Running integration, system, and package tests

- Introduction
- Running smoke-tests
- Running bigtop's integration tests
- Cluster Failure Tests
- Package tests
- Things to keep in mind

Introduction

Bigtop is based on iTest which has a clear separation between test artifacts and test execution. Test artifacts happen to be arbitrary Maven artifacts with classes containing @Test methods. Test execution phase is being driven by maven pom.xml files where you'd define dependencies on test artifacts that you would like to execute and bind everything to the maven-failsafe-plugin's verify goal.

These tests can also be run with a new feature (pending BIGTOP-1388) - cluster failure tests, which is explained at the end of this page.

There are 3 levels of testing that bigtop supports.

1. **smoke-tests** : Basic hadoop ecosystem interoperability. These are gradle based, and can be run from the bigtop-tests/smoke-tests directory, and are super easy to modify at run time (no jar file is built - they are run from groovy source and compiled on the fly at runtime)
2. **integration tests** : Advanced, maven based tests which run from jar files. These are maven based, and are run from the bigtop-tests/test-execution directory.
3. **integration tests** : with test failures : Same as (2), but also featuring cluster failures during the tests, to test resiliency.

Since many of the integration tests are simply smoke tests, we will hope to see convergence of much of (2) into (1),over time.

Running smoke-tests

If you are looking to simply test that your basic ecosystem components are working, most likely the **smoke-tets** will suite your needs.

Also, note that the smoke tests can be used to call the tets from the integration tests directory quite easily : and run them from source without needing a jar file intermediate.

To see examples of how to do this, check out the mapreduce/ and mahout/ tests, which both reference groovy files in bigtop-tests/test-artifacts.

To avoid redundant documentation, you can read about how to run these tests in the **README file under *bigtop-tests/smoke-tests/***.

These tests are particularly easy to modify - without requiring precompiled jars, and run directly from groovy scripts.

Running bigtop's integration tests

For the testing of binary compatibility with particular versions of hadoop ecosystem components, or for other integration tests, the original bigtop tests, which live in the **bigtop-tests/test-artifacts** directory, can be used.

This maven project contains a battery of junit based tests which are built as jar files, and compiled against **a specific hadoop version**, and is executed using the **bigtop-tests/test-execution** project.

These can be a **good way to do fine grained integration testing of your bigtop based hadoop installation.**

- Make sure that you have the latest Maven installed (3.3.3+)
- Make sure that you have the following defined in your environment:

```
export JAVA_HOME=/usr/lib/jvm/java-openjdk
export HADOOP_HOME=/usr/lib/hadoop
export HADOOP_CONF_DIR=/etc/hadoop/conf
export HBASE_HOME=/usr/lib/hbase
export HBASE_CONF_DIR=/etc/hbase/conf
export ZOOKEEPER_HOME=/usr/lib/zookeeper
export HIVE_HOME=/usr/lib/hive
export PIG_HOME=/usr/lib/pig
export FLUME_HOME=/usr/lib/flume
export SQOOP_HOME=/usr/lib/sqoop
export HCAT_HOME=/usr/lib/hcatalog
export OOZIE_URL=http://localhost:11000/oozie
export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce
export SPARK_HOME=/usr/lib/spark
export SPARK_MASTER=spark://localhost:7077
```

- Given the on-going issues with Apache Jenkins builds you might need to deploy everything locally:

```
# Under bigtop home dir
mvn install
mvn -f bigtop-test-framework/pom.xml -DskipTests install
mvn -f bigtop-tests/test-execution/conf/pom.xml install
mvn -f bigtop-tests/test-execution/common/pom.xml install
mvn -f bigtop-tests/test-artifacts/pom.xml install
```

- Start test execution:

```
mvn -f bigtop-tests/test-execution/smokes/<subsystem> verify
```

- [OPTIONAL] If you want to run a specific class of test:

```
mvn -f bigtop-tests/test-execution/smokes/hadoop failsafe:integration-test -Dit.test=TestWebHDFS
```

- [OPTIONAL] If you want to run a specific test in a class:

```
mvn -f bigtop-tests/test-execution/smokes/hadoop failsafe:integration-test -Dit.
test=TestWebHDFS#testGetFileChecksum
```

Cluster Failure Tests

The purpose of this test is to check whether or not mapreduce jobs complete when failing the nodes of the cluster that is performing the job. When applying these cluster failures, the mapreduce job should complete with no issues. If mapreduce jobs fail as a result of any of the cluster failure tests, the user may not have a functional cluster or implementation of mapreduce.

Cluster failures are handled by three classes - **ServiceKilledFailure.groovy**, **ServiceRestartFailure.groovy**, and **NetworkShutdownFailure.groovy**.

We will call the functionality of these classes "cluster failures." The cluster failures extend an abstract class called AbstractFailure.groovy, which is a runnable. Each of these runnable classes execute specific shell commands that purposely fail a cluster. When cluster failures are executed, they call a function populateCommandsLIst(), which will fill up the datastructures failCommands and restoreCommands with values pertaining to the cluster failure. The values include a shell string such as "sudo pkill -9 -f %s" and a specified host to run the command on. From this, shell commands are generated and executed. Note: the host can be specified when instantiating the cluster failure or configured in /resources/vars.properties

- ServiceKilledFailure will execute commands that will kill a specified service.

```
  private static final String KILL_SERVICE_TEMPLATE = "sudo pkill -9 -f %s"
  private static final String START_SERVICE_TEMPLATE = "sudo service %s start"
```

- ServiceRestartFailure will execute commands that will stop and start a service.

```
private static final String STOP_SERVICE_TEMPLATE = "sudo service %s stop"
private static final String START_SERVICE_TEMPLATE = "sudo service %s start"
```

- NetworkShutdownFailure will execute a series of commands that restarts the network.

```
private static final String DROP_INPUT_CONNECTIONS = "sudo iptables -A INPUT -s %s -j DROP"
private static final String DROP_OUTPUT_CONNECTIONS = "sudo iptables -A OUTPUT -d %s -j DROP"
private static final String RESTORE_INPUT_CONNECTIONS = "sudo iptables -D INPUT -s %s -j DROP"
private static final String RESTORE_OUTPUT_CONNECTIONS = "sudo iptables -D OUTPUT -d %s -j DROP"
```

Two other classes that must be mentioned are FailureVars.groovy and FailureExecutor.groovy. FailureVars, when instantiated, will load configurations from /resources/vars.properties to prepare for cluster failing. The configurations dictate which cluster failures will be executed along with a variety of different timing options. More information in "How to run cluster failure tests." FailureExecutor is the main driver that creates and runs cluster failure threads (threads run parallel to hadoop and mapreduce jobs). The sequence of execution are as follows:

- FailureVars will configure all variables that are necessary for cluster failures.
- For configuration of FailureVars, see the properties file associated with it (in FailureVars.groovy).
- FailureExecutor will then spawn and execute cluster failure threads.
- The threads will then run its respective shell commands on hosts specified by the user.

- How to run cluster failure tests:

  Since the cluster failures are all runnable, the user just has to instantiate the objects and execute them in the tests they are running. If the user wishes to run cluster failures in parallel to hadoop and mapreduce jobs to test for job completion, the user must utilize FailureVars and FailureExecutor. Let's say we want to run cluster failures test while a mapreduce test such as TestDFSIO is running:
  - First step is to create a FailureVars object before the test is run inside TestDFSIO.groovy.

```
@Before
void configureVars() {
  def failureVars = new FailureVars();
}
```

  - Next step is to insert code to spawn and start a FailureExecutor thread inside the test body of TestDFSIO

```
@Test
public void testDFSIO() {
      FailureExecutor failureExecutor = new FailureExecutor();
  Thread failureThread = new Thread(failureExecutor, "DFSIO");
  failureThread.start();

      //the test
      ...
      ...
}
```

  - Now the user just has to execute the test. When the test is run, the cluster failures will run in parallel to the mapreduce test.
  - To configure the hosts as well as various timing options, open /resources/vars.properties. There, you can specify hosts, which cluster failures to run, and when the cluster failures start. You can also specify the time in between cluster failures and how long services can be killed before being brought back up. Refer to the /bigtop/bigtop-test-framework/README for more information on vars.properties.

Package tests

There's a special kind of tests designed to validate and find bugs in the packages before they are getting deployed. The source code of the tests could be found in `bigtop-tests/test-artifacts/package`. Before you can run tests you actually have to specify the testsuite that you want to use. You can pick from the following list:

- TestPackagesBasicsWithRM
- TestPackagesBasics
- TestPackagesPseudoDistributedServices
- TestPackagesPseudoDistributedDependency
- TestPackagesPseudoDistributedFileContents
- TestPackagesPseudoDistributedWithRM
  (you can open up corresponding class implementation to see how they are different from each other).

As the first step, pick `TestPackagesBasicsWithRM`. With that in mind your **last** command line is going to look something like:

```
$ mvn clean verify -f bigtop-tests/test-execution/package/pom.xml -Dorg.apache.bigtop.itest.log4j.
level=TRACE -Dlog4j.debug=true -Dorg.apache.maven-failsafe-plugin.testInclude="**
/TestPackagesBasicsWithRM.*" -Dbigtop.repo.file.url=http://xxxxxxx
```

The last two -D settings are for the name of the test suite and for the URL of the repo file describing the repo with your packages.

Things to keep in mind

- If you want to select a subset of tests you can use -Dorg.apache.maven-failsafe-plugin.testInclude='**/Mask*'. e.g., mvn verify -Dorg.apache.maven-failsafe-plugin.testInclude='**/TestHDFSBalancer*'
- It is helpful to add -Dorg.apache.bigtop.itest.log4j.level=TRACE to your mvn verify command
- These tests are not currently executed via our smoke tests - which remains a separate testing package.