# Tutorial: Introduction to Storm

## 1 Overview

**Welcome**

Welcome to the Storm tutorial. This tutorial covers the critical skills needed to develop a Storm application. It starts with an already deployed Storm environment where students will execute a series of hands-on labs.

**Objectives**

Upon completing this tutorial, students will be able to:

- Set up an all-in-one Storm installation
- Develop a simple Storm application (Word Count)
- Compile Storm applications
- Run Storm applications and examine the output

**Structure**

This guide is designed as step-by-step instructions for hands-on exercises. It teaches you, in the examples that follow, how to operate Storm service in a functional test environment. While working through this tutorial, you will:

1. Download and Setup an all-in-one Storm VM through a process similar to the one you followed in the MapReduce tutorial
2. Develop a simple "**Word Count**" application and build it
3. Run the application on Storm and examine the output

## 2 Requirements

This section is similar to week 2 Tutorial 1, which focused on MapReduce. If you have already successfully completed that tutorial, you can skip to Step 10.

To run the all-in-one virtual machine you need to have a hypervisor such as **Virtual Box** installed. You can grab a copy of **Virtual Box** for free from the following URL:

**https://www.virtualbox.org/wiki/Downloads**

For command line steps, you need a SSH client:

- If using Linux and OS X, you should already have it installed.
- If using Windows, you can download a free copy of the **Putty** from the following URL:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

# 3 Set Up Storm Virtual Machine

Before diving into Storm, we need to set up the environment. This tutorial uses an all-in-one Storm virtual machine made by Hortonworks. Later this virtual machine is also used for Programming Assignments.

**Step 1:** Download the Hortonworks Sandbox:
Download the **Virtual Machine for VirtualBox** from the following link:

http://hortonworks.com/products/hortonworks-sandbox/#install

You may want to download the latest version. This tutorial uses "HDP 2.3 Preview on Hortonworks Sandbox".

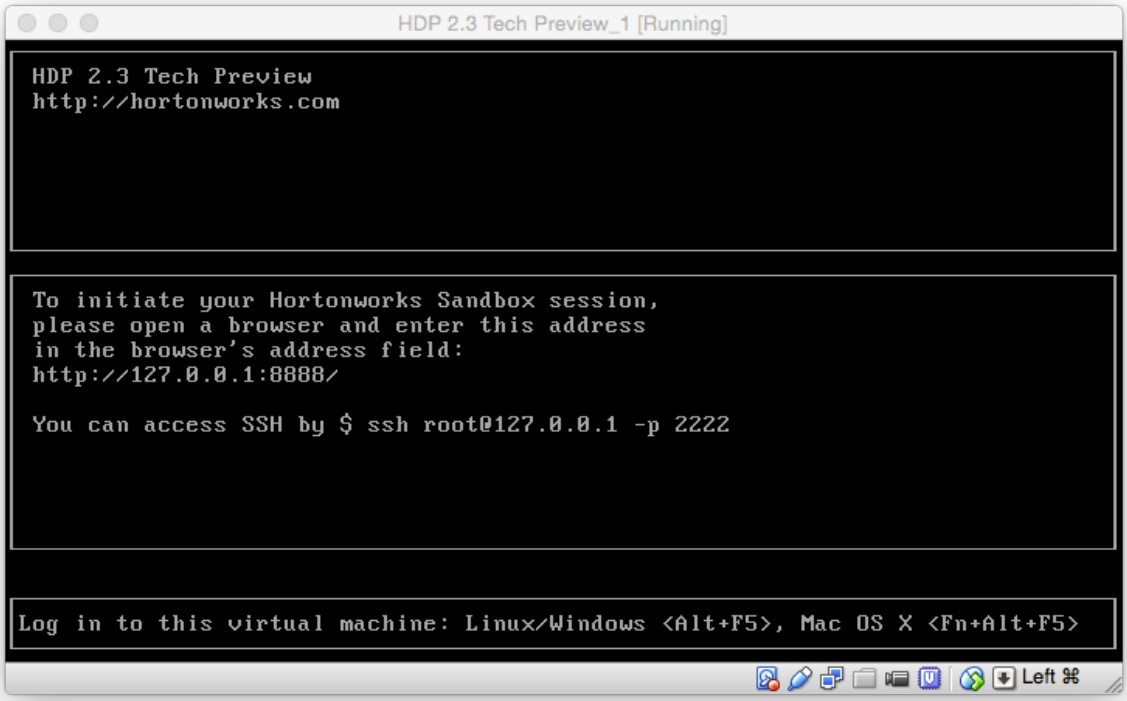
For more information about this virtual machine and details about the installation process, please refer to the Install Guide at:

http://hortonworks.com/products/hortonworks-sandbox/#install

**Step 2:** Open **VirtualBox**; then select **File > Import Appliance …**.
**Step 3:** Follow the instructions to import the downloaded file. Make sure the VM uses the following settings:

| | |
|---|---|
| **Ram** | 2048 MB |
| **CPU** | 2 |

**Step 4:** Start the virtual machine. After a few moments, you should be able to see the login screen on the virtual machine.




**Step 5 (OS X, Linux):** Open the terminal on your machine (*not* on the virtual machine), and log in to the virtual machine via SSH protocol. The password is **hadoop**. (Ignore the leading # in the commands. It is just an indicator that the command has to run in a terminal.)

```
# ssh root@127.0.0.1 -p 2222
```

**Step 5 (Windows):** Open **Putty** on your machine, and log in to the virtual machine via SSH protocol using following information:

| | |
|---|---|
| **Server** | 127.0.0.1 |
| **Port** | 2222 |
| **Username** | root |
| **Password** | hadoop |

**Step 6:** After successfully log in, you should see a prompt similar to the following:

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

**Step 7:** Before continuing, you need to install the **nano** text editor. Run the following command, and then complete the installation instructions:

```
# yum install nano
```

**Step 8:** After the installation is done, check the installation using the following command:

```
# nano
```

**Step 9:** Quit **nano**.

**Step 10:** In order to build and package Storm applications, we need **maven**. In order to install maven, run the following commands:

```
# wget http://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
# yum install apache-maven
```

**Step 11:** Check the installation using the following command:

```
# mvn -version
```

**Step 12:** Create a new folder for this tutorial:

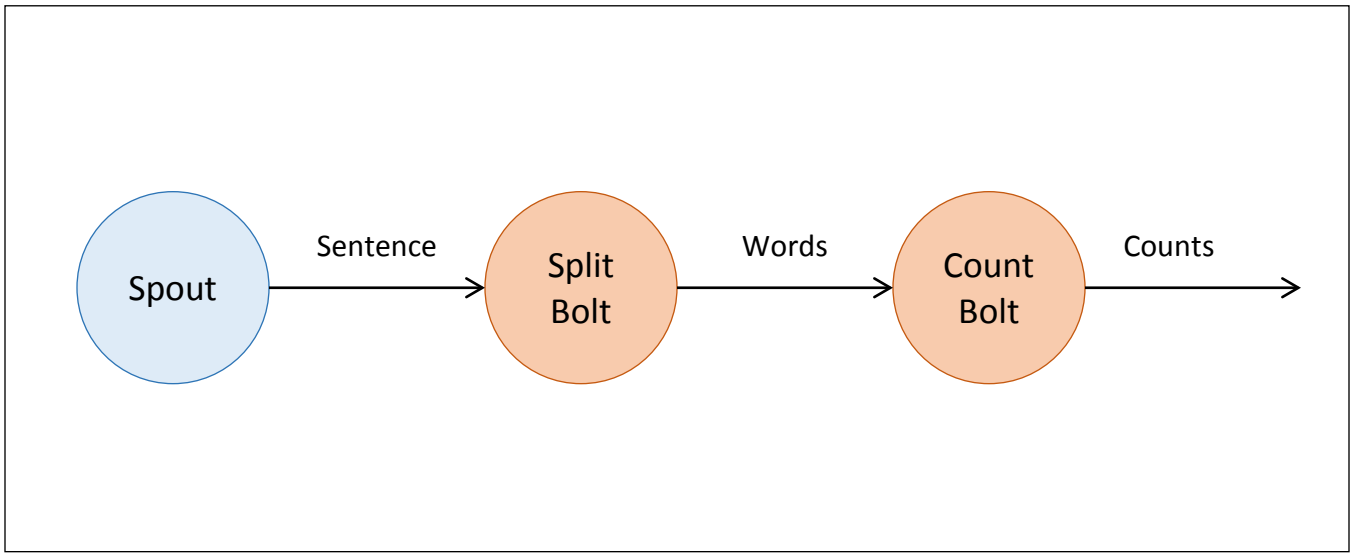
```
# mkdir storm-tutorial
```

**Step 13:** Change the current folder:

```
# cd storm-tutorial
```

## 4 Hello World (Word Count)

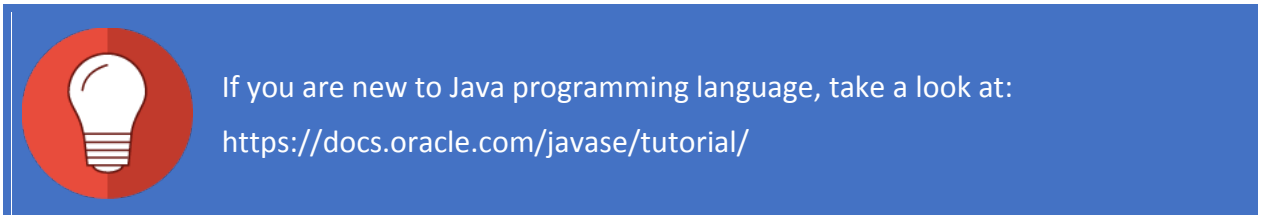
The **Word Count** application is the canonical example in Strom. In this tutorial, we are going to build all the components needed step-by-step and then build the whole application. In order to have a Word Count application, you need to create a Storm Topology as below:




As is seen above, this topology has three main components:

- **Spout**: In this topology, the Spout is emitting sentences as an output. In this tutorial, we are going to build a Spout that randomly emits a sentence from a predefined set in the Spout.
- **Split Bolt**: This bolt inputs a tuple in the form of a sentence and then emits a tuple for each word in the sentence.
- **Count Bolt**: This bolt keeps track of the count of each word. It does so by inputting tuples in form of words and incrementing the count for each word accordingly. Additionally, it emits the new count for each word. For example, if the tuple being read is the word "apple" and the current count for "apple" is 5 in the bolt, it will increment the internal count to 6 and emit a tuple in form of ("apple", 6).

In the remaining of the tutorial, we are going to explain how to build each of the components mentioned above and how to wire them up to create the whole topology.




**Step 1:** Make sure you are in the "storm-tutorial" directory. You can go to that directory by using the following command:

```
# cd ~/storm-tutorial/
```

**Step 2:** Create a "pom.xml" file by using the following command, which is used by maven in order to build and package the project:

```
# nano pom.xml
```

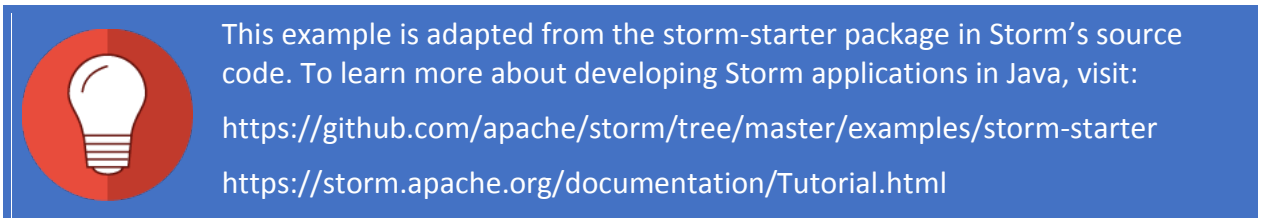
**Step 3:** Type (or copy/paste) the XML code of **"Appendix A"** in the editor; then quit **nano,** and save the file.

**Step 3 (Alternative):** Quit **nano**. Download the source from the github:

```
# wget -c https://github.com/s-noghabi/storm-tutorial/raw/master/pom.xml
```

**Step 4:** Create a "src" directory and go into that directory by using the command below. This directory is where we will place all the java classes. In the following steps, we will create each component and then create the whole topology

```
# mkdir src
# cd src
```




## 4.1 Spout

**Step 5:** Create a new Java source code file using the following command:

```
# nano RandomSentenceSpout.java
```

**Step 6:** Type (or copy/paste) the Java code of **"Appendix B"** in the editor; then quit **nano,** and save the file.

**Step 6 (Alternative):** Quit **nano**. Download the source from the github:

```
# wget -c https://github.com/s-noghabi/storm-
tutorial/raw/master/src/RandomSentenceSpout.java
```

As a result, this spout randomly chooses one of the sentences in "String[] sentences", and emits it as a tuple.

## 4.2 Split Bolt

**Step 7:** Create a new Java source code file using the following command:

```
# nano SplitSentenceBolt.java
```

**Step 8:** Type (or copy/paste) the Java code of **"Appendix C"** in the editor; then quit **nano,** and save the file.

**Step 8 (Alternative):** Quit **nano**. Download the source from the github:

```
# wget -c https://github.com/s-noghabi/storm-
tutorial/raw/master/src/SplitSentenceBolt.java
```

As a result, this Bolt reads a sentence and splits it into words, then it emits a tuple for each word.

## 4.3 Count Bolt

**Step 9:** Create a new Java source code file using the following command:

```
# nano WordCountBolt.java
```

**Step 10:** Type (or copy/paste) the Java code of **"Appendix D"** in the editor; then quit **nano,** and save the file.

**Step 10 (Alternative):** Quit **nano**. Download the source from the github:

```
# wget -c https://github.com/s-noghabi/storm-
tutorial/raw/master/src/WordCountBolt.java
```

This bolt keeps an in-memory hash map of words to their count value. Upon receipt of a tuple it increments the count for that word and emits the new count as a tuple in the format of ("word", count).

## 4.4 Word Count Topology

Now that we have all the components, we have to build the Storm Topology.

**Step 11:** Create a new Java source code file using the following command:

```
# nano WordCountTpology.java
```

**Step 12:** Type (or copy/paste) the Java code of **"Appendix E"** in the editor; then quit **nano,** and save the file.

**Step 12 (Alternative):** Quit **nano**. Download the source from the github:

```
# wget -c https://github.com/s-noghabi/storm-
tutorial/raw/master/src/WordCountTopology.java
```

This class creates a "ToplogyBuilder" instance which wires up all the components and submits the topology. It first adds the 5 instance of the spout. Then it connects 8 instances of the SplitBolt to the spout, and 12 instances of the CountBolt to the SplitBolt.

After building the topology, it submits the topology, runs the topology for 10 seconds, and then stops the application.

## 4.5 Building and Submitting the Job

**Step 13:** Go to the root directory of the project using the following command:

```
# cd ..
```

**Step 14:** Compile the source code using the following command, which will create a folder called “target” and build a fat jar under that folder:

```
# mvn clean package
```

**Step 15:** Go to the target folder using the following command:

```
# cd target
```

**Step 16:** Run the application using the following command:

```
# storm jar storm-example-0.0.1-SNAPSHOT.jar WordCountTopology
```

You should see an output similar to the one below:

```
# [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]
12611 [Thread-18-count] INFO  backtype.storm.daemon.executor -
Processing received message source: split:7, stream: default, id: {},
[over]
12611 [Thread-18-count] INFO  backtype.storm.daemon.task - Emitting:
```

```
count default [over, 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, 214]
12611 [Thread-18-count] INFO  backtype.storm.daemon.executor -
Processing received message source: split:7, stream: default, id: {},
[moon]
```

# Appendix A: “pom.xml” Source Code

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <groupId>storm.book</groupId>
  <artifactId>Getting-Started</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <build>
   <plugins>
     <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <compilerVersion>1.6</compilerVersion>
      </configuration>
     </plugin>
          </plugins>
  </build>

  <repositories>

      <!-- Repository where we can found the storm dependencies  -->
      <repository>
         <id>clojars.org</id>
         <url>http://clojars.org/repo</url>
      </repository>

  </repositories>

  <dependencies>

      <!-- Storm Dependency -->
      <dependency>
        <groupId>storm</groupId>
        <artifactId>storm</artifactId>
        <version>0.7.1</version>
      </dependency>
```

```
  </dependencies>

</project>
```

# Appendix B: Random Sentence Spout Source Code

```
import backtype.storm.spout.SpoutOutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseRichSpout;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Values;
import backtype.storm.utils.Utils;

import java.util.Map;
import java.util.Random;

public class RandomSentenceSpout extends BaseRichSpout {
  SpoutOutputCollector _collector;
  Random _rand;


  @Override
  public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    _collector = collector;
    _rand = new Random();
  }

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an apple a day keeps the
doctor away",
        "four score and seven years ago", "snow white and the seven dwarfs", "i am at two with
nature" };
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence));
  }

  @Override
  public void ack(Object id) {
  }

  @Override
  public void fail(Object id) {
  }
```

```
  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }

}
```

# Appendix C: Split Sentence Bolt Source Code

```
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;

public class SplitSentenceBolt extends BaseBasicBolt {

@Override
  public void execute(Tuple tuple, BasicOutputCollector collector) {
      String sentence = tuple.getString(0);
      String[]words=sentence.split(" ");

      for(String word:words){
          collector.emit(new Values(word));
      }
  }
  @Override
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }
 }
```

# Appendix D: Word Count Bolt Source Code

```
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;

import java.util.HashMap;
import java.util.Map;

public class WordCountBolt extends BaseBasicBolt {
   Map<String, Integer> counts = new HashMap<String, Integer>();

   @Override
   public void execute(Tuple tuple, BasicOutputCollector collector) {
     String word = tuple.getString(0);
     Integer count = counts.get(word);
     if (count == null)
       count = 0;
     count++;
     counts.put(word, count);
     collector.emit(new Values(word, count));
   }

   @Override
   public void declareOutputFields(OutputFieldsDeclarer declarer) {
     declarer.declare(new Fields("word", "count"));
   }
}
```

# Appendix E: Word Count Topology Source Code

```
import backtype.storm.Config;
import backtype.storm.LocalCluster;
import backtype.storm.StormSubmitter;
import backtype.storm.topology.BasicOutputCollector;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.TopologyBuilder;
import backtype.storm.topology.base.BaseBasicBolt;
import backtype.storm.tuple.Fields;
import backtype.storm.tuple.Tuple;
import backtype.storm.tuple.Values;

/**
 * This topology demonstrates Storm's stream groupings and multilang capabilities.
 */
public class WordCountTopology {


  public static void main(String[] args) throws Exception {

    TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("spout", new RandomSentenceSpout(), 5);

    builder.setBolt("split", new SplitSentenceBolt(), 8).shuffleGrouping("spout");
    builder.setBolt("count", new WordCountBolt(), 12).fieldsGrouping("split", new
Fields("word"));

    Config conf = new Config();
    conf.setDebug(true);


    if (args != null && args.length > 0) {
      conf.setNumWorkers(3);

      StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
    }
    else {
      conf.setMaxTaskParallelism(3);
```

```
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("word-count", conf, builder.createTopology());

    Thread.sleep(10000);

    cluster.shutdown();
   }
  }
}
```