of partition 1
                                    |- 00000000000000000000.index
                                    |- 00000000000000000000.log
                          |- kafka_server_9091.log                   # kafka log on port 9091
                     |- /broker-2
                          . . .
                     |- /broker-3
                          . . .
                     |- /console_consumer-5
                          . . .
                     |- /producer_performance-4
                          . . .
                     |- /zookeeper-0
                          . . .
```

Troubleshooting a failing case

The following describes the steps to troubleshoot a failing case running in a local machine.

Refer to Running System Test on how to quick starting System Test

```
1. Under <kafka>/system_test, execute the following command to start System Test :

   $ python -u -B system_test_runner.py 2>&1 | tee system_test_output.log
```

2. When the test is completed, the following may be showing in the output:

```
        _test_case_name  :  testcase_5003
        _test_class_name  :  MirrorMakerTest
        arg : bounce_leader  :  false
        arg : bounce_mirror_maker  :  true
        arg : bounced_entity_downtime_sec  :  30
        arg : message_producing_free_time_sec  :  15
        arg : num_iteration  :  1
        arg : num_messages_to_produce_per_producer_call  :  50
        arg : num_partition  :  1
        arg : replica_factor  :  3
        arg : sleep_seconds_between_producer_calls  :  1
        validation_status  :
             Unique messages from consumer on [test_1]  :  2440                    <- consumer has consumed
60 messsages less than producer produced
             Unique messages from producer on [test_1]  :  2500                    <- producer has produced
2500 messages
             Validate for data matched on topic [test_1]  :  FAILED                <- data matched is "FAILED"
             Validate for merged log segment checksum in cluster [source]  :  PASSED
             Validate for merged log segment checksum in cluster [target]  :  PASSED
```

3. The status of item "Validate for data matched on topic [test_1]" is FAILED. Take a look at system_test_output.log and search for "validating data":

```
      i. Producer produces 2500 messages
-------------------------------------------------------------------------------------

|
     ii. Consumer consumes 2440 messages
-------------------------------------------------------------------------------------|---

|   |
    iii. Data loss threshold 5% for Ack=1 case
-------------------------------------------------------------------------------------|---|---

|   |   |
     iv. But this is not a Ack=1 test case
-------------------------------------------------------------------------------------|---|---|---
         (Therefore, this case
FAILED)
|   |   |   |

|   |   |   |
        2013-07-23 09:25:53,769 - INFO -
==================================================                                          |   |
|   |
        2013-07-23 09:25:53,769 - INFO - validating data
matched                                                                                 |   |   |   |
        2013-07-23 09:25:53,769 - INFO -
==================================================                                          |   |
|   |
        2013-07-23 09:25:53,769 - DEBUG - #### Inside validate_data_matched
(kafka_system_test_utils)                                       |   |   |   |
        2013-07-23 09:25:53,770 - DEBUG - working on topic : test_1
(kafka_system_test_utils)                                           |   |   |   |
        2013-07-23 09:25:53,770 - DEBUG - matching consumer entity id found
(kafka_system_test_utils)                                       |   |   |   |
        2013-07-23 09:25:53,826 - INFO - no. of unique messages on topic [test_1] sent from publisher  : 2500
(kafka_system_test_utils)  < ---    |   |   |
        2013-07-23 09:25:53,826 - INFO - no. of unique messages on topic [test_1] received by consumer : 2440
(kafka_system_test_utils)  < -------    |   |
        2013-07-23 09:25:53,826 - INFO - Data loss threshold % : 5.0
(kafka_system_test_utils)                                        < -----------    |
        2013-07-23 09:25:53,826 - WARNING - Data loss % on topic : test_1 : 2.4
(kafka_system_test_utils)                                 < ---------------
```

4. To further troubleshoot the failure, take a look at the log4j messages of each entity under the logs folder:

```
    <kafka>/system_test $ ls -l mirror_maker_testsuite/testcase_5003/logs
```

```
    broker-4  broker-6  broker-8  console_consumer-11  mirror_maker-12  producer_performance-10  zookeeper-1
zookeeper-3
    broker-5  broker-7  broker-9  dashboards           mirror_maker-13  zookeeper-0              zookeeper-2

5. Check if there are any errors in producer performance log:

    <kafka>/system_test $ grep -i error mirror_maker_testsuite/testcase_5003/logs/producer_performance-10
/producer_performance.log
    <kafka>/system_test $ grep -i exception mirror_maker_testsuite/testcase_5003/logs/producer_performance-10
/producer_performance.log
    <kafka>/system_test $ grep -i fail mirror_maker_testsuite/testcase_5003/logs/producer_performance-10
/producer_performance.log

6. Check if there are any errors in mirror maker logs:

    <kafka>/system_test $ grep -i exception mirror_maker_testsuite/testcase_5003/logs/mirror_maker-12
/mirror_maker_12.log

    java.nio.channels.ClosedByInterruptException
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    No partition metadata for topic test_1 due to kafka.common.LeaderNotAvailableException}] for topic
[test_1]: class kafka.common.LeaderNotAvailableException  (kafka.producer.BrokerPartitionInfo)
    java.nio.channels.ClosedByInterruptException

    <kafka>/system_test $ grep -i exception mirror_maker_testsuite/testcase_5003/logs/mirror_maker-13
/mirror_maker_13.log

    java.nio.channels.ClosedByInterruptException
    java.nio.channels.ClosedByInterruptException
```

Adding a Test Suite

```
To create a new test suite called "broker_testsuite" :
  1. Copy and paste system_test/replication_testsuite => system_test/broker_testsuite
  2. Rename system_test/broker_testsuite/replica_basic_test.py => system_test/broker_testsuite
/broker_basic_test.py
  3. Edit system_test/broker_testsuite/broker_basic_test.py and update all ReplicaBasicTest related class name
to BrokerBasicTest (as an example)
  4. Follow the flow of system_test/broker_testsuite/broker_basic_test.py and modify the necessary test logic
accordingly.
  5. Most of the cases, you may remove the code in the indicated area as shown below

                .
                .
                .

                # ============================================================================== #
                # ============================================================================== #
                #                   Product Specific Testing Code Starts Here:                   #
                # ============================================================================== #
                # ============================================================================== #

                .
                .     REMOVE THE EXISTING CODE IN THIS AREA FOR YOUR NEW TESTSUITE
                .

            except Exception as e:
                self.log_message("Exception while running test {0}".format(e))
                traceback.print_exc()
            finally:
                if not skipThisTestCase and not self.systemTestEnv.printTestDescriptionsOnly:
                    self.log_message("stopping all entities - please wait ...")
                    kafka_system_test_utils.stop_all_remote_running_processes(self.systemTestEnv, self.
testcaseEnv)
```

Adding a Test Case

```
To create a new test case under "replication_testsuite" :
  1. Copy and paste system_test/replication_testsuite/testcase_1 => system_test/replication_testsuite/testcase_2
  2. Rename system_test/replication_testsuite/testcase_2/testcase_1_properties.json => system_test
/replication_testsuite/testcase_2/testcase_2_properties.json
  3. Update system_test/replication_testsuite/testcase_2/testcase_2_properties.json with the corresponding
settings for testcase 2.
```

Logging of System Test Framework

```
System Test has its own logging messages to facilitate troubleshooting. This is not the same as the logging
messages in Broker, ZK, Producer, Consumer (which are specified in <kafka>/config/log4j.properties)

By default, System Test has log level configured in INFO. To change to DEBUG level :
1. Modify <kafka>/system_test/logging.conf
2. In the section shown below, change level=INFO => level=DEBUG

   # ========================================
   # handlers session
   # ** Change 'level' to INFO/DEBUG in this session
   # ========================================
   [handler_namedConsoleHandler]
   class="StreamHandler"
   level=INFO
   formatter=namedFormatter
   args=[]
```

Supported Platform

- Linux

Obsolete Files

The following are not part of this Python based System Test framework

- system_test/broker_failure
- system_test/common
- system_test/mirror_maker
- system_test/producer_perf