Chapter 1. HBase Engine

Table of Contents

HBase

Apache HBase is an open-source, distributed, versioned, column-oriented store.

NoSQLUnit supports *HBase* by using next classes:

Table 1.1. Lifecycle Management Rules

Embedded	`com.lordofthejars.nosqlunit.hbase.EmbeddedHBase`
Managed	`com.lordofthejars.nosqlunit.hbase.ManagedHBase`

Table 1.2. Manager Rule

NoSQLUnit Management	`com.lordofthejars.nosqlunit.hbase.HBaseRule`

Maven Setup

To use **NoSQLUnit** with HBase you only need to add next dependency:

Example 1.1. NoSqlUnit Maven Repository

```
<dependency>
 <groupId>com.lordofthejars</groupId>
 <artifactId>nosqlunit-hbase</artifactId>
 <version>${version.nosqlunit}</version>
</dependency>
```

Dataset Format

Default dataset file format in *HBase* module is json. Dataset in HBase is the same used by Cassandra-Unit [https://github.com/jsevellec/cassandra-unit/] but not all fields are supported. Only fields available in TSV HBase application can be set into dataset.

So as summary datasets must have next format :

Example 1.2. Example of HBase Dataset

```
{
    "name" : "tablename",
    "columnFamilies" : [{
        "name" : "columnFamilyName",
        "rows" : [{
            "key" : "key1",
            "columns" : [{
                "name" : "columnName",
                "value" : "columnValue"
            },
            ...
            ]
        },
        ...
        ]
    },
    ...
    ]
}
```

Getting Started

Lifecycle Management Strategy

First step is defining which lifecycle management strategy is required for your tests. Depending on kind of test you are implementing (unit test, integration test, deployment test, ...) you will require an embedded approach, managed approach or remote approach.

Embedded Lifecycle

To configure **embedded** approach you should only instantiate next rule :

Example 1.3. Embedded HBase

```
@ClassRule
public static EmbeddedHBase embeddedHBase = newEmbeddedHBaseRule().build();
```

By default embedded *Embedded* rule uses `HBaseTestingUtility` default values:

Table 1.3. Default Embedded Values

Target path	This is the directory where *HBase* stores data and is `target/data` .
Host	localhost
Port	By default port used is 60000.
File Permissions	Depending on your umask configuration, `HBaseTestingUtility` will create some directories that will not be accessible during runtime. By default this value is set to 775, but depending on your OS you may require a different value.

Managed Lifecycle

To configure **managed** approach you should only instantiate next rule :

Example 1.4. Managed HBase

```
@ClassRule
public static ManagedHBase managedHBase = newManagedHBaseServerRule().build();
```

By default managed *HBase* rule uses next default values but can be configured programmatically:

Table 1.4. Default Managed Values

Target path	This is the directory where *HBase* server is started and is `target/hbase-temp` .
CassandraPath	*HBase* installation directory which by default is retrieved from `HBASE_HOME` system environment variable.
Port	By default port used is 60000. If port is changed in *HBase* configuration file, this port should be configured too here.

> **Warning**
> To start *HBASE*`JAVA_HOME` must be set. Normally this variable is already configured, so you would need to do nothing.

Remote Lifecycle

Configuring **remote** approach does not require any special rule because you (or System like Maven) is the responsible of starting and stopping the server. This mode is used in deployment tests where you are testing your application on real environment.

Configuring HBase Connection

Next step is configuring **HBase** rule in charge of maintaining *HBase* columns into known state by inserting and deleting defined datasets. You must register `HBaseRule` *JUnit* rule class, which requires a configuration parameter with some information.

To make developer's life easier and code more readable, a fluent interface can be used to create these configuration objects. Three different kind of configuration builders exist.

Embedded Connection

The first one is for configuring a connection to embedded *HBase* .

Example 1.5. HBase with embedded configuration

```
import static com.lordofthejars.nosqlunit.hbase.EmbeddedHBase.EmbeddedHBaseRuleBui

@Rule
public HBaseRule hBaseRule = newHBaseRule().defaultEmbeddedHBase();
```

Embedded HBase does not require any special parameter. Configuration object is copied from Embedded rule directly to HBaseRule.

Managed Connection

This is for configuring a connection to managed *HBase* .

Example 1.6. HBase with managed configuration

```
import static com.lordofthejars.nosqlunit.hbase.ManagedHBaseConfigurationBuilder.n

@Rule
public HBaseRule hbaseRule = new HBaseRule(newManagedHBaseConfiguration().build())
```

By default configuration used is the one loaded by calling HBaseConfiguration.create() method. HBaseConfiguration.create() [http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/HBaseConfiguration.html#create()] which uses hbase-site.xml and hbase-default.xml classpath files.

But also a method `setProperty` method is provided to modify any parameter of generated configuration object.

Remote Connection

Configuring a connection to remote *HBase* uses same approach like ManagedHBase configuration object but using `com.lordofthejars.nosqlunit.hbase.RemoteHBaseConfigurationBuilder` class instead of com.lordofthejars.nosqlunit.hbase.ManagedHBaseConfigurationBuilder. .

Warning

Working with Apache HBase required a bit of knowledge about how it works. For example your /etc/hosts file cannot contain a reference to your host name with ip 127.0.1.1.

Moreover **NoSQLUnit** uses *HBase-0.94.1* and this version should be also installed in your computer to work with managed or remote approach. If you install another version, you should exclude these artifacts from **NoSQLUnit** dependencies, and add the new ones manually to your pom file.

Verifying Data

`@ShouldMatchDataSet` is also supported for *HBase* data but we should keep in mind some considerations.

If you plan to verify data with `@ShouldMatchDataSet` in Managed and Remote approach, you should enable Aggregate coprocessor by editing hbase-site-xml file and adding next lines:

Example 1.7. HBase with coprocessor

```
<property>
    <name>hbase.coprocessor.user.region.classes</name>
    <value>org.apache.hadoop.hbase.coprocessor.AggregateImplementation</value>
</property>
```

Full Example

To show how to use **NoSQLUnit** with *HBase* , we are going to create a very simple application.

PersonManager is the business class responsible of getting and updating person's car.

Example 1.8. PersonCar cassandra with manager.

```
public class PersonManager {

 private Configuration configuration;

 public PersonManager(Configuration configuration) {
  this.configuration = configuration;
 }

 public String getCarByPersonName(String personName) throws IOException {
  HTable table = new HTable(configuration, "person");
  Get get = new Get("john".getBytes());
  Result result = table.get(get);

  return new String(result.getValue(toByteArray().convert("personFamilyName"), toB
 }

 private Converter<String, byte[]> toByteArray() {
  return new Converter<String, byte[]>() {

   @Override
   public byte[] convert(String element) {
    return element.getBytes();
   }
  };
 }

}
```

And now one unit test is written:

For unit test we are going to use embedded approach:

Example 1.9. HBase with embedded configuration

```
public class WhenPersonWantsToKnowItsCar {

 @ClassRule
 public static EmbeddedHBase embeddedHBase = newEmbeddedHBaseRule().build();

 @Rule
 public HBaseRule hBaseRule = newHBaseRule().defaultEmbeddedHBase(this);

 @Inject
 private Configuration configuration;


 @Test
 @UsingDataSet(locations="persons.json", loadStrategy=LoadStrategyEnum.CLEAN_INSER
 public void car_should_be_returned() throws IOException {

  PersonManager personManager = new PersonManager(configuration);
  String car = personManager.getCarByPersonName("john");

  assertThat(car, is("toyota"));
 }

}
```

And dataset used is:

Example 1.10. persons.json HBase file

```
{
    "name" : "person",
    "columnFamilies" : [{
        "name" : "personFamilyName",
        "rows" : [{
            "key" : "john",
            "columns" : [{
                "name" : "age",
                "value" : "22"
            },
            {
                "name" : "car",
                "value" : "toyota"
            }]
        },
        {
            "key" : "mary",
            "columns" : [{
                "name" : "age",
                "value" : "33"
            },
            {
                "name" : "car",
                "value" : "ford"
            }]
        }]
    }]
}
```