Bandit Test Writing Manual

Raymond Poling

November 6, 2014

Contents

Abstract

Bandit is a simple structure for writing correctness tests. It uses a simple, straight forward CSV style format for structuring a series of synchronous, serial steps. It is not the purpose of Bandit to perform performance tests, which should be mostly asynchronous. Bandit is also designed to be an outside observer, which is only able to observe the actions of the system(s) under test. It a goal Bandit that except for shell scripts, no code should be written to test a system, by using simple generic components and simple reasoning.

1 Introduction to Bandit

The format of this document is that each section following this one will be written:

- Configuration: A section on the module configurations
- Test: Actions and their arguments for each module

In that vein this portion of the document covers this information in brief for all modules.

1.1 Configuration

Configuration tests should be written against environments, and tests should be as abstracted as possible from these environments. It is not always possible, and tests have no mutable features.

Configurations are written as Clojure maps. The reason for this is to allow for certain language features, should they be useful, such as allowing modules to require functions to be passed in for more elaborate capabilities.

In order for modules to be activated, and thus for their features to be available in tests, they must minimally be represented with a structure:

```
{ "module" {} }
```

This means that the above module "module" has no arguments. A better example would be this:

```
{ "rest" {}}
```

As the rest module has no arguments.

1.2 Test

All test modules have the following format:

```
identifier|module|action|argument{|arguments}*
```

Where:

identifier A string of the writer's choosing to desribe this step of the test, intended to aid the readability of output.

module The module that runs this step.

action The action the module should perform.

argument All actions have at least one required argument.

arguments Most actions take more than one argument, and many will take varargs.

Test steps are performed one after the other. A step will always be completed when pass is displayed (although scripts may launch background processes, or the system under test could still be processing messages from a message broker).

Tests can also contain blank lines and also any line beginning with # is treated as a comment.

2 Rest Module

The rest module allows for testing rest interfaces using the common verbs GET, POST, PUT, and DELETE. Each of these are defined as an action.

2.1 Configuration

No options, to ativate:

```
{"rest" {}}
```

2.2 Test

Actions are one of:

1. get
2. post
3. delete
4. put

All tests using the rest module will have rest in the second field of the test csv. The rest module assumes responses in JSON format.

All actions have the same set of arguments:

```
url|body|code{|regex}+
```

Where:

url The url to test.

body A body to send with the resut query (may be empty).

code Expected status code (this is tested against a field within the returned JSON body).

regex Regular expressions ran against the result, of which all must pass. If regex has a ! as the first character, it will attempt to negate the match.

3 Mongo Module

This module provides a number of features for testing or waiting for data in MongoDB.

3.1 Configuration

Example of all features:

```
{
  "mongo" {
    "host" "localhost"
    "port" 27017
    "writeable" true
    "db-user-pass" [{"user" "some"
      "password" "one"
      "db" "test"}]
  }
}
```

Where:

host The hostname running the mongodb that should be ran against.

port The port number for mongodb.

writeable Setting this to true (no quotes) enables destructive mongodb operations. In case live tests are setup, this will prevent damaging environments by allowing potentially catastrophic operations run. If not set, or if set to anything other than `true`.

db-user-pass The username, password, and database for authorization. db :: Database to authenticate against. user :: Username to authenticate with. password :: Password to authenticate with.

3.2 Test

The mongo module has various different actions for dealing with different needs on mongodb. Only a handful are potentially destructive, and these actions are managed by the *"writeable"* configuration setting.

All actions have the following arguments after the action argument:

```
db|collection
```

Where:

db The database to use for this action.

collection The collection to use for this action.

All following argument lists will include all arguments, for simplicity of reading.

3.2.1 compare Action

This action compares two different records (excluding certain fields) to see if they are the same based on clojure value comparisons. Used to check if a sequence of actions is equivalent to simply processing the last document of a series in our testing.

Arguments are:

```
id|mongo|compare|db|collection|query1|query2{|excludes}+
```

Where:

query1 A strict (keys must be wrapped in quotes) JSON query document for a single document that has to be compared.

query2 A strict (keys must be wrapped in quotes) JSON query document for a single document that has to be compared.

excludes Fields to be excluded from the comparison. By design _id must be specified minimally, but any number of pipe delimited excludes may be included.

3.2.2 count Action

This action counts all documents in a collection and compares it to an expected number of documents.

Arguments are:

```
id|mongo|count|db|collection|expectation
```

Where:

expectation The number of documents expected to be in the collection.

3.2.3 exists Action

Waits for a document to appear in the document, based on a standard query. One of the few actions that wait a set maximum time (no minimum) for an external action to be completed.

Arguments are:

```
id|mongo|exists|db|collection|query|wait
```

Where:

query A strictly formatted JSON document to find the document.

wait The maximum amount of time to wait for the document to appear in the database.

3.2.4 remove Action

Remove all documents from a mongo db database, based on a query. Requires *"writeable"* to be set.

Arguments are:

```
id|mongo|remove|db|collection{|queries}+
```

Where:

queries One or more pipe delimitted queries describing the documents to remove from the collection.

4 JMS Module

This module provides the ability to both send to and receive from JMS brokers. Queues and topics are supported, however durable connections are not. The publishers will publish all xml in a blank line separated file, or a set of files in a directory. Only two real actions.

4.1 Configuration

JMS currently only supports a single JMS broker endpoint (not queue or topic, only the server to which one connects). As this limitation poses problems, it is a todo to extend configuration to allow this. Such an extention should not impact current tests.

```
{
  "jms" {
    "context" {
      "java.naming.security.principal" "admin"
      "java.naming.security.credentials" ""
      "java.naming.factory.initial" "org.apache.activemq.jndi.ActiveMQInitialContextF
      "java.naming.provider.url" "tcp://localhost:61616"
      "topic.topic" "topic"
      "queue.queue" "queue"
    },
    "destinations" [{
      "destination" "topic"
      "type" "topic"
      "factory" "TopicConnectionFactory"
      "producer-consumer" "producer"
    },{
      "destination" "queue"
      "type" "queue"
      "factory" "QueueConnectionFactory"
      "producer-consumer" "producer"
  },{
      "destination" "topic"
      "type" "topic"
      "factory" "TopicConnectionFactory"
      "producer-consumer" "consumer"
    },{
      "destination" "queue"
      "type" "queue"
      "factory" "QueueConnectionFactory"
      "producer-consumer" "consumer"}]
  }
}
```

Where:

context Standard JNDI key value pairs, using the string representation of context private static fields. Generally required is:

1. "java.naming.factory.initial" - Context.INITIAL_CONTEXT_FACTORY.
2. "java.naming.provider.url" - Context.PROVIDER_URL.
3. "java.naming.security.principal" - Context.SECURITY_PRINCIPAL

4. "java.naming.security.credentials" - Context.SECURITY_CREDENTIALS
5. "topic.topic" "topic" - Represents a topic such that context lookups will work if the resource is undefined currently on the broker for ActiveMQ.

destinations A list of destinations that the test can use. In tests, a given test destination is identified by its destination and whether it publishes or consumes.

destination The destination object (name of a queue or topics).

type Whether this destination is a queue or a topic.

factory Factory object used to construct this destination.

producer-consumer Whether this object creates or consumes messages. Valid values are producer or consumer, as in the key.

4.2 Tests

There are only two actions in the module, publish and consume.

4.2.1 publish Action

The publish action will publish all data from a file or set of files from a directory. If more than on set of data resides within a file, it should be separated by empty lines.

Action is:

```
id|jms|publish|destination|file-or-directory
```

Where:

destination The JMS object as defined by a destination in the config file.

file-or-directory The file with message body data to send, or a directory of such files, separated by blank lines.

4.2.2 consume Action

Allows for reading a sequend from a JMS destination, and compares it to a set of regex of which only one need match.

Arguments are:

```
id|jms|consume|destination|number{|regex}+
```

Where:

destination A JMS destination defined in the configuration file, as defined by the destination field.

number The number of messages to consume.

regex One or more patterns of which received messages must match at least one.

5 Shell Module

The shell module invokes commandline arguments and shell scripts with arguments, allowing for use of native shell tools to support tests in live environments.

5.1 Configuration

No configuration, although a todo is to add a path to a predefined scripts directory. To activate configuration must include:

```
{"shell" {}}
```

5.2 Test

The only action at present is run.

```
id|shell|run|command with arguments
```

Where:

command with arguments A command, followed by is arguments, space separated. Environment variables cannot be used in the arguments.

6 Websocket Module

The websocket module is a rare stateful module, that allows for the testing of websocket services. It is stateful to prevent timeout issues, or for messages to queue up without consumption if other components are under test.

6.1 Configuration

You do not need to provide an alias to construct a URL, but using aliases means that the config file can be setup per environment, and tests needn't change (for tests that can be promoted through environments). Urls should be preceded with ws://.

```
{“websocket” {“some-alias” “beginning of the url”
“another-alias” “its beginning”}}
```

6.2 Test

6.2.1 open Action

Opens a websocket connection.

```
open|alias|url|handshake
```

Where:

alias used to keep track of open connections (need not be reflected in config).

url is either appended to the end of the url referred to in the config file reference for that alias, or it will be used as is if not defined.

handshake is a string that will be sent over the websocket.

6.2.2 receive Action

receive|alias|number|wait|or-pattern
Where:

alias reference to the websocket to send messages on.

number The numeber of messages to receive form the websocket.

wait Max amount of time to wait between message receives.

or-pattern A set of regex patterns, one of which must match each received message.

6.2.3 send Action

send|alias|text
Where:

alias reference to the websocket to send messages to.

text text to send through web socket.

6.2.4 close Action

close|alias
Where:

alias reference to the websocket connection to close.

7 Kestrel Module

7.1 Configuration

Kestrel uses aliases exclusively. Each alias has a map, which defines its servers. Other options could be added to the map in the future. Server entries must include the port. Despite the key name, only a single server is supported at present.

```
{“kestrel” {“some-alias” {"servers" “host:port”
“another-alias” {"servers" “host:port”}}
```

7.2 Test

7.2.1 consume Action

Consume text only messages from a kestrel queue.

`consume|alias|queue|number|wait{|regex}+`

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to consume from.

number the number of messages to pull from Kestrel Queue.

wait max period of time to wait for a message to become available on Kestrel Queue.

regex One or more patterns to test the text of the message against. The patterns are logically ORed together, so that if any one passes, they all pass.

7.2.2 bin-consume Action

`bin-consume|alias|queue|number|wait`

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to consume from.

number the number of messages to try to consume from Kestrel queue.

wait how long to wait for each message to arrive at the queue.

7.2.3 publish Action

`publish|alias|queue|file-or-dir`

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to publish to.

file-or-directory load an xml file or directory of files. Uses the same module as jms.

7.2.4 bin-publish Action

`bin-publish|alias|queue|file`

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to publish to.

file load a binary file into a Kestrel Queue. At present it will load an entire file. Other options may present themselves later.

7.2.5 peek Action

```
peek|alias|queue|wait{|regex}+
```

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to peek at.

wait The period of time to wait for a message to appear on the top of the queue.

regex A list of regex patterns, one of which must match the message peeked at from the top of the queue.

7.2.6 delete Action

```
publish|alias|queue
```

Where:

alias used to keep track of Kestrel connections. Must exist in config files.

queue the name of the queue to delete.

8 Running Tests

To run a test:

```
cd test/directory # should have the test .csv
java -jar path/to/bandit.jar bandit.core config.clj test.csv
```

It is useful to write scripts to wrap up the configuration, and to move into directories holding tests with related data.

NOTE: In lab7 you can use:

```
./check test
```

This will move into the directory with the test, and run it, so all relative paths will be related directly to the test.csv file.

9 Sample Tests

9.1 Simple Product Text

```
#Clear everything, just this test
Remove all product|mongo|remove|smcdb          |product|{}
Remove journal    |mongo|remove|smcJournal     |product|{}
Remove historical |mongo|remove|smcHistoricalDb|product|{}
Remove failed     |mongo|remove|failedMessages |smcFailedMessages|{}

#publish messages to a queue
```

```
publish all xml messages|jms|publish|cmb.cibtech.na.smc_160829_isgcloud.ABS|products.xml

#Verify that all messages are consumed, pause until they are consumed
Verify exists _id|mongo|exists|smcdb|product|{"_id":"417062831"}|600
Verify exists _id|mongo|exists|smcdb|product|{"_id":"722591051"}|600
Verify exists _id|mongo|exists|smcdb|product|{"_id":"990415051"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"141895090"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"141884611"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"141907120"}|600

#Verify counts
Verify 9244 messages in smcdb|mongo|count|smcdb|product|9244|{}
Verify 756 messages in failedMessages|mongo|count|failedMessages|smcFailedMessages|756|{}
```

9.2 Failed Messages Test

```
#Clear everything, just this test
Remove all product|mongo|remove|smcdb          |product|{}
Remove journal    |mongo|remove|smcJournal     |product|{}
Remove historical |mongo|remove|smcHistoricalDb|product|{}
Remove failed     |mongo|remove|failedMessages |smcFailedMessages|{}

#Backup current properties
Backup properties|shell|run|../scripts/backup-properties.sh

#Break the topology
Use historicalDB free properties|shell|run|../scripts/load-properties.sh ../properties/no-smcHistoricalDb.properties

#Start the topology
Start small-products for test|shell|run|../scripts/start-bos.sh product-small

#publish messages to a queue
Publish all xml messages|jms|publish|cmb.cibtech.na.smc_160829_isgcloud.ABS|only-good.xml

#Verify that all messages are consumed
#It is okay if some of theese fail, due to processing time in large loads
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"138260714"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"150195227"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"150514099"}|600

#Verify counts in failure
Verify 0 messages in db|mongo|count|smcHistoricalDb|product|0|{}
Verify 9244 messages in failedMessages|mongo|count|failedMessages|smcFailedMessages|9244|{}
Verify 9244 messages in smcdb|mongo|count|smcdb|product|9244|{}

#Stop the topology
Stop the topology|shell|run|../scripts/stop-bos.sh product-small

#Remove all of the failed messages for PRODUCT, they are known bad anyways
Remove failed PRODUCT|mongo|remove|failedMessages|smcFailedMessages|{"Type":"PRODUCT"}

#Install no smcdb properties
Install correct properties|shell|run|../scripts/load-properties.sh ../properties/no-smcdb.properties

#Restart the topology
Start small-products for test|shell|run|../scripts/start-bos.sh product-small

#publish a few more messages, so we have bad xml that's valid
Publish smcdb miss messages|jms|publish|cmb.cibtech.na.smc_160829_isgcloud.ABS|smcdb-fail.xml

#Wait for messages to stop processing
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"98572177","Type":"PRODUCT"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"12680143","Type":"P
RODUCT"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"150060330","Type":"P
RODUCT"}|600
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"150149037","Type":"P
RODUCT"}|600
```

```
Verify exists failed ObjectId|mongo|exists|failedMessages|smcFailedMessages|{"ObjectId":"150343828","Type":"P
RODUCT"}|600

#Stop the topology
Stop small products|shell|run|../scripts/stop-bos.sh product-small

#run the kestrel topology
Run Kestrel Topology|shell|run|../scripts/start-bos.sh product lab7-dl380-11

#Run the failure pump
Run smcFailedMessagePump|shell|run|../scripts/start-pump.sh SmcFailJournal

#Verify all records are pushed to db
Verify exists _id|mongo|exists|smcHistoricalDb|product|{"id":"417062831"}|600
Verify exists _id|mongo|exists|smcHistoricalDb|product|{"id":"722591051"}|600
Verify exists _id|mongo|exists|smcHistoricalDb|product|{"id":"990415051"}|600

#Verify counts are correct
#Unless we figure out how to 'fix' failedMessages to seed good xml for bad
Verify counts 9244 + 666|mongo|count|smcHistoricalDb|product|9910|{}
#Consumed messages should be removed from failed messages
Verify failed messages empty|mongo|count|failedMessages|smcFailedMessages|0|{}

#shutdown the pump
Stop pump|shell|run|../scripts/stop-pump.sh

#stop topology
Stop kestrel topology|shell|run|../scripts/stop-bos.sh product

#restore old properties
Restore old properties|shell|run|../scripts/restore-properties.sh
```