Hadoop:
Big Data Stacks Validation w/ iTest
How to tame the elephant?

Konstantin Boudnik Ph.D.,
Hadoop Committer
Roman Shaposhnik,
Hadoop Contributor, Oozie Committer
Andre Arcilla
This work is licensed under Creative Commons 3.0
Agenda
(in no particular order)

● Problem we are facing
  ○ Big Data Stacks
  ○ Why validation
● Solutions
  ○ Ops testing
  ○ Platform certification
  ○ Application testing
● Stack on stack
  ○ Test artifacts are First Class Citizen
  ○ Assembling validation stack (vstack)
  ○ Tailoring vstack for target clusters
  ○ D³: Deployment/Dependencies/Determinism
Not on Agenda

● Development cycles
  ○ Features, fixes, commits, patch testing
● Release engineering
  ○ Release process
  ○ Package preparations
  ○ Release notes
  ○ RE deployments
  ○ Artifact management
  ○ Branching strategies
● Application deployment
  ○ Cluster update strategies (rolling vs. offline)
  ○ Cluster upgrades
  ○ Monitoring
  ○ Statistics collection
● We aren't done yet, but...
What's a Big Data Stack anyway?

Just a base layer!

HDFS
What is Big Data Stack?

Guess again...
What is Big Data Stack?

A bit more...
What is Big Data Stack?

A bit or two more...
What is Big Data Stack?

A bit or two more + a bit = HIVE
What is Big Data Stack?

- HDFS
- MapReduce
- HIVE
- PIG 0.8
- PIG 0.7
- Oozie
What is Big Data Stack?

And a Sqoop of flavor...
What is Big Data Stack?

A Babylon tower?

- HBase
- Sqoop
- Oozie
- PIG 0.7
- PIG 0.8
- HIVE
- MapReduce
- HDFS
What is the glue that holds the bricks?

- Packaging
  - RPM, DEB, YINST, EINST?
  - Not the best fit for Java
- Maven
  - Part of the Java ecosystem
  - Not the best tool for non-Java artifacts
- Common APIs we will assume
  - Versioned artifacts
  - Dependencies
  - BOMs
How can you possibly guarantee that everything is right?
Development & Deployment Discipline !
Is it enough really?
Of course not...
Components:
- I want Pig 0.7
- You need Pig 0.8
Components:
● I want Pig 0.7
● You need Pig 0.8

Configurations:
● Have I put in 5th data volume to DN's conf of my 6TB nodes?
● Have I _not_ copied it accidently to my 4TB nodes?
Components:
- I want Pig 0.7
- You need Pig 0.8

Configurations:
- Have I put in 5th data volume to DN's conf of my 6TB nodes?
- Have I _not_ copied it accidently to my 4TB nodes?

Auxiliary services:
- Does my Sqoop has Oracle connector?
Honestly:

- Can anyone remember all these details?
What if you've missed some?
How would you like your 10\textsuperscript{th} re-spin of a release? Make it bloody this time, please...
Redeployments...
Angry customers...
LOST REVENUE ;(
And don't you have anything better to do with that life of yours?
You need AUTOMATIC VALIDATION!!!
Validation Stack for Big Data

A Babylon tower vs Tower of Hanoi
Validation Stack (tailored)

Deployed Test Artifacts

HBase

HBase

PIG 0.7

PIG 0.7

HIVE

MapReduce

MapReduce

HDFS

HDFS

Or something like this...
Use accepted platform, tools, practices

- JVM is simply The Best
  - Disclaimer: not to start religious war
- Yet, Java isn't dynamic enough (as in JDK6)
  - But we don't care what's your implementation language
    - Groovy, Scala, Clojure, JPython (?)
- Everyone knows JUnit/TestNG
  - alas not everyone can use it effectively
- Dependency tracking and packaging
  - Maven
- Information radiators facilitate data comprehension and sharing
  - TeamCity
  - Jenkins
Few more pieces

- Tests/workloads have to be artifact'ed
  - It's not good to go fishing for test classes
- Artifacts have to be self-contained
  - Reading 20 pages to find a URL to copy a file from?
    "Forget about it" (C)
- Standard execution interface
  - JUnit's Runner is as good as any custom one
- A recognizable reporting format
  - XML sucks, but at least it has a structure
A test artifact (PigSmoke 0.9-SNAPSHOT)

<project>
  <groupId>org.apache.pig</groupId>
  <artifactId>pigsmoke</artifactId>
  <packaging>jar</packaging>
  <version>0.9.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pigunit</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pig</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>
A test artifact (PigSmoke 0.9-SNAPSHOT)

<project>
  <groupId>org.apache.pig</groupId>
  <artifactId>pigsmoke</artifactId>
  <packaging>jar</packaging>
  <version>0.9.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pigunit</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pig</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>

What wrong with it?
A test artifact (PigSmoke 0.9-SNAPSHOT)

<project>
  <groupId>org.apache.pig</groupId>
  <artifactId>pigsmoke</artifactId>
  <packaging>jar</packaging>
  <version>0.9.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pigunit</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.apache.pig</groupId>
      <artifactId>pig</artifactId>
      <version>0.9.0-SNAPSHOT</version>
    </dependency>
  </dependencies>
</project>
Before you start!
Add suitable dependencies (if desired)

```xml
<project>
...
<dependency>
  <groupId>org.apache.pig</groupId>
  <artifactId>pigsmoke</artifactId>
  <version>0.9-SNAPSHOT</version>
  <scope>test</scope>
</dependency>
</dependency>
<!-- OMG: Hadoop dependency _WAS_ missed -->
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-core</artifactId>
  <version>0.20.2-CDH3B4-SNAPSHOT</version>
</dependency>
</project>
...
Unpack data (if needed)

...<execution>
  <id>unpack-testartifact-jar</id>
  <phase>generate-test-resources</phase>
  <goals>
    <goal>unpack</goal>
  </goals>
  <configuration>
    <artifactItems>
      <artifactItem>
        <groupId>org.apache.pig</groupId>
        <artifactId>pigsmoke</artifactId>
        <version>0.9-SNAPSHOT</version>
        <type>jar</type>
        <outputDirectory>${project.build.directory}</outputDirectory>
        <includes>test/data/**/*</includes>
      </artifactItem>
    </artifactItems>
  </configuration>
</execution>
...
Find runtime libraries (if required)

... 

<execution>
    <id>find-lzo-jar</id>
    <phase>pre-integration-test</phase>
    <goals> <goal>execute</goal>  </goals>
    <configuration>
        <source>
            try {
                project.properties['lzo.jar'] = new File("${HADOOP_HOME}/lib").list(
                    [accept:{d, f-> f ==~ /hadoop.*lzo.*.jar/ }] as FilenameFilter
                ).toList().get(0);
            } catch (java.lang.IndexOutOfBoundsException ioob) {
                log.error "No lzo.jar has been found under ${HADOOP_HOME}/lib. Check your installation."
                throw ioob;
            }
        </source>
    </configuration>
</execution>
</project>
Take it easy: iTest will do the rest

...<execution>
  <id>check-testslist</id>
  <phase>pre-integration-test</phase>
  <goals>
    <goal>execute</goal>
  </goals>
  <configuration>
    <source><![CDATA[
      import org.apache.itest.*

      if (project.properties['org.codehaus.groovy.maven.destination'] &&
          project.properties['org.codehaus.groovy.maven.jar']) {
        def prefix = project.properties['org.codehaus.groovy.maven.destination'];
        JarContent.listContent(project.properties['org.codehaus.groovy.maven.jar']).
        each {
          TestListUtils.touchTestFiles(prefix, it);
        }
    ]]>}
    </source>
  </configuration>
</execution>
...
Tailoring validation stack

<project>
    <groupId>com.cloudera.itest</groupId>
    <artifactId>smoke-tests</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>hadoop-stack-validation</name>
...

    <!-- List of modules which should be executed as a part of stack testing run -->
    <modules>
        <module>pig</module>
        <module>hive</module>
        <module>hadoop</module>
    </modules>
...

</project>
How do we write iTest artifacts

```groovy
$ cat TestHadoopTinySmoke.groovy
    class TestHadoopTinySmoke {
    ....

        @BeforeClass
        static void setUp() throws IOException {
            String pattern = null; //Let's unpack everything
            JarContent.unpackJarContainer(TestHadoopSmoke.class, '.', pattern);
            ....
        }

        @Test
        void testCacheArchive() {
            def conf = (new Configuration()).get("fs.default.name");
            ....
            sh.exec("hadoop fs -rmr ${testDir}/cachefile/out",
                "hadoop ....
            
```
Just let Jenkins do its job
Just let Jenkins do its job

### Test Result

1 failures (+1), 1 skipped (+1)

19 tests (+18)  
Took 12 min.

### All Failed Tests

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Duration</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;&gt; com.cloudera.itest.hadoopsmoke.TestHadoopSmoke.testArchives</td>
<td>14.544s</td>
<td>2</td>
</tr>
</tbody>
</table>

### All Tests

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Fail</th>
<th>(diff)</th>
<th>Skip</th>
<th>(diff)</th>
<th>Total</th>
<th>(diff)</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.cloudera.itest.hadoopsmoke</td>
<td>43 sec</td>
<td>1</td>
<td>+1</td>
<td>0</td>
<td></td>
<td>2</td>
<td>+2</td>
</tr>
<tr>
<td>com.cloudera.itest.hivesmoke</td>
<td>48 sec</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>org.apache.pig.test.pigunit</td>
<td>10 min</td>
<td>0</td>
<td></td>
<td>1</td>
<td>+1</td>
<td>12</td>
<td>+12</td>
</tr>
<tr>
<td>org.apache.pig.test.pigunit.pig</td>
<td>0.54 sec</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>4</td>
<td>+4</td>
</tr>
</tbody>
</table>
What else needs to be taken care of?

● Packaged deployment
  ○ packaged artifact verification
  ○ stack validation

Think little bit of Puppet implemented on top of JVM:

```java
static PackageManager pm;

@BeforeClass
static void setUp() {
    ....
    pm = PackageManager.getPackageManager()
    pm.addBinRepo("default", "http://archive.canonical.com/", key)
    pm.refresh()
    pm.install("hadoop-0.20")
```
The coolest thing about single platform:

```java
void commonPackageTest(String[] gpkgs, Closure smoke, ...) {
    pkgs.each { pm.install(it) }
    pkgs.each { assertTrue("package ${it.name} is not installed",
                        pm.isInstalled(it)) }
    pkgs.each { pm.svc_do(it, "start") }
    smoke.call(args)
}

@Test
void testHadoop() {
    commonPackageTest(['hadoop-0.20', ...],
                        this.&commonSmokeTestsuiteRun,
                        TestHadoopSmoke.class)
    commonPackageTest(['hadoop-0.20', ...],
                        { sh.exec("hadoop fs -ls /") })
}```
Putting it all together:

- Puppet, iTest, Whirr:
  1. Change hits a SCM repo
  2. Hudson build produces Maven + Packaged artifacts
  3. Automatic deployment of modified stacks
  4. Automatic validation using corresponding stack of integration tests
  5. Rinse and repeat

- Challenges:
  - Maven versions vs. packaged versions vs. source
  - Strict, draconian discipline in test creation
  - Battling combinatoric explosion of stacks
  - Size of the cluster (pseudo-distributed <-> 500 nodes)
  - Self contained dependencies (JDK to the rescue!)
  - Sure, but does it brew espresso?
iTest: current status

- is in Alpha state
- all work is under Apache2.0 license
- to be available from github.com/cloudera/iTest shortly
Demo
Q & A