BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

Topic: Apache Storm

Introduction

What is Apache Storm?

Apache Storm is a distributed real-time big data-processing system. Storm is designed to process vast amount of data in a fault-tolerant and horizontal scalable method. It is a streaming data framework that has the capability of highest ingestion rates. Though Storm is stateless, it manages distributed environment and cluster state via Apache Zookeeper. It is simple and you can execute all kinds of manipulations on real-time data in parallel.

Apache Storm Benefits

Storm is open source, robust, and user friendly. It could be utilized in small companies as well as large corporations.

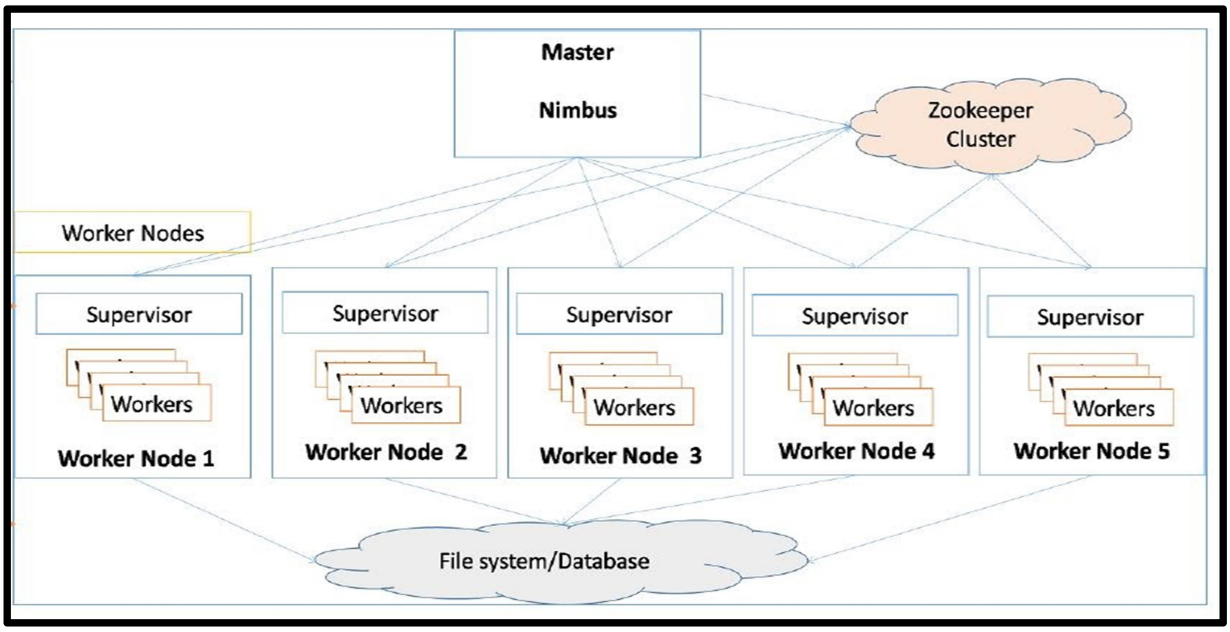
- Storm is fault tolerant, flexible, reliable, and supports any programming language.
- Allows real-time stream processing.
- Storm is unbelievably fast because it has enormous power of processing the data.
- Storm can keep up the performance even under increasing load by adding resources linearly. It is highly scalable.
- Storm performs data refresh and end-to-end delivery response in seconds or minutes depends upon the problem. It has very low latency.
- Storm provides guaranteed data processing even if any of the connected nodes in the cluster die or messages are lost.

Storm Architecture

An Apache Storm cluster is superficially similar to a Hadoop cluster. Storm has a master-slave architecture. There is a master server called **Nimbus** running on a single node called **master** node. There are slave services called **supervisors** that are running on each worker node. Supervisors start one or more worker processes called workers that run in parallel to process the input.

Worker processes store the output to a file system or database. Storm uses **Zookeeper** for distributed process coordination.

The diagram shows the Storm architecture with one master node and five worker nodes. The Nimbus process is running on the master node. There is one supervisor process running on each worker node. There are multiple worker processes running on each worker node. The workers get the input from the file system or database and store the output also to a file system or database.



Nimbus

Nimbus is a master node of Storm cluster. All other nodes in the cluster are called as worker nodes. Master node is responsible for distributing data among all the worker nodes, assign tasks to worker nodes and monitoring failures.

The master node runs a daemon called Nimbus that is similar to Hadoop's Job Tracker.

Supervisor

The nodes that follow instructions given by the nimbus are called as Supervisors. A supervisor has multiple worker processes and it governs worker processes to complete the tasks assigned by the nimbus.

Each worker node runs a daemon called the Supervisor. The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.

Worker process

A worker process will execute tasks related to a specific topology. A worker process will not run a task by itself. Instead it creates executors and asks them to perform a particular task. A worker process will have multiple executors.

Executer

An executor is nothing but a single thread spawn by a worker process. An executor runs one or more tasks but only for a specific spout or bolt.

Task

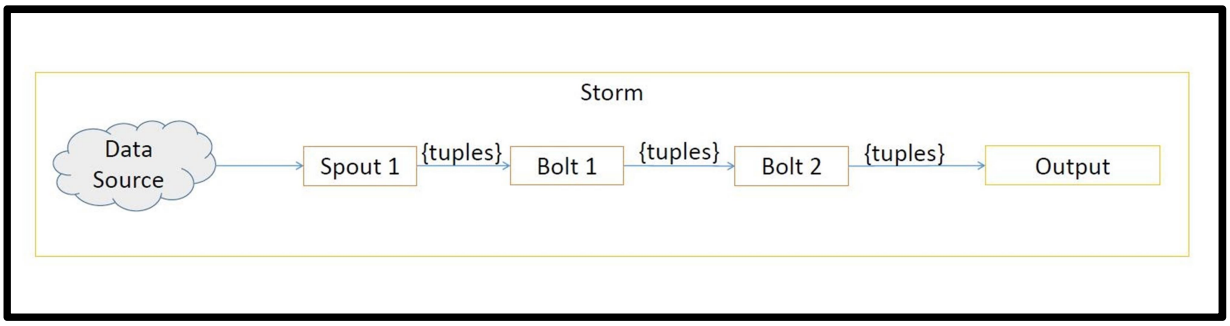
A task performs actual data processing. So, it is either a spout or a bolt.

ZooKeeper framework

Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintaining shared data with robust synchronization techniques. Nimbus is stateless, so it depends on ZooKeeper to monitor the working node status. ZooKeeper helps the supervisor to interact with the nimbus. It is responsible to maintain the state of nimbus and supervisor.

Storm Components

Storm provides two types of components that process the input stream, **spouts** and **bolts**. Spouts process external data to produce streams of tuples. Spouts produce tuples and send them to bolts. Bolts process the tuples from input streams and produce some output tuples. Input streams to bolt may come from spouts or from another bolt.

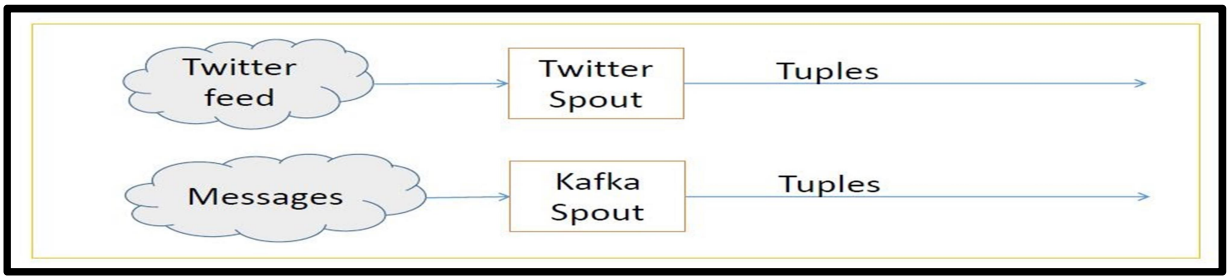


The diagram shows a Storm cluster consisting of one spout and two bolts. The spout gets the data from an external data source and produces a stream of tuples. The first bolt takes the output tuples from the spout and processes them to produce another set of tuples. The second bolt takes the output tuples from bolt 1 and stores them into an output stream.

Storm Spout

A spout can create a stream of tuples from the input, and it automatically serializes the output data. It can get data from other queuing systems like Kafka, Twitter, RabitMQ, etc. Spout implementations are available for popular message producers such as Kafka and Twitter. A single spout can produce multiple streams of output. Each stream output can be consumed by one or more bolts.

The diagram shows a twitter spout that gets twitter posts from a twitter feed and converts them into a stream of tuples. It also shows a Kafka spout that gets messages from Kafka server and produces a tuple of messages.



Storm Bolt

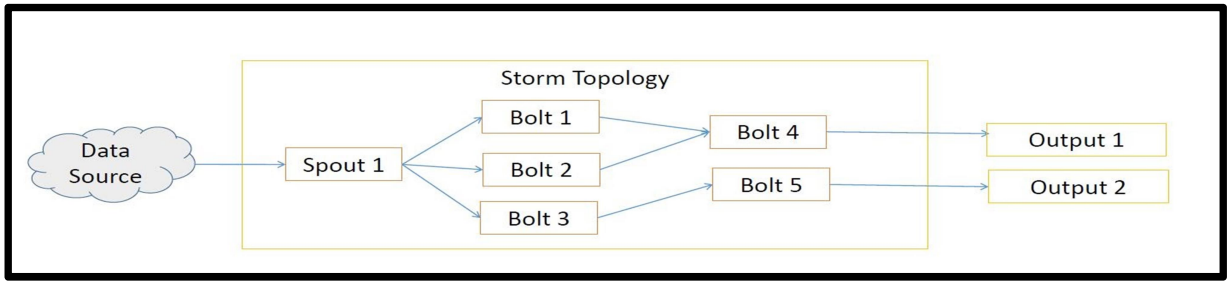
Storm Bolt processes the tuples from spouts and outputs to external systems or other bolts. The processing logic in a bolt can include filters, joins, and aggregation.
Filter data examples include processing only records with STATUS = GOOD, processing only records with volume > 100, etc. Aggregation examples include calculating the sum of sale amount, calculating the max stock value, etc.
A bolt can process any number of input streams. Input data is deserialised, and the output data is serialized.
Bolts run in parallel and can distribute across machines in the Storm cluster.

Tuple

The tuple is the main data structure in Storm. A tuple is a named list of values, where each value can be any type. Tuples are dynamically typed — the types of the fields do not need to be declared. Tuples have helper methods like getInteger and getString to get field values without having to cast the result.

Storm Topology

A group of spouts and bolts running in a Storm cluster form the Strom Topology. Spouts and bolts run in parallel. There can be multiple spouts and bolts. Topology determines how the output of a spout is connected to the input of bolts and how the output of a bolt is connected to the input of other bolts.



The diagram shows a Storm Topology with one spout and five bolts. The output of spout 1 is processed by three bolts: bolt 1, bolt 2 and bolt 3. Bolt 4 gets the output from bolt 2 and bolt 3. Bolt 5 gets the input from Bolt 5. This represents the storm topology. The input to spout 1 is coming from an external data source. The output from bolt 4 goes to Output 1 and the output from bolt 5 goes to output 2.

Apache Storm – Installation

Step 1: First we will update the system using the command:

$ *sudo apt-get update*

Step 2: Install java and python in your system

>>> Installing **Java.** Use the following commands:

$ *sudo apt-get install openjdk-8-jre-headless*

$ *sudo apt-get install openjdk-8-jdk-headless*

>>> To see the location of java installation, use the command:

$ *sudo update-alternatives --config java*

>>> To set environment variable use the command:

$ *sudo gedit /etc/environment*

>>> A file will open. Add the following line

JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"

>>> Save the file and use the below command to refresh.

$ *source /etc/environment*

>>> Check the java version by using

$ *java –version*

>>> Installing **python** using the command:

$ *sudo apt install python-is-python3*

Step 3: Make any folder on desktop where you want to install the Apache storm setup. Let's say "**Apache_Storm"**

Step 4: Download the Apache Storm and Apache Zookeeper files.

$ *wget https://dlcdn.apache.org/storm/apache-storm-2.4.0/apache-storm-2.4.0.tar.gz*

$ *wget https://dlcdn.apache.org/zookeeper/zookeeper-3.7.1/apache-zookeeper-3.7.1-bin.tar.gz*

Step 4: Extract both the files and put them inside the folder you created in Step 3.

Step 5: We have to edit the .**bashrc** file. Open it using the command:

$ *gedit ~/.bashrc*

>>> Insert these lines in .bashrc file

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
export STORM_HOME=/home/iamnitishsaxena/Apache_Storm/apache-storm-2.4.0
export PATH=$PATH:$STORM_HOME/bin
export ZOOKEEPER_HOME=/home/iamnitishsaxena/Apache_Storm/apache-zookeeper-3.7.1-bin
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

***Note*: Change the path according to your system.**

>>> Now, for applying all the changes, use the below command:

$ *source ~/.bashrc*

Step 6: Now, go to the apache zookeeper folder, open conf folder, then rename the **zoo_sample.cfg** file to **zoo.cfg**

Step 7: Now, open the **zoo.cfg** file. It will have many parameters. For the details of these parameters, you can refer to the below link:

https://zookeeper.apache.org/doc/r3.7.1/zookeeperAdmin.html

Step 8: In the zoo.cfg file, change the location of **dataDir**

dataDir=/var/zoo

>>> Save and close the file.

>>> Go to the **var** folder and then make a directory in var using following command:

$ *cd /var*

$ *sudo mkdir zoo*

>>> Grant read, write, and execute permissions for everyone using the command:

$ *sudo **chmod 777 zoo***

Step 9: Now, we have to add the location where storm data should be stored. For this, make a directory inside the folder you created in step 3 i.e. **Apache_Storm**

>>> Go inside **Apache_storm** folder and create **storm1** directory using the command

$ *cd Apache_Storm*

$ *sudo mkdir storm1*

Step 10: Now, we have to change some parameters in storm folder.

>>> Go to the folder where apache Storm files have been extracted, then go to conf folder and open **storm.yaml** file.

>>> We have to modify some parameters in this file.

Change the "**server** 1" as "***localhost***" and uncomment that line

>>> Now, add the following lines in **storm.yaml** file:

storm.local.dir: /home/iamnitishsaxena/Apache_Storm/storm1

Note: ***Change the path according to your system***

>>> Now, we have to specify nimbus host and ui port

nimbus.host: "localhost"

ui.port: 8090

>>> For each worker machine, you configure how many workers run on that machine with this config. By default, this setting is configured to run 4 workers on the ports 6700, 6701, 6702, and 6703. Copy paste the below lines in storm.yaml file.

supervisor.slots.ports:

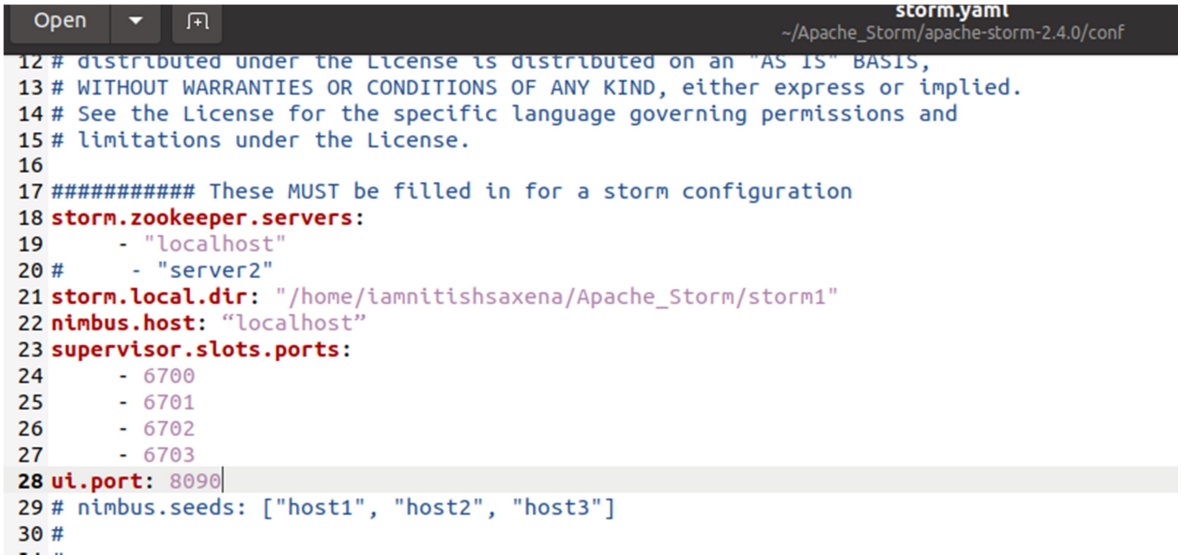
- 6700

- 6701

- 6702

- 6703

>>> Save the file and close it.



Step 11: Now, we are ready to run storm. Execute the below commands.

>>> first we have to start zookeeper server by:

$ *zkServer.sh start*

***Note*:** To stop a zookeeper, use this command *$ zkServer.sh stop*

>>> Now, we have to start the storm daemons. Execute these commands, each in a new terminal window.

$ *storm nimbus*

$ *storm supervisor*

$ *storm ui*

>>> Now, to check if everything is working properly or not, open browser and type:

http://localhost:8090

Apache Storm UI should appear.

Programs in Apache Storm

In order to run the programs in Apache Storm, we will first install **eclipse** IDE.

>>> Go to the official site, download and install the eclipse IDE

https://www.eclipse.org/downloads/

>>> Once Eclipse is installed, open it and start a new java project.

Program 1: In this program we will have a stream of integers from 0 to 99 as input and we will multiply this continuous stream of integers by 2 and produce the output stream of resultant integers as output.

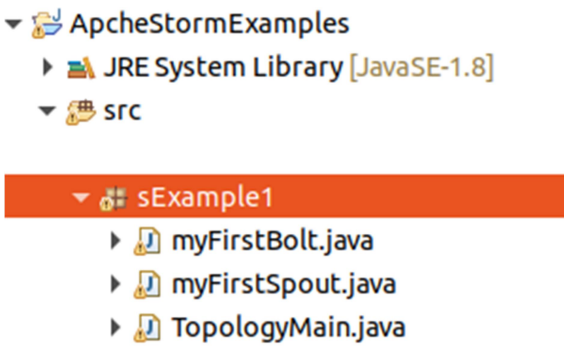
Note: *All the java files are attached with the labsheet.*

Step 1: Create a directory structure as shown in the below figure.

>>> Package name: **sExample1**

>>> Inside sExample1, 3 java classes will be there (*files provided*).

>>> Copy the file contents.

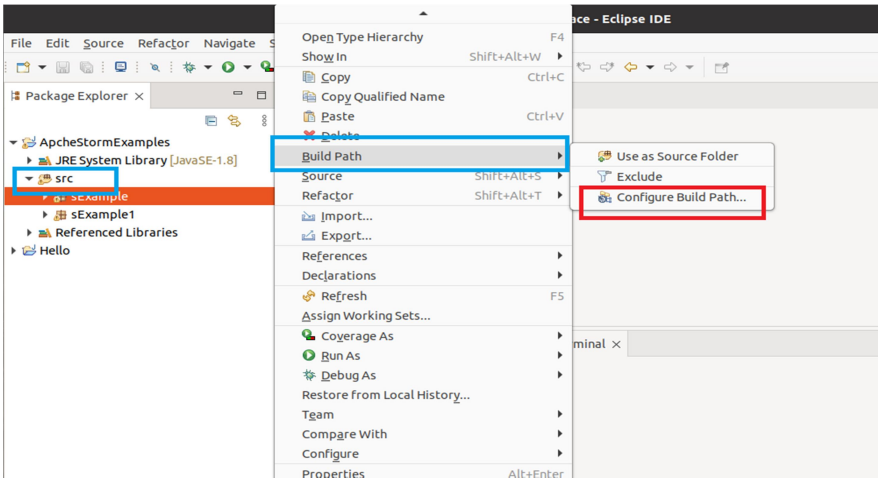


Step 2: Now, we need to include the external jar files we obtained with Apache Storm in our java project.

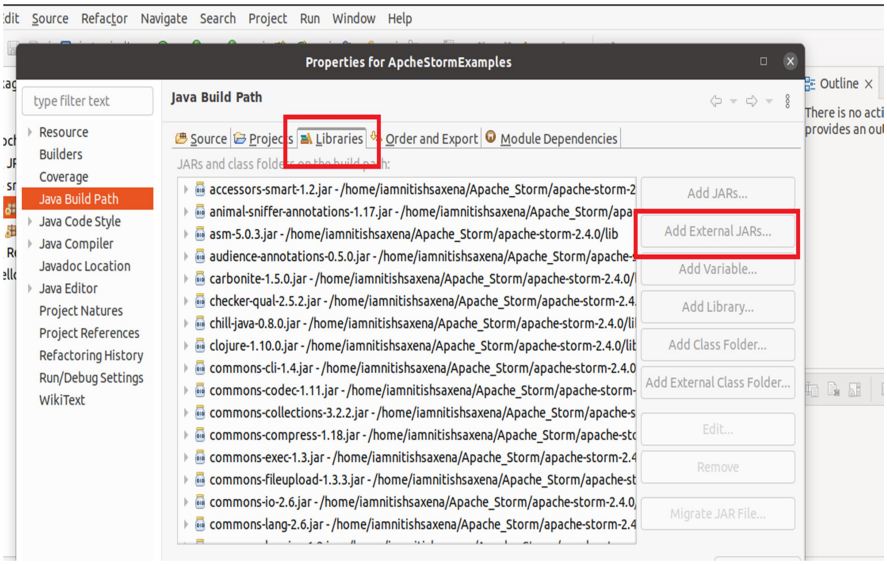
1. Right click on **src**.

2. Select **build path**.

3. Select **configure build path**.



4. A dialog box will appear. Select **libraries** tab and then select **external jars.**

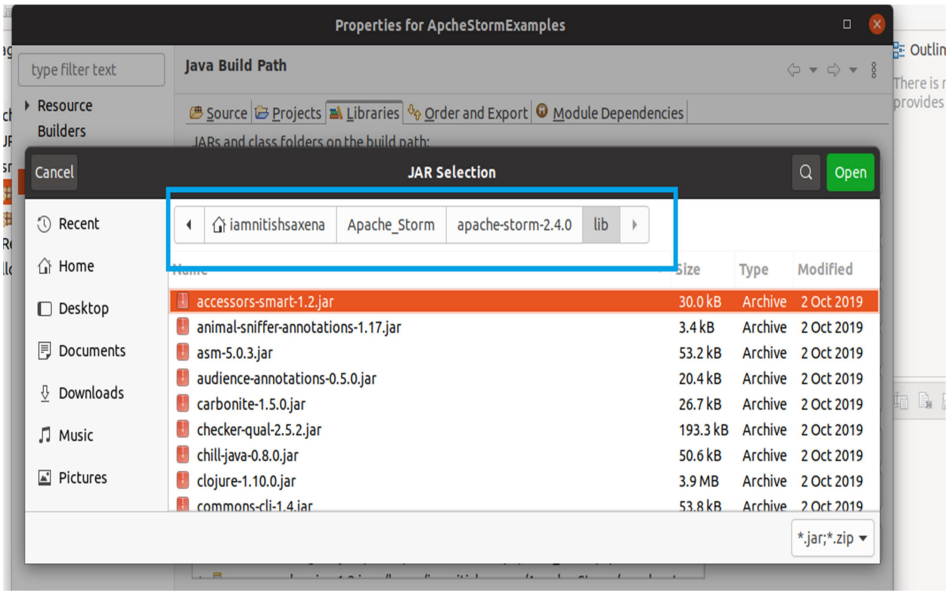


5. Go to **Apache storm** folder, then **lib** directory.

6. Select all the jar files using **ctrl+A.**

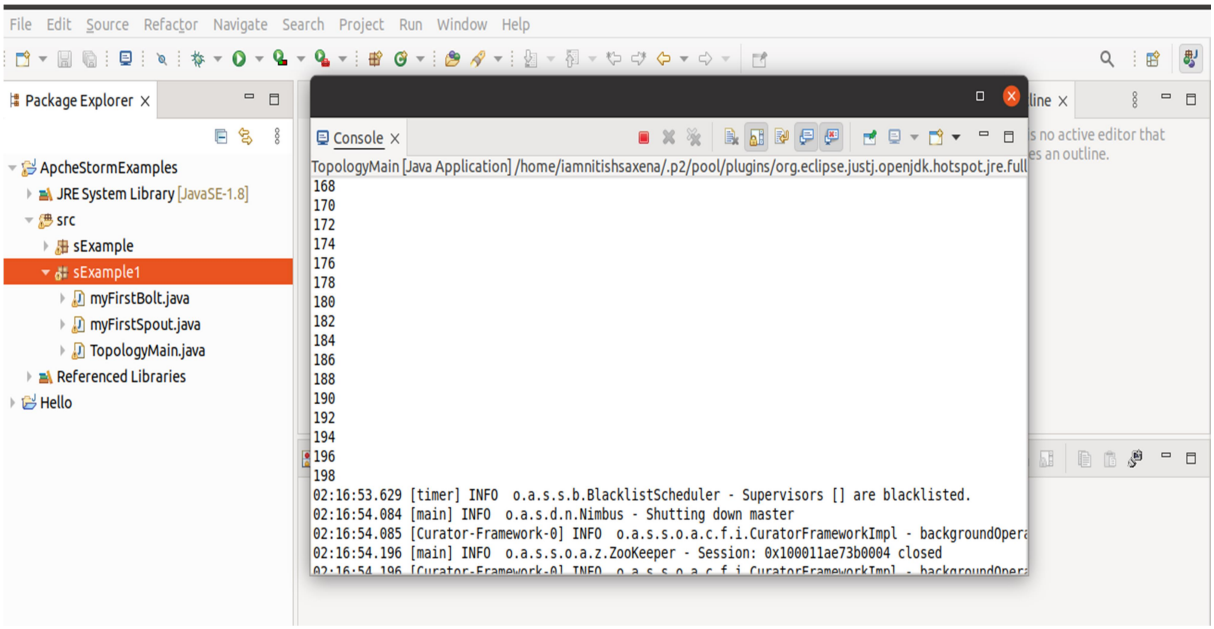
7. Click on open.

8. Apply the changes.



Step 3: Now, run the program sExample1.

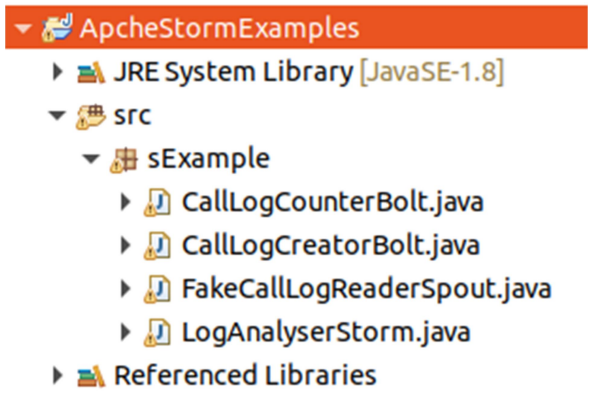
>>> The following output should appear. All the numbers from 0 to 99 given as input stream are multiplied by 2 and produced as output stream.



Program 2: Mobile call and its duration will be given as input to Apache Storm and the Storm will process and group the call between the same caller and receiver and their total number of calls.

Note: *All the java files are attached with the labsheet.*

Step 1: Create a directory structure as shown in the below figure.



>>> Package name: **sExample**

>>> Inside sExample, 4 java classes will be there (files provided).

>>> Copy the file contents.

Step 2: Run the program.

The following output should appear on the console.