Notes on building and using Apache storm on Ubuntu

Java

Install Java-8 if you don't have it:

```
sudo apt install openjdk-8-jre-headless
sudo apt install openjdk-8-jdk-headless
```

Downloading Apache Tools

Create main apache folder on home directory:

```
mkdir ~/apache
```

Download apache zookeeper and apache storm
Zookeeper_Latest_Stable_Release:
https://zookeeper.apache.org/releases.html
Storm_Latest_Release
https://storm.apache.org/downloads.html

Move the dowloaded archive files into `~/apache` and extract them:
We refer path to storm as STORM_PATH
We refer path to zookeeper as ZOOKEEPER_PATH
For example, on my computer,
STORM_PATH is `/home/sefik/apache/apache-storm-2.4.0`
ZOOKEEPER_PATH is `/home/sefik/apache/apache-zookeeper-3.7.1-bin`

Go into `.bashrc` and add these lines

```
export STORM_HOME=STORM_PATH
export PATH=$PATH:$STORM_HOME/bin
export ZOOKEEPER_HOME=ZOOKEEPER_PATH
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

Apply the changes with the command

```
source ~/.bashrc
```

Now we can use the apache executables from anywhere in the command line.

Python Requirement

But we need to do one more thing. Install Python3 if you haven't yet.

```
sudo apt install python3
```

Storm uses python while calling it python instead of python3. In your computer you may have installed python3 and python may result in undefined command. To solve this install python-is-python3 package

```
sudo apt install python-is-python3
```

Now storm should work from the command line
Run

```
storm version
```

to verify that it works

Zookeeper should also be working

Configuration Files

Now cd into `ZOOKEEPER_PATH/conf`
You should see a file called `zoo_sample.cfg`
Copy this file into `zoo.cfg` with the command

```
cp zoo_sample.cfg zoo.cfg
```

Now cd into `STORM_PATH/conf`
You should see a file called `storm.yaml`
Create a backup for this with the command

```
cp storm.yaml storm_yaml.backup
```

Now, edit `storm.yaml`
Add the line

```
storm.zookeeper.servers:
    - "localhost"
```

This will tell storm where to look for a zookeeper server, since we are trying to make a local cluster, we write "`localhost`" here. We could write multiple servers.

Add the line

```
nimbus.seeds: ["localhost"]
```

This will tell storm where to look for nimbus seed. Since we are running nimbus on local cluster, we write localhost here. You could write multiple servers also.

Add the line

```
storm.local.dir: "~/apache/storm1"
```

Or any folder you like. This will be used to store snapshot files like .jar files.

Add the line

```
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

Lastly, add the line

```
ui.port: 8087
```

To the end of the file. Port 8080 is the default one but often it is occupied. You can write any port you wish here but remember it when we try to open ui window on a browser.

Source for the configuration file

You can see detailed explanation of the storm.yaml file here. And default values on the storm.yaml here.

Starting a Local Cluster

Start zookeeper with

```
zkServer.sh start
```

Start nimbus with

```
storm nimbus
```

Start supervisor with

```
storm supervisor
```

Start ui with

```
storm ui
```

You can do these on different terminals or “Ctrl-z” and “bg” to make them work at background.

Open browser and go to “`localhost:8087`” to see the ui

Stopping storm daemons:
If you have sent them to background,

```
kill %3
kill %2
kill %1
```

Stopping zookeeper

```
zkServer.sh stop
```

Further Documentation

For further documentation refer to usage of zkCli.sh and zkServer.sh tools for Zookeeper and refer to Apache Storm website.

Building a Topology

For building topologies, you need the build tool maven.
Go to

```
apache-storm-2.4.0/examples/storm-starter
```

and run,

```
mvn clean
```

This will clean the project and now you can

```
mvn package
```

to build the target directory.
Hoping all went well, if you investigate the target directory, you will see the .jar file:

```
storm-starter-2.4.0.jar
```

This is where our projects code live, and this is the project .jar file that we will reference further down this tutorial.

Investigation of a Topology

Now, let us examine ExclamationTopology in this package. This topology's spout sends one of 5 strings at random to a bolt. This bolt appends "!!!" to that string and sends it to the last bolt. The last bolt also appends "!!!" to the string it received.

Examine the file:

```
src/jvm/org/apache/storm/starter/ExclamationTopology.java
```

```
public class ExclamationTopology extends ConfigurableTopology {

    public static void main(String[] args) throws Exception {
        ConfigurableTopology.start(new ExclamationTopology(), args);
    }
…
```

We pass down the arguments to our topology here. Typically, name of the topology is given in the arguments.

```
…
protected int run(String[] args) {
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("word", new TestWordSpout(), 10);
        builder.setBolt("exclaim1", new ExclamationBolt(),
3).shuffleGrouping("word");
```

```
        builder.setBolt("exclaim2", new ExclamationBolt(),
2).shuffleGrouping("exclaim1");

        conf.setDebug(true);

        String topologyName = "test";

        conf.setNumWorkers(3);

        if (args != null && args.length > 0) {
            topologyName = args[0];
        }

        return submit(topologyName, conf, builder);
    }
```

…

In the run() function, we actually build our topology.
We create a new TopologyBuilder Object.
We set a spout on this object with a previously prepared spout class (we will examine this too).
Configure the spout's id ("word") and an implementation of the IRichSpout interface (TestWordSpout()), and the parallelism hint (10).
We set a bolt on this object with the bolt's id ("exclaim1") and an implementation of the IRichBolt interface (ExclamationBolt()), and configure it so it will receive inputs from the component "word" (which is the spout we just set) and its grouping (shuffle grouping in this case).
Shuffle grouping distributes tuples evenly across each worker of the bolt. So, in this case, the words coming out of the spout "word", will go to the workers of "exclaim1" bolt evenly.
"Exclaim2" bolt is constructed similarly.
Set Debug configuration to true to get some debug info in logs.
Set a topology name.
Set number of workers across the cluster. See this explanation: How many processes should be spawned around the cluster to execute this topology. Each process will execute some number of tasks as threads within them. This parameter should be used in conjunction with the parallelism hints on each component in the topology to tune the performance of a topology. The number of workers will be dynamically calculated when the Resource Aware scheduler is used, in which case this parameter will not be honored.
See here.
Set the topology name if one is provided in the arguments, if not it will remain "test".
Submit the topology and return.

…

```
public static class ExclamationBolt extends BaseRichBolt {
        OutputCollector collector;

        @Override
        public void prepare(Map<String, Object> conf, TopologyContext context,
OutputCollector collector) {
            this.collector = collector;
        }

        @Override
        public void execute(Tuple tuple) {
```

```
            collector.emit(tuple, new Values(tuple.getString(0) + "!!!"));
            collector.ack(tuple);
        }

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

    }
}
```

Here, we define the ExclamationBolt. The most important function is the execute function which appends "!!!" to given tuple's 0^{th} element and emits it down the stream. Note that the 0^{th} element is pulled by getString function so it is a string and "!!!" can be appended naturally in java with "+" operator.
Then, we send an ack packet to all interested parties, this is not important for now.

declareOutputFields declares the name of the fields in the tuple. Right now, we have only one field and its name is "word".

Now let us examine the spout.
The spout is in

`storm-client/src/jvm/org/apache/storm/testing/TestWordSpout.java`

The most important function is the nextTuple().

```
@Override
public void nextTuple() {
        Utils.sleep(100);
        final String[] words = new String[]{ "nathan", "mike", "jackson", "golda",
"bertels" };
        final Random rand = new Random();
        final String word = words[rand.nextInt(words.length)];
        collector.emit(new Values(word));
    }
```

We can see that this chooses a random name in that array of names and emits a tuple that contains only that name.

Also see the declareOutputFields() function:

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

Here, we see that the output has one field and its name is "word".

Submitting a Topology

Submitting a Topology in Local Mode

We will work with the previous example.

Go to storm-started directory and run:

```
storm local target/storm-starter-2.4.0.jar
org.apache.storm.starter.ExclamationTopology
```

Command.
This will run a local cluster without you needing to do anything and submit the ExclamationTopology into that cluster. This is useful for debugging and testing but useless in real life.
Note that we use the .jar file that our code lives in which was mentioned at the start of this tutorial.

Storm local command takes at least two arguments. One, the .jar file ("storm-starter-2.4.0.jar"), two, the topology identifier ("ExclamationTopology").

Submitting a Topology to a Cluster

Same as local mode, only difference is you write "storm jar…" instead of "storm local …". And also you need to have a working cluster and connection to a nimbus machine before you run this command.