Steps for Apache Storm Single Node Cluster Setup

1) From Windows, copy apache-storm-1.0.1.tar.gz and zookeeper-3.4.6.tar.gz to the downloads directory. Check the proper version available to you.

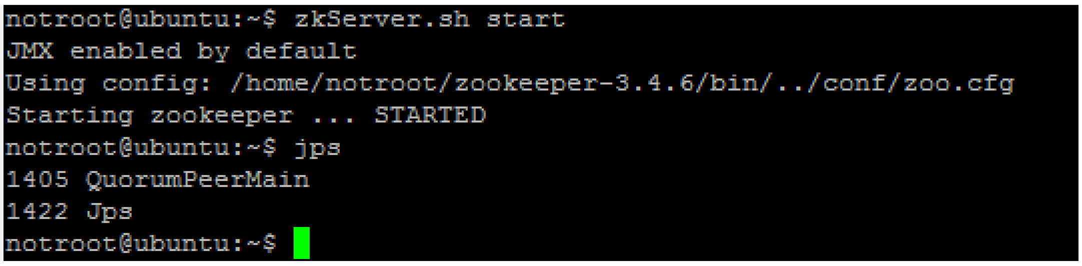
2) Extract apache-Storm and Zookeeper inside the lab/software directory

3) Make changes in the .bashrc [STORM_HOME and ZK_HOME] and execute it and check env to see if the settings have been made in the path. Also remember to add the PATH variable also

4) Make a copy of zoo_sample.cfg to zoo.cfg. The location is zookeeper conf directory.

5) Edit the zoo.cfg to include dataDir=/home/notroot/lab/data/zookeeper and create that directory also.

6) Start Zookeeper : zkServer.sh start



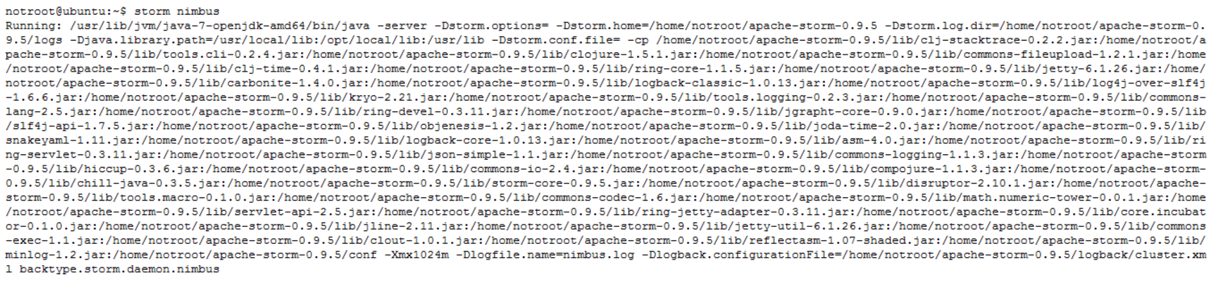
```
notroot@ubuntu:~$ zkServer.sh start
JMX enabled by default
Using config: /home/notroot/zookeeper-3.4.6/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
notroot@ubuntu:~$ jps
1405 QuorumPeerMain
1422 Jps
notroot@ubuntu:~$
```

7) copy the storm.yaml from windows to Storm folder in lab/software inside the conf directory

Note: Create the directory storm inside "/home/notroot/lab/data

Note: Also look at the additional lines added in the storm.yaml file.

8) start nimbus daemon by keying in : storm nimbus & [The & is to run it in the background]



```
notroot@ubuntu:~$ storm nimbus
Running: /usr/lib/jvm/java-7-openjdk-amd64/bin/java -server -Dstorm.options= -Dstorm.home=/home/notroot/apache-storm-0.9.5 -Dstorm.log.dir=/home/notroot/apache-storm-0.
9.5/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file= -cp /home/notroot/apache-storm-0.9.5/lib/clj-stacktrace-0.2.2.jar:/home/notroot/a
pache-storm-0.9.5/lib/tools.cli-0.2.4.jar:/home/notroot/apache-storm-0.9.5/lib/clojure-1.5.1.jar:/home/notroot/apache-storm-0.9.5/lib/commons-fileupload-1.2.1.jar:/home
/notroot/apache-storm-0.9.5/lib/clj-time-0.4.1.jar:/home/notroot/apache-storm-0.9.5/lib/ring-core-1.1.5.jar:/home/notroot/apache-storm-0.9.5/lib/jetty-6.1.26.jar:/home/
notroot/apache-storm-0.9.5/lib/carbonite-1.4.0.jar:/home/notroot/apache-storm-0.9.5/lib/logback-classic-1.0.13.jar:/home/notroot/apache-storm-0.9.5/lib/log4j-over-slf4j
-1.6.6.jar:/home/notroot/apache-storm-0.9.5/lib/kryo-2.21.jar:/home/notroot/apache-storm-0.9.5/lib/tools.logging-0.2.3.jar:/home/notroot/apache-storm-0.9.5/lib/commons-
lang-2.5.jar:/home/notroot/apache-storm-0.9.5/lib/ring-devel-0.3.11.jar:/home/notroot/apache-storm-0.9.5/lib/jgrapht-core-0.9.0.jar:/home/notroot/apache-storm-0.9.5/lib
/slf4j-api-1.7.5.jar:/home/notroot/apache-storm-0.9.5/lib/objenesis-1.2.jar:/home/notroot/apache-storm-0.9.5/lib/joda-time-2.0.jar:/home/notroot/apache-storm-0.9.5/lib/
snakeyaml-1.11.jar:/home/notroot/apache-storm-0.9.5/lib/logback-core-1.0.13.jar:/home/notroot/apache-storm-0.9.5/lib/asm-4.0.jar:/home/notroot/apache-storm-0.9.5/lib/ri
ng-servlet-0.3.11.jar:/home/notroot/apache-storm-0.9.5/lib/json-simple-1.1.jar:/home/notroot/apache-storm-0.9.5/lib/commons-logging-1.1.3.jar:/home/notroot/apache-storm
-0.9.5/lib/hiccup-0.3.6.jar:/home/notroot/apache-storm-0.9.5/lib/commons-io-2.4.jar:/home/notroot/apache-storm-0.9.5/lib/compojure-1.1.3.jar:/home/notroot/apache-storm-
0.9.5/lib/chill-java-0.3.5.jar:/home/notroot/apache-storm-0.9.5/lib/storm-core-0.9.5.jar:/home/notroot/apache-storm-0.9.5/lib/disruptor-2.10.1.jar:/home/notroot/apache-
storm-0.9.5/lib/tools.macro-0.1.0.jar:/home/notroot/apache-storm-0.9.5/lib/commons-codec-1.6.jar:/home/notroot/apache-storm-0.9.5/lib/math.numeric-tower-0.0.1.jar:/home
/notroot/apache-storm-0.9.5/lib/servlet-api-2.5.jar:/home/notroot/apache-storm-0.9.5/lib/ring-jetty-adapter-0.3.11.jar:/home/notroot/apache-storm-0.9.5/lib/core.incubat
or-0.1.0.jar:/home/notroot/apache-storm-0.9.5/lib/jline-2.11.jar:/home/notroot/apache-storm-0.9.5/lib/jetty-util-6.1.26.jar:/home/notroot/apache-storm-0.9.5/lib/commons
-exec-1.1.jar:/home/notroot/apache-storm-0.9.5/lib/clout-1.0.1.jar:/home/notroot/apache-storm-0.9.5/lib/reflectasm-1.07-shaded.jar:/home/notroot/apache-storm-0.9.5/lib/
minlog-1.2.jar:/home/notroot/apache-storm-0.9.5/conf -Xmx1024m -Dlogfile.name=nimbus.log -Dlogback.configurationFile=/home/notroot/apache-storm-0.9.5/logback/cluster.xm
l backtype.storm.daemon.nimbus
```

9) start supervisor daemon by keying in : storm supervisor &

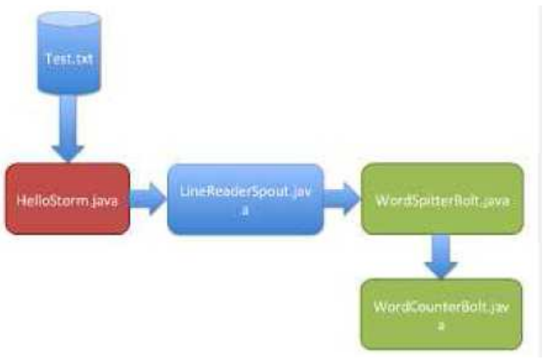
10) start the ui daemon by typing: storm ui &

11) Then navigate to the storm UI: http://IPADDRESS:8080/index.html

Working with a sample example in Storm

The basic idea behind the sample program is it takes .txt file as input and passes it to LineReaderSpout.java, which reads the file one line at a time and passes it to Storm for further processing. Storm will pass each line to WordSpitterBolt.java, this class is responsible for splitting the line into multiple words and passing them back to Storm for further processing, The last part is

WordCounterBolt.java which takes each of the word and maintains a HashMap of words with their frequency count. At the end the WordCounetrBolt.java will print all the words to the console.



1) Create a new Java Project in Eclipse called Storm_WordCount

2) Create a lib directory and all the jars inside the lib folder of apache-storm-1.0.1into it.

3) Create a class called LineRSpout inside the com.jpmc package

```
public class LineRSpout implements IRichSpout {

        private SpoutOutputCollector collector;
        private FileReader fileReader;
        private boolean completed = false;
        private TopologyContext context;

        @Override
        public void open(Map conf, TopologyContext context,
                        SpoutOutputCollector collector) {
                try {
                        this.context = context;
                        this.fileReader = new
FileReader(conf.get("inputFile").toString());
                } catch (FileNotFoundException e) {
                        throw new RuntimeException("Error reading file "
                                        + conf.get("inputFile"));
                }
                this.collector = collector;
        }

        @Override
        public void nextTuple() {
                if (completed) {
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {

                        }
                }
                String str;
                BufferedReader reader = new BufferedReader(fileReader);
                try {
                        while ((str = reader.readLine()) != null) {
                                this.collector.emit(new Values(str), str);
                        }
                } catch (Exception e) {
```

```
			throw new RuntimeException("Error reading typle", e);
		} finally {
			completed = true;
		}

	}

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

	@Override
	public void close() {
		try {
			fileReader.close();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}

	public boolean isDistributed() {
		return false;
	}

	@Override
	public void activate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void deactivate() {
		// TODO Auto-generated method stub

	}

	@Override
	public void ack(Object msgId) {

	}

	@Override
	public void fail(Object msgId) {

	}

	@Override
	public Map<String, Object> getComponentConfiguration() {
		return null;
	}

}
```

Note: The LineRSpout code has three important methods

1. open(): This method would get called at the start and will give you context information. You read value of inputFile configuration variable and read that file

2.nextTuple(): This method would allow you to pass one tuple to storm for processing at a time, in this method I am just reading one line from file and pass it to tuple

3. declareOutputFields(): This method declares that LineRSpout is going to emit line tuple

4) Create a new class called WordSplitBolt inside the package

```
import java.util.Map;

public class WordSplitBolt implements IRichBolt{
	private OutputCollector collector;

	@Override
	public void prepare(Map stormConf, TopologyContext context,
			OutputCollector collector) {
		this.collector = collector;

	}

	@Override
	public void execute(Tuple input) {
		String sentence = input.getString(0);
		String[] words = sentence.split(" ");
		for(String word: words){
			word = word.trim();
			if(!word.isEmpty()){
				word = word.toLowerCase();
				collector.emit(new Values(word));
			}
		}
		collector.ack(input);
	}

	@Override
	public void cleanup() {
		// TODO Auto-generated method stub

	}

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

	}

	@Override
	public Map<String, Object> getComponentConfiguration() {
		// TODO Auto-generated method stub
		return null;
	}

}
```

Note:

1. prepare(): This method is similar to open() method in LineReaderSpout, it allows you to initialize your code and get access to OutputCollector object for passing output back to Storm

2. declareOutputFields(): This method is similar to declareOutputFields() method in LineReadSpout, it declares that it is going to return word tuple for further processing

3. execute(): This is the method where you implement business logic of your bolt, in this case i am splitting the input line into words and passing them back to Storm for further processing

5) Create a class called WordCountBolt inside the package

```
public class WordCountBolt implements IRichBolt{

	Integer id;
	String name;
	Map<String, Integer> counters;
	private OutputCollector collector;

	@Override
	public void prepare(Map stormConf, TopologyContext context,
			OutputCollector collector) {
		this.counters = new HashMap<String, Integer>();
		this.collector = collector;
		this.name = context.getThisComponentId();
		this.id = context.getThisTaskId();

	}

	@Override
	public void execute(Tuple input) {
		String str = input.getString(0);
		if(!counters.containsKey(str)){
			counters.put(str, 1);
		}else{
			Integer c = counters.get(str) +1;
			counters.put(str, c);
		}
		collector.ack(input);
	}

	@Override
	public void cleanup() {
		System.out.println(" -- Word Counter ["+ name + "-"+id +"]");
		for(Map.Entry<String, Integer> entry:counters.entrySet()){
			System.out.println(entry.getKey()+" : " + entry.getValue());
		}
	}

	@Override
	public void declareOutputFields(OutputFieldsDeclarer declarer) {
		// TODO Auto-generated method stub

	}
```

```
        @Override
        public Map<String, Object> getComponentConfiguration() {
                // TODO Auto-generated method stub
                return null;
        }

}
```

1. prepare(): In this method i am creating a HashMap that would be used for maintaining list of words to their frequency count

2. declareOutputFields(): This method is empty because we dont want to return any tuples for further processing

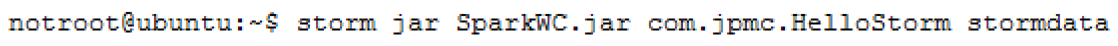
3. execute(): This method takes care of building/maintaining a HashMap for counting the frequency of the words

4. cleanup(): This method would be called at the end and we are using it to print all the words with their frequency.

6) Create a Driver Class called Hello Storm to tie up everything.

```
public class HelloStorm {

        public static void main(String[] args) throws Exception{
                Config config = new Config();
                config.put("inputFile", args[0]);
                config.setDebug(true);
                config.put(Config.TOPOLOGY_MAX_SPOUT_PENDING, 1);

                TopologyBuilder builder = new TopologyBuilder();
                builder.setSpout("line-reader-spout", new LineRSpout());
                builder.setBolt("word-spitter", new
WordSplitBolt()).shuffleGrouping("line-reader-spout");
                builder.setBolt("word-counter", new
WordCountBolt()).shuffleGrouping("word-spitter");

                LocalCluster cluster = new LocalCluster();
                cluster.submitTopology("HelloStorm", config,
builder.createTopology());
                Thread.sleep(10000);

                cluster.shutdown();
        }

}
```

7) create a jar file called SparkWC [could be any name] by exporting the project, move it to /home/notroot via winscp and then execute the same in the cluster.



```
notroot@ubuntu:~$ storm jar SparkWC.jar com.jpmc.HelloStorm stormdata
```

Note: stormdata is the data file which we are using in this case.

Check the output in the console.