JBoss JMS Reliability Testing

Introduction

LODH has high expectations regarding the reliability of the underlying software that it is using. This is true in particular for pieces of the JBoss application server such as the transaction manager and messaging product (hornetq). To ensure that applications can rely on a reliable stack, LODH has built a reliability test platform.

The overall principle is to deploy applications that have been written using LODH standards (like is any business application deployed in production) on standard LODH JBoss clusters (that look exactly like the servers we are running in production). Under some special circumstances (aka the incidents), we look at the behavior of the system, and verify that expectations are met. Specifically, we verify that messages injected into the system cannot be lost, or processed multiple times (ie: no duplicates) : This is an 'exactly once' quality of service. To be able to assess this behavior, we bring in a single global JTA transaction actions on 2 different resource managers (XA DB+HornetQshare the s or HornetQ+HornetQ).

Tests can be used to validate a specific platform version (eap5, or eap6) or the interoperability between 2 versions of the platform (eap5 with eap6, or later eap6 with eap7).

To maximize the chances of finding issues, the test platform allows tests to be run an infinite number of times, during several hours, automating tasks that a sysop would be typically doing as normal production operations (eg: republish messages from error queues, forcing shutdown or kill + restart on inactive servers). The test platform considers each run has a different one, and automate also getting the test to be able to start from a fresh clean slate (running processes are stopped or force stopped, temporary directories from servers are removed, servers get started, ...). Finally, it has facilities to assemble a snapshot of the environment in case of a run failure: partial copy of the servers including the configuration, the log files, the data directory, the list of errors (lost messages or duplicates), ... This means that when a run fails, we have a reproductible way of assembling all resources that support needs to investigate an issue.

Test cases

There are 3 main use cases being tested:

- Local consume + Remote post + write to DB: within a global transaction, a MDB consumes one message from a local queue, inserts a row in the DB and posts a message into a remote queue. This use case has name SEND_DB. Once in the remote queue, messages get consumed again, and persisted into another table in the DB
- Remote consume + write to DB: within a global transaction, a MDB consumes a message from a remote queue and inserts a row in the DB. This use case has prefix CONS_DB.
- Remote consume + remote post: within a global transaction, a MDB consumes a message from a remote queue and posts a message into a remote queue. This use case has prefix CONS_SEND. Once in the remote queue, messages get consumed again, and persisted into another table in the DB

The complete name of a test case is <use case>_<eap versions>_<incident>

As we saw, <use case> can be one of: SEND_DB, CONS_DB or CONS_SEND.

<eap versions> can be one of: 55, 66, 56, 65:

- 55 means that there is an EAP5 application on the left cluster, and another EAP5 application on the right cluster
- 66 means that there is an EAP6 application on the left cluster, and another EAP6 application on the right cluster
- 56 means that there is an EAP5 application on the left cluster, and an EAP6 application on the right cluster
- 65 means that there is an EAP6 application on the left cluster, and an EAP5 application on the right cluster

<incident> can be one of: SHUT, KILL, NETW, OOM, CPU

- SHUT: a jboss node is shut down (a normal jboss server shutdown) during the test then restarted
- KILL: a jboss node is killed during the test then restarted
- NETW: a host is disconnected from the network during the test (as if the network cable had been unplugged suddenly) during 1 minute before its gets reconnected
- OOM: an out of memory is generated on one jboss node during the test (a thread is very quickly consuming all possible memory), and after 5 minutes the node is killed then restarted
- CPU: 100 processes that are looping to eat as much cpu as possible are launched and run for 10 minutes before exiting

This makes up (3 use cases) * (4 platform versions) * (5 incidents) = 60 possible scenarii

Topology

For all tests, we use 2 applications deployed on 2 clusters, each cluster is made of 2 members, each on a different machine.

The hosts we are using are leto01 (10.23.92.15), leto02 (10.23.92.16), leto03 (10.23.32.48) and leto04 (10.23.32.49). There is a first cluster on leto01/leto02, and the second cluster on leto03/leto04.

For eap6/eap6 use cases for instance we have applications JMSEAP6 deployed on cluster leto03/leto04 and application JMSEAP6Bis deployed on leto01/leto02.

For eap5/eap5 use cases for instance we have applications JMSEAP5 deployed on cluster leto01/leto02 and application JMSEAP5Bis deployed on leto03/leto04.

Finally, for eap5/eap6 use cases we have applications JMSEAP5 deployed on cluster leto01/leto02 and application JMSEAP6 deployed on cluster leto03/leto04.

In our diagrams we use denominations “left cluster” and “right cluster”. The left and right cluster are either a EAP5 cluster running applications JMSEAP5 or JMSEAP5Bis, or a EAP6 cluster running applications JMSEAP6 or JMSEAP6Bis depending on the test case.

For instance in test case SEND_DB_56_SHUT, the left cluster is JMSEAP5 running on leto01/leto02, whereas the right cluster is JMSEAP6 running on leto03/leto04. And in test case CONS_DB_66_SHUT the left cluster is JMSEAP6 running on leto03/leto04. The applications and clusters being used as left or right clusters is not influenced by what incident we are running. For instance CONS_DB_66_SHUT shares the same test topology with CONS_DB_66_KILL, CONS_DB_66_OOM, …

During the test, we fire an incident on the first node (leto01 for cluster leto01/leto02, and leto03 for cluster leto03/leto04) of the cluster where is started the global transaction. For SEND_DB use cases, the incident is always triggered on the first node of the left cluster. For CONS_DB and CONS_SEND the incident is triggered on the first node of the right cluster.

Expectations

For all test cases, we inject 50000 persistent messages. Each message will lead to an insert into 1 or 2 tables in the DB depending on the use case:

- SEND_DB: we expect 50000 rows in table of application deployed in left cluster, and 50000 rows in table of application deployed in right cluster
- CONS_DB: we expect 50000 rows in table of application deployed in right cluster
- CONS_SEND : we expect 50000 rows in table of application deployed in left cluster

Each row must be mapped uniquely back to a particular message. That is for each message, for the CONS_SEND use case for instance, we expect one and only one row for that message (exactly once semantic: no message loss and no duplicates).

If we reach the expected number of rows, the test case is considered to be a success. If we make 10 successful runs, the test case is considered to be validated.

DB

We are using oracle with XA driver for the DB. But the test could be run on other RDBMS.

Server Configuration

Hornet and arjuna should be in trace mode.

journal-min-files should be 100, journal-compact-min-files should be 0, journal-file-size should be 10485760

in eap6, max-backup-index should be 100, rotate-size value should be 100M

max-size-bytes should be -1 by default

address-full-policy should be PAGE

Running a test case

The tests case is orchestrated by an external programs and follows these steps:

- Kill all servers
- Clean the temp directories (eap5: data, log, tmp, work; eap6: data, log, tmp)
- Start the 4 servers for a given test case
- Wait for the servers to be online, and verify that remote MDB have connected
- Inject 50000 messages into the system (SETUP queue for SEND_DB use case, OUT queue for CONS_DB and CONS_SEND)
- Wait some time (eg: 30 secs)

- Fire incident on the appropriate node
- Wait some time (depends on the type incident)
- Restore from incident
- Start looping on DB tables to see if we receive the exact number of expected rows
    - During this loop, check if there are messages that went into error queues, and if that is the case, republish them automatically into the corresponding business queue
    - If we detect more than 50000 rows into one of the DB tables, stop the test immediately and make it a failure
    - If we detect exactly 50000 rows, exit the loop
- When we detect exactly 50000 rows, wait some more time, and check that we have exactly one row per original message (ie: no loss and no duplicates)
    - If True: test is a success
    - If False: test is a failure

For SEND_DB there are 2 tables to check. For the 2 other use cases there is only one table to check.

Use case instantiation

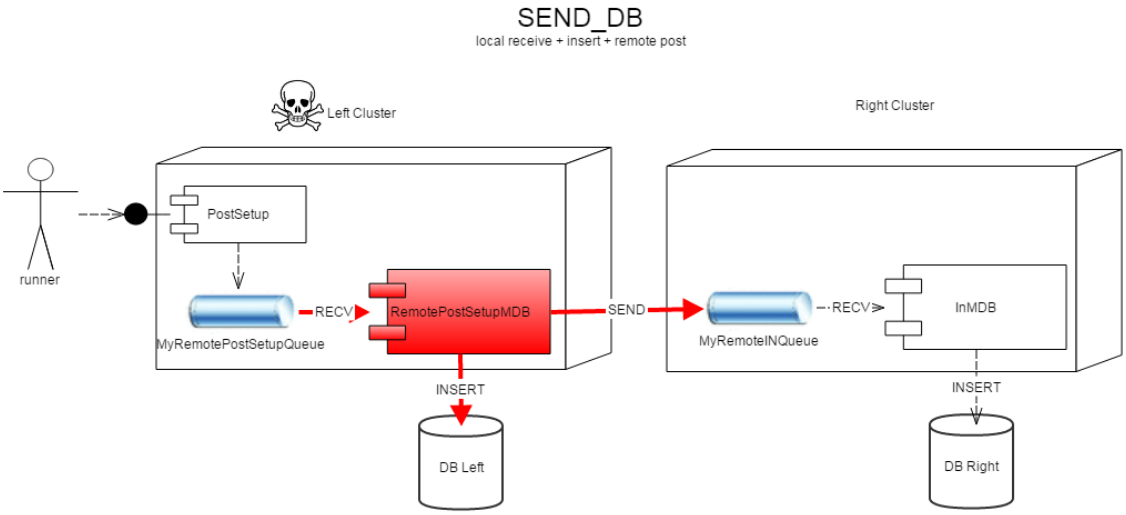
We have 3 use cases * 4 eap versions combination (55, 56, 65, 66) * 5 incidents (SHUT, KILL, NETW, OOM, CPU)

That makes 60 test cases. Those 60 test cases should be executed in the default configuration, and other configurations as well:

- Activate hornet paging, and send messages with such a length in the message body that 25000 messages is going to create a few hornetq pages
- Replace queues by topics, and receiver by durable subscribers, and setup incident during message injection

Test cases

SEND_DB

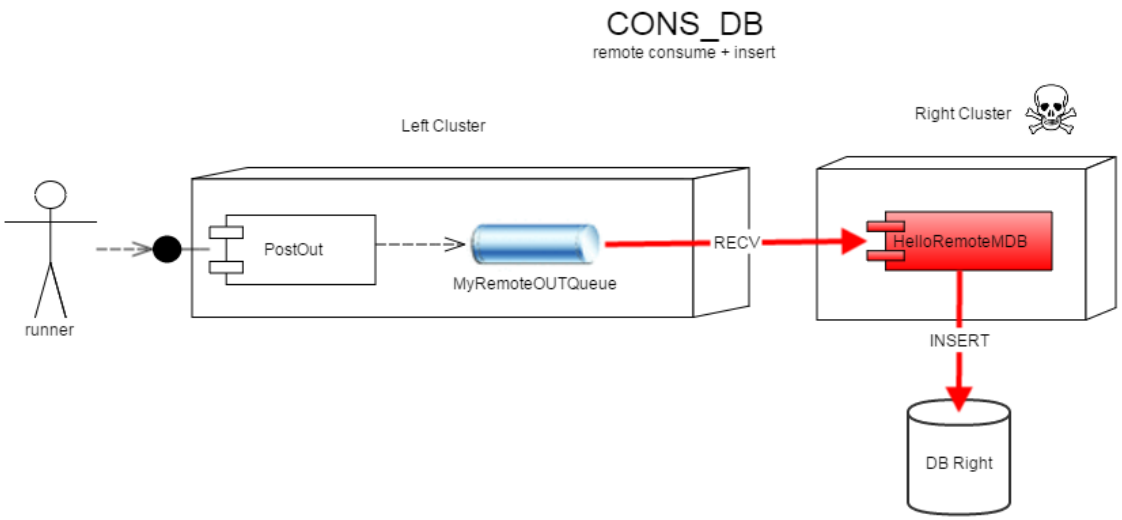


Use case	Left Cluster	Right Cluster
SEND_DB_56_<incident>	EAP5 on leto01/02	EAP6 on leto03/04
SEND_DB_65_<incident>	EAP6 on leto03/04	EAP5 on leto01/02
SEND_DB_55_<incident>	EAP5Bis on leto03/04	EAP5 on leto01/02
SEND_DB_66_<incident>	EAP6Bis on leto01/02	EAP6 on leto03/04

The node that has the incident is the first node of the Left Cluster (eg : leto01 on SEND_DB_56_<incident>)

The JTA transaction that is likely to have problems is shown in red.

CONS_DB

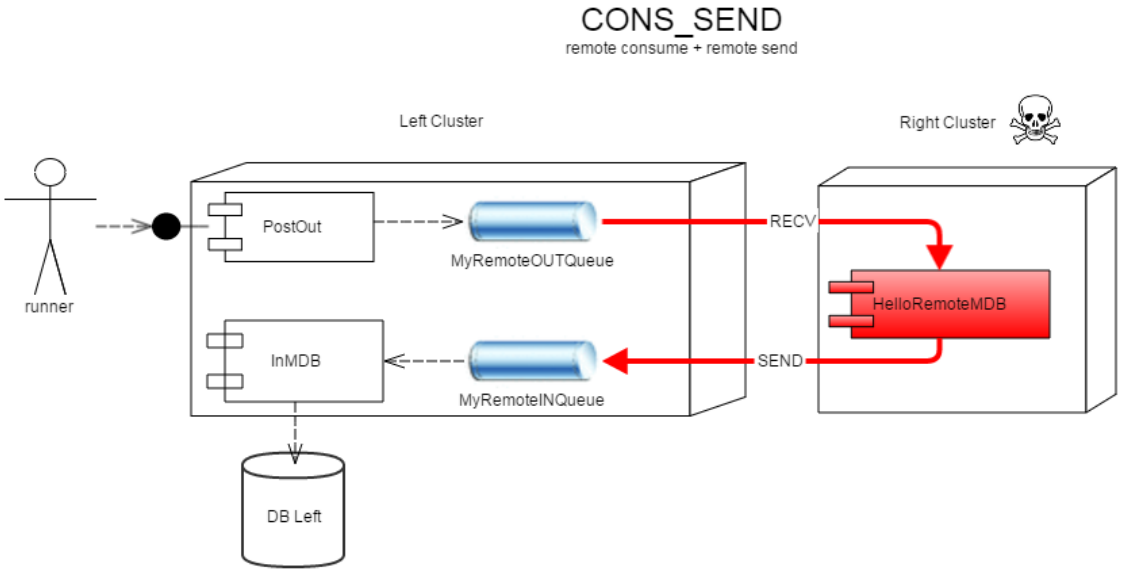


Use case	Left Cluster	Right Cluster
CONS_DB_56_<incident>	EAP5 on leto01/02	EAP6 on leto03/04
CONS_DB_65_<incident>	EAP6 on leto03/04	EAP5 on leto01/02
CONS_DB_55_<incident>	EAP5 on leto01/02	EAP5Bis on leto03/04
CONS_DB_66_<incident>	EAP6 on leto03/04	EAP6Bis on leto01/02

The node that has the incident is the first node of the Right Cluster (eg : leto03 on CONS_SEND_56_<incident>)

The JTA transaction that is likely to have problems is shown in red.

CONS_SEND



Use case	Left Cluster	Right Cluster
CONS_SEND_56_<incident>	EAP5 on leto01/02	EAP6 on leto03/04
CONS_SEND_65_<incident>	EAP6 on leto03/04	EAP5 on leto01/02

CONS_SEND_55_<incident>	EAP5 on leto01/02	EAP5Bis on leto03/04
CONS_SEND_66_<incident>	EAP6 on leto03/04	EAP6Bis on leto01/02

The node that has the incident is the first node of the Right Cluster (eg : leto03 on CONS_SEND_56_<incident>)

The JTA transaction that is likely to have problems is shown in red.