1 Overview

Welcome to the Storm tutorial. This tutorial covers the critical skills needed to develop a Storm application in Java.

Objectives

Upon completing this tutorial, students will be able to:

- Set up the 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 a set of step-by-step instructions for hands-on exercises. It teaches you how to operate the MapReduce service in a functional test environment through a series of examples. Exercises should be completed in the order described in the following steps:

1. Download and Setup Storm
2. Develop a simple “Word Count” application and build it
3. Run the application on Storm and examine the output

2. Requirements

This tutorial is designed to work on the sample Docker image that we provide. This application will be based on **JDK 8**

3. Prepare the Docker image and install Storm

Step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine start default
docker-machine env
# follow the instruction to configure your shell: eval $(...)
```

Step 2: Download the Dockerfile and related files for this tutorial, change the current folder, build, and run the docker image, run:

```
git clone https://github.com/UIUC-public/tutorial_docker.git
cd tutorial_docker
docker build -t tutorial_docker .
docker run -it tutorial_docker bin/bash
```

Step 3: Install storm on the virtual machine and change the PATH system variable:

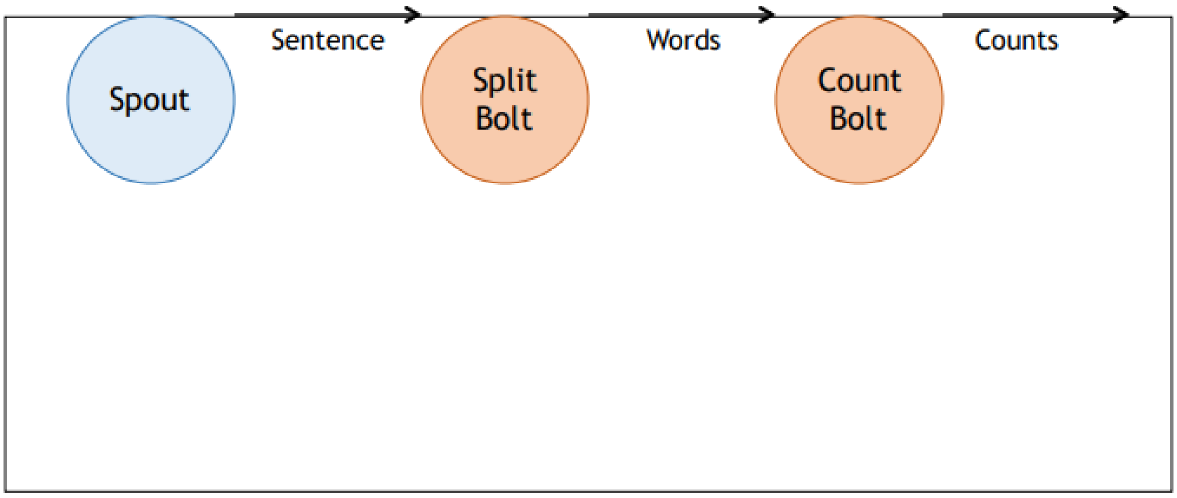
```
curl -s "http://mirrors.advancedhosters.com/apache/storm/apache-storm-1.0.6
  /apache-storm-1.0.6.tar.gz" | tar -xz -C /usr/local/
ln -s /usr/local/apache-storm-1.0.6 /usr/local/storm
PATH="/usr/local/storm/bin:${PATH}"
```

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:

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: create a temporary folder for this tutorial and enter

```
mkdir storm-tutorial
cd storm-tutorial
```

step 2: Create a “pom.xml” file by using the following command, which will be used by maven in order to build and package the project

(option 1)

```
vim pom.xml
#then type or copy/paste the XML code at the end of this tutorial in the editor;
  save and quit
```

(option 2) Download the pom.xml from the github:

```
wget -c https://github.com/UIUC-public/storm-tutorial/raw/master/pom.xml
```

step 3: 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

```

step 4: [Spout]

(option 1) Create a new Java source code file "RandomSentenceSpout.java" using the following command:

```
vim RandomSentenceSpout.java
#then type or copy/paste the RandomSentenceSpout java code at the end of this
  tutorial in the editor; save and quit
```

(option 2) Download the RandomSentenceSpout.java from the github:

```
wget -c https://github.com/UIUC-public/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.

step 5: [Split Bolt]

(option 1) Create a new Java source code file "SplitSentenceBolt.java" using the following command:

```
vim SplitSentenceBolt.java
#then type or copy/paste the SplitSentenceBolt java code at the end of this
  tutorial in the editor; save and quit
```

(option 2) Download the SplitSentenceBolt.java from the github:

```
wget -c https://github.com/UIUC-public/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.

step 6: [Count Bolt]

(option 1) Create a new Java source code file "WordCountBolt.java" using the following command:

```
vim WordCountBolt.java
#then type or copy/paste the WordCountBolt java code at the end of this tutorial
  in the editor; save and quit
```

(option 2) Download the WordCountBolt.java from the github:

```
wget -c https://github.com/UIUC-public/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).

step 7: Word Count Topology

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

(option 1) Create a new Java source code file "WordCountTopology.java" using the following command:

```
vim WordCountTopology.java
#then type or copy/paste the WordCountTopology java code at the end of this
  tutorial in the editor; save and quit
```

(option 2) Download the WordCountTopology.java from the github:

```
wget -c https://github.com/UIUC-public/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.

step 8: Build and Submitting the Job

8.1: Go to the root directory of the project using the following command:

```
cd ..
```

8.2: Compile the source code using the following command:

```
mvn clean package
# this will create a folder called "target" and build a fat jar under that
  folder
```

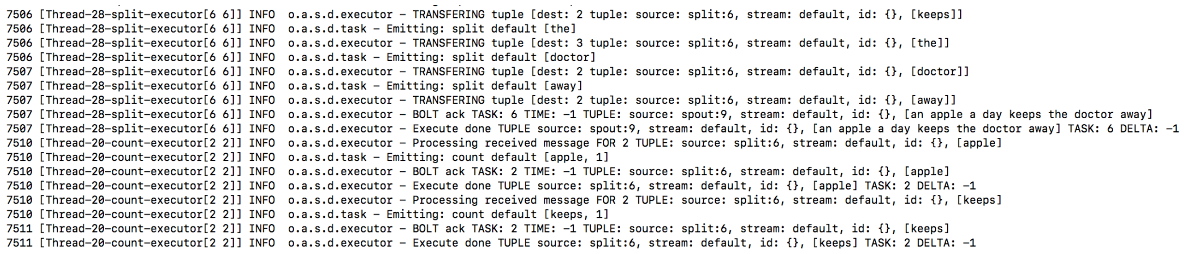
8.3: Go to the target folder using the following command:

```
cd target
```

8.4: Run the application and save its output using the following command:

```
storm jar storm-example-0.0.1-SNAPSHOT.jar WordCountTopology > wordcount.txt
```

You should see an output similar to the one below in wordcount.txt:



```
7506 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple [dest: 2 tuple: source: split:6, stream: default, id: {}, [keeps]]
7506 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split default [the]
7506 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple [dest: 3 tuple: source: split:6, stream: default, id: {}, [the]]
7506 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split default [doctor]
7507 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple [dest: 2 tuple: source: split:6, stream: default, id: {}, [doctor]]
7507 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.task - Emitting: split default [away]
7507 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - TRANSFERING tuple [dest: 2 tuple: source: split:6, stream: default, id: {}, [away]]
7507 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - BOLT ack TASK: 6 TIME: -1 TUPLE: source: spout:9, stream: default, id: {}, [an apple a day keeps the doctor away]
7507 [Thread-28-split-executor[6 6]] INFO  o.a.s.d.executor - Execute done TUPLE source: spout:9, stream: default, id: {}, [an apple a day keeps the doctor away] TASK: 6 DELTA: -1
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Processing received message FOR 2 TUPLE: source: split:6, stream: default, id: {}, [apple]
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.task - Emitting: count default [apple, 1]
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - BOLT ack TASK: 2 TIME: -1 TUPLE: source: split:6, stream: default, id: {}, [apple]
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Execute done TUPLE source: split:6, stream: default, id: {}, [apple] TASK: 2 DELTA: -1
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Processing received message FOR 2 TUPLE: source: split:6, stream: default, id: {}, [keeps]
7510 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.task - Emitting: count default [keeps, 1]
7511 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - BOLT ack TASK: 2 TIME: -1 TUPLE: source: split:6, stream: default, id: {}, [keeps]
7511 [Thread-20-count-executor[2 2]] INFO  o.a.s.d.executor - Execute done TUPLE source: split:6, stream: default, id: {}, [keeps] TASK: 2 DELTA: -1
```

To learn more about developing Storm applications in Java, visit:

https://github.com/apache/storm/tree/master/examples/storm-starter

http://storm.apache.org/releases/current/Tutorial.html

pom.xml

```xml
<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.example</groupId>
  <artifactId>storm-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <repositories>
    <repository>
      <id>clojars.org</id>
      <url>http://clojars.org/repo</url>
    </repository>
  </repositories>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.storm/storm -->
    <dependency>
      <groupId>org.apache.storm</groupId>
      <artifactId>storm-core</artifactId>
      <version>1.0.5</version>
      <!-- keep storm out of the jar-with-dependencies -->
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>

    </plugins>
  </build>
</project>
```

RandomSentenceSpout.java

```java
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.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"));
  }

}
```

SplitSentenceBolt.java

```java
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.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("[\\s~`!@#$%^&*(-)+=_:;'\",.<>?/\\\\0-9" +"\\]\\[\\}\\{]+");

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

WordCountBolt.java

```java
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.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"));
    }
}
```

WordCountTopology.java

```java
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.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();
    }
  }
}
```