Lab: Word Count Topology

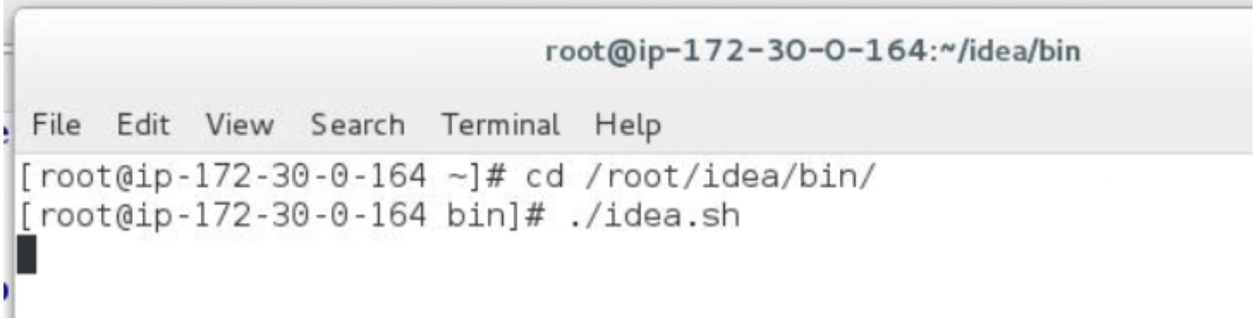
Real-time Word Count with Storm

About this Lab

Objective:	Use Storm to implement the canonical Word Count application
File locations:	`/labs/rtlabs/projects/storm/word-count`
Successful outcome:	Have successfully written a complete Storm topology with custom spout and bolt components
Before you begin	N/A
Related lesson:	Storm Architecture

Validate Skelton Project Maven Build

Launch the IntelliJ IDE by opening a new Terminal window in the desktop UI and navigating to `/root/idea/bin` and executing the `./idea.sh` script.


root@ip-172-30-0-164:~/idea/bin
File Edit View Search Terminal Help

```
[root@ip-172-30-0-164 ~]# cd /root/idea/bin/
[root@ip-172-30-0-164 bin]# ./idea.sh
```

Then open the skeleton of the Word Count project by selecting *File* > *Open* > */root/rtlabs/projects/storm/word-count* > *OK*.

NOTE: If given the option to "Enable Auto-Import" via a hover box in the lower-right corner of the IDE, click on that link.

Ensure the Maven tooling is accessible by selecting *View* > *Tool Windows* > *Maven Projects*.

Build the Maven project by selecting *Run Maven Build* after right-clicking on *Maven Projects (window)* > *Lifecycle* > *package*.

Verify the jar file was created by opening a new Terminal window in the desktop UI and navigating to `/root/rtlabs/projects/storm/word-count/target` and seeing the compiled jar files highlighted in red below.

```
[root@ip-172-30-0-164 target]# pwd
/root/rtlabs/projects/storm/word-count/target
[root@ip-172-30-0-164 target]# ls -l
total 75528
drwxr-xr-x. 3 root root       83 Jun  3 18:51 classes
drwxr-xr-x. 2 root root       27 Jun  3 18:51 maven-archiver
drwxr-xr-x. 3 root root       34 Jun  3 18:51 maven-status
-rw-r--r--. 1 root root     6830 Jun  3 18:51 original-storm-word-count-1.0-
SNAPSHOT.jar
-rw-r--r--. 1 root root 77332126 Jun  3 18:51 storm-word-count-1.0-
SNAPSHOT.jar
[root@ip-172-30-0-164 target]#
```

Complete the Sentence Generating Spout

Examine the `RandomSentenceSpout` and make the necessary code addition for declaring the field it will emit when the `nextTuple()` method is call.

```
    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer)
{
        outputFieldsDeclarer.declare(new Fields("sentence"));
    }
```

Save the changes and ensure the project can be built.

Create a Sentence Splitting Bolt

Create a new class called `SplitSentenceBolt` that subclasses `BaseBasicBolt` and allow IDEA to create stubs for any required methods.

HINT: Right-click on class name (in source code, not left-hand nav) > Generate… > Implement Methods > OK

```
public class SplitSentenceBolt extends BaseBasicBolt {

    public void execute(Tuple tuple, BasicOutputCollector
basicOutputCollector) {

    }

    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer)
{

    }
}
```

The class will break apart the sentence into individual words and emit each one separately, so as with the spout, implement the `declareOutputFields()` method.

HINT: Same code as the spout, but we are now emitting a `word` *instead of a* `sentence`

NOTE: If you have any trouble with auto-completing imports, put the cursor on the class name and press ALT-ENTER to see a list to choose from.

For the `execute()` method, tokenize the sentence that will later be wired up to be passed into this method. Emit each individual word back into the workflow.

HINT: Use the `org.apache.storm.shade.org.apache.commons.lang` *package for the* `StringUtils` *class*

```
    public void execute(Tuple tuple, BasicOutputCollector
basicOutputCollector) {
        String[] words =
StringUtils.split(tuple.getStringByField("sentence"));
        for( String word : words ) {
            System.out.println("\n*** Split Sentence Bolt *** " + word + "
***\n");
            basicOutputCollector.emit(new Values(word));
        }
    }
```

Ensure the project can be built.

Create a Word Counting Bolt

Create another bolt that extends `BaseBasicBolt`. This one will emit two values; a word and a running total.

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

For the implementation of the `execute()` method, we will need a class-level private `Map<String, Integer>` to store the rolling results with. Once you retrieve the word from the passed in `Tuple` you can create a simple block of code to update any existing count for that word; or initialize it if it is new. Then you can emit out the running total for the particular word.

```
        String word = tuple.getStringByField("word");
        Integer count = counts.get(word);
        if( count == null ) {
            count = 0;
        }
        count++;
        counts.put(word, count);
```

HINT: Don’t forget to emit the these values. Like with the declare output `Fields` object, the `Values` constructor can take more than one attribute.

Ensure the project can be built.

Assemble the Topology Submitter

Now we need to put it all together. First, build a basic Java “main” program to house the topology submission code.

```
package wordcount;

public class RandomWordCountTopology {

    public static void main(String[] args) {

    }
}
```

Instantiate a `TopologyBuilder` object and add our sentence generating spout to it.

```
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout("generator",
                new RandomSentenceSpout(), 1);
```

Wire up an instance of the word tokenizing bolt to the sentence generating spout and have the sentences be delivered in a round-robin manner.

```
        builder.setBolt("splitter",
                new SplitSentenceBolt(), 1)
                .shuffleGrouping("generator");
```

For the final bolt, use the previously created word counter. Wire it up to receive the words that come from the splitter and ensure that specific instances of the words being received are always routed to the same bolt instance.

```
        builder.setBolt("counter",
                new WordCountBolt(), 2)
                .fieldsGrouping("splitter", new Fields("word"));
```

Add the necessary code to subit the topology to a Storm cluster.

```
        Config conf = new Config();
        conf.setDebug(true);
        conf.setNumWorkers(1);

        StormSubmitter.submitTopologyWithProgressBar(
                "word-count", conf,
                builder.createTopology());
```

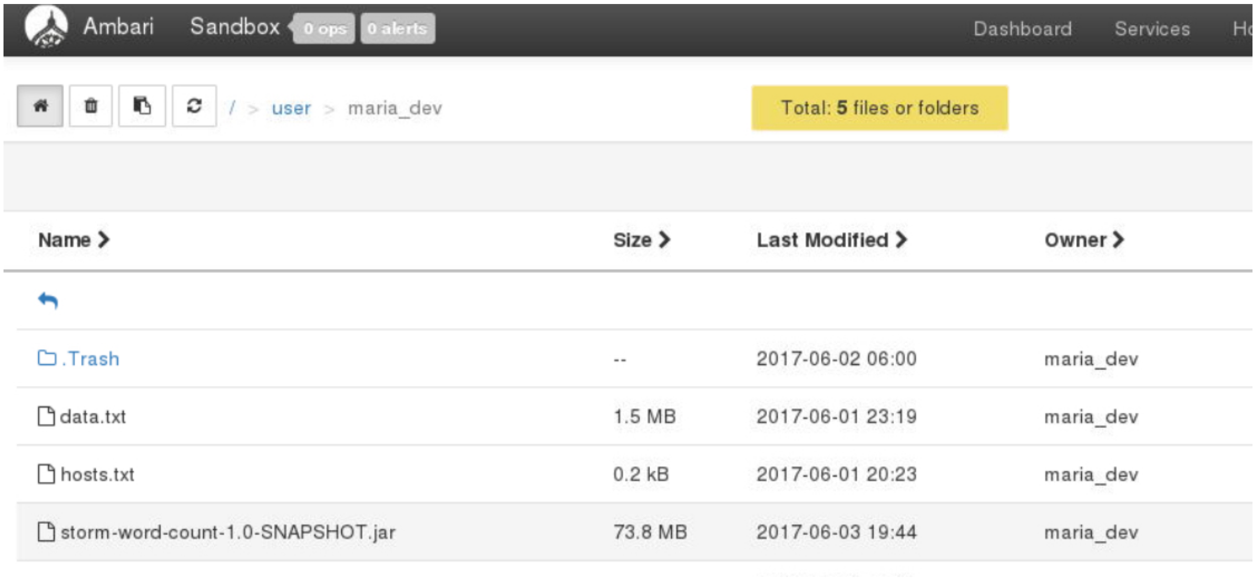
HINT: Either wrap the `submit()` method with a try/catch block or declare the exceptions on the `main()` method’s signature.

Ensure the project can be built.

Submit, Monitor, and Kill the Topology

The jar was built on the host that supports a desktop UI, but the HDP cluster is running on the Docker sandbox instance we created earlier. We will need to move the jar file to that machine, but in this configuration the scp command will not work. We will take a two-phased approach of loading it into HDFS and then pulling it back down to the sandbox.

Log into `http://127.0.0.1:8081` as user `maria_dev` with password `maria_dev`, use the *Files View* to upload `storm-word-count-1.0-SNAPSHOT.jar` into the `/user/maria_dev`.



Name	Size	Last Modified	Owner
.Trash	--	2017-06-02 06:00	maria_dev
data.txt	1.5 MB	2017-06-01 23:19	maria_dev
hosts.txt	0.2 kB	2017-06-01 20:23	maria_dev
storm-word-count-1.0-SNAPSHOT.jar	73.8 MB	2017-06-03 19:44	maria_dev

Then, log into the sandbox, become user `maria_dev` and download the jar to that local file system.

```
[root@ip-172-30-0-164 ~]# ssh -p 2222 root@127.0.0.1
root@127.0.0.1's password:
Last login: Fri Jun  2 18:49:27 2017 from 172.17.0.1
[root@sandbox ~]# su - maria_dev
[maria_dev@sandbox ~]$ hdfs dfs -get storm*.jar
[maria_dev@sandbox ~]$ ls -l storm*
-rw-r--r--. 1 maria_dev maria_dev 77335451 Jun  3 19:47 storm-word-count-1.0-
SNAPSHOT.jar
[maria_dev@sandbox ~]$
```

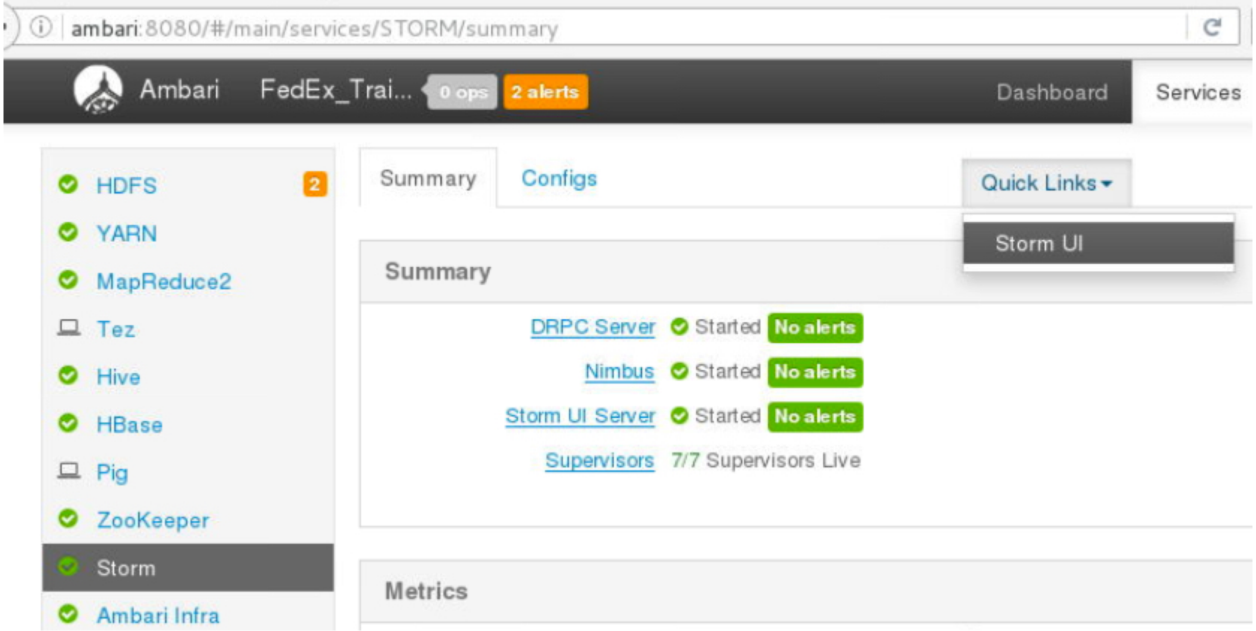
Now, you can submit the job to the Storm cluster.

```
[maria_dev@sandbox ~]$ storm jar storm-word-count-1.0-SNAPSHOT.jar
wordcount.RandomWordCountTopology
...
   ...
   ...
1532 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: word-
count
[maria_dev@sandbox ~]$
```

NOTE: You can safely ignore the following exception if it is raised.

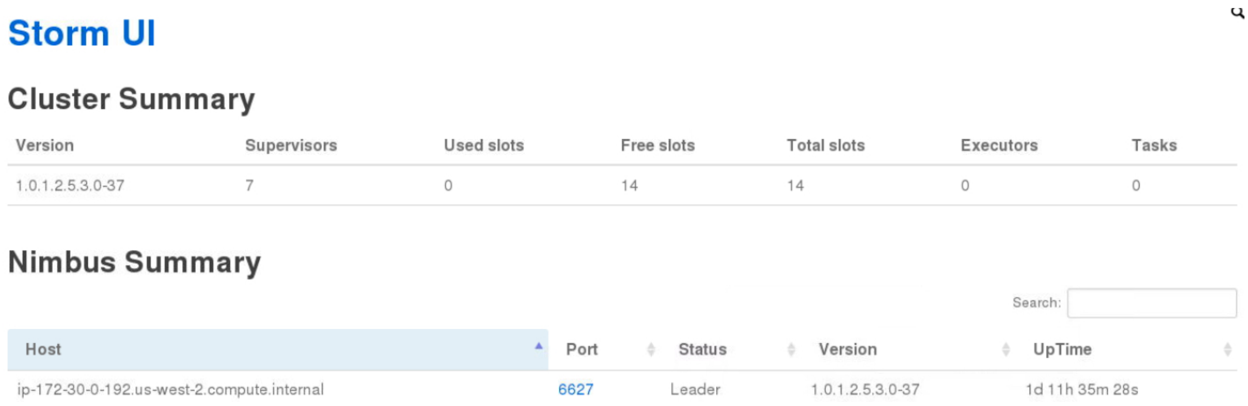
```
1532 [main] INFO  o.a.s.StormSubmitter - Initializing the registered
ISubmitterHook [org.apache.atlas.storm.hook.StormAtlasHook]
1532 [main] WARN  o.a.s.StormSubmitter - Error occurred in invoking submitter
hook:[org.apache.atlas.storm.hook.StormAtlasHook]
java.lang.ClassNotFoundException: org.apache.atlas.storm.hook.StormAtlasHook
```

From Ambari, launch the Storm UI from the link shown below.



DRPC Server ✓ Started No alerts
Nimbus ✓ Started No alerts
Storm UI Server ✓ Started No alerts
Supervisors 7/7 Supervisors Live

This will surface the Storm UI in another tab of your browser.



Storm UI

Cluster Summary

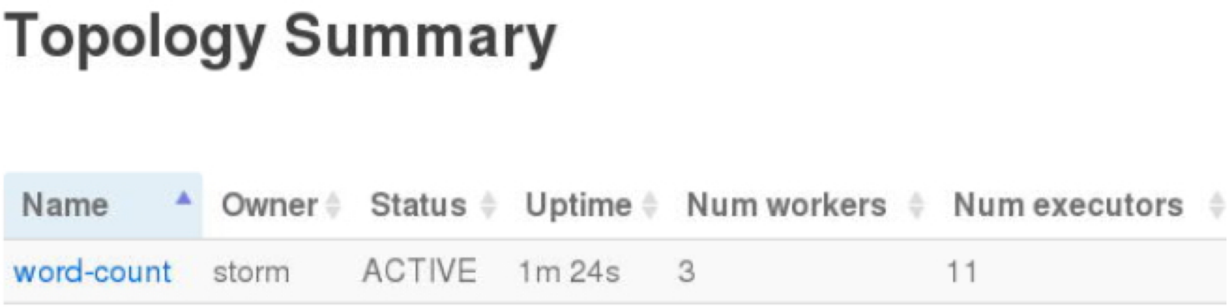
Version	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
1.0.1.2.5.3.0-37	7	0	14	14	0	0

Nimbus Summary

Search:

Host	Port	Status	Version	UpTime
ip-172-30-0-192.us-west-2.compute.internal	6627	Leader	1.0.1.2.5.3.0-37	1d 11h 35m 28s

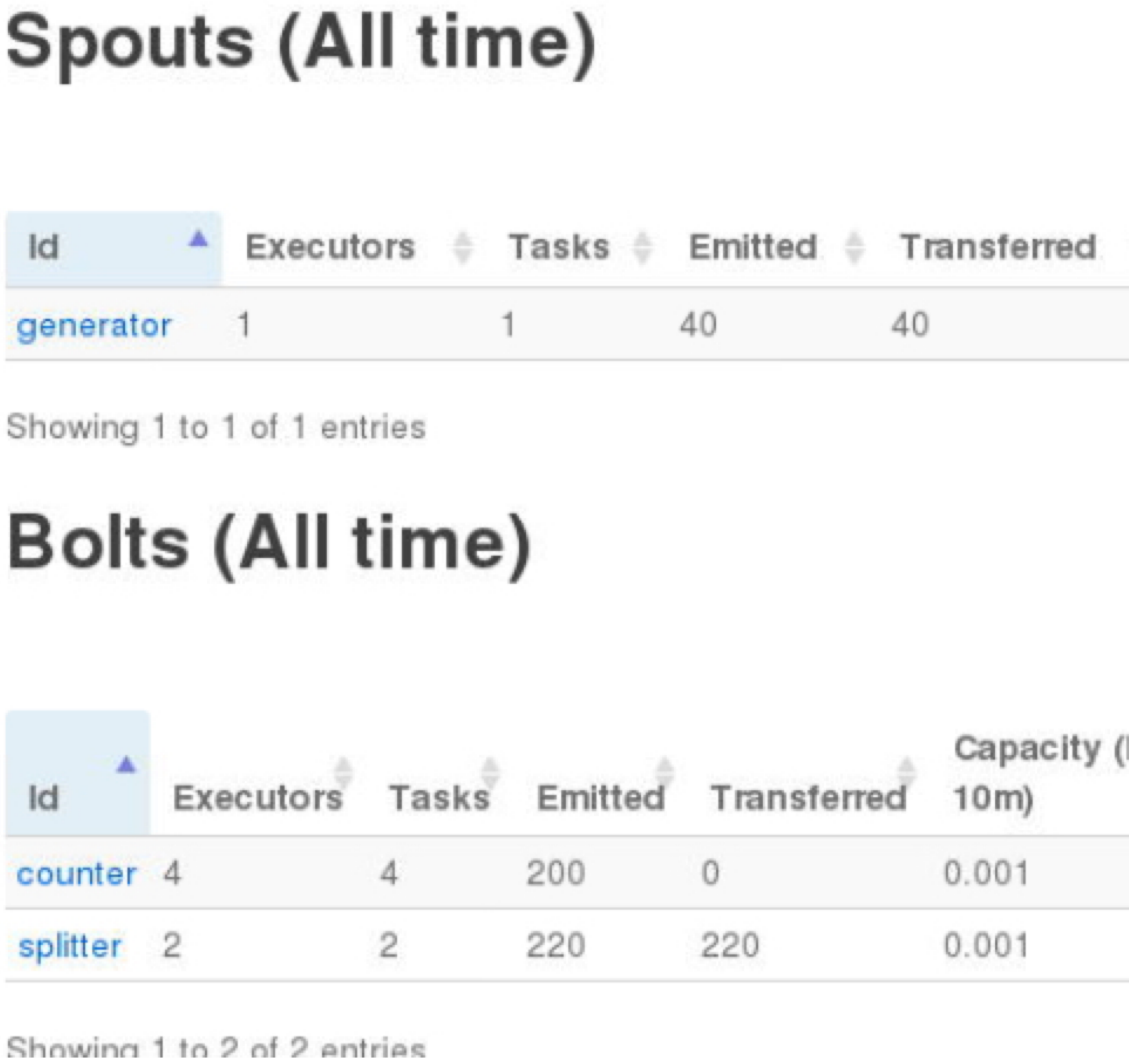
Click on the word-count link within the Topology Summary section.



Topology Summary

Name	Owner	Status	Uptime	Num workers	Num executors
word-count	storm	ACTIVE	1m 24s	3	11

You can now see links for the spout and the two bolts along with some statistics about how many items where emitted from each.



Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred
generator	1	1	40	40

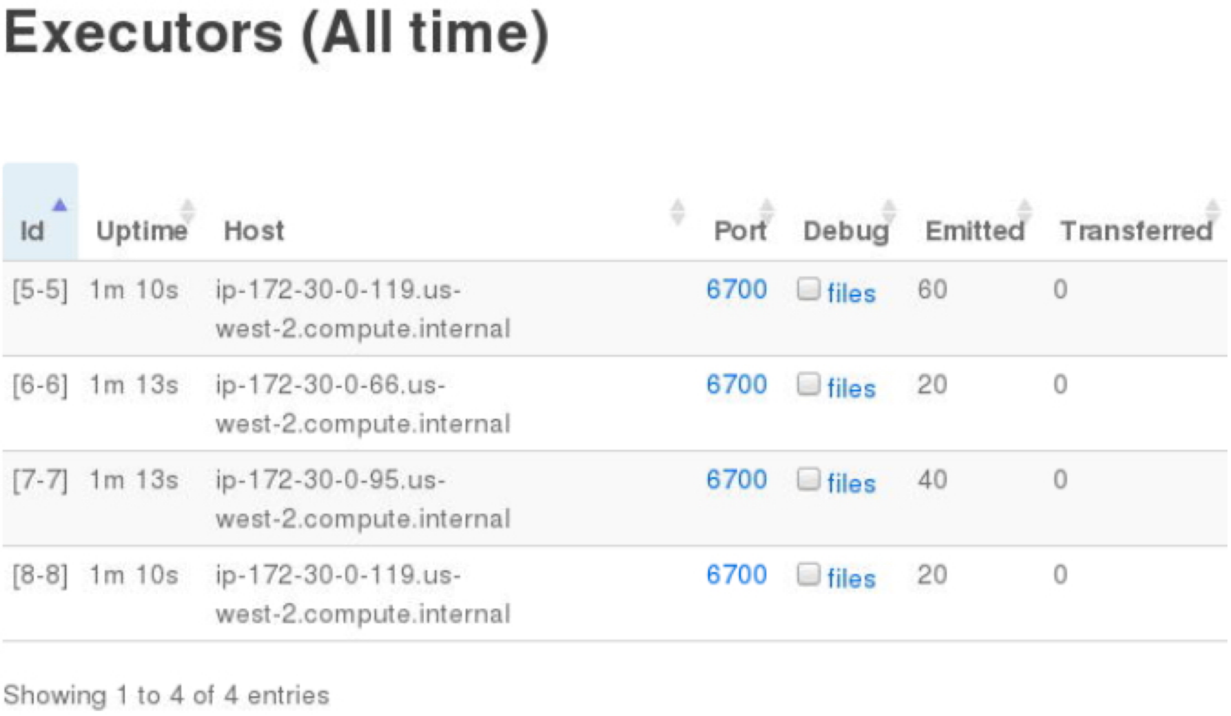
Showing 1 to 1 of 1 entries

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (10m)
counter	4	4	200	0	0.001
splitter	2	2	220	220	0.001

Showing 1 to 2 of 2 entries

Click into the spout and bolts and scroll down to the Executors section to see the unique instances of each.



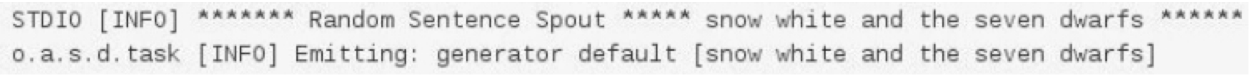
Executors (All time)

Id	Uptime	Host	Port	Debug	Emitted	Transferred
[5-5]	1m 10s	ip-172-30-0-119.us-west-2.compute.internal	6700	☐ files	60	0
[6-6]	1m 13s	ip-172-30-0-66.us-west-2.compute.internal	6700	☐ files	20	0
[7-7]	1m 13s	ip-172-30-0-95.us-west-2.compute.internal	6700	☐ files	40	0
[8-8]	1m 10s	ip-172-30-0-119.us-west-2.compute.internal	6700	☐ files	20	0

Showing 1 to 4 of 4 entries

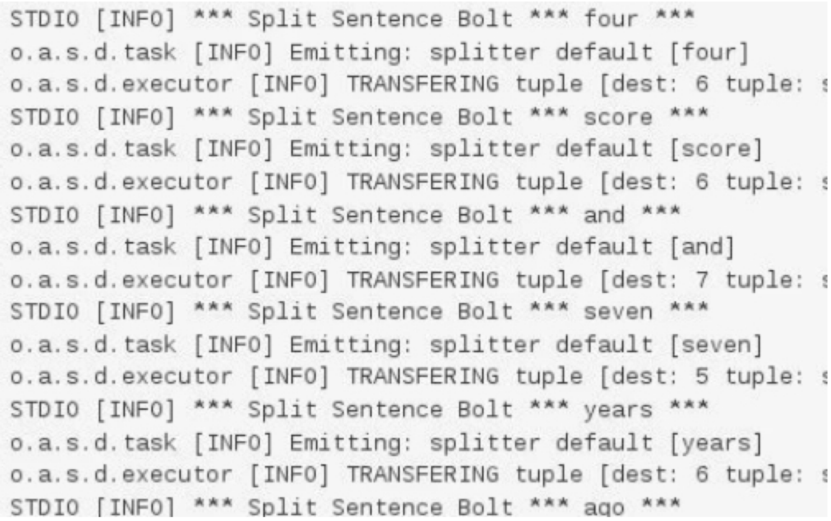
From here, click on the Port link to get log-level details of what is going on in the specific components.

Here are some log events from the spout that is generating sentences.



```
STDIO [INFO] ******* Random Sentence Spout ***** snow white and the seven dwarfs ******
o.a.s.d.task [INFO] Emitting: generator default [snow white and the seven dwarfs]
```

Details from the splitter bolt.

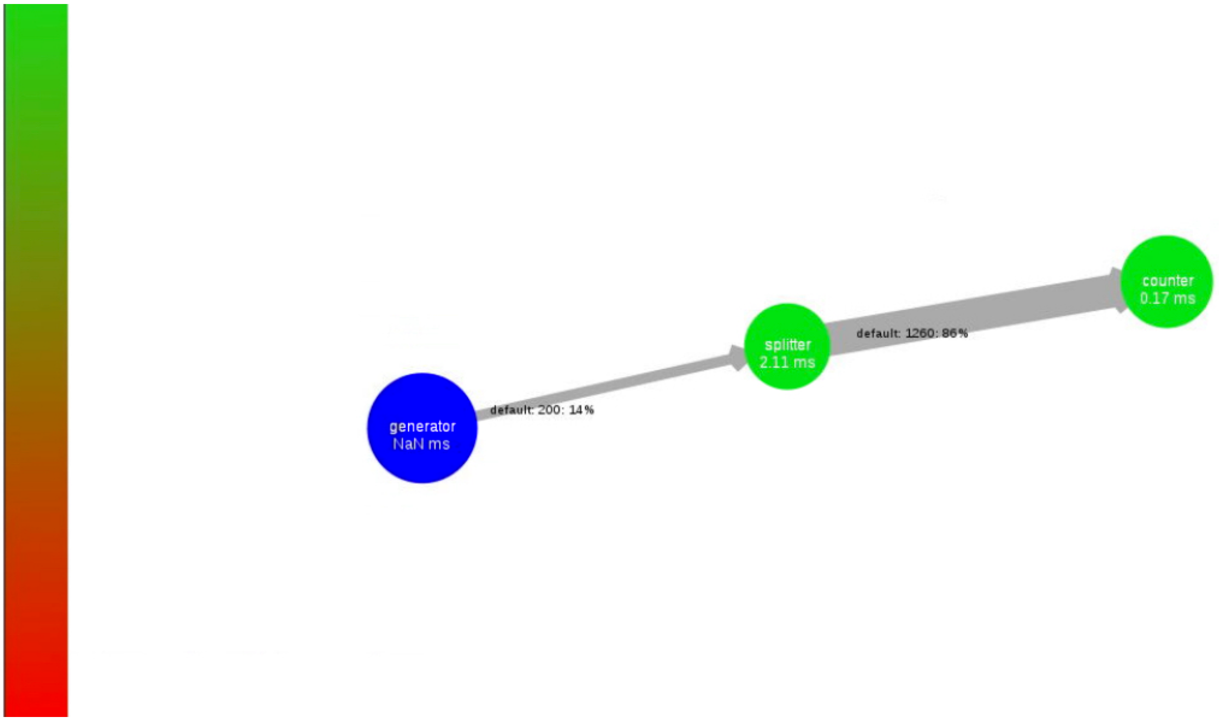


```
STDIO [INFO] *** Split Sentence Bolt *** four ***
o.a.s.d.task [INFO] Emitting: splitter default [four]
o.a.s.d.executor [INFO] TRANSFERING tuple [dest: 6 tuple:
STDIO [INFO] *** Split Sentence Bolt *** score ***
o.a.s.d.task [INFO] Emitting: splitter default [score]
o.a.s.d.executor [INFO] TRANSFERING tuple [dest: 6 tuple:
STDIO [INFO] *** Split Sentence Bolt *** and ***
o.a.s.d.task [INFO] Emitting: splitter default [and]
o.a.s.d.executor [INFO] TRANSFERING tuple [dest: 7 tuple:
STDIO [INFO] *** Split Sentence Bolt *** seven ***
o.a.s.d.task [INFO] Emitting: splitter default [seven]
o.a.s.d.executor [INFO] TRANSFERING tuple [dest: 5 tuple:
STDIO [INFO] *** Split Sentence Bolt *** years ***
o.a.s.d.task [INFO] Emitting: splitter default [years]
o.a.s.d.executor [INFO] TRANSFERING tuple [dest: 6 tuple:
STDIO [INFO] *** Split Sentence Bolt *** ago ***
```

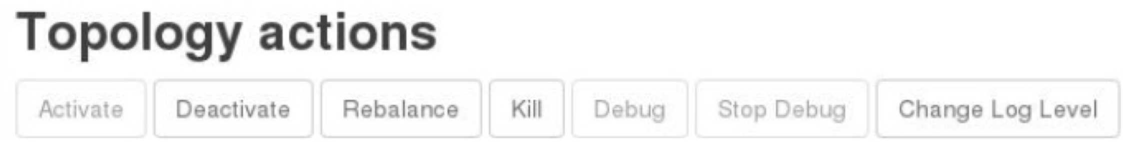
Lastly, some output from the counter bolt.

```
STDIO [INFO] *** Word Count Bolt *** day = 26 ***
STDIO [INFO] *** Word Count Bolt *** an = 26 ***
o.a.s.d.task [INFO] Emitting: counter default [day, 26]
o.a.s.d.task [INFO] Emitting: counter default [an, 26]
```

Back on the Topology Summary page you can click on the Show Visualization button to get a real-time visual perspective on the topology similar to the following.



To stop the topology from running, the Storm UI offers a Kill button.

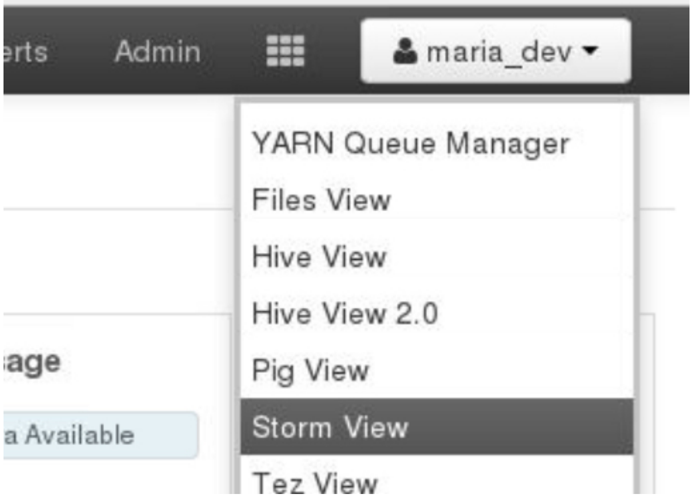


Topology actions

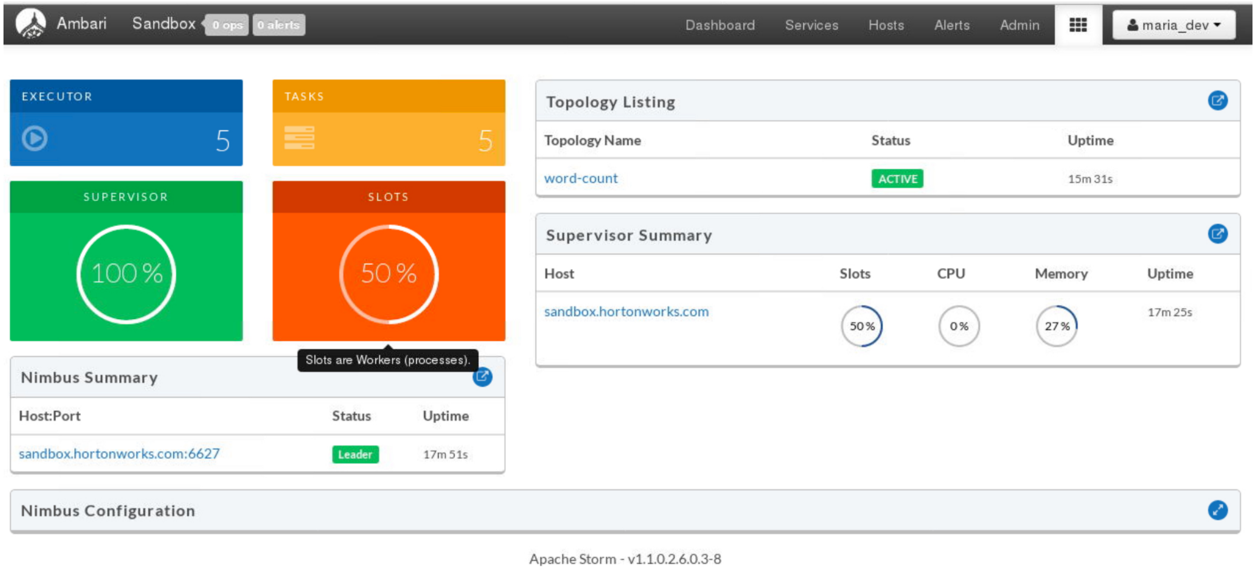
You can also shut down the execution from the command line.

```
[maria_dev@sandbox ~]$ storm kill word-count
...
    ...
        ...
2387 [main] INFO  o.a.s.c.kill-topology - Killed topology: word-count
[maria_dev@sandbox ~]$
```

Before killing the topology, or after resubmitting it, select the new Storm View in Ambari.



This provides a cleaner user experience to the same information.



Topology Listing

Topology Name	Status	Uptime
word-count	ACTIVE	15m 31s

Supervisor Summary

Host	Slots	CPU	Memory	Uptime
sandbox.hortonworks.com	50 %	0 %	27 %	17m 25s

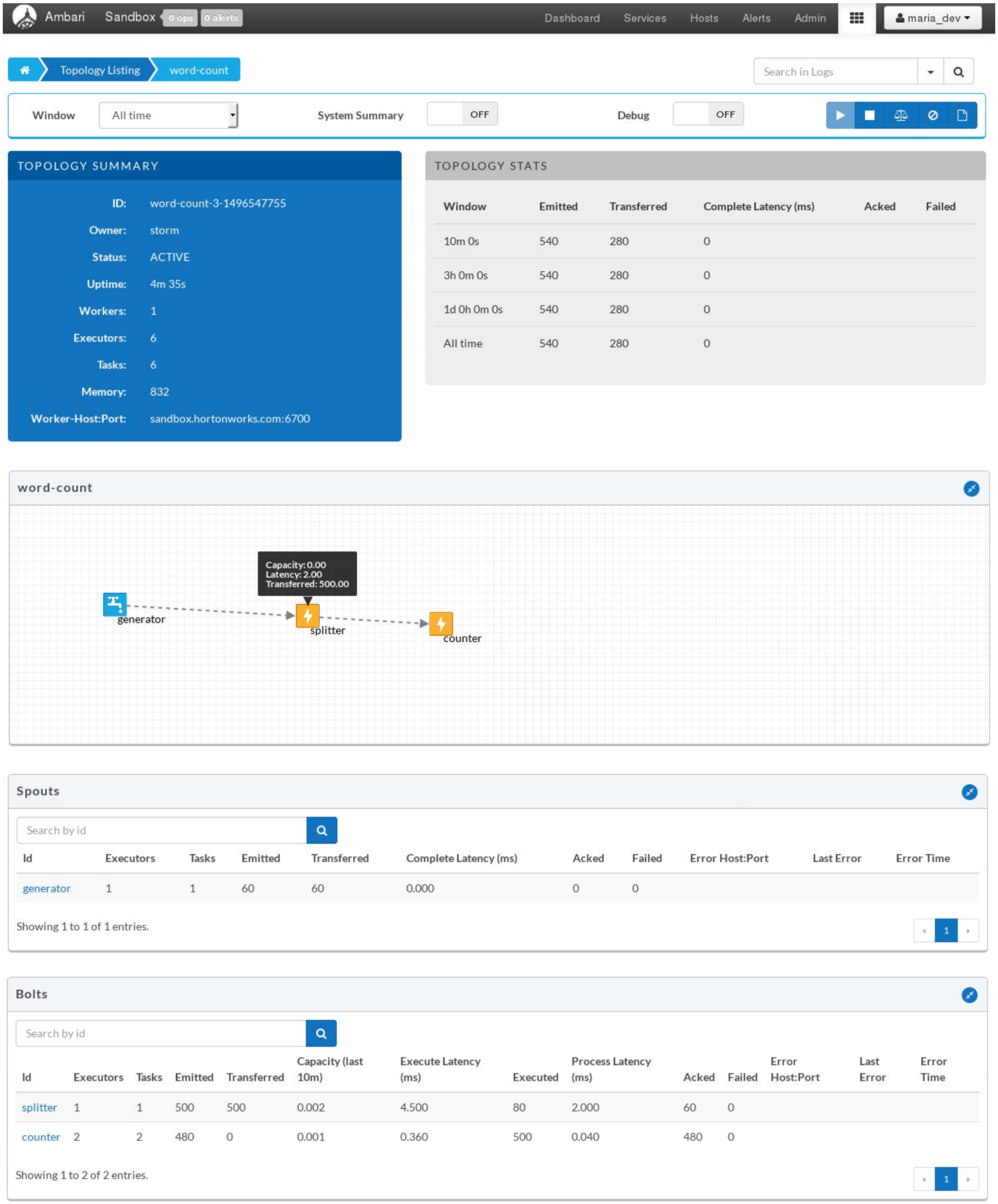
Nimbus Summary

Host:Port	Status	Uptime
sandbox.hortonworks.com:6627	Leader	17m 51s

Nimbus Configuration

Apache Storm - v1.1.0.2.6.0.3-8

Explore this alternative UI to the “classic” Storm UI.



Window: All time

System Summary: OFF

Debug: OFF

TOPOLOGY SUMMARY

ID:	word-count-3-1496547755
Owner:	storm
Status:	ACTIVE
Uptime:	4m 35s
Workers:	1
Executors:	6
Tasks:	6
Memory:	832
Worker-Host:Port:	sandbox.hortonworks.com:6700

TOPOLOGY STATS

Window	Emitted	Transferred	Complete Latency (ms)	Acked	Failed
10m 0s	540	280	0		
3h 0m 0s	540	280	0		
1d 0h 0m 0s	540	280	0		
All time	540	280	0		

word-count

Spouts

Id	Executors	Tasks	Emitted	Transferred	Complete Latency (ms)	Acked	Failed	Error Host:Port	Last Error	Error Time
generator	1	1	60	60	0.000	0	0			

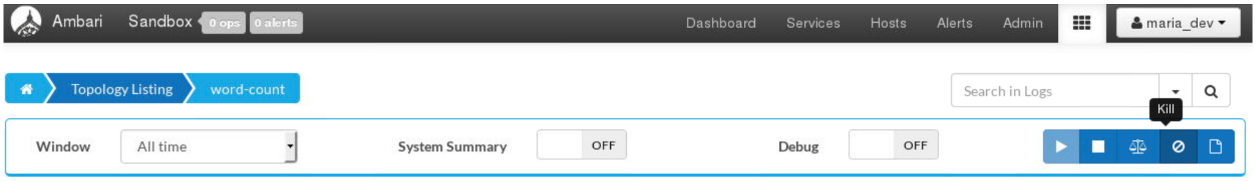
Showing 1 to 1 of 1 entries.

Bolts

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute Latency (ms)	Executed	Process Latency (ms)	Acked	Failed	Error Host:Port	Last Error	Error Time
splitter	1	1	500	500	0.002	4.500	80	2.000	60	0			
counter	2	2	480	0	0.001	0.360	500	0.040	480	0			

Showing 1 to 2 of 2 entries.

Be sure to kill the running topology when done either from the command-line or this Storm View UI's icon.



Summary

You have successful built a complete Storm topology, submitted it to the Storm cluster, and monitored this real-time application.

A complete solution to this lab can be found in `/root/rtlabs/proj-solns/storm/world-count`.