# Programming Assignment 3: Apache Storm

## 1 Overview

Welcome to Storm Machine Practice. This assignment builds on **Tutorial 3: Introduction to Storm**. It is highly recommended that you practice that tutorial before starting this assignment.

The ultimate goal of this assignment is to build a topology that finds the top N words in one of Shakespeare's articles. We will build the assignment step-by-step on top of the topology in the tutorial.

## 2 Requirements

All these assignments are designed to work and will be graded based on the **Hortonworks Sandbox 2.3** virtual machine. You need to have a working HortonWorks Sandbox machine either locally or on the Amazon Web Services.

Also, all assignments are designed based on **JDK 7** (included in the virtual machine).

Please refer to "Tutorial: Run HortonWorks Sandbox 2.3 Locally" or "Tutorial: Run HortonWorks Sandbox 2.3 on AWS" for more information.

For a quick review of how to use this virtual machine and Storm, take a look at "Tutorial: Introduction to Storm."

## 3 Setup

**Step 1:** Start the virtual machine, and then connect to it through the SSH.

**Step 2:** After successfully logging in, you should see a prompt similar to the following:

```
[root@sandbox ~]#
```

**Step 3:** Download the following assignments files:

```
# git clone https://github.com/przybylek/Storm.git
```

**Step 4:** Change the current folder to:

```
# cd Storm/
```

This folder includes some boilerplate files which you are going to fill in while completing the exercises. All the parts that need to be implemented are marked as “TODO”.

**Step 6:** Finish the exercises by editing the provided template files. All you need to do is complete the parts marked with **TODO**.

- Each exercise has a Java code template. All you need to do is edit this file.
- Each exercise should be implemented in one file only. Multiple file implementation is not allowed.
- Only standard JDK 7 and Apache Storm 0.10.0 libraries can be used. In other words, the code should be compiled and run on a vanilla HortonWorks Sandbox VM.

More information about the exercises is provided below.

# Exercise A: Simple Word Count Topology

In this exercise, you are going to build a simple word count that counts the words generated in a random sentence spout. This first exercise is similar to **Tutorial 3: Introduction to Storm**.

In this exercise, we are going to use the "RandomSentenceSpout" class as the spout, the "SplitSentenceBolt" class to split sentences into words, and the "WordCountBolt" class to count the words. These components are exactly the same as in the tutorial. You can find the implementation of these classes in:

```
Storm/src
```

All you need to do for this exercise is to wire up these components, build the topology, and submit the topology, which is exactly the same as in the tutorial. To make things easier, we have provided a boilerplate for building the topology in the following file:

```
src/ TopWordFinderTopologyPartA.java
```

All you have to do is to complete the parts marked as "TODO". Note that this topology will run for 60 seconds and will automatically be killed after that.

**NOTE:** When connecting the component in the topology (using builder.setSpout() and builder.setBolt() ), make sure to use the following names for each component. You might not receive full credit if you don't use these names accordingly:

| Component | Name |
|---|---|
| **RandomSentanceSpout** | "spout" |
| **SplitSentenceBolt** | "split" |
| **WordCountBolt** | "count" |

After completing the implementation of this file, you have to build the application using the command below from the "cloudapp-mp3" directory:

```
# mvn clean package
```

Next, you have to run the application and store the output using the command below from the "cloudapp-mp3" directory:

```
# storm jar target/storm-example-0.0.1-SNAPSHOT.jar
TopWordFinderTopologyPartA > output-part-a.txt
```

Here is **a part** of a sample output of this application:

```
...
# [Thread-28-spout] INFO  backtype.storm.daemon.task - Emitting: spout
```

```
default [the cow jumped over the moon]
12608 [Thread-26-split] INFO  backtype.storm.daemon.executor -
Processing received message source: spout:8, stream: default, id: {},
[the cow jumped over the moon]
12608 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [the]
12608 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [cow]
12608 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [jumped]
12608 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [over]
12608 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [the]
12609 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [moon]
12611 [Thread-16-count] INFO  backtype.storm.daemon.executor -
Processing received message source: split:7, stream: default, id: {},
[cow]
12611 [Thread-16-count] INFO  backtype.storm.daemon.task - Emitting:
count default [cow, 55]
12611 [Thread-16-count] INFO  backtype.storm.daemon.executor -
Processing received message source: split:7, stream: default, id: {},
[jumped]
12611 [Thread-16-count] INFO  backtype.storm.daemon.task - Emitting:
count default [jumped, 55]
12611 [Thread-18-count] INFO  backtype.storm.daemon.executor -
Processing received message source: split:7, stream: default, id: {},
[the]
12611 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [the, 213]
...
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-a.txt” file. You will be graded based on the content of your “output-part-a.txt” file.

# Exercise B: Input Data from a File

As you may have noticed, the spout used in the topology of Exercise A is generating random sentences from a predefined set in the spout's class. However, we want to count words from one of Shakespeare's articles. Thus, in this exercise, you are going to create a new spout that reads data from an input file and emits each line as a tuple.

To make the implementation easier, we have provided a boilerplate for the spout needed in the following file:

```
src/ FileReaderSpout.java
```

All you need to do is to make the necessary changes in the file by implementing the sections marked as "TODO".

After finishing the implementation of this class, you have to wire up the topology with this new spout. Thus, you have to change the "TopWordFinderTopologyPartB" class accordingly. Note that this topology will run for two minutes and will automatically be killed after that. There is a chance that you might not process all the data in the input file in this time. However, that is fine and incorporated in the grader.

**NOTE:** When connecting the component in the topology (using builder.setSpout() and builder.setBolt() ), make sure to use the following names for each component. You might not receive full credit if you don't use these name accordingly:

| Component | Name |
|---|---|
| **FileReaderSpout** | "spout" |
| **SplitSentenceBolt** | "split" |
| **WordCountBolt** | "count" |

**NOTE:** You probably want to set the number of executors of the spout to "1" so that you don't read the input file more than once. However, that depends on your implementation.

We are going to test this application on a Shakespeare article, which is stored in the file "data.txt". When you are finished with the implementation, you should build and run the application again using the following command from the "cloudapp-mp3" directory:

```
# mvn clean package
# storm jar target/storm-example-0.0.1-SNAPSHOT.jar
TopWordFinderTopologyPartB data.txt > output-part-b.txt
```

Note that this command assumes you are giving the input file name as an input argument. If needed, you can change the command accordingly.

Here is **a part** of a sample output of this application:

```
...
4297 [Thread-28-spout] INFO  backtype.storm.daemon.task - Emitting:
spout default [********************The Tragedie of
Macbeth*********************]
4297 [Thread-26-split] INFO  backtype.storm.daemon.executor - Processing
received message source: spout:8, stream: default, id: {},
[********************The Tragedie of Macbeth*********************]
4298 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [The]
4298 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Tragedie]
4298 [Thread-20-count] INFO  backtype.storm.daemon.executor - Processing
received message source: split:7, stream: default, id: {}, [The]
4298 [Thread-20-count] INFO  backtype.storm.daemon.task - Emitting:
count default [The, 2]
4298 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [of]
4298 [Thread-26-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Macbeth]
4300 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
received message source: split:7, stream: default, id: {}, [Tragedie]
4300 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [Tragedie, 1]
4300 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
received message source: split:7, stream: default, id: {}, [of]
4300 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [of, 2]
4300 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
received message source: split:7, stream: default, id: {}, [Macbeth]
4300 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [Macbeth, 1]
...
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-b.txt” file. You will be graded based on the content of your “output-part-b.txt” file.

# Exercise C: Normalizer Bolt

The application we have developed in Exercise B counts the words “Apple” and “apple” as two different words. However, if we want to find the top N words, we have to count these words the same. Additionally, we don’t want to take common English words into consideration.

Therefore, in this part we are going to normalize the words by adding a normalizer bolt that gets the words from the splitter, normalizes them, and then sends them to the counter bolt. The responsibility of the normalizer is to:

1. Make all input words lowercase.
2. Remove common English words.

To make the implementation easier, we have provided a boilerplate for the normalizer bolt in the following file:

```
src/ NormalizerBolt.java
```

All you need to do is make the necessary changes in the file by implementing the sections marked as “TODO”. There is a list of common words to filter in this class, so please make sure you use this exact list in order to be fully graded for this part.

After finishing the implementation of this class, you have to wire up the topology with this bolt added to the topology. Thus, you have to change the “TopWordFinderTopologyPartC” class accordingly. Note that this topology will run for 2 minutes and will automatically be killed after that. There is a chance that you might not process all the data in the input file in this time. However, that is fine and incorporated in the grader.

**NOTE:** When connecting the component in the topology (using builder.setSpout() and builder.setBolt() ), make sure to use the following names for each component. You might not receive full credit if you don’t use these name accordingly:

| Component | Name |
|---|---|
| **FileReaderSpout** | “spout” |
| **SplitSentenceBolt** | “split” |
| **WordCountBolt** | “count” |
| **NormalizerBolt** | “normalize” |

When you are finished with the implementation, you should build and run the application again using the following command from the “cloudapp-mp3” directory:

```
# mvn clean package
# storm jar target/storm-example-0.0.1-SNAPSHOT.jar
TopWordFinderTopologyPartC data.txt > output-part-c.txt
```

Here is **a part** of a sample output of this application:

```
...
4904 [Thread-34-spout] INFO  backtype.storm.daemon.task - Emitting:
spout default [***The Project Gutenberg's Etext of Shakespeare's First
Folio***]
4905 [Thread-32-split] INFO  backtype.storm.daemon.executor - Processing
received message source: spout:11, stream: default, id: {}, [***The
Project Gutenberg's Etext of Shakespeare's First Folio***]
4906 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default []
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Gutenberg]
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [s]
4906 [Thread-22-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[The]
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Etext]
4907 [Thread-26-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[Project]
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Shakespeare]
4907 [Thread-22-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[Gutenberg]
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [First]
4907 [Thread-22-normalize] INFO  backtype.storm.daemon.task - Emitting:
normalize default [gutenberg]
4907 [Thread-32-split] INFO  backtype.storm.daemon.task - Emitting:
split default [Folio]
4911 [Thread-16-count] INFO  backtype.storm.daemon.executor - Processing
received message source: normalize:6, stream: default, id: {}, []
4911 [Thread-24-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[s]
4911 [Thread-16-count] INFO  backtype.storm.daemon.task - Emitting:
count default [, 1]
4911 [Thread-24-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[Shakespeare]
4912 [Thread-24-normalize] INFO  backtype.storm.daemon.task - Emitting:
normalize default [shakespeare]
4912 [Thread-24-normalize] INFO  backtype.storm.daemon.executor -
Processing received message source: split:10, stream: default, id: {},
[Folio]
4912 [Thread-16-count] INFO  backtype.storm.daemon.task - Emitting:
count default [shakespeare, 1]
4912 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
```

```
received message source: normalize:5, stream: default, id: {},
[gutenberg]
4912 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [gutenberg, 1]
...
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-c.txt” file. You will be graded based on the content of your “output-part-c.txt” file.

# Exercise D: Top N Words

In this exercise, we are going to find the top N words. To complete this part, we have to build a topology that reads from an input file, splits the information into words, normalizes the words, and counts the number of occurrences of each word. In this exercise, we are going to use the output of the count bolt to keep track of and periodically report the top N words.

For this purpose, you have to implement a bolt that keeps count of the top N words. Upon receipt of a new count from the count bolt, it updates the top N words. Then, it reports the top N words periodically. To make the implementation easier, we have provided a boilerplate for the top-N finder bolt in the following file:

```
src/TopNFinderBolt.java
```

All you need to do is to make necessary changes in the file by implementing the sections marked as "TODO". There is a printMap() method that prints the top-N words. Please make sure you use that function.

After finishing the implementation of this class, you have to wire up the topology with this bolt added to the topology. Thus, you have to change the "TopWordFinderTopologyPartD" class accordingly. Note that this topology will run for 2 minutes and will automatically be killed after that. There is a chance that you might not process all the data in the input file in this time. However, that is fine and incorporated in the grader.

**NOTE:** When connecting the component in the topology (using builder.setSpout() and builder.setBolt() ), make sure to use the following names for each component. You might not receive full credit if you don't use these name accordingly:

| Component | Name |
|---|---|
| **FileReaderSpout** | "spout" |
| **SplitSentenceBolt** | "split" |
| **WordCountBolt** | "count" |
| **NormalizerBolt** | "normalize" |
| **TopNFinderBolt** | "top-n" |

When you are finished with the implementation, you should build and run the application again using the following command from the "cloudapp-mp3" directory:

```
# mvn clean package
# storm jar target/storm-example-0.0.1-SNAPSHOT.jar
TopWordFinderTopologyPartD data.txt > output-part-d.txt
```

Here is **a part** of a sample output of this application:

```
...
8069 [Thread-36-top-n] INFO  backtype.storm.daemon.task - Emitting: top-
n default [top-words = [ (copyright , 2) , (laws , 2) , (world , 2) ,
(donations , 2) , (electronic , 2) , (macbeth , 2) , (project , 3) ,
(etexts , 3) , (information , 3) , (gutenberg , 2) ]]
8070 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
received message source: normalize:5, stream: default, id: {}, [first]
8070 [Thread-36-top-n] INFO  backtype.storm.daemon.executor - Processing
received message source: count:3, stream: default, id: {}, [gutenberg,
3]
8070 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [first, 2]
8070 [Thread-36-top-n] INFO  backtype.storm.daemon.executor - Processing
received message source: count:2, stream: default, id: {}, [etext, 3]
8070 [Thread-18-count] INFO  backtype.storm.daemon.executor - Processing
received message source: normalize:6, stream: default, id: {}, [folio]
8070 [Thread-36-top-n] INFO  backtype.storm.daemon.executor - Processing
received message source: count:2, stream: default, id: {}, [shakespeare,
3]
8070 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
count default [folio, 2]
8070 [Thread-36-top-n] INFO  backtype.storm.daemon.executor - Processing
received message source: count:3, stream: default, id: {}, [first, 2]
8070 [Thread-36-top-n] INFO  backtype.storm.daemon.executor - Processing
received message source: count:3, stream: default, id: {}, [folio, 2]
8172 [Thread-34-spout] INFO  backtype.storm.daemon.task - Emitting:
spout default [********************The Tragedie of
Macbeth*********************]
...
```

This is just a sample. Data may not be accurate.

Note that you can view the output of the program in the “output-part-d.txt” file. You will be graded based on the content of your “output-part-d.txt” file.

**NOTE:** After you are done with the MP, be sure to submit your results for grading.