# JStrom test

[Zhongyan.feng@alipay.com](mailto:Zhongyan.feng@alipay.com) -- Longda

## Summary

JStorm is more stable and faster than storm in the following test. For example, in the shutdown node’s test, storm will fail, but JStorm pass. In the performance test, JStorm performance is triple of storm.

In the document, it will indicate stability/functionality/performance test/correctness result. If you have an idea of testing JStorm, please let me know. Thanks in advance.

目录

[JStrom test 1](#_Toc353092531)

[Summary 1](#_Toc353092532)

[Stability tests 1](#_Toc353092533)

[Kill Nimbus 1](#_Toc353092534)

[Kill supervisor 10](#_Toc353092535)

[Kill worker 20](#_Toc353092536)

[Shutdown non-nimbus node 26](#_Toc353092537)

[Shutdown nimbus node 32](#_Toc353092538)

[Functionality test 42](#_Toc353092539)

[Basic tests 42](#_Toc353092540)

[Transaction topology 42](#_Toc353092541)

[DRPC topology 43](#_Toc353092542)

[Special topology 43](#_Toc353092543)

[Non Grouping method 48](#_Toc353092544)

[Topology bumping 49](#_Toc353092545)

[Performance test 54](#_Toc353092546)

[Simplest test 54](#_Toc353092547)

[Program correctness 57](#_Toc353092548)

[Simplest test 57](#_Toc353092549)

## Stability tests

### Kill Nimbus

#### When topology is in active, kill nimbus

**Test purpose:**

The topology won’t be influence, after restart nimbus

**Test result:**

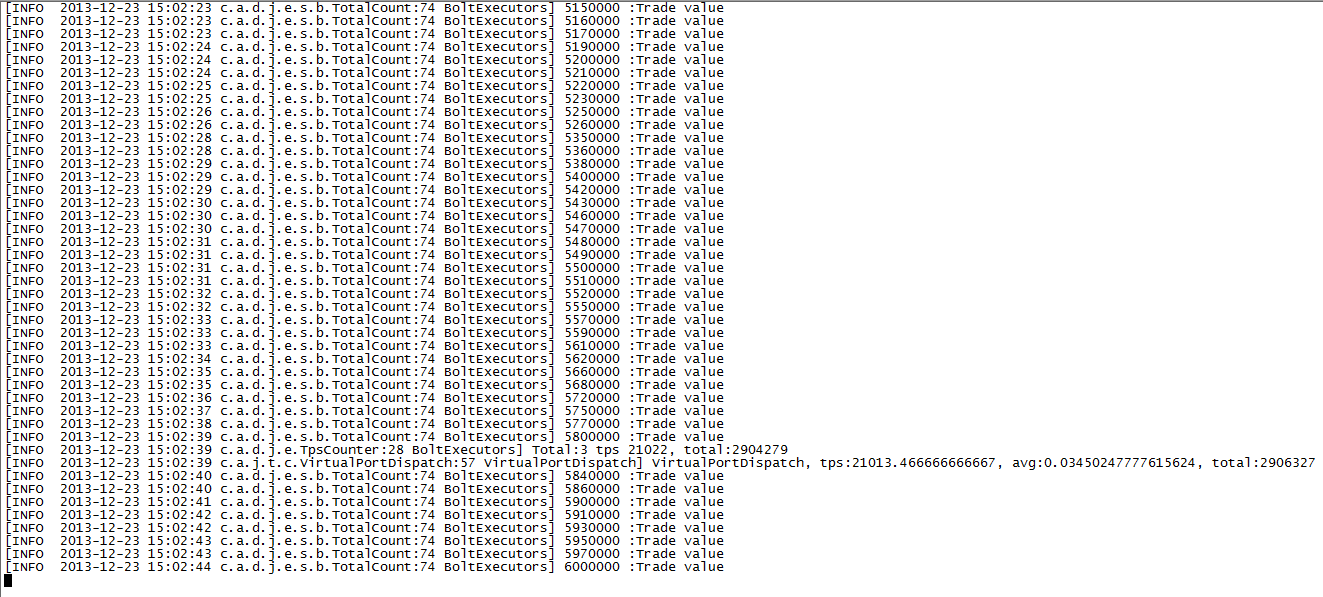
Pass, the active topology still work correctly

**Test process:**

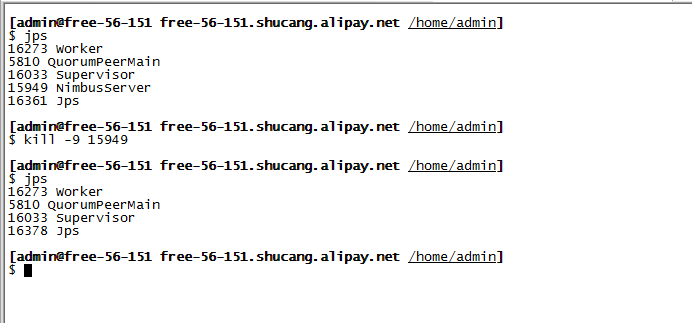
1. Submit one topology
2. Wait several minutes, kill the nimbus
3. Check whether supervisor is alive
4. Check Whether workers is alive
5. Check Whether workers work correctly
6. Start nimbus

**Test snapshots:**

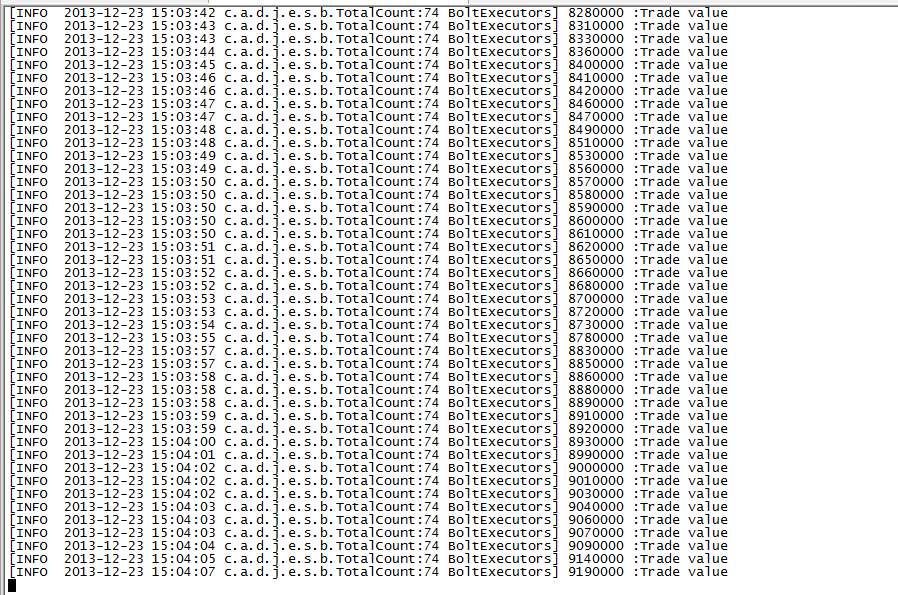
Before kill:



Kill action



After kill:



#### When topology is in inactive, kill nimbus

**Test purpose:**

The topology still be inactive, after restart nimbus

**Test result:**

Pass, the topology still be inactive, all worker process are alive.

**Test process:**

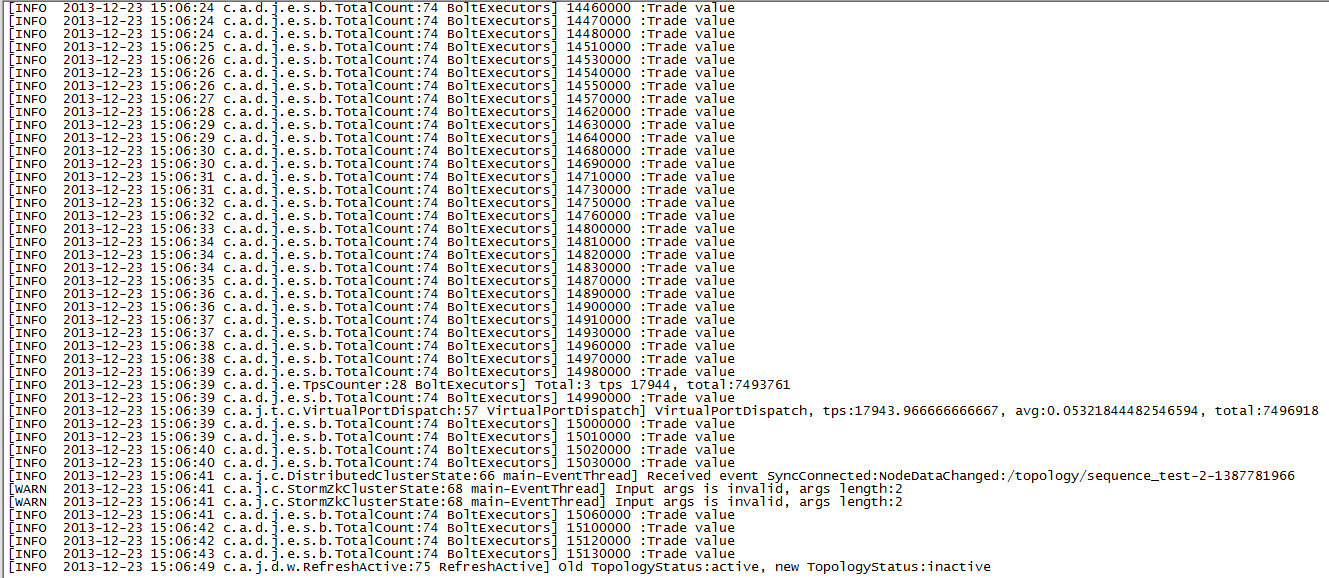
1. Submit one topology
2. Wait several minutes, deactivate the topology
3. Wait one minute, kill the nimbus
4. Check whether supervisor is alive
5. Check Whether workers is alive
6. Check Whether workers work correctly, Check whether spout send tuples
7. Start nimbus

**Test snapshots:**

Before kill



Worker log



Read topology status:



Do kill operation:

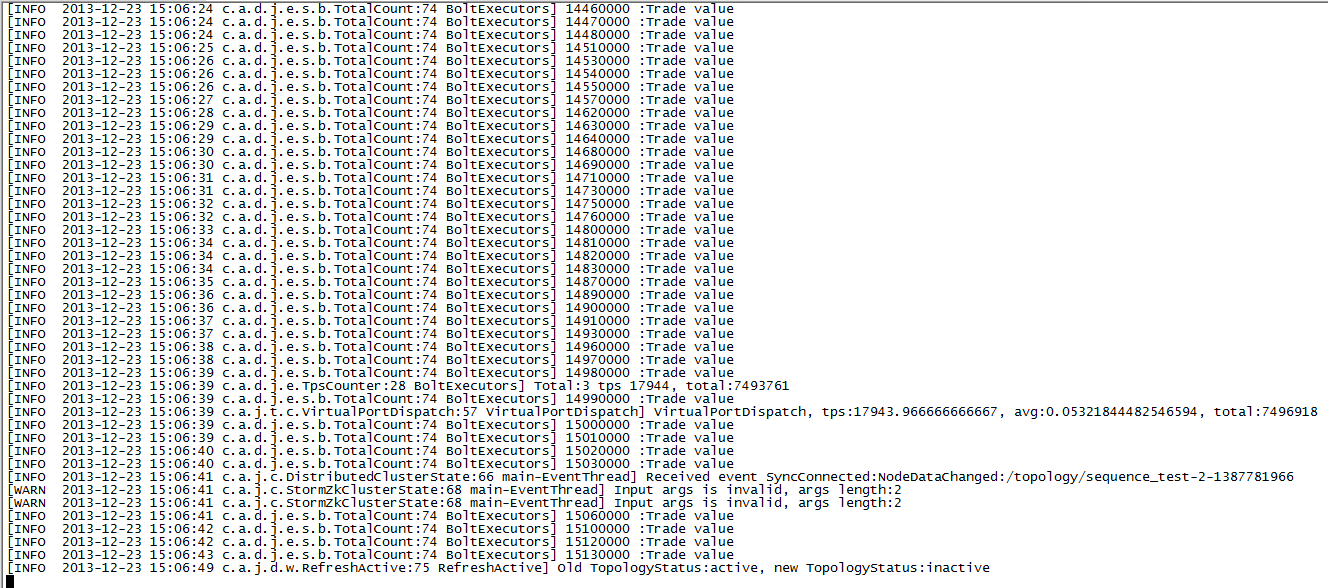


No worker has been killed

After kill:



Logs:



#### When topology is in rebalancing, kill nimbus

**Test purpose:**

The topology can continue rebalancing, after restart nimbus

**Test result:**

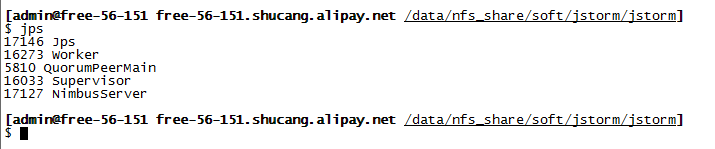
Pass, the topology continue doing rebalancing

**Test process:**

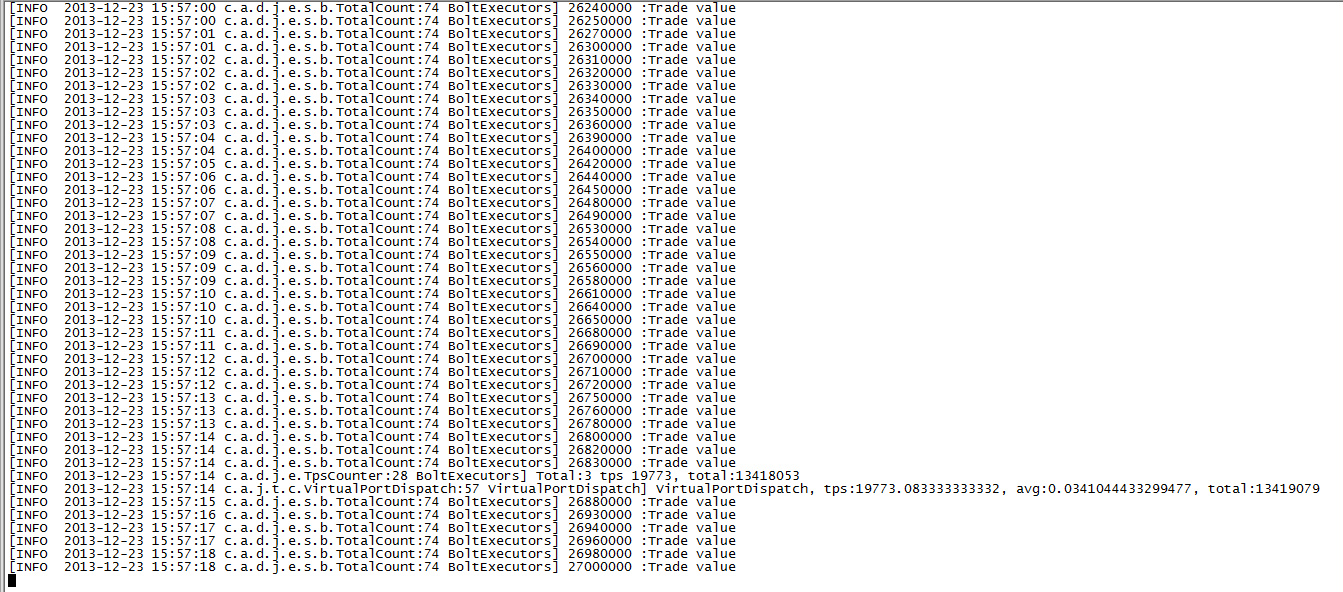
1. Submit one topology
2. Wait several minutes, commit rebalance the topology command
3. Wait one minute, kill the nimbus
4. Check whether supervisor is alive
5. Check Whether workers is alive
6. Check Whether workers work correctly, Check whether spout send tuples
7. Start nimbus
8. Check whether workers has been restart
9. Check whether workers work correctly.

**Test snapshot:**

Before submit rebalance



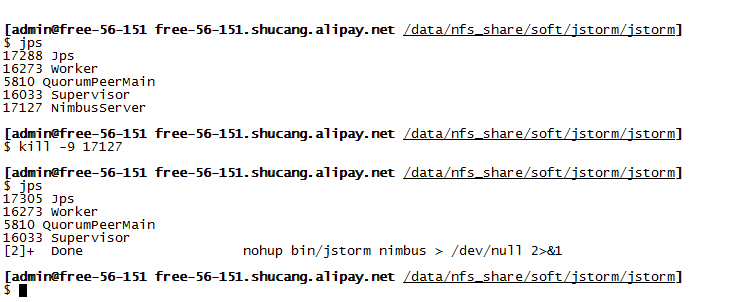
Worker logs:



Submit rebalance

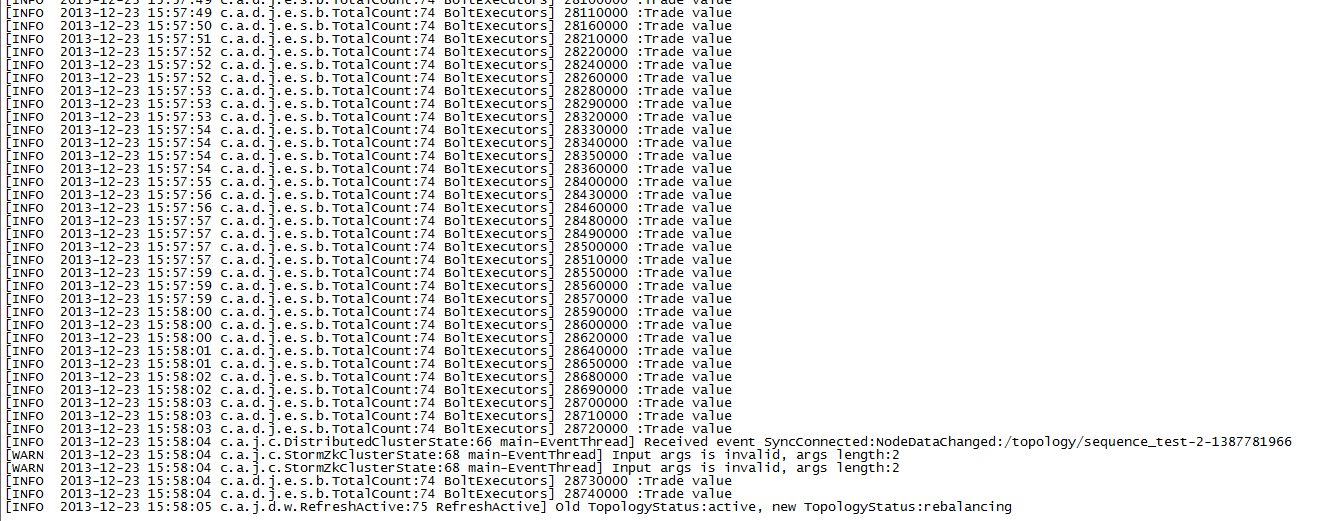


Kill nimbus:

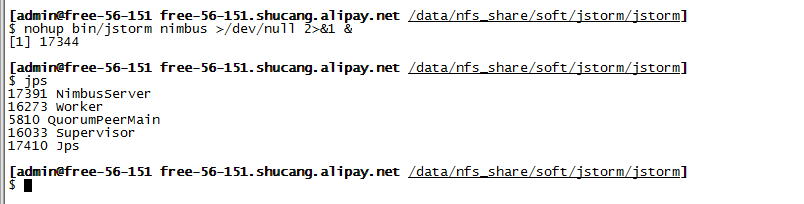


No worker has been killed

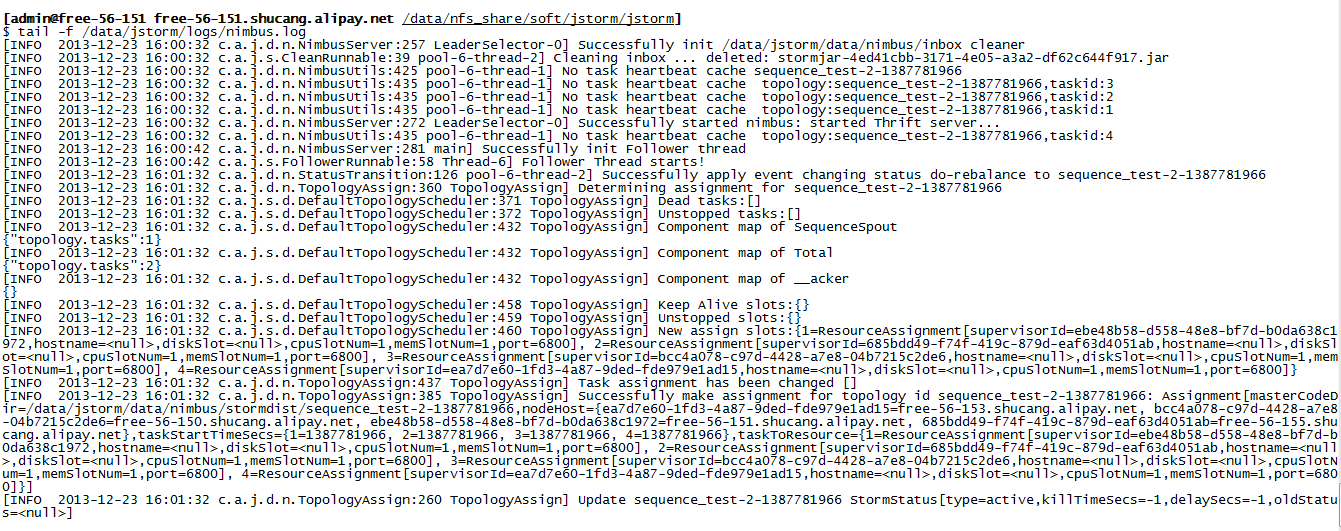
Worker log:



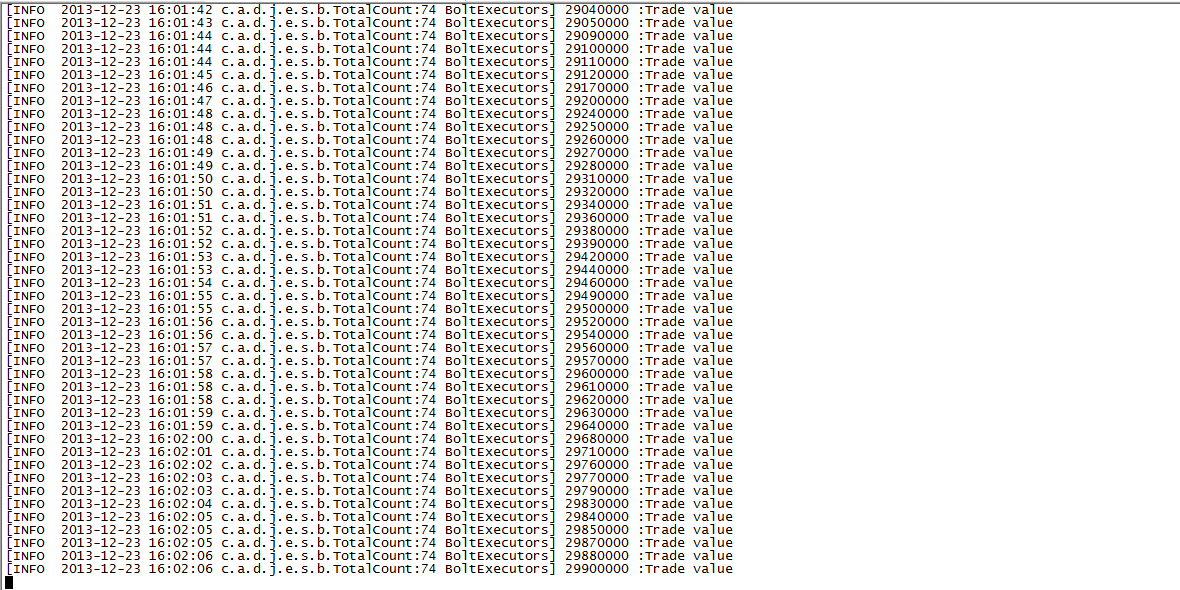
After start nimbus:



Nimbus continue doing rebalancing



Worker’s continue work



#### When topology is in killing, kill nimbus

**Test purpose:**

The topology can continue being killed, after restart nimbus

**Test result:**

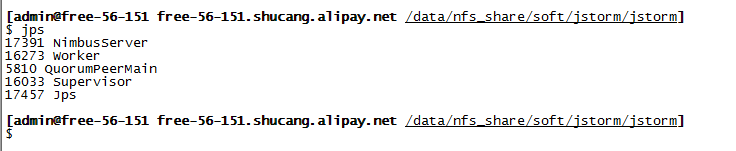
Pass, the topology continue being killed

**Test process:**

1. Submit one topology
2. Wait several minutes, commit kill the topology command
3. Wait one minute, kill the nimbus
4. Check whether supervisor is alive
5. Check Whether workers is alive
6. Check Whether workers work correctly, Check whether spout send tuples
7. Start nimbus
8. Check whether workers has been killed

**Test snapshot:**

Before kill topology:



Kill topology:



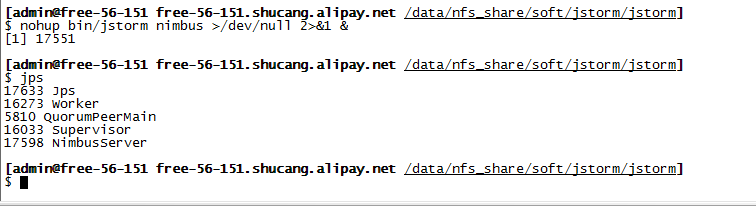


Kill nimbus:

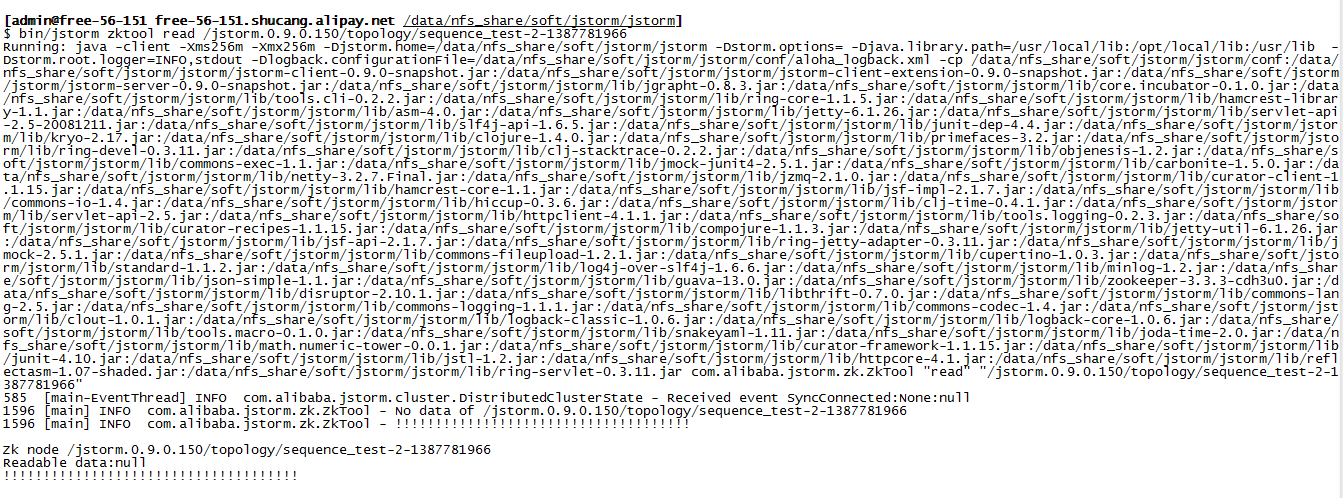


No worker has been killed at this time

After start nimbus



The topology has been killed



### Kill supervisor

#### When topology is in active, kill supervisor

**Test purpose:**

The topology won’t be influence, when restart supervisor

**Test result:**

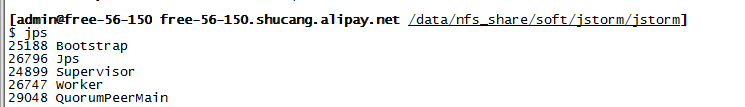
Pass, the active topology still work correctly

**Test process:**

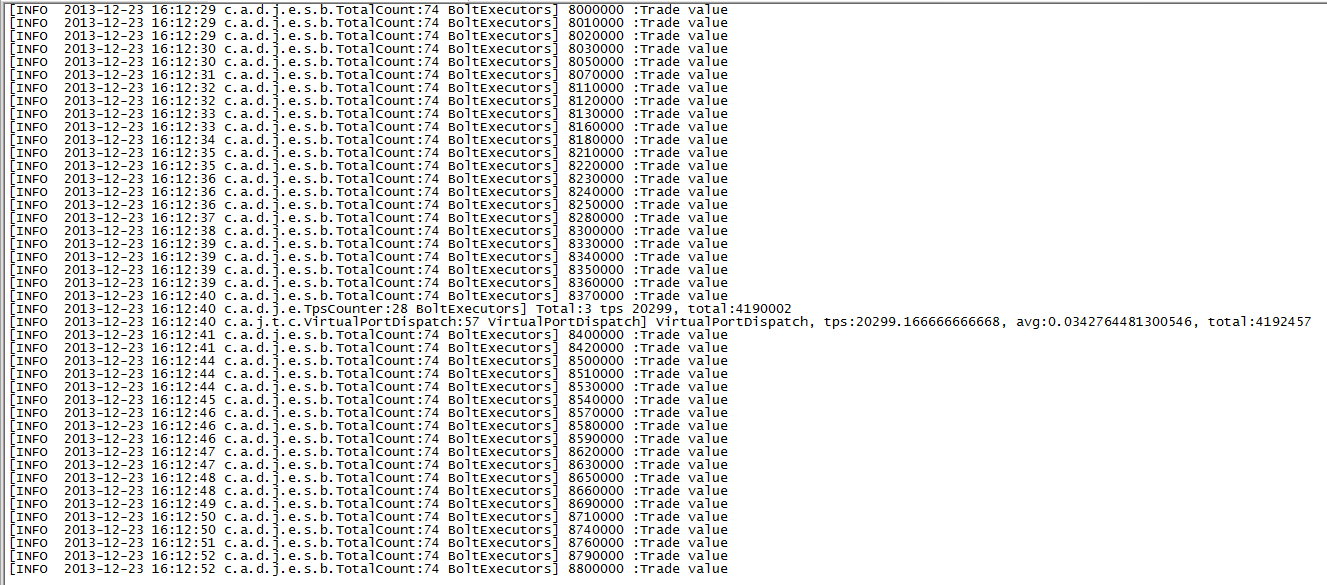
1. Submit one topology
2. Wait several minutes, kill the supervisor
3. Check whether supervisor is alive
4. Check Whether workers is alive
5. Check Whether workers work correctly
6. Start supervisor

**Test snapshots:**

Before kill supervisor:



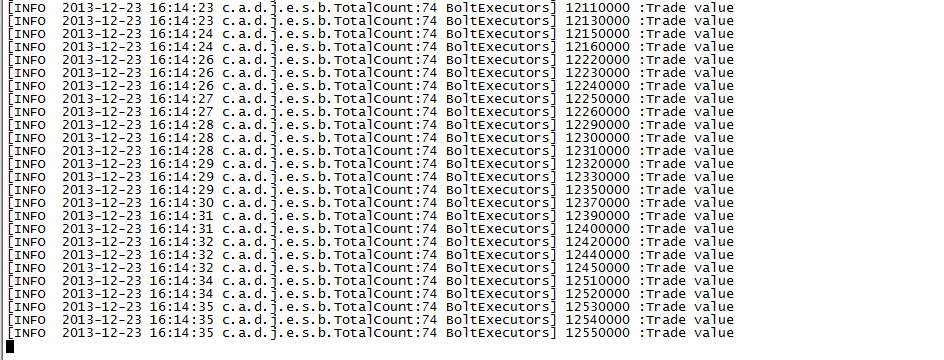
Worker’s log:



Kill supervisor : supervisor is dead



Workers status:



#### When topology is in inactive, kill supervisor

**Test purpose:**

The topology still be inactive, after restart supervisor

**Test result:**

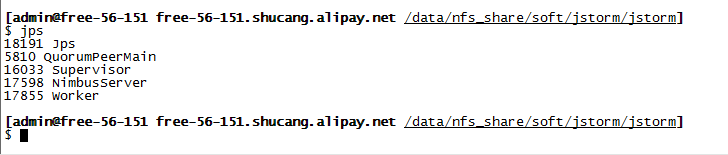
Pass, the topology still be inactive, all worker are alive.

**Test process:**

1. Submit one topology
2. Wait several minutes, deactivate the topology
3. Wait one minute, kill the supervisor
4. Check whether supervisor is alive
5. Check Whether workers is alive
6. Check Whether workers work correctly, Check whether spout send tuples
7. Start supervisor

**Test snapshots:**

Before deactivate topology:



Worker’s log

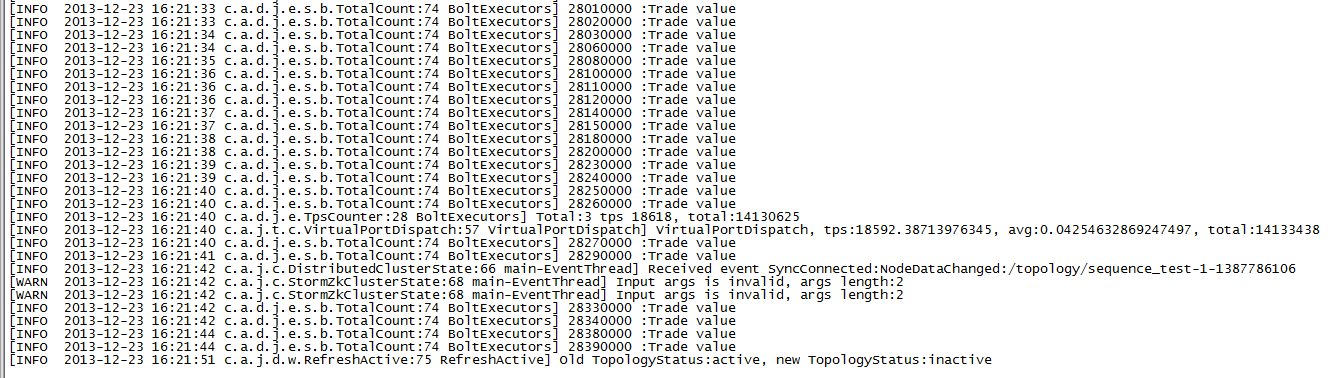


Deactivate topology:

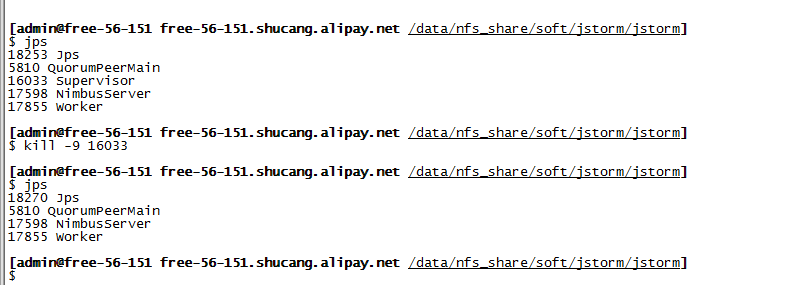




Worker stop working:

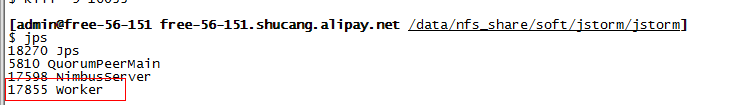


Kill supervisor:



After kill:

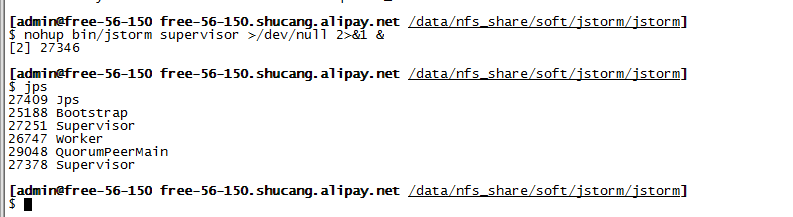
Worker process is still alive



Worker still in inactive:



Start supervisor:



Worker’s status is inactive:



#### When topology is in rebalancing, kill supervisor

**Test purpose:**

The supervisor continue doing rebalancing and the alive worker don’t change, after restarting

**Test result:**

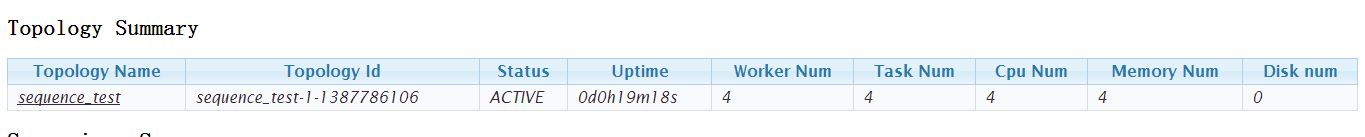
Pass, the supervisor continue doing rebalancing and the alive worker change nothing.

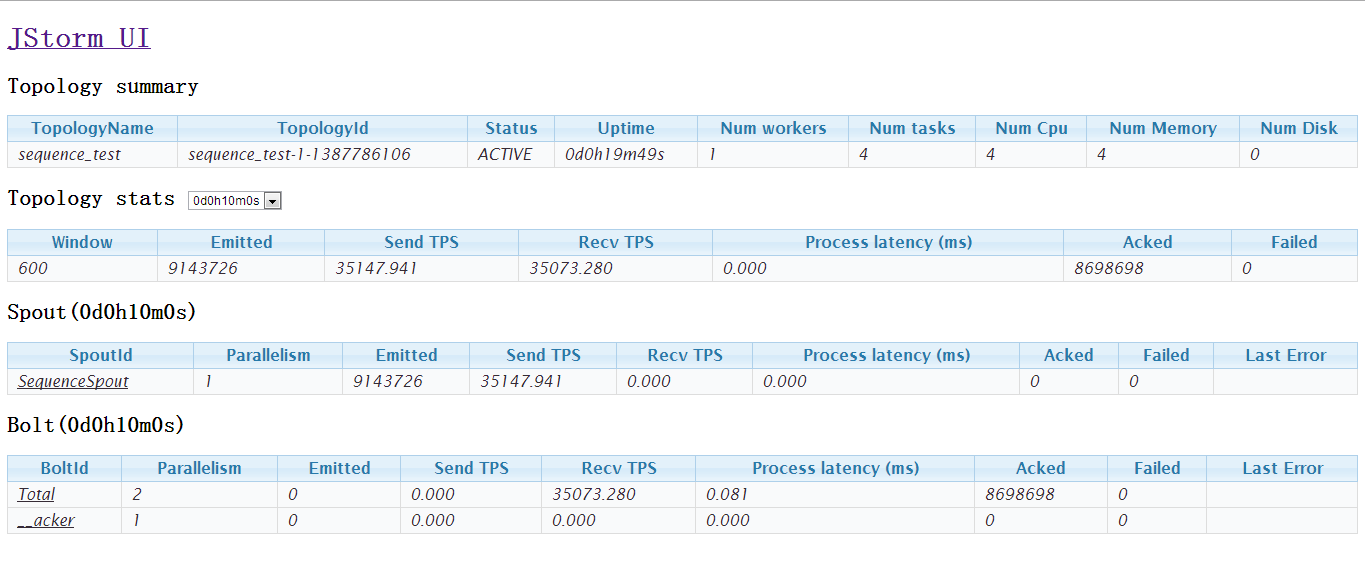
**Test process:**

1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Add supervisor to JStorm
4. Wait one minute, kill the supervisor
5. Check whether supervisor is alive
6. Check Whether workers is alive
7. Check Whether workers work correctly, Check whether spout send tuples
8. Start supervisor

**Test snapshots:**

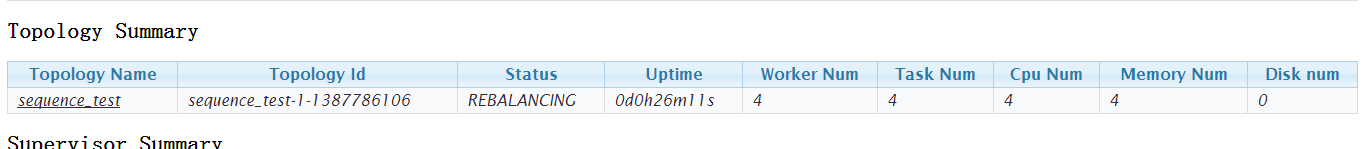
Before add supervisor

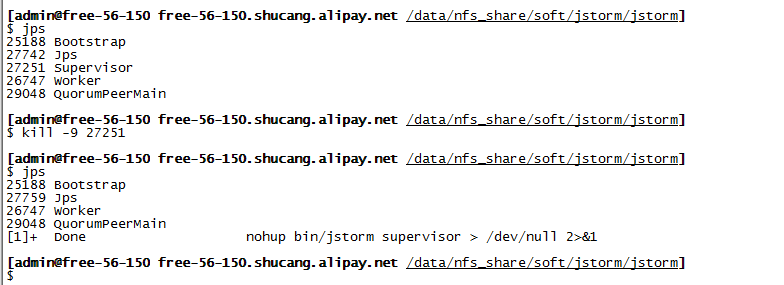




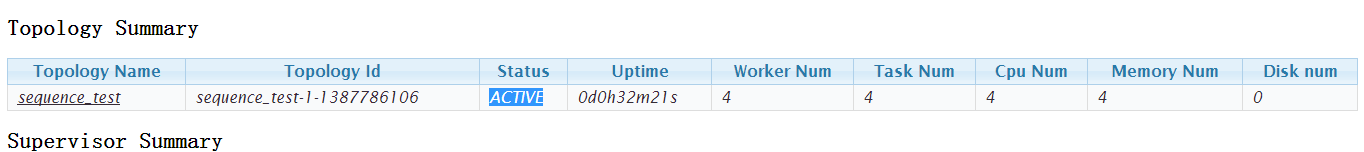
Submit rebalance command and kill supervisor:







After start supervisor:



#### When topology is in killing, kill supervisor

**Test purpose:**

The supervisor continue killing topology, after restarting

**Test result:**

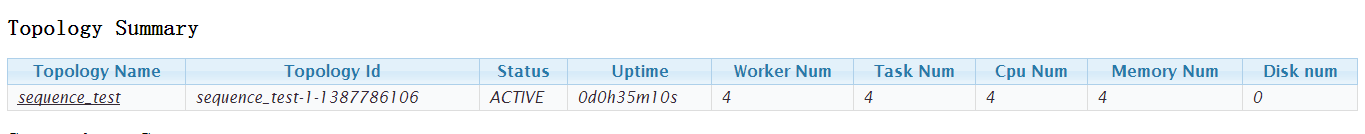
Pass, the supervisor continue killing topology

**Test process:**

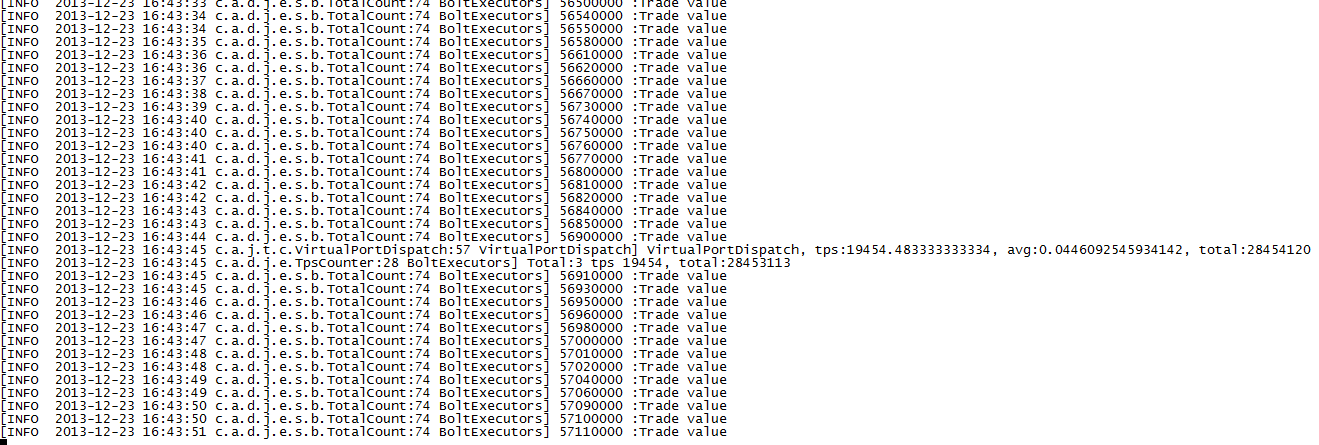
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Wait one minute, kill the supervisor
4. Check whether supervisor is alive
5. Check Whether workers is alive
6. Check Whether workers work correctly, Check whether spout send tuples
7. Start supervisor
8. Check whether workers have been killed

**Test snapshot:**

Before kill topology

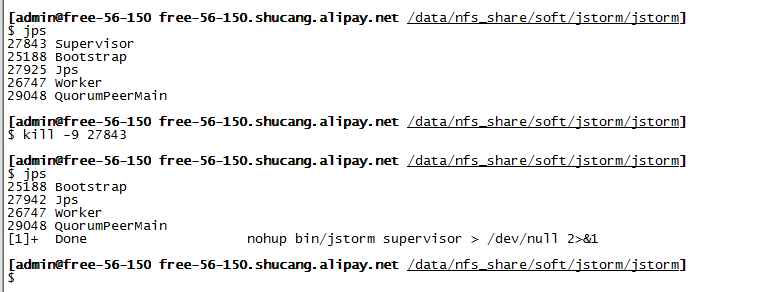


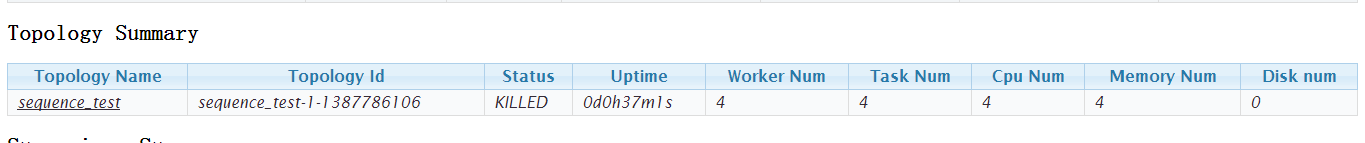
Worker’s log



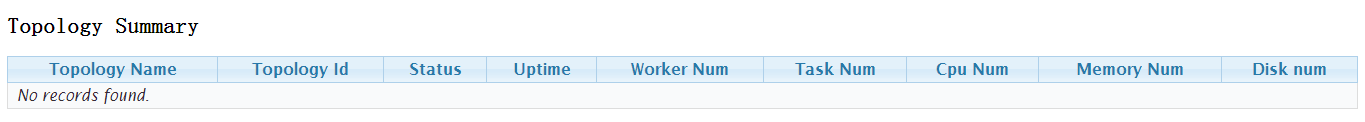
Kill topology:



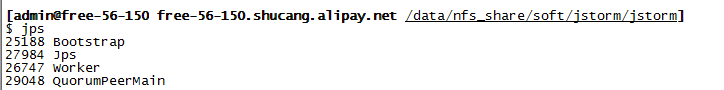




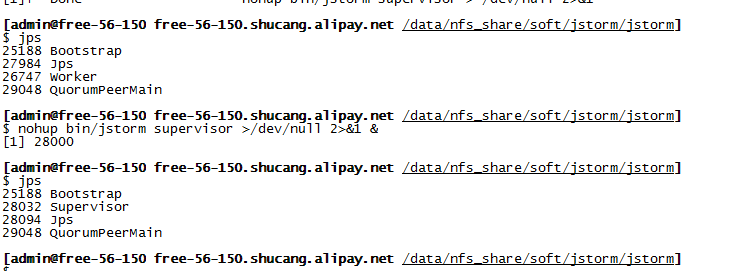
After 60 seconds, UI doesn’t show the topology



But due to supervisor is down, the workers is still alive:



After start supervisor, workers has been killed



### Kill worker

#### When topology is in active, kill worker

**Test purpose:**

Another worker will replace the killed worker, and it is alive.

**Test result:**

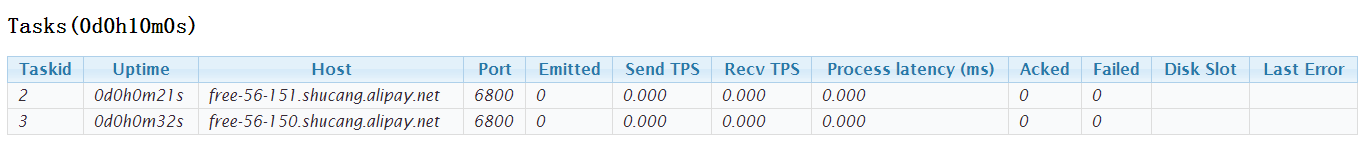
Pass, one same worker has replaced the killed worker.

**Test process:**

1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Kill one worker
4. One minute later, check whether another worker will replace the killed worker

**Test snapshot:**

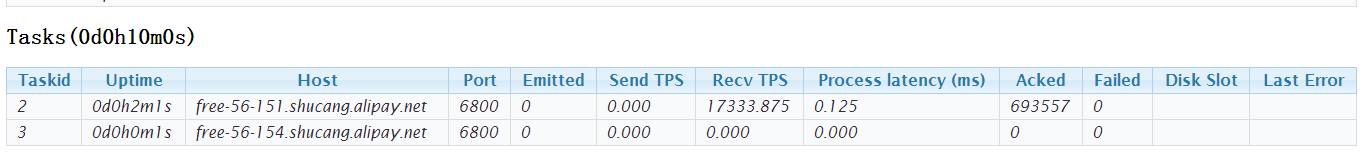
Before kill worker:



Kill worker:



After kill worker:



#### When topology is in inactive, kill worker

**Test purpose:**

Check whether a new worker replaces the killed worker and it’s status is inactive

**Test result:**

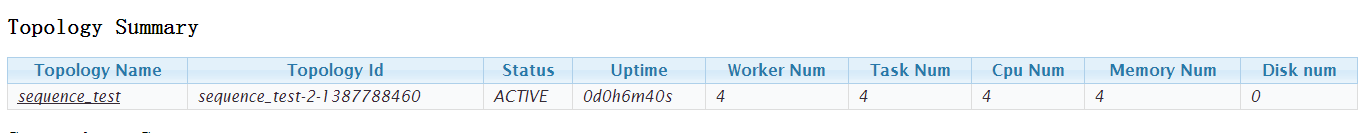
Pass, A new worker replace the killed worker and it’s status is inactive

**Test process:**

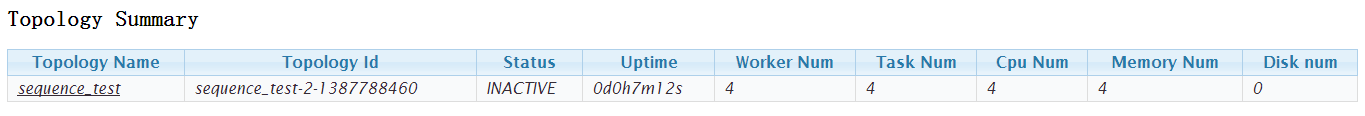
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Submit inactive the topology command
4. Kill one worker
5. One minute later, check whether another worker will replace the killed worker
6. Check the new worker’s status is inactive or not?

**Test snapshot:**

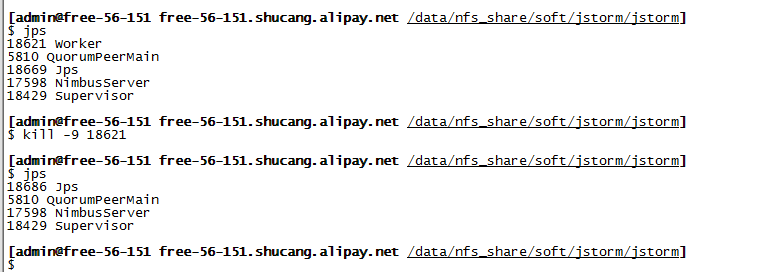
Before submit deactivate command:



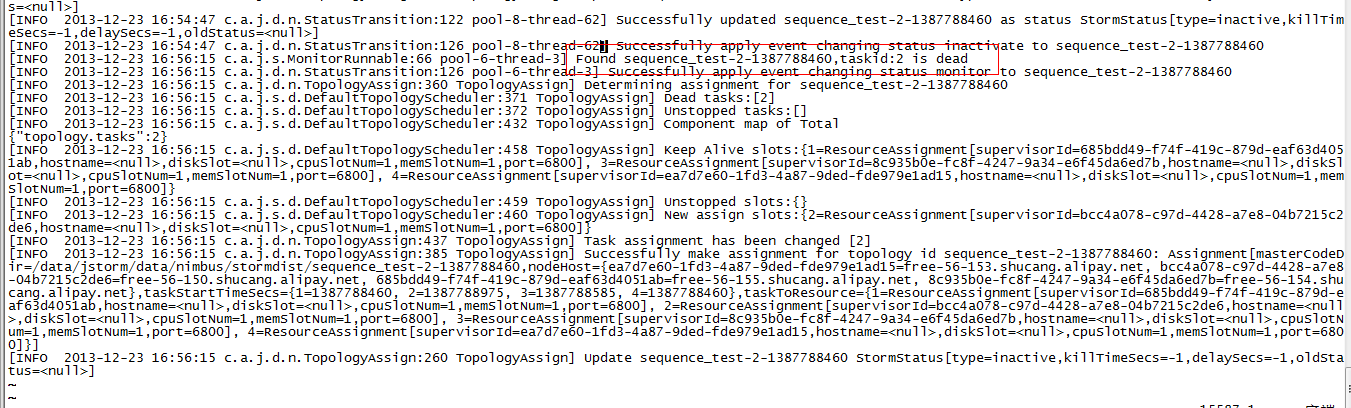
Submit deactivate command:



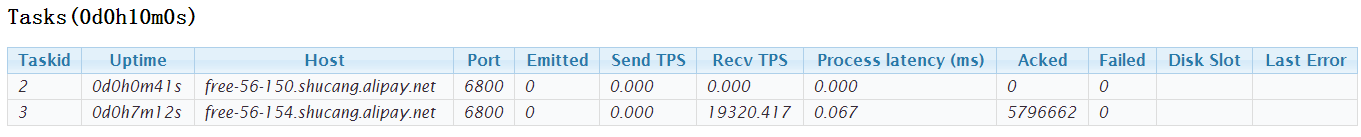
Kill bolt worker:



Nimbus found the task is dead:



Start a new worker, and the worker is in inactive



#### When topology is in rebalancing, kill worker

**Test purpose:**

Check whether a new worker replace the killed worker

**Test result:**

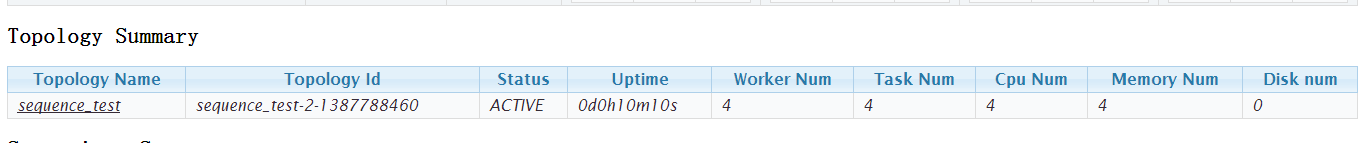
Pass, A new worker replace the killed worker and it’s status is active

**Test process:**

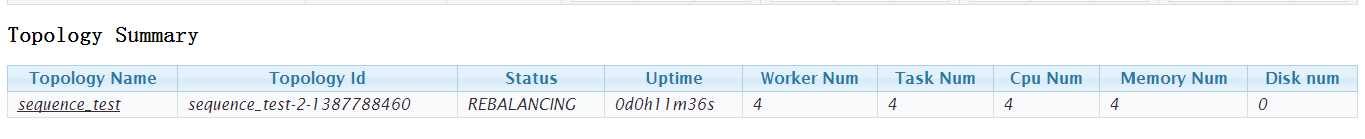
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Submit rebalance the topology command
4. Kill one worker
5. One minute later, check whether another worker will replace the killed worker
6. Check the new worker’s status is inactive or not?

**Test snapshot:**

Before submit rebalance command:



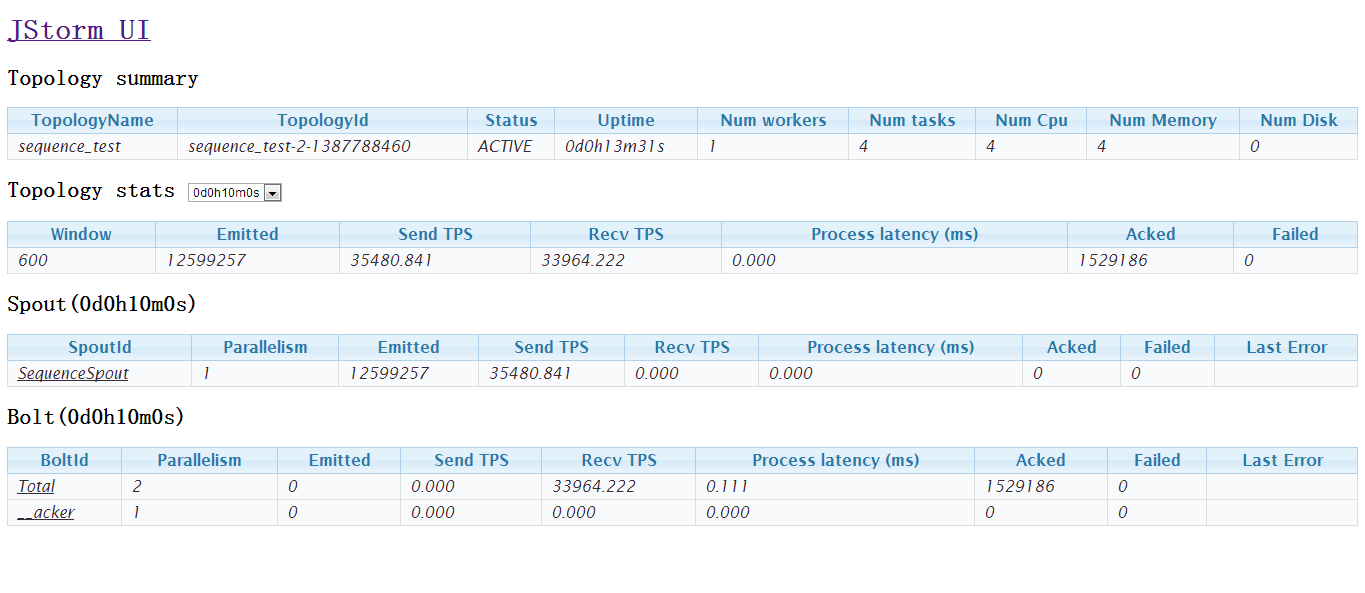
Submit rebalance command and kill the worker



Kill worker:



Wait one minutes:



#### When topology is in killing, kill worker

**Test purpose:**

Check whether the topology has been killed

**Test result:**

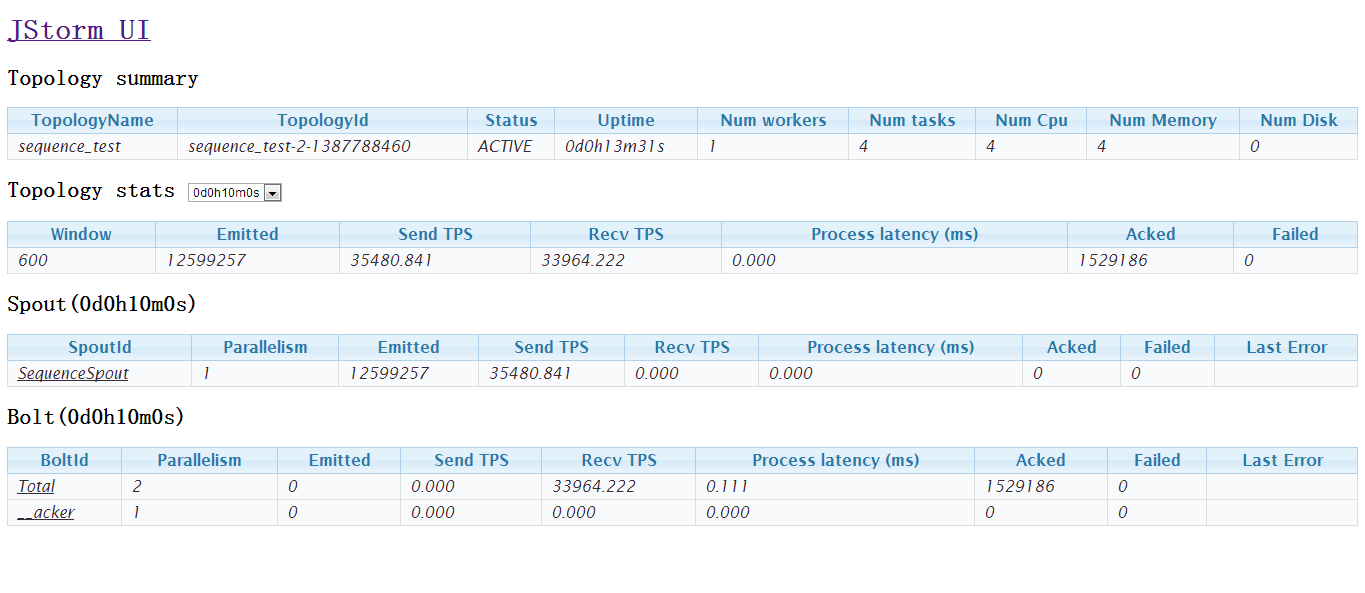
Pass, the topology has been killed

**Test process:**

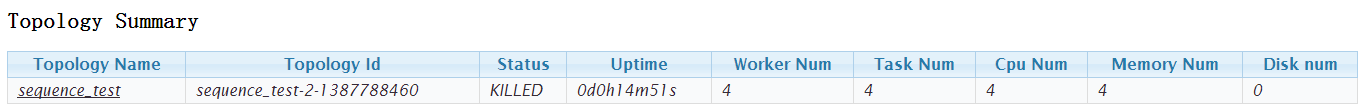
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Submit kill the topology command
4. Kill one worker
5. One minute later, check whether another worker will replace the killed worker
6. Check the topology has been killed or not?

**Test snapshot:**

Before submit kill command:

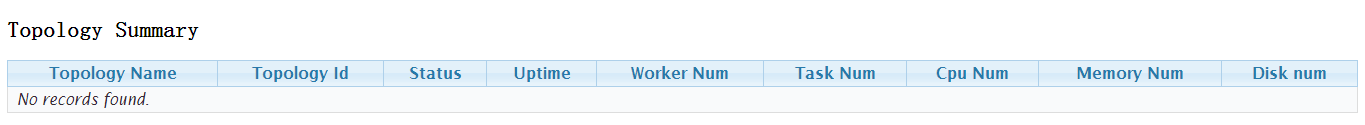


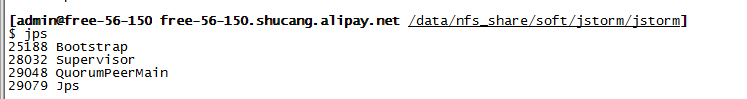
Submit kill command and kill the worker:





Wait 2 minute:





### Shutdown non-nimbus node

#### When topology is in active, shutdown the node

**Test purpose:**

Check whether new workers has replaced workers in the shutdown node

**Test result:**

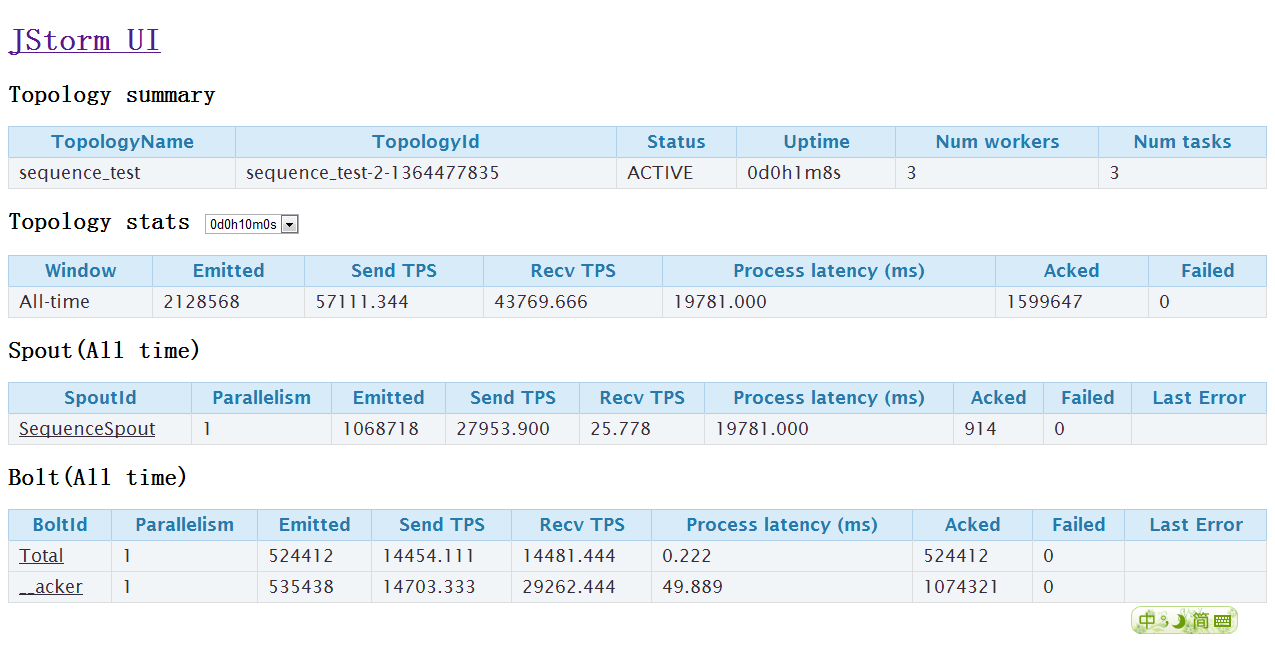
Pass, new workers has replaced workers in the shutdown’s node

**Test process:**

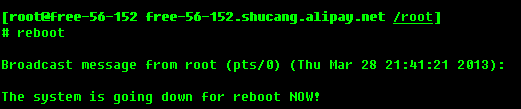
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Shutdown the non-nimbus node
4. Wait 1 minute, check whether new workers has replace workers in the shutdown node

**Test snapshot:**

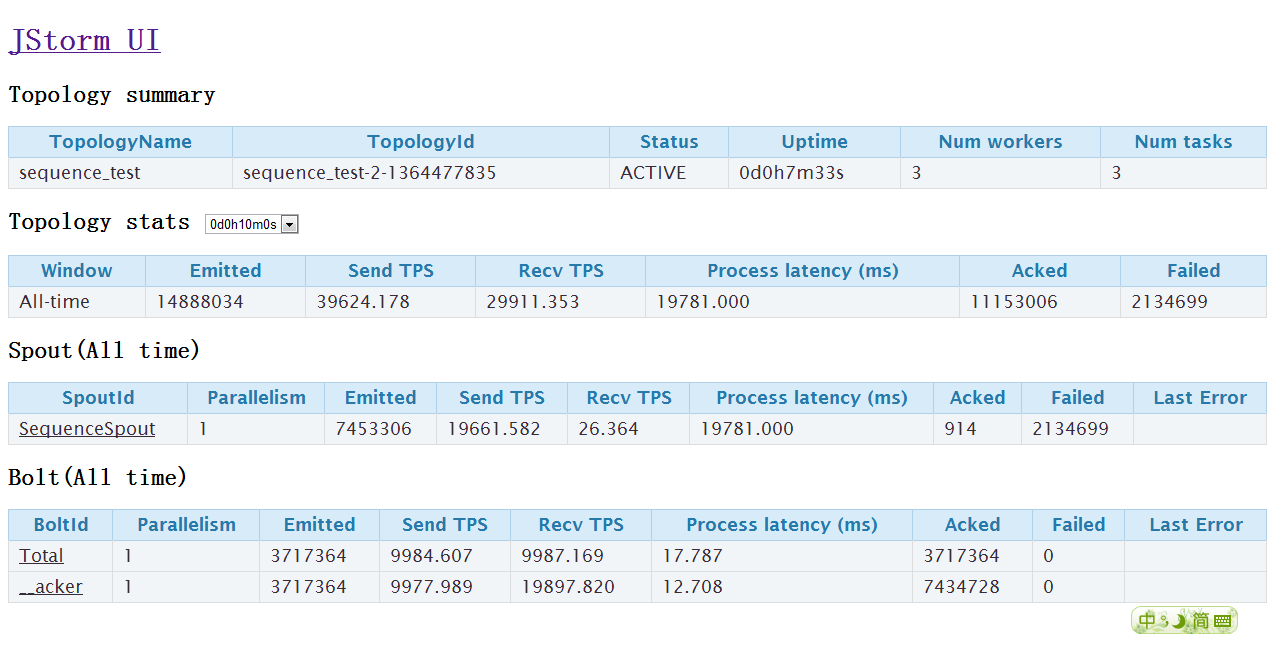
Before shutdown:

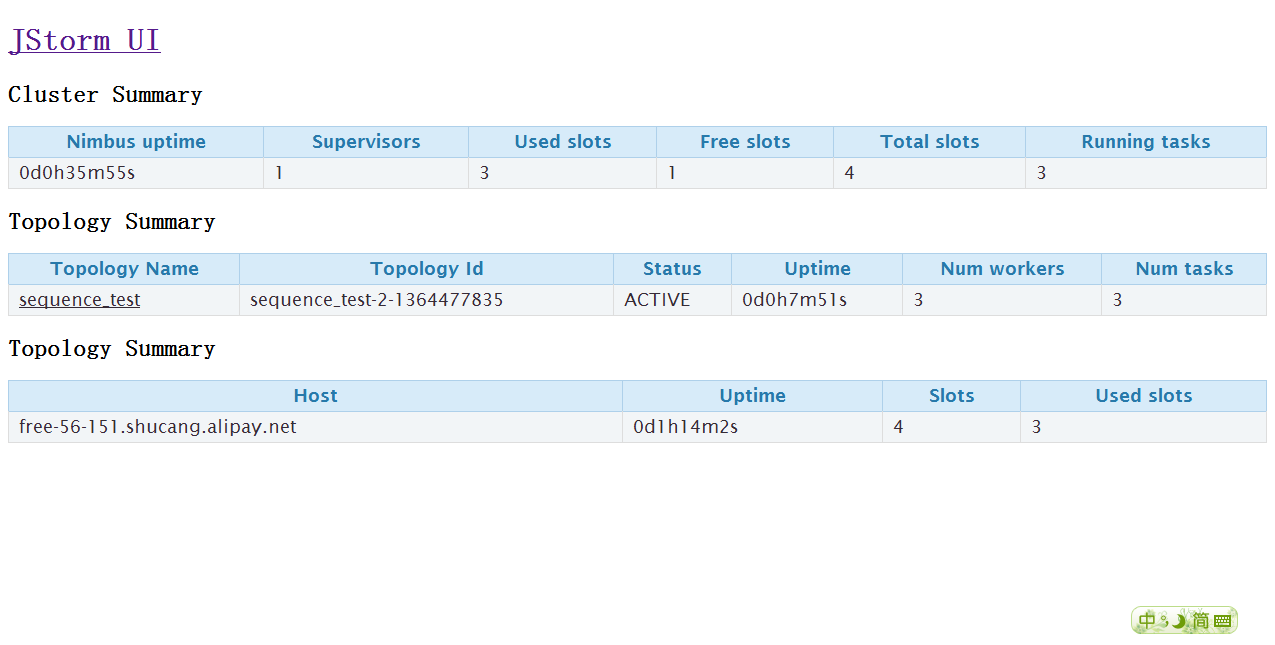


Shutdown non-nimbus node:



After reboot





#### When topology is in inactive, shutdown the node

**Test purpose:**

Check whether new workers has replaced workers in the shutdown node and the worker’s status is inactive

**Test result:**

Pass, new workers has replaced workers in the shutdown’s node

**Test process:**

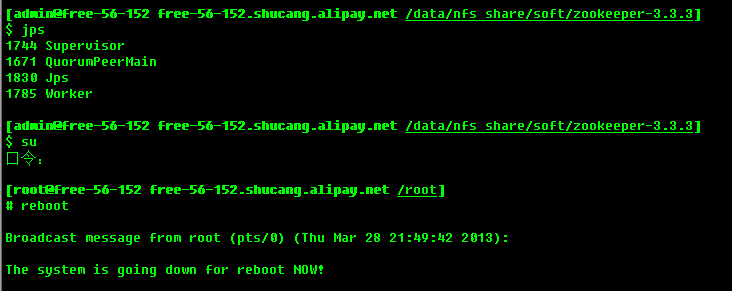
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Commit deactivate command
4. Shutdown the non-nimbus node
5. Wait 1 minute, check whether nimbus kill the topology
6. Start the supervisor
7. Check whether the supervisor start the task of the topology

**Test snapshot:**

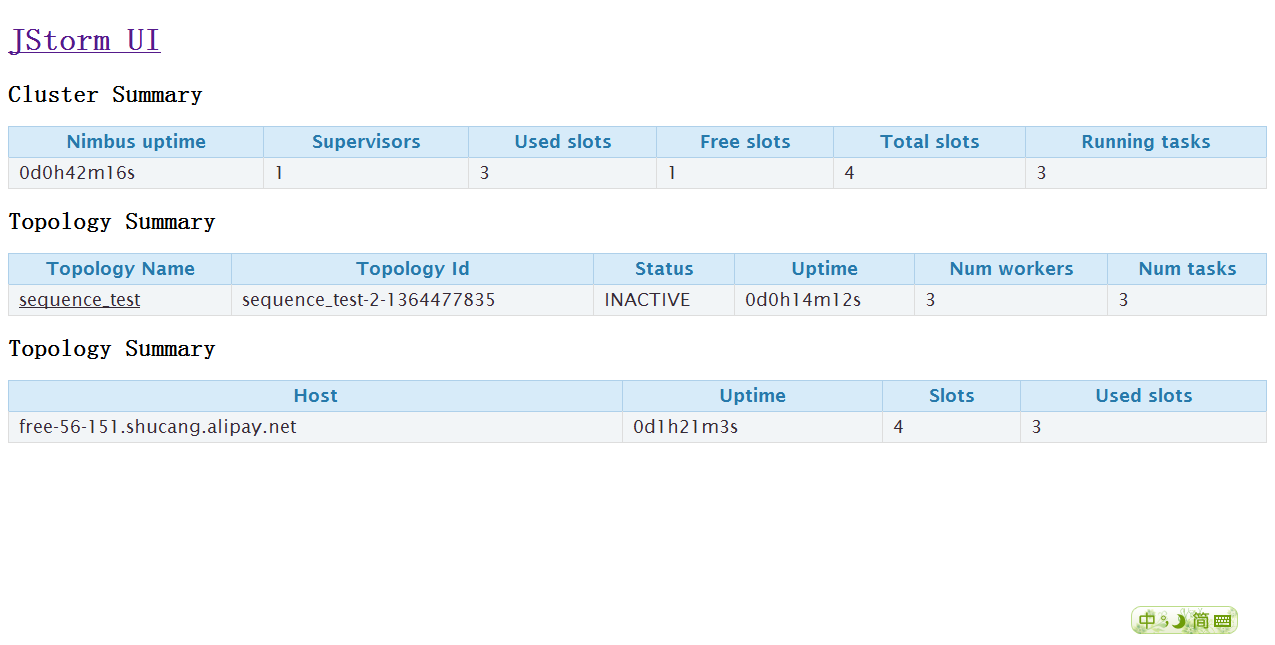
Before shutdown:



Do reboot:



After reboot:



#### When topology is in rebalancing, shutdown the node

**Test purpose:**

Check whether nimbus continue doing reblance

**Test result:**

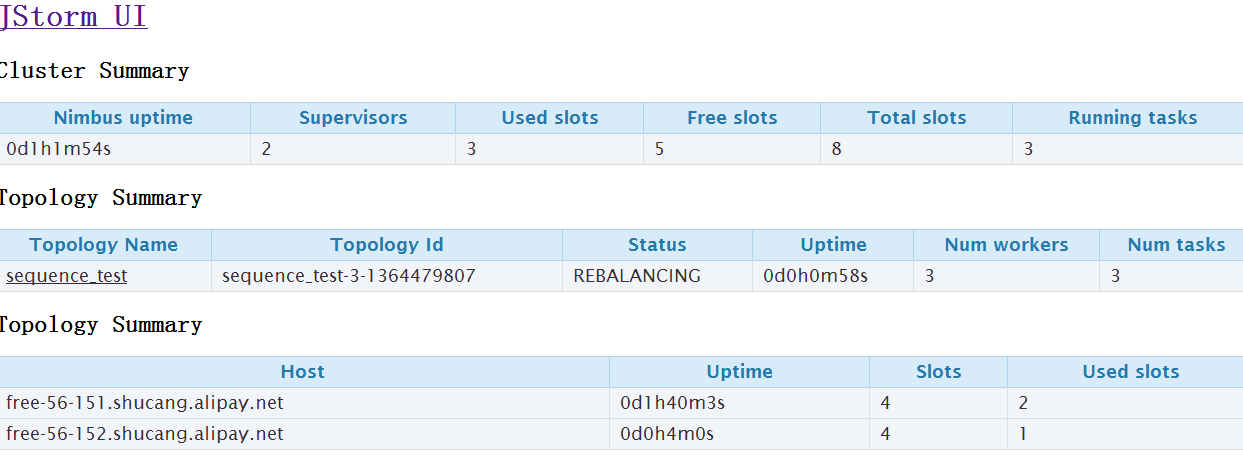
Pass, nimbus continue doing reblance

**Test process:**

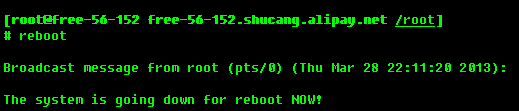
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Commit kill command
4. Shutdown the non-nimbus node
5. Wait 1 minute, check whether new workers has replace workers in the shutdown node

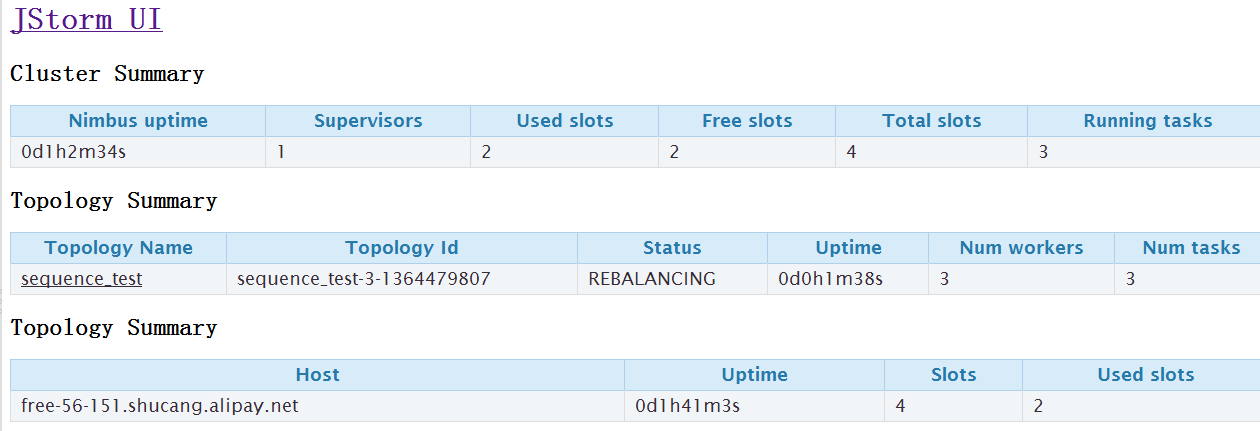
**Test snapshot:**

Before shutdown:



Do reboot





After one minute:



#### When topology is in killing, shutdown the node

**Test purpose:**

Check whether nimbus continue doing kill topology and supervisor will shut down all tasks after restart

**Test result:**

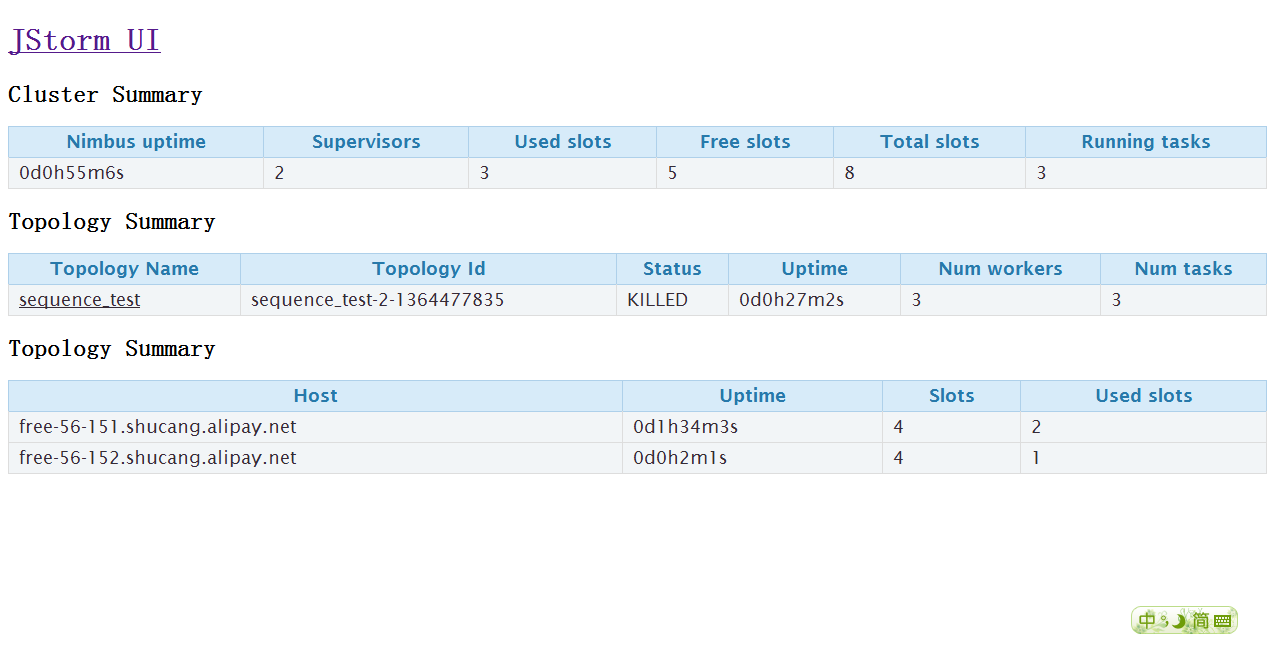
Pass, nimbus continue doing kill topology and supervisor will shut down all tasks after restart

**Test process:**

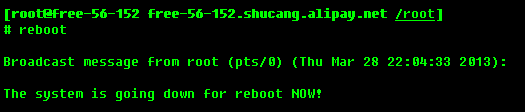
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Commit reblance command
4. Shutdown the non-nimbus node
5. Wait 1 minute, check whether new workers has replace workers in the shutdown node

**Test snapshot:**

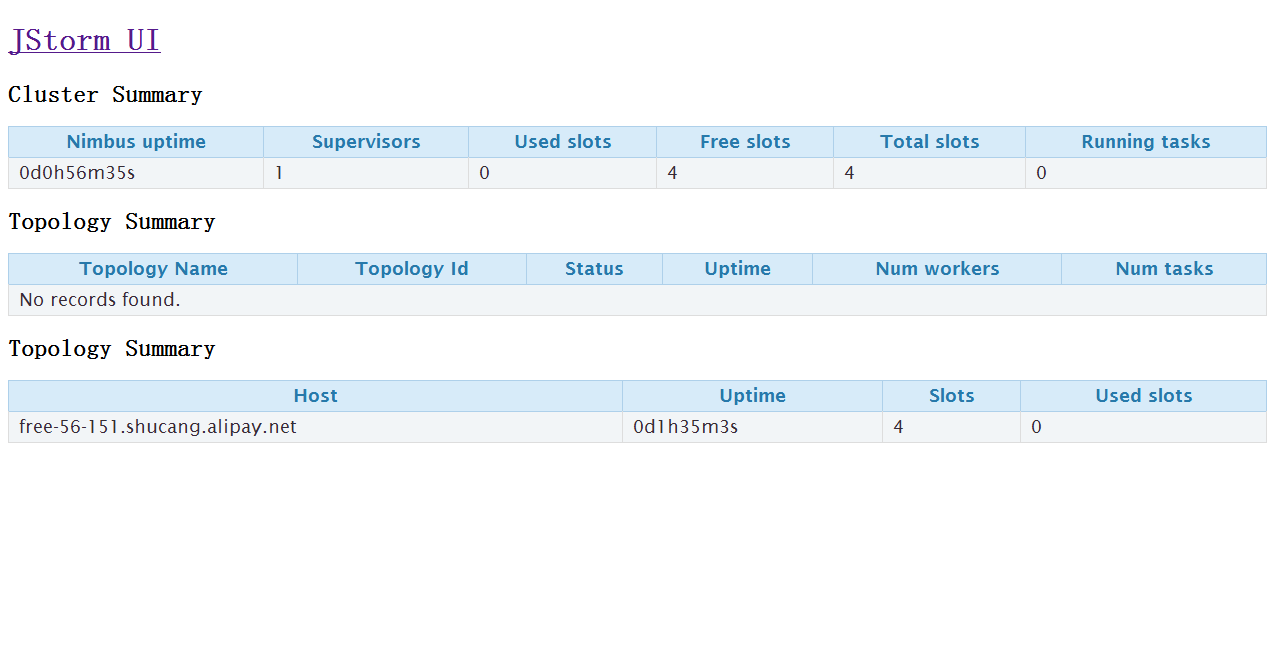
Before shutdown:



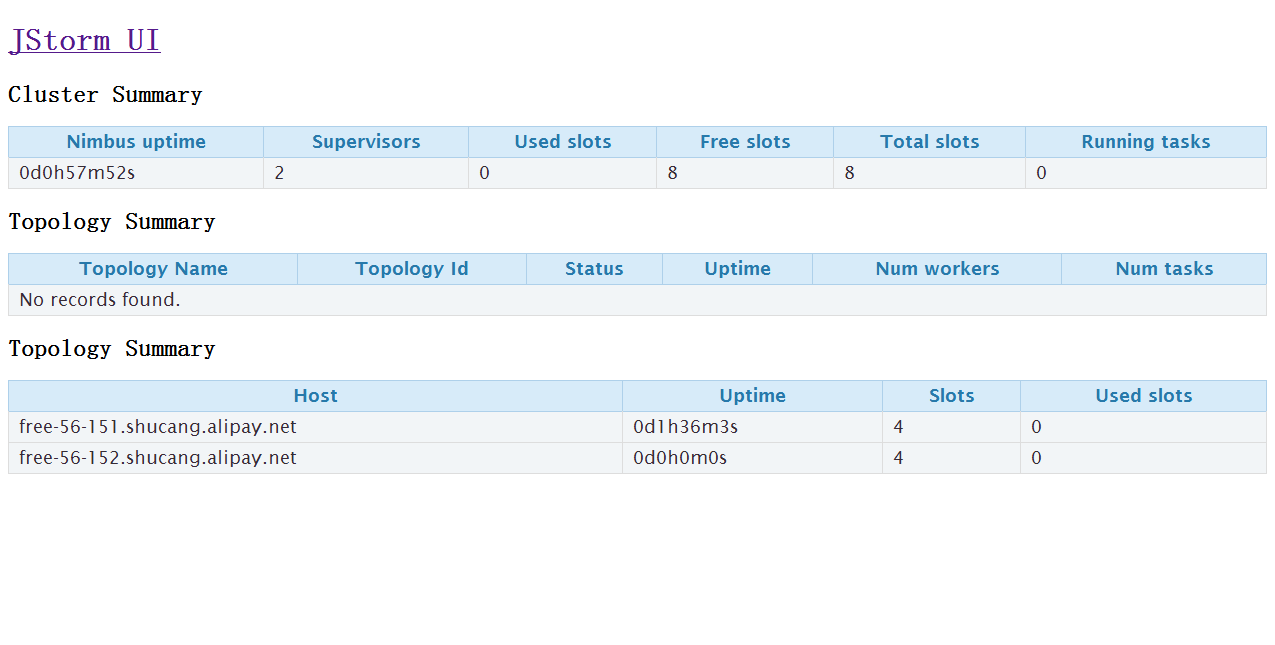
Do reboot



After one minute:



After start the shutdown node’s supervisor



### Shutdown nimbus node

#### When topology is in active, shutdown the node

**Test purpose:**

Check whether topology continue after restart the nimbus node and topology status is active

**Test result:**

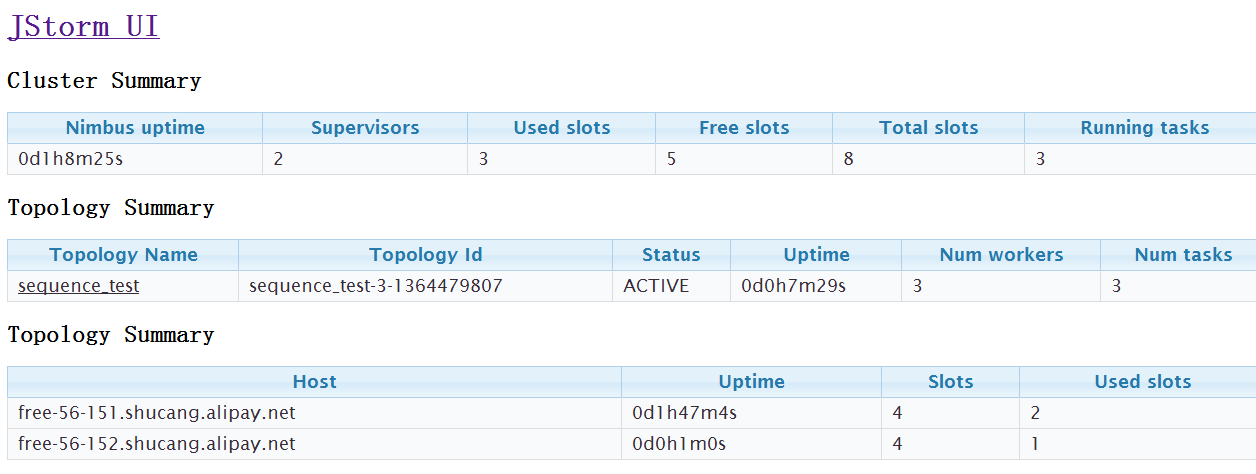
Pass, topology continue after restart the nimbus node and topology status is active

**Test process:**

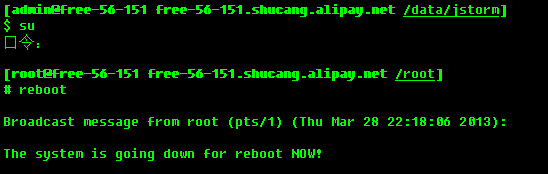
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Shutdown the nimbus node
4. Wait 1 minute, check whether the alive node’s worker status
5. Start the nimbus node, start nimbus and supervisor on the nimbus ‘s node

**Test snapshot:**

Before shutdown:

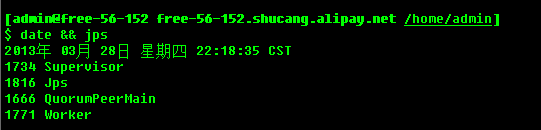


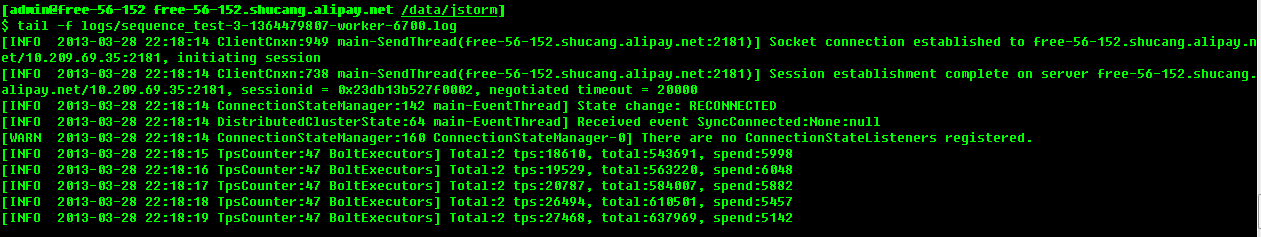
Do shutdown:



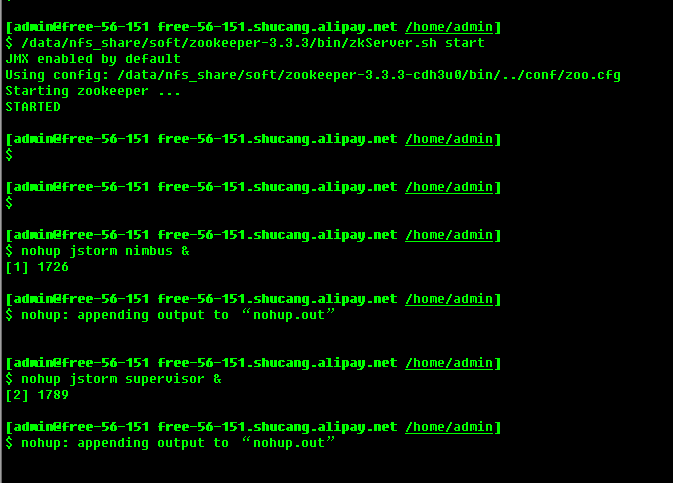
Web ui is unavailable:

Another node’s worker is alive:

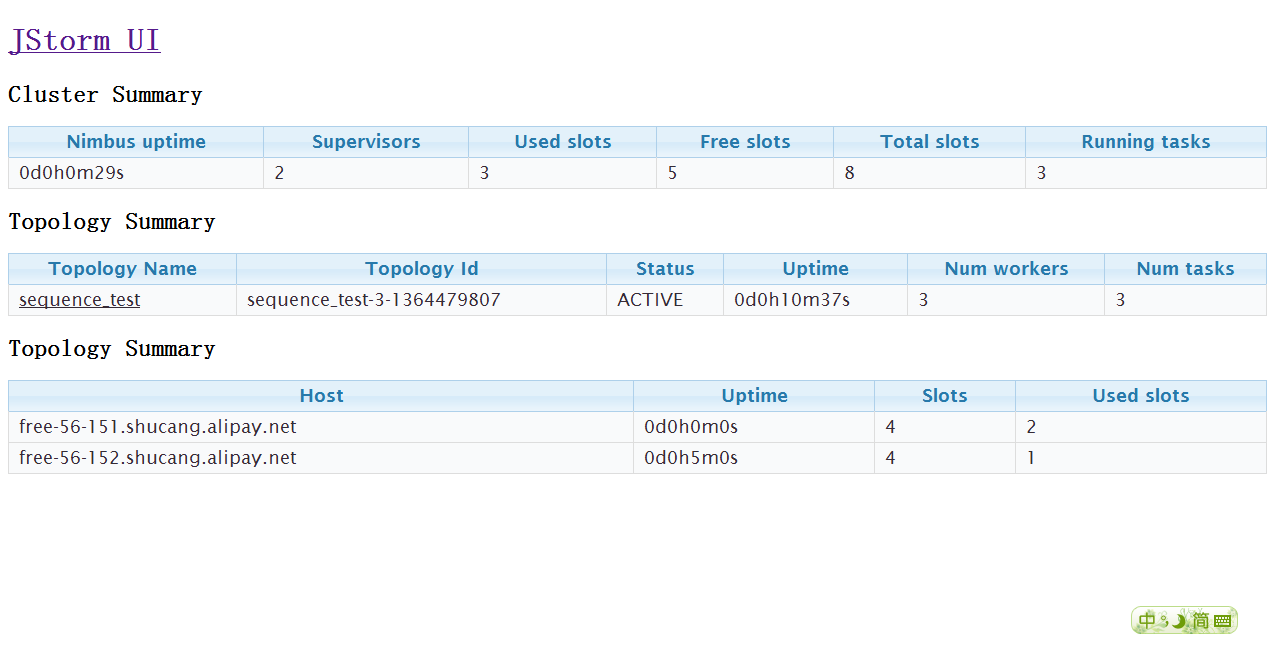




After start nimbus node, start nimbus and supervisor:



Check web ui:



#### When topology is in inactive, shutdown the node

**Test purpose:**

Check whether topology continue after restart the nimbus node and topology status is inactive

**Test result:**

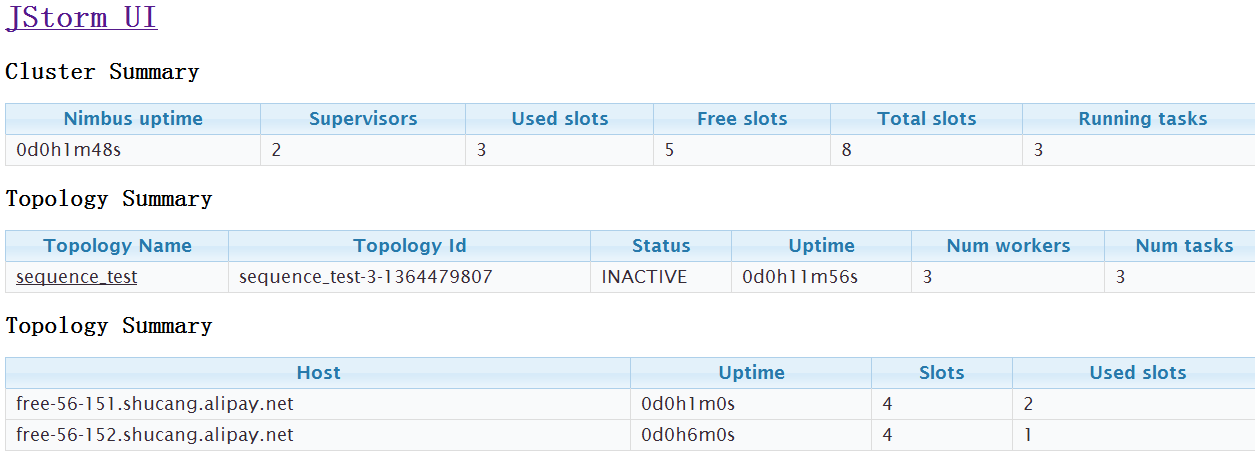
Pass, topology continue after restart the nimbus node and topology status is inactive

**Test process:**

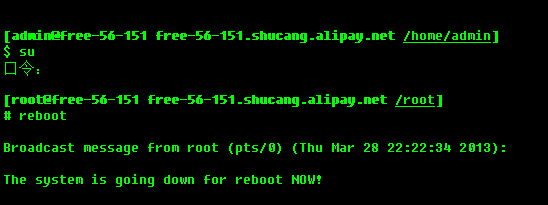
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Submit inactive command
4. Shutdown the nimbus node
5. Wait 1 minute, check whether the alive node’s worker status
6. Start the nimbus node, start nimbus and supervisor on the nimbus ‘s node

**Test snapshot:**

Before shutdown:

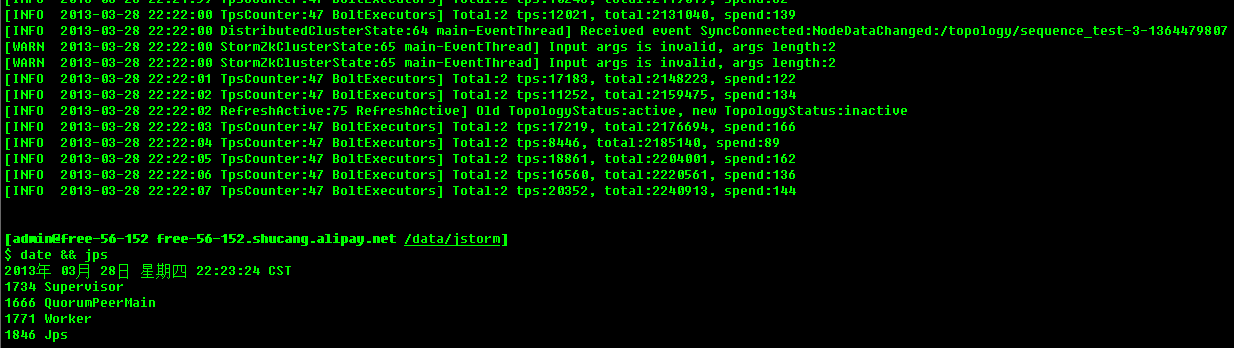


Doing shutdown:

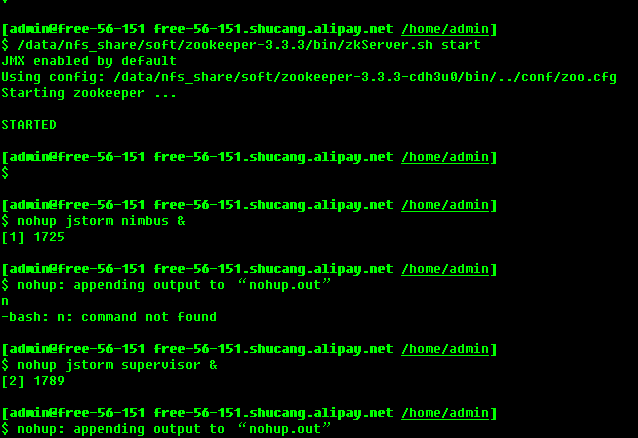


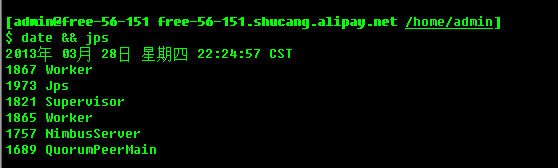
Web ui is unavailable

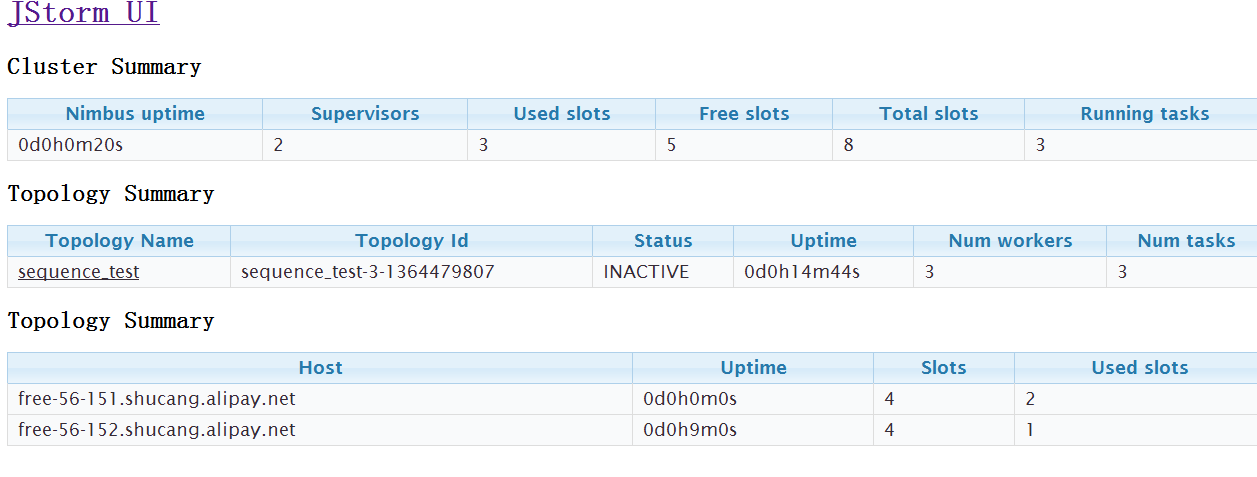
Check other node’s tasks’ status, worker is alive, and topology status is inactive



Start the nimbus node, start nimbus and supervisor in the nimbus node:







#### When topology is in rebalancing, shutdown the node

**Test purpose:**

Check whether nimbus continue doing rebalancing and the topology is active after restart the nimbus node

**Test result:**

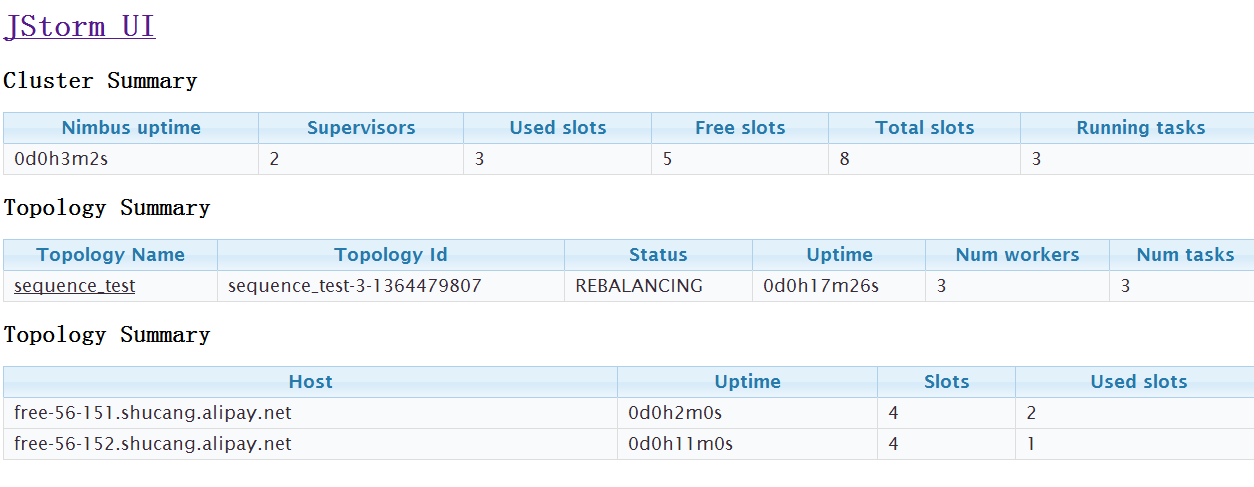
Pass, nimbus continue doing rebalancing and the topology is active after restart the nimbus node

**Test process:**

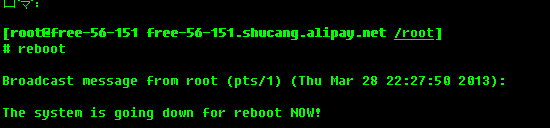
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Commit reblance command
4. Shutdown the nimbus node
5. Wait 1 minute, check whether other node’s worker is alive
6. Start the nimbus node, start nimbus and supervisor in the nimbus node
7. Check the topology status

**Test snapshot:**

Before shutdown:



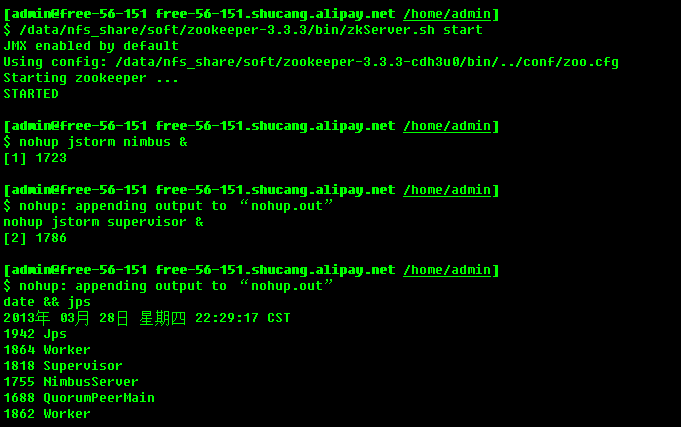
Doing reboot:

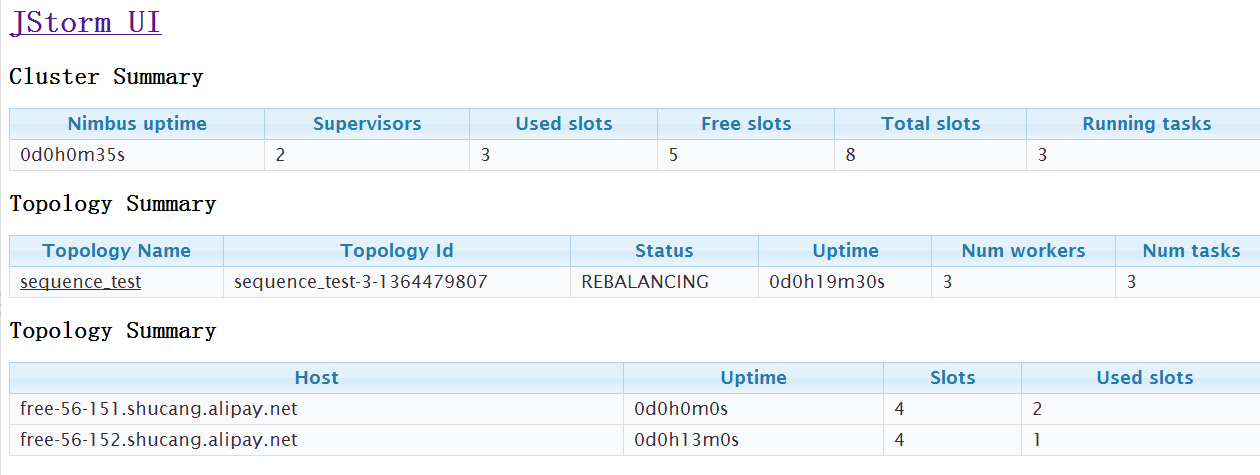


Check other node’s workers and log

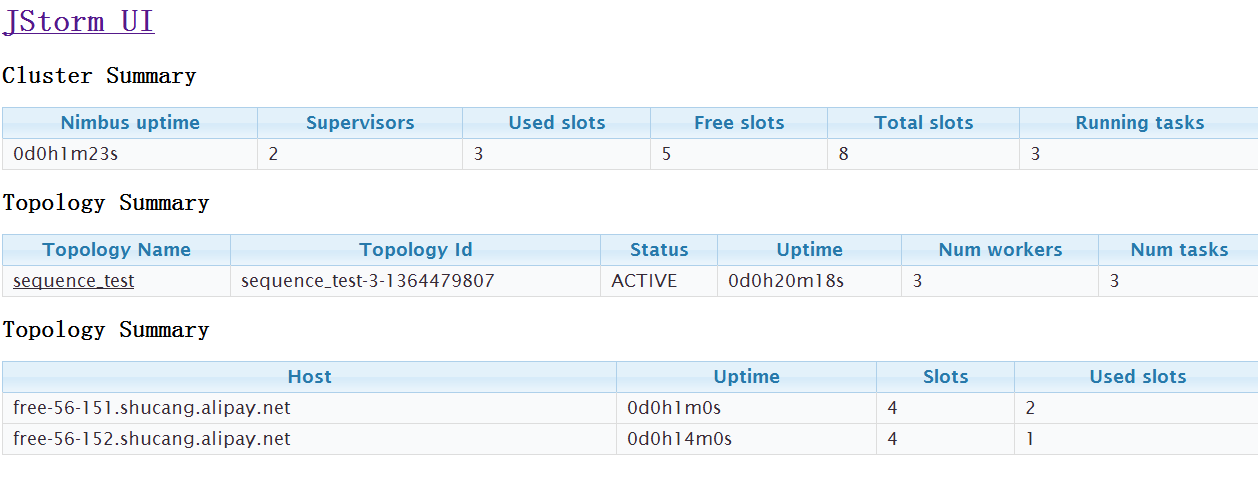


After nimbus node reboot, start nimbus and supervisor :





Wait one minute, nimbus finish rebalance operation



#### When topology is in killing, shutdown the node

**Test purpose:**

Check whether nimbus continue doing kill topology

**Test result:**

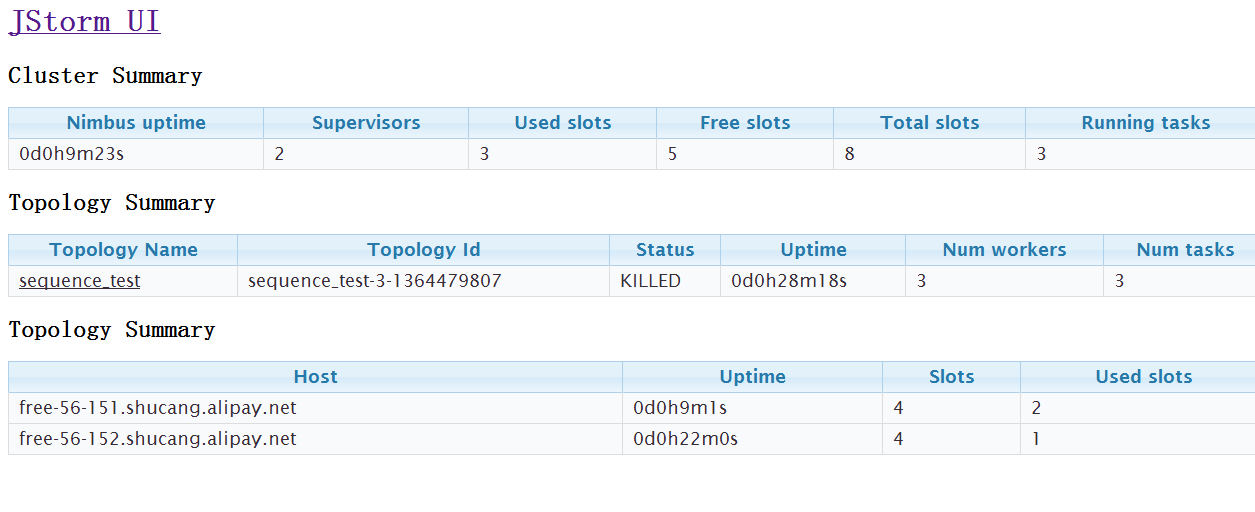
Pass, nimbus continue doing kill topology

**Test process:**

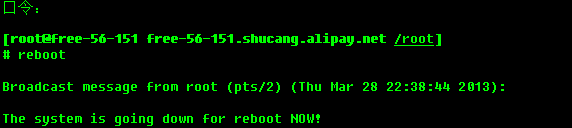
1. Submit one topology
2. Wait several minutes, let the topology runs well
3. Commit kill command
4. Shutdown the non-nimbus node
5. Wait 1 minute, check whether other node’s worker is alive
6. After nimbus restart, start nimbus and supervisor
7. Check the topology status
8. Wait 1 minute, the topology should be killed

**Test snapshot:**

Before shutdown:

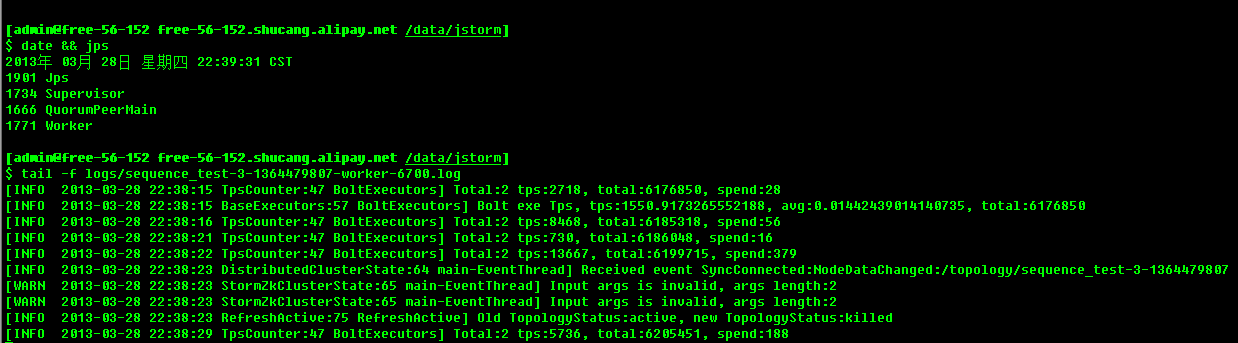


Doing reboot:

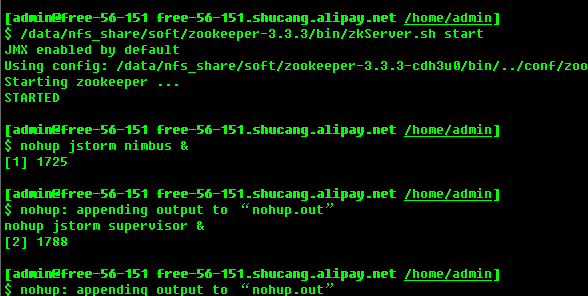
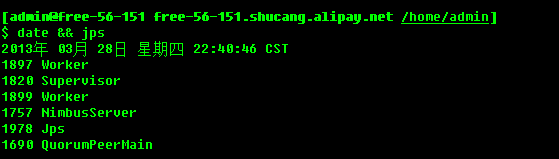


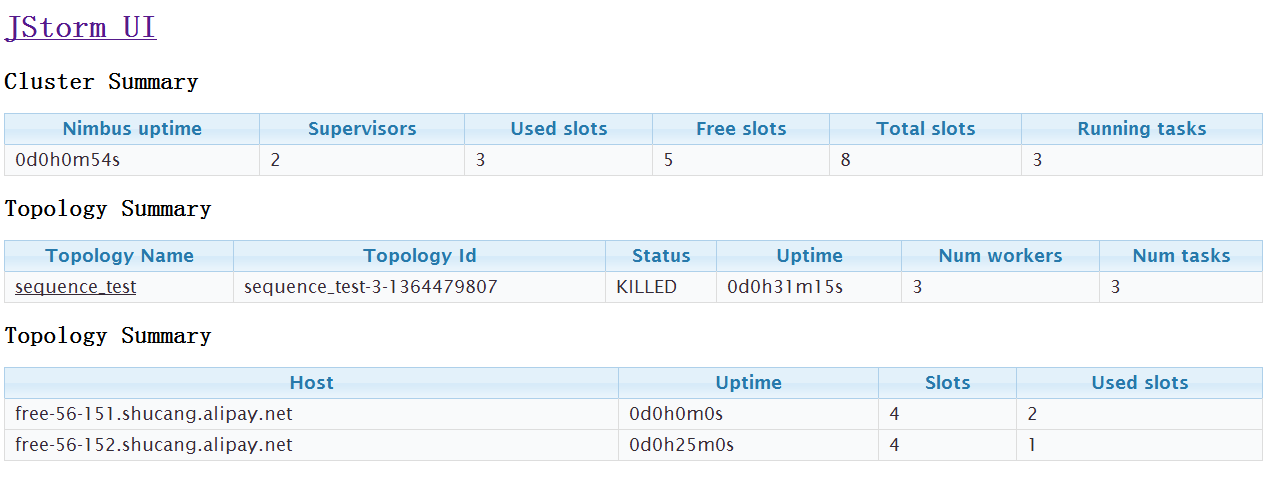
Web ui is unavailable

Check other node’s worker status, they are alive right now

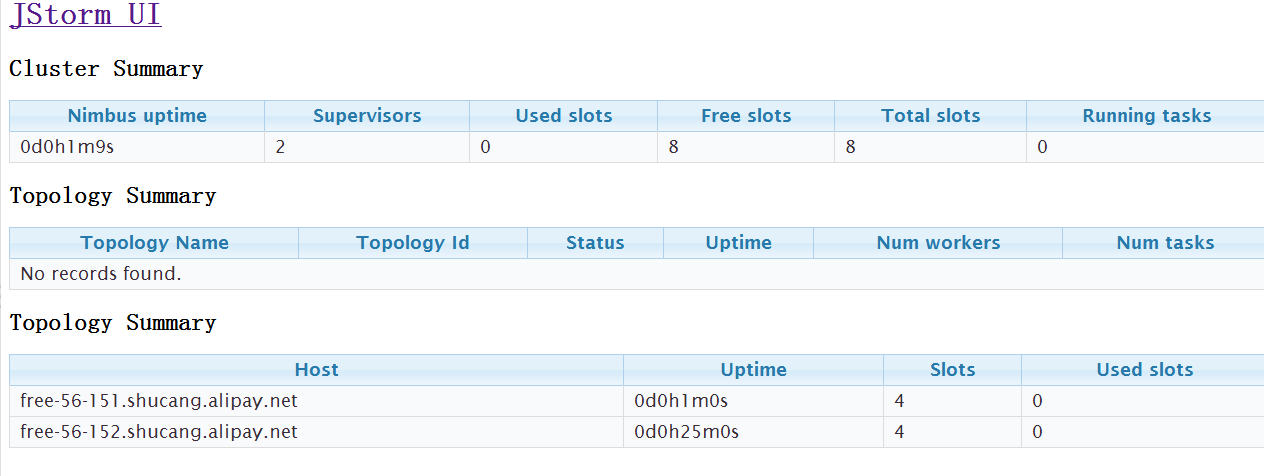


After nimbus node restart, start nmbus and supervisor



Wait one minutes:



## Functionality test

### Basic tests

Skip these basic tests, due to if one of the following tests fails, JStorm can’t work, so they are definitely working correctly.

Skip these tests

1. Submit topology
2. Kill topology
3. Active topology
4. Deactivate topology
5. Rebalance topology
6. Support split operations, which one bolt or spout can send tuple to multiple bolts, due to spout would send one user tuple to next bolt, at the same time, it will also send one init ack tuple to Acker, so skip this test
7. Support merge operation, which one bolt can receive tuples from multiple bolts or spout, due to Acker would receive tuples from difference component.
8. Shuffle/field/direct/Random/ shuffle method, due to Acker has use these grouping method, if one of them fails, JStorm can’t work,

### Transaction topology

It will also check Global/All grouping method

### DRPC topology

### Special topology

#### Acker’s number is 0

**Test purpose:**

Check whether topology works well when acker’s number is 0

**Test result:**

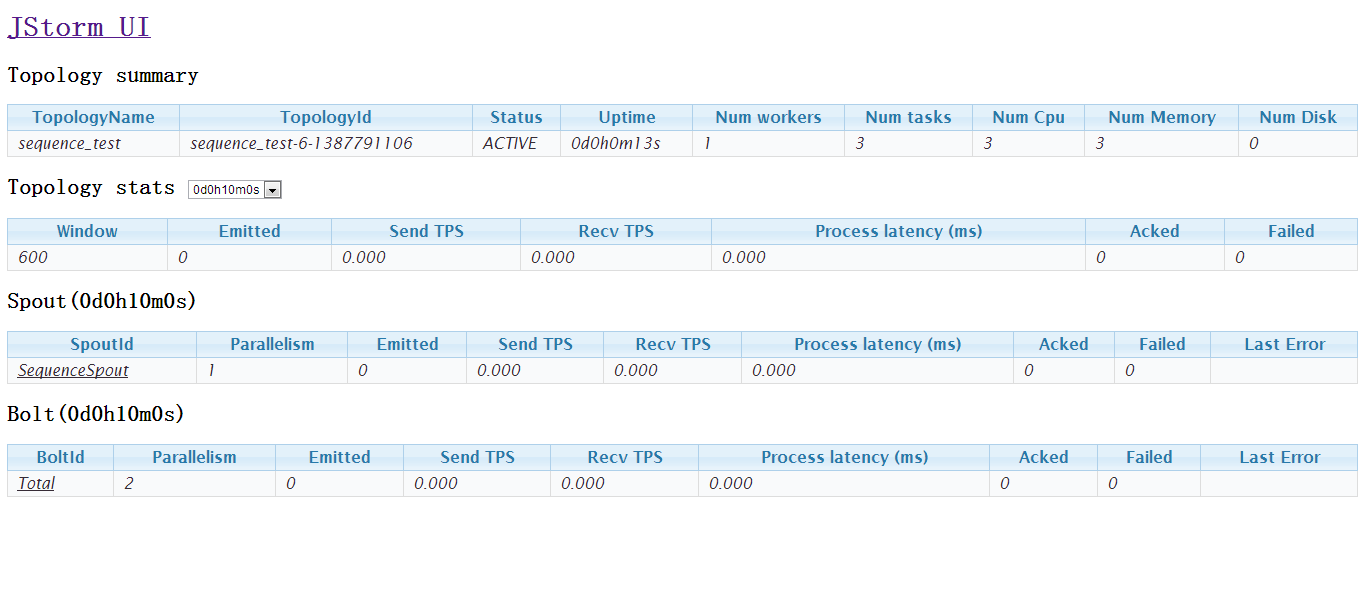
Pass, topology works well when acker’s number is 0

**Test process:**

1. Submit one topology which acker’s number is 0
2. Wait one minutes to let topology runs well
3. Check topology status

**Test snapshot:**

Topology status:



#### Worker’s number is less than available slots

**Test purpose:**

Check whether topology works well when worker’s number is less than available slots

**Test result:**

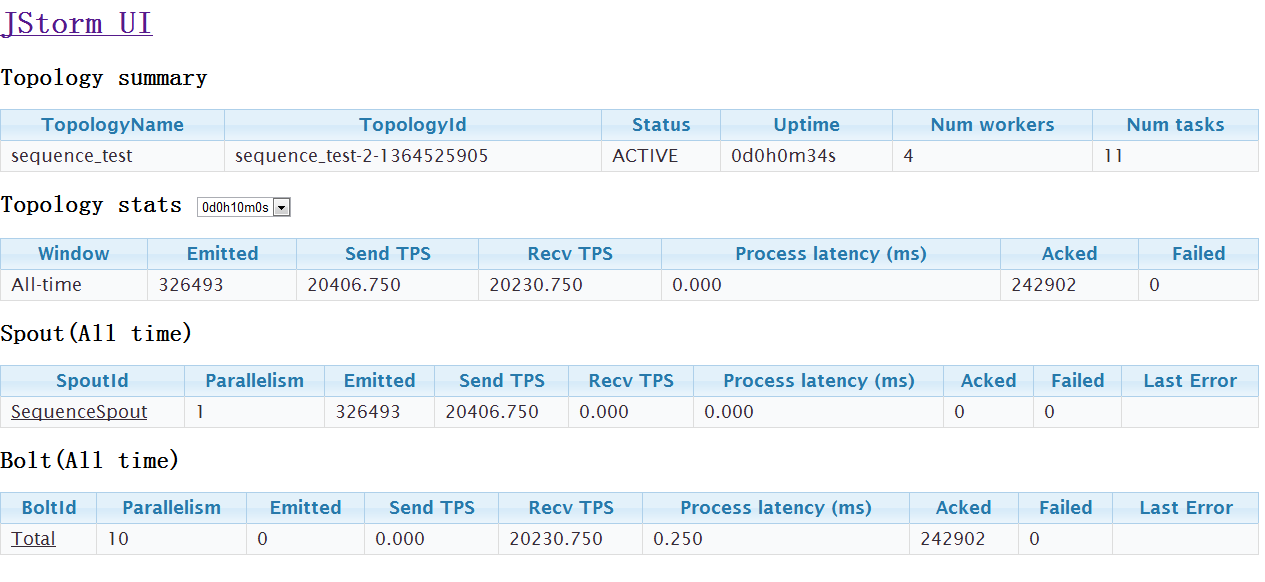
Pass, topology works well

**Test process:**

1. Submit one topology
2. Wait one minutes to let topology runs well
3. Check topology status

**Test snapshot:**

Topology status:



#### Worker’s number is less 0

**Test purpose:**

Check whether topology can submit or not?

**Test result:**

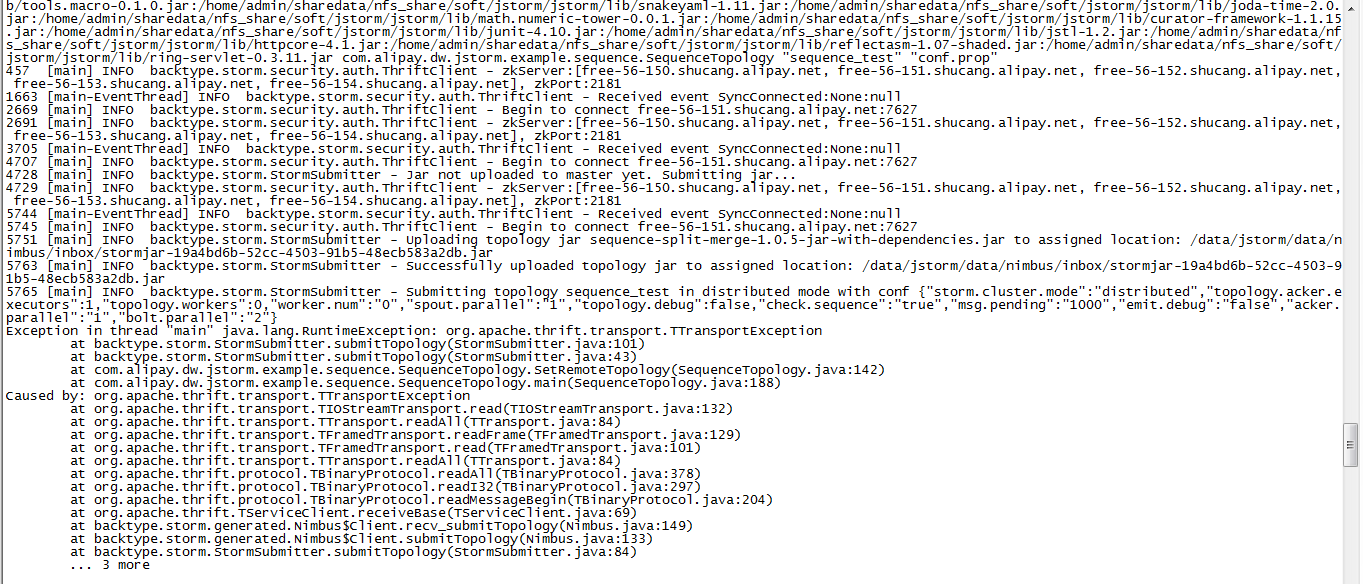
Pass, topology can’t submit

**Test process:**

1. Submit one topology which acker’s number is less 0

**Test snapshot:**

Topology can’t submit:



Nimbus log:



#### Bolt or spout occur exception in initialization

**Test purpose:**

Check whether topology can run?

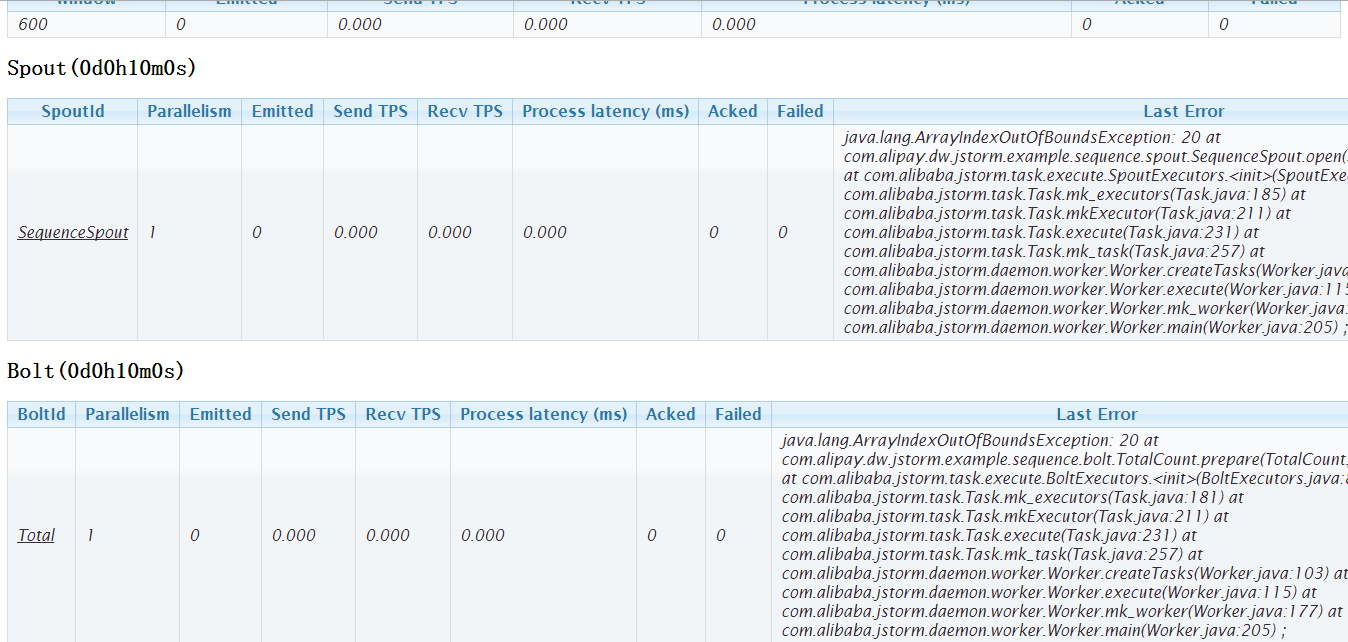
**Test result:**

Pass, topology can’t run

**Test process:**

1. Submit one topology which spout will throw exception in spout initialization

**Test snapshot:**



#### It take long time to do initialization for bolt or spout

**Test purpose:**

Check whether topology can run?

**Test result:**

Pass,

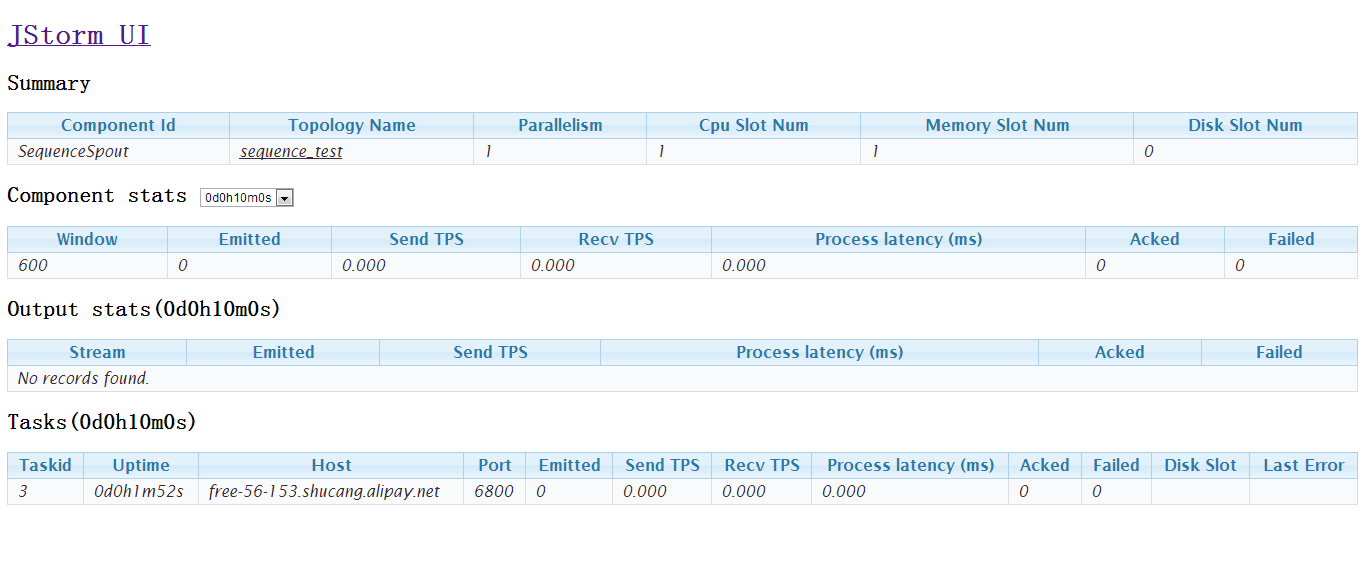
If it takes more than **“supervisor.worker.start.timeout.secs”**(default is 120) seconds, the worker can’t start

If not, the worker can be started

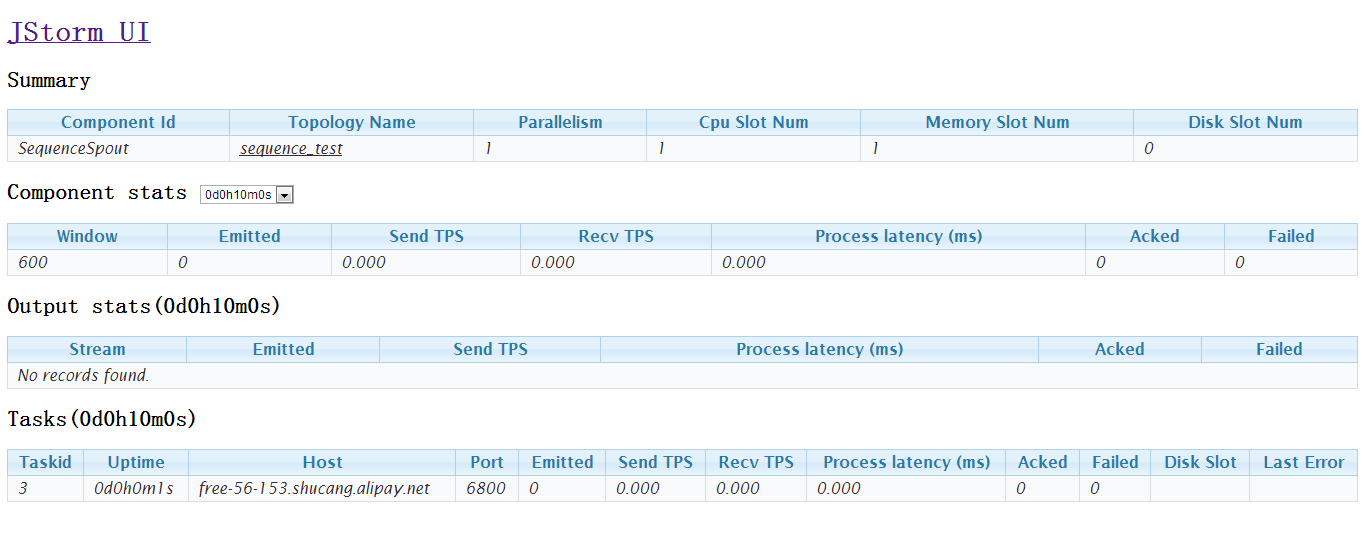
**Test process:**

1. Submit one topology which bolt sleep 120 s in bolt’s prepare
2. Check topology status

**Test snapshot:**

****

Spout will restart when it takes time more than 120s to prepare:



#### Bolt or spout occur exception in normal run

**Test purpose:**

Check whether topology can run?

**Test result:**

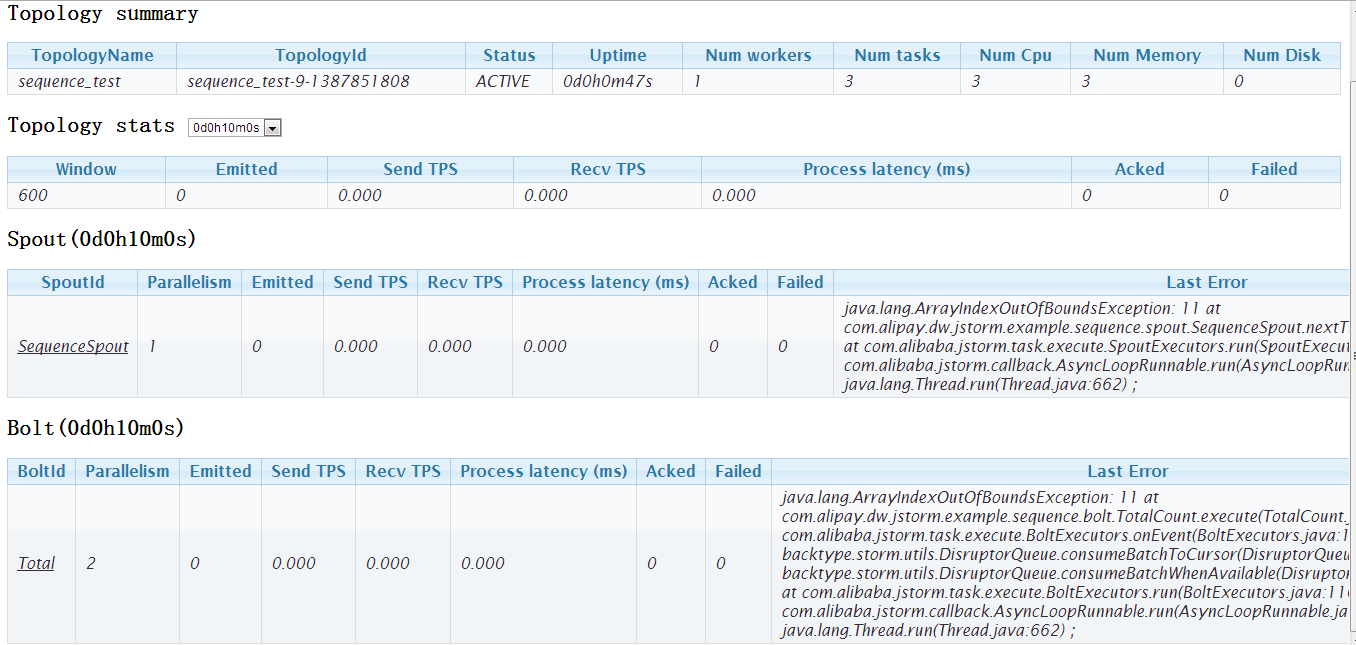
Pass, topology can’t run, web ui will display exception.

**Test process:**

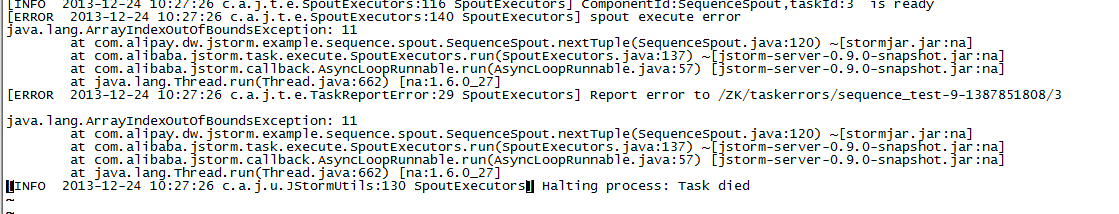
1. Submit one topology which bolt throw exception in execute
2. Check topology status

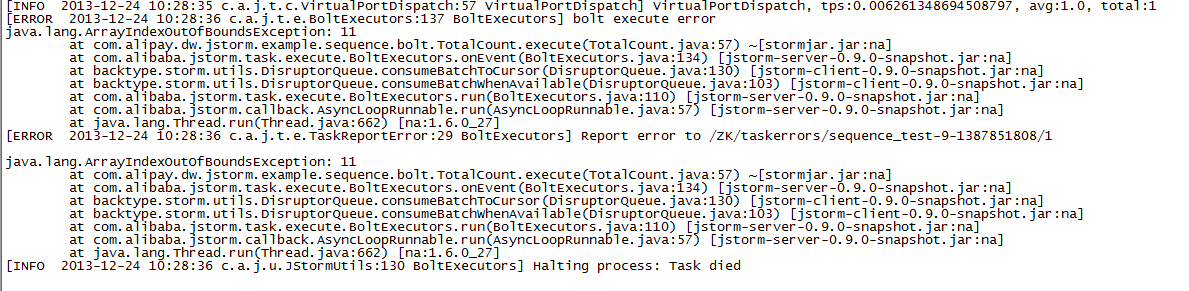
**Test snapshot:**

Topology status just after submit



Worker halt :





### Non Grouping method

**Test purpose:**

Check whether JStorm support noneGrouping method

**Test result:**

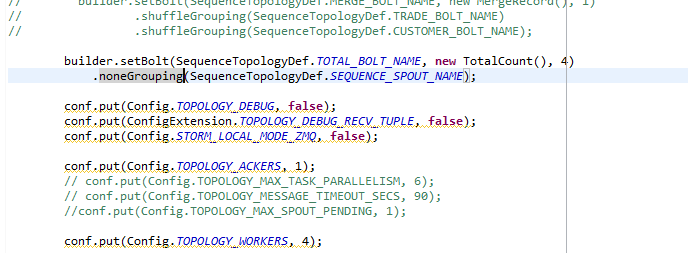
Pass, JStorm support noneGrouping method

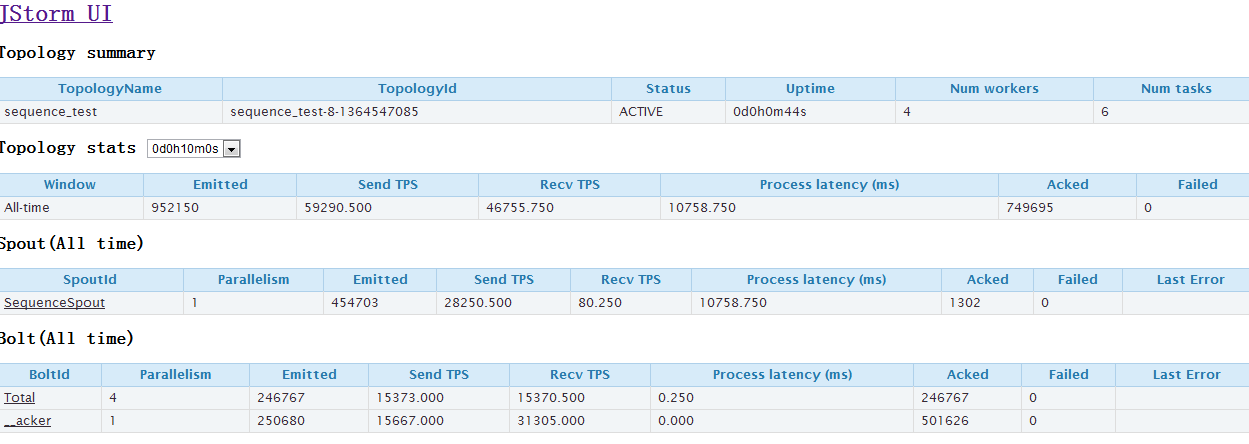
**Test process:**

1. Submit one topology which use nonGroupong method
2. Check topology status

**Test snapshot:**

Topology use nonGrouping method





### Topology bumping

#### Add supervisor when alive topologies use slots number is less than cluster can provide

**Test purpose:**

Check whether nimbus do rebalance for the old topology

**Test result:**

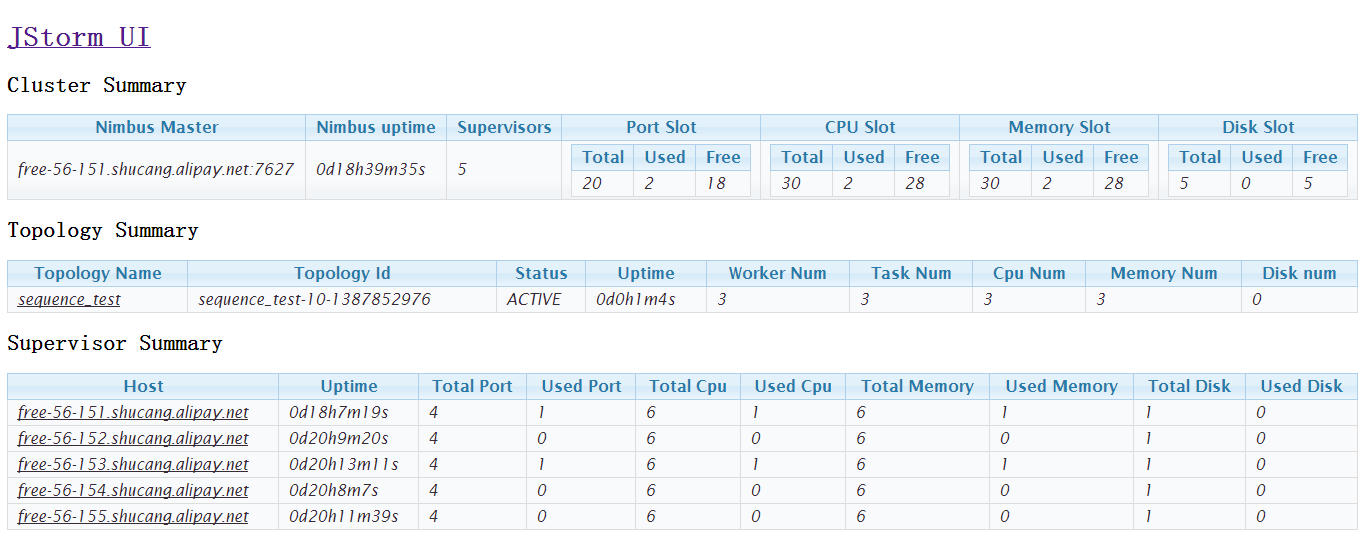
Pass, Nimbus hasn’t done rebalance for the old topology

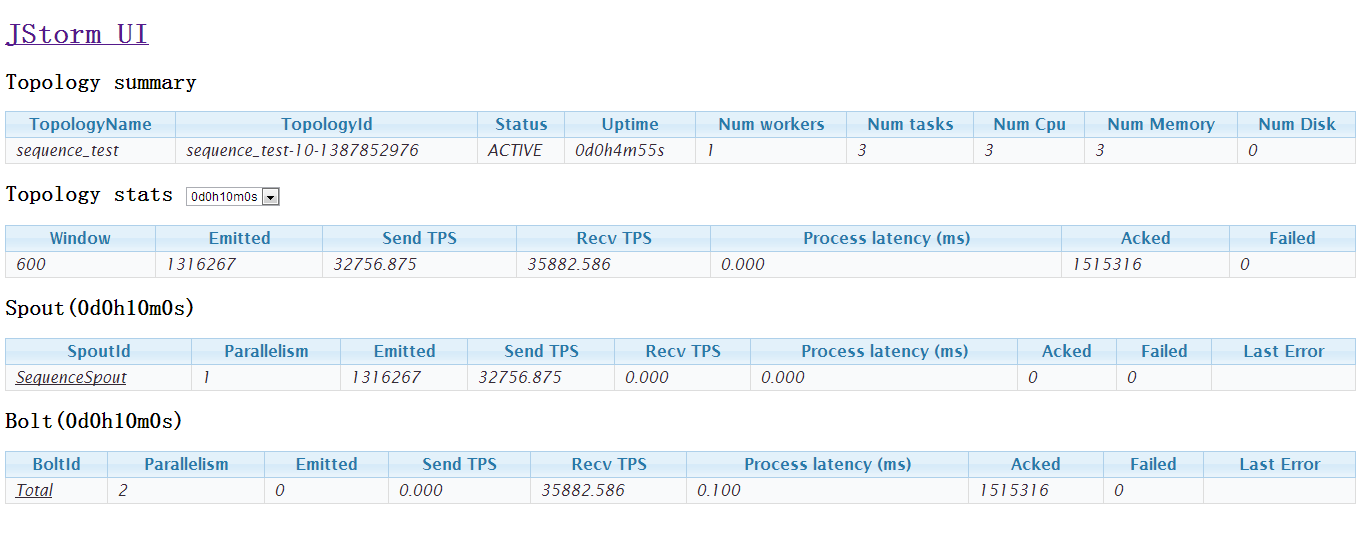
**Test process:**

1. Submit one topology which use slots number is less than JStorm cluster can provided
2. Check topology status
3. Add a new topology
4. Check the topology status, check whether the topology has been rebalanced.

**Test snapshot:**

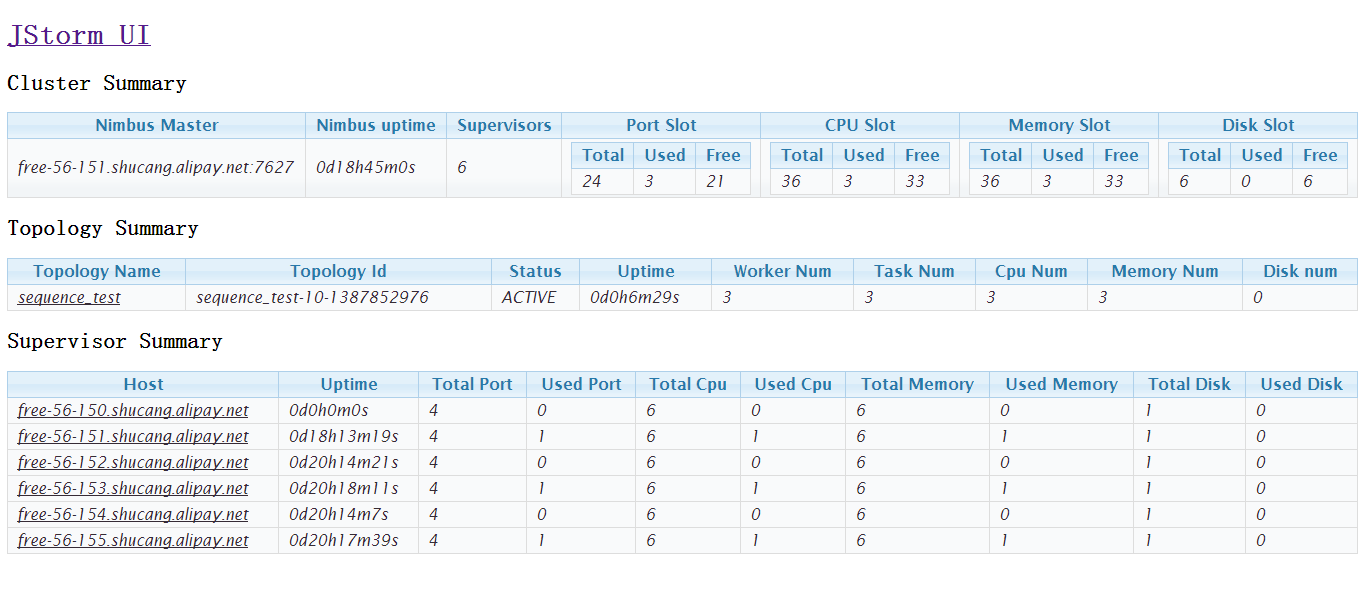
Before add supervisor:

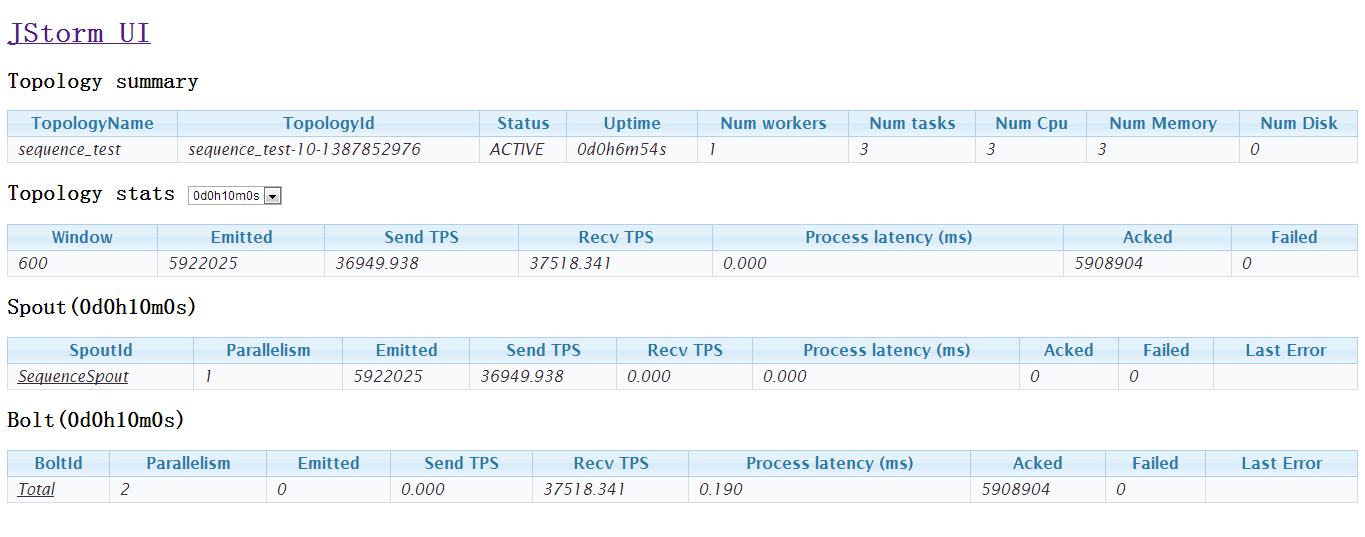




Add a supervisor :

Old topology no change





#### Add supervisor when alive topologies use slots number is more than cluster can provide

**Test purpose:**

Check whether nimbus do rebalance for the old topology

**Test result:**

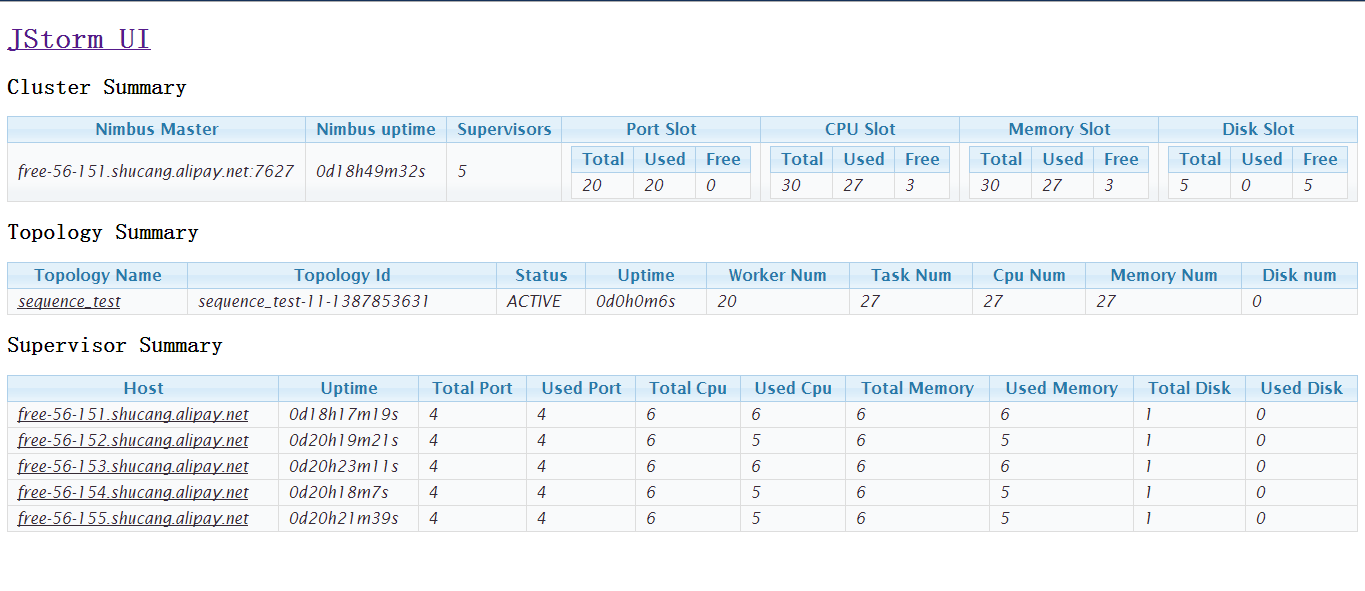
Pass, Nimbus hasn’t done rebalance for the old topology

**Test process:**

1. Submit one topology which use slots number is than than JStorm cluster can provided
2. Check topology status
3. Add a new topology
4. Check the topology status, check whether the topology has been rebalanced.

**Test snapshot:**

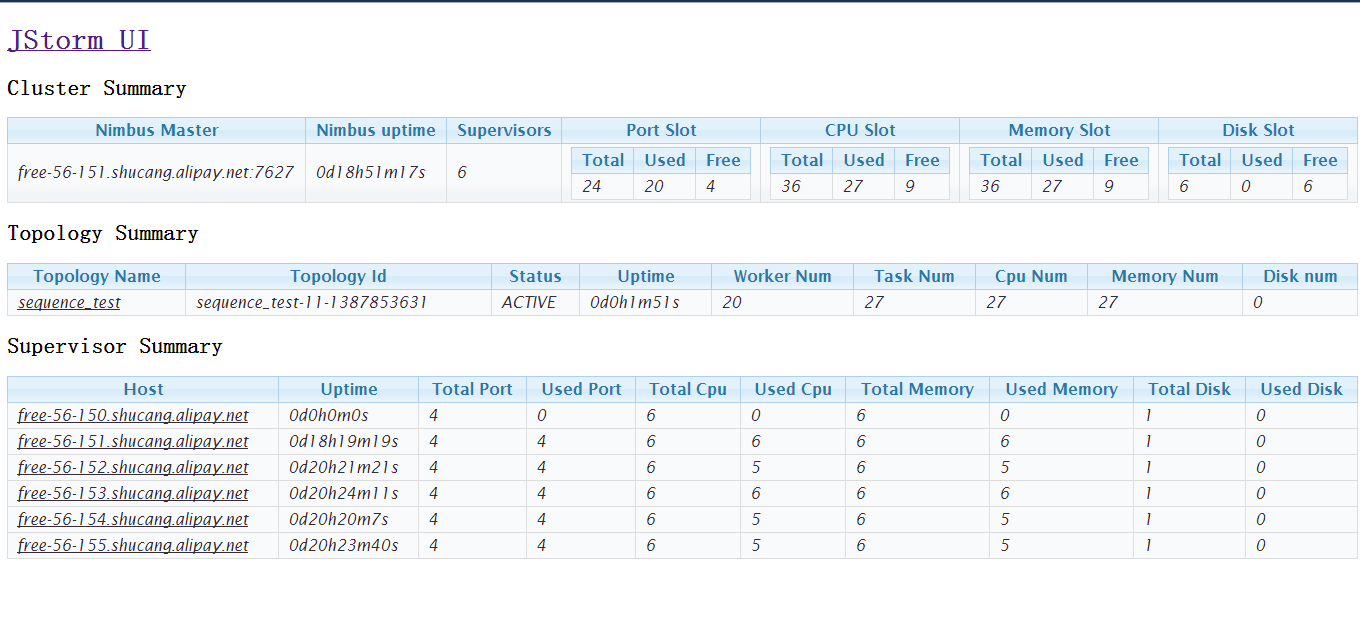
Before add supervisor:



But topology allocate 6 workers



Add one supervisor, nimbus hasn’t done rebalance the topology



#### Add topology which allocate slots is more than cluster can provide

**Test purpose:**

Check whether nimbus reduce the old topology’s worker slot

**Test result:**

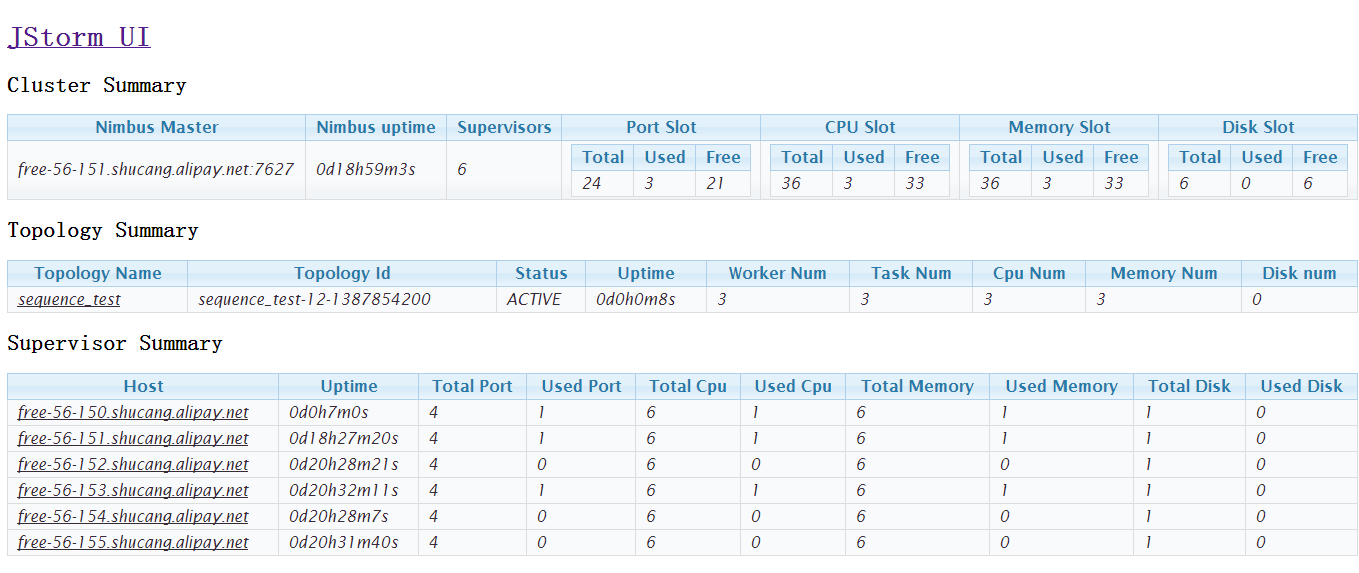
Pass, no change on the old topology

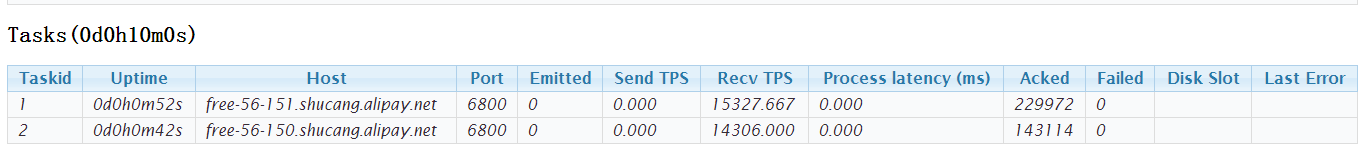
**Test process:**

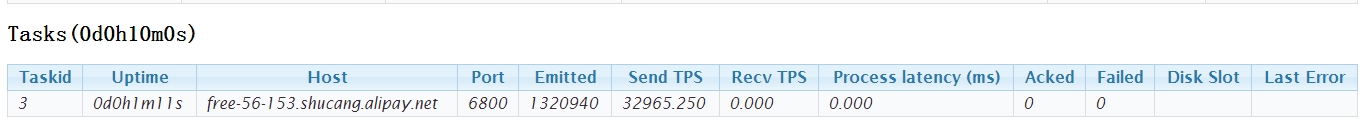
1. Submit one topology
2. Wait several minutes
3. Submit a new topology, which need more slots than the JStorm can provide.
4. Check whether the old topology has been impacted

**Test snapshots:**

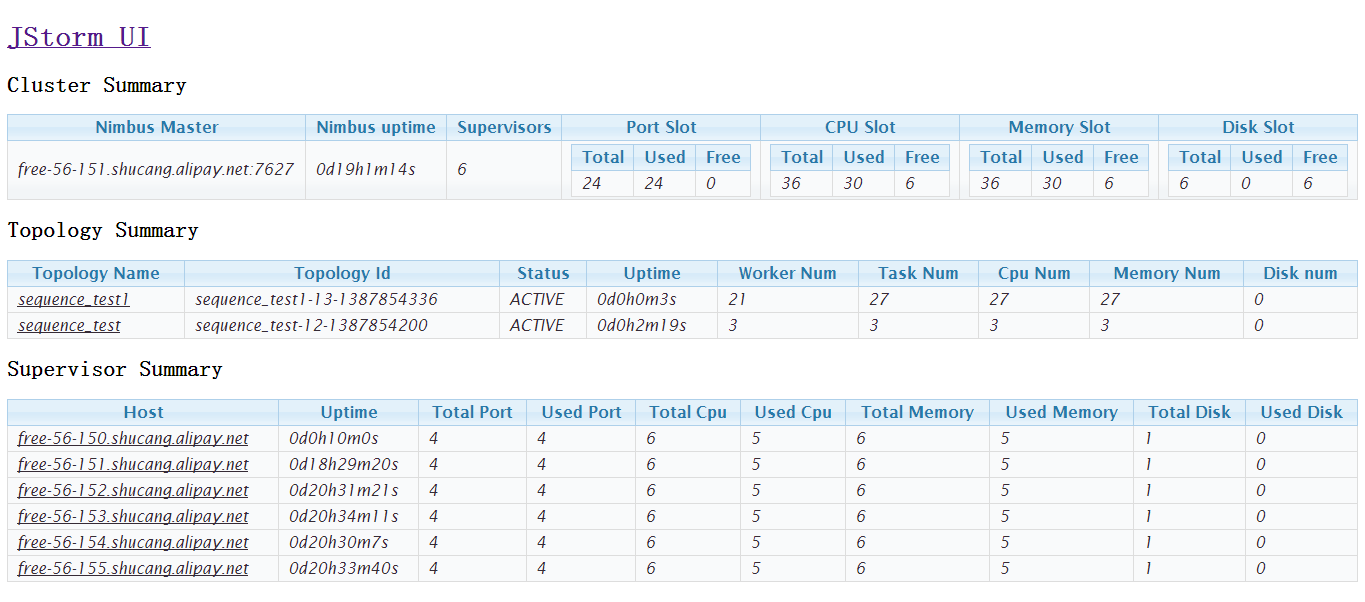
Before submit new topology:



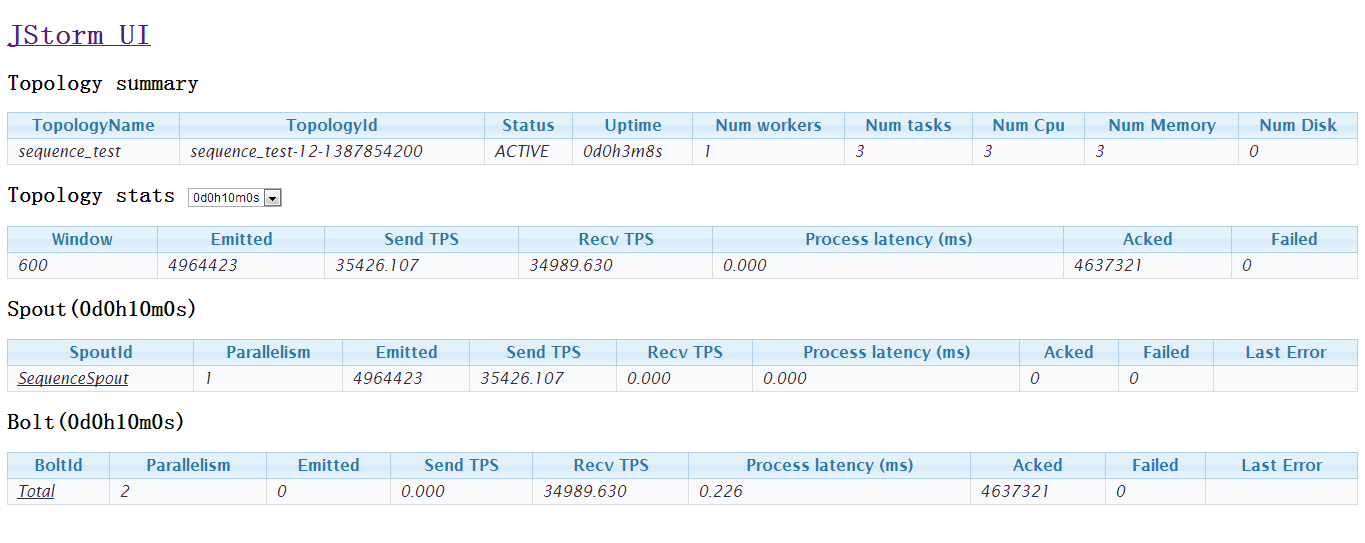


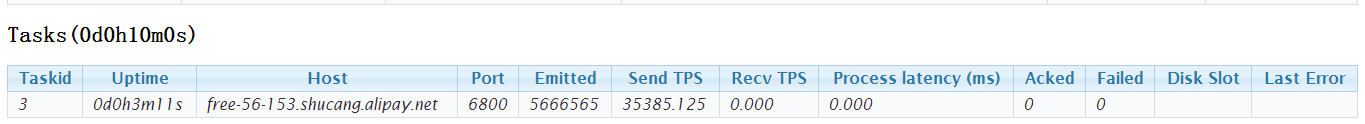


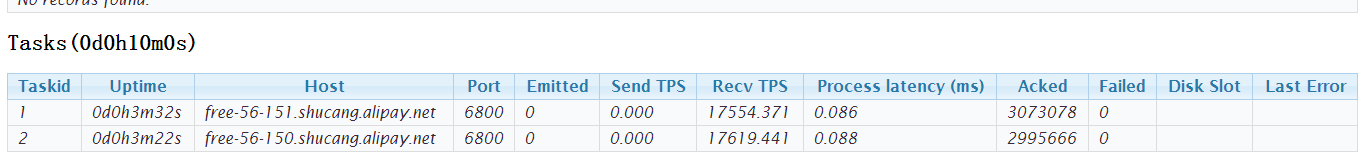
Submit the other topology:



The old topology status no change:







#### Kill topology when left topologies allocate slots number is more than cluster can provide

**Test purpose:**

Check whether nimbus do rebalance for left topologies after kill a topology

**Test result:**

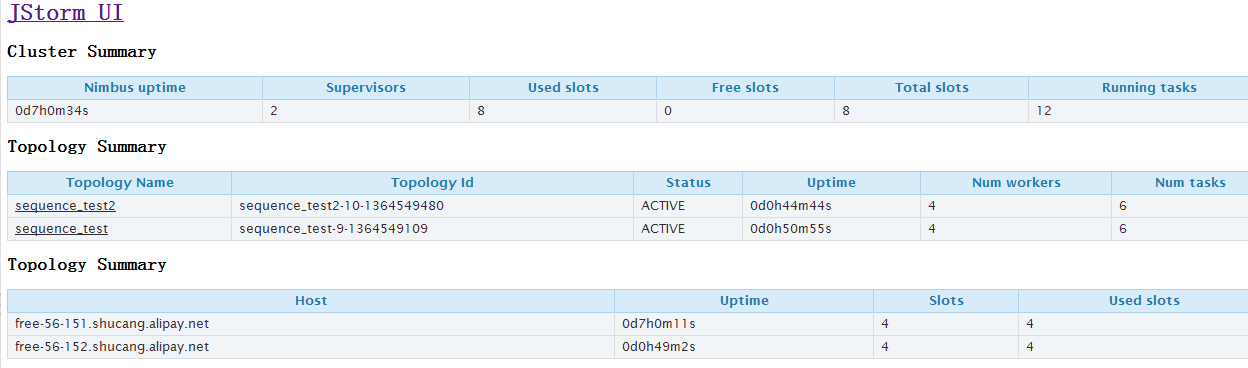
Pass, no change on the left topology, nimbus don’t rebalance for left topology after kill a topology, even when left topologies allocate slots number is more than they uses right now

**Test process:**

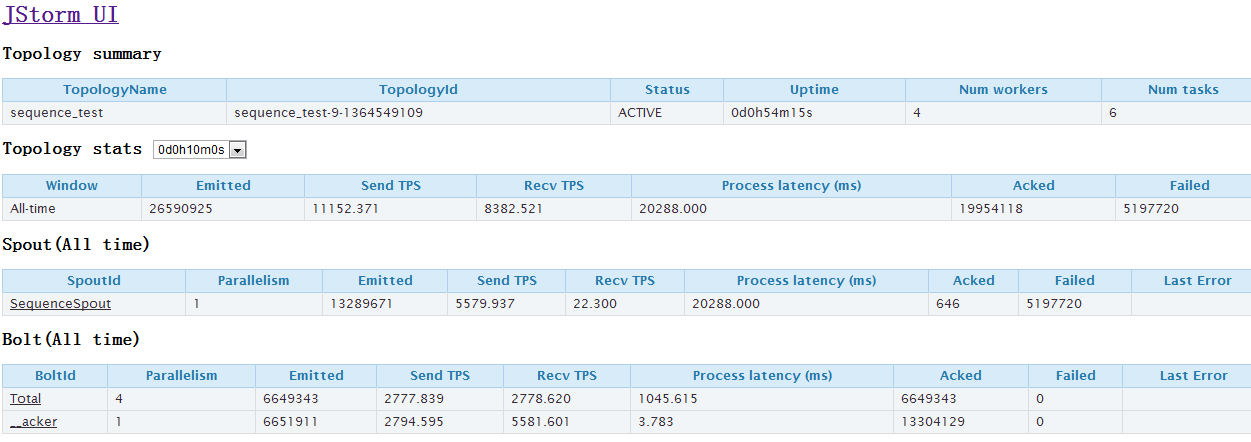
1. Submit one topology, which allocate slots number is more than cluster can provide
2. Wait several minutes
3. Add one supervisor
4. Submit a new topology
5. Wait several minutes, kill the new topology
6. Check whether nimbus do rebalance the old topology

**Test snapshots:**

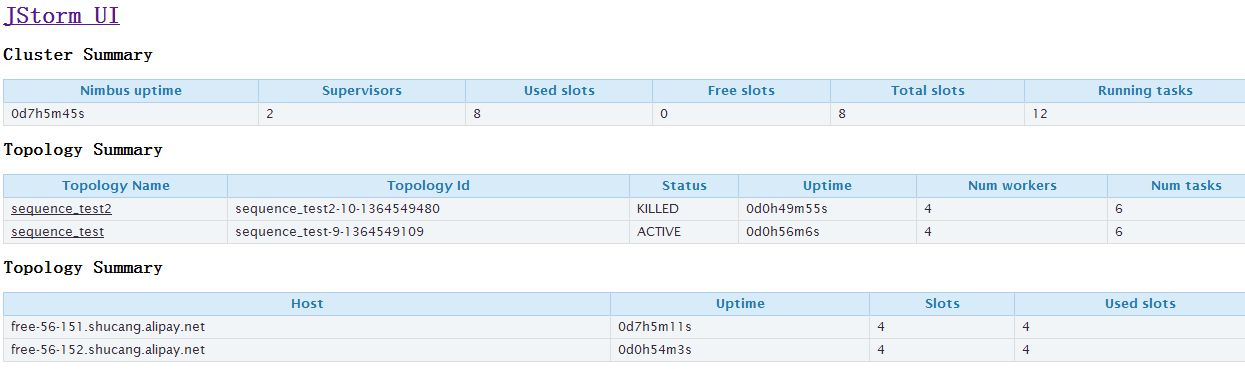
Before kill the new topology



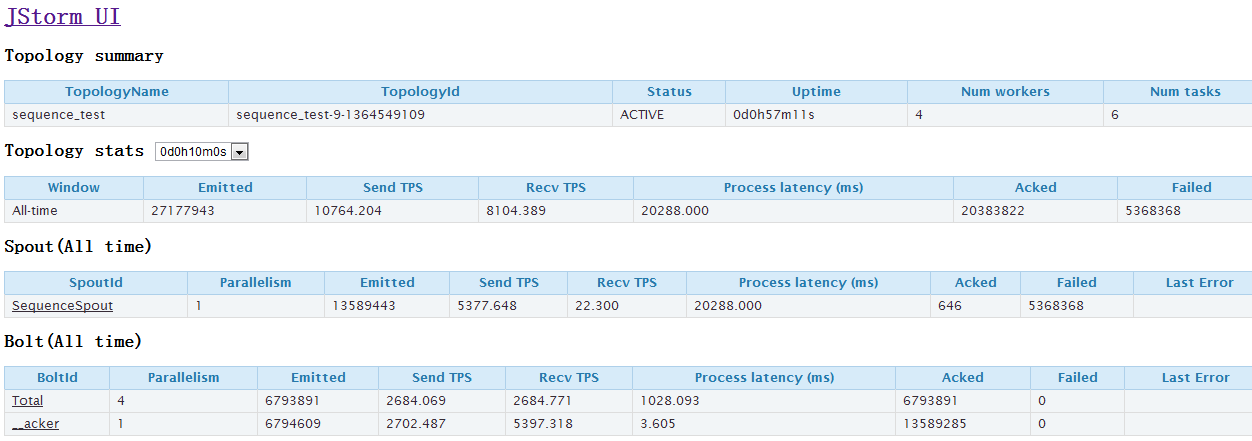
The old topology status:



After kill topology:



The old topology no change, if nimbus do rebalance, bolt or spout’s emit number will be reset



## Performance test

### Simplest test

**Test purpose:**

Check how much JStorm is faster than storm

**Test result:**

JStorm speed is triple of storm

**Test process:**

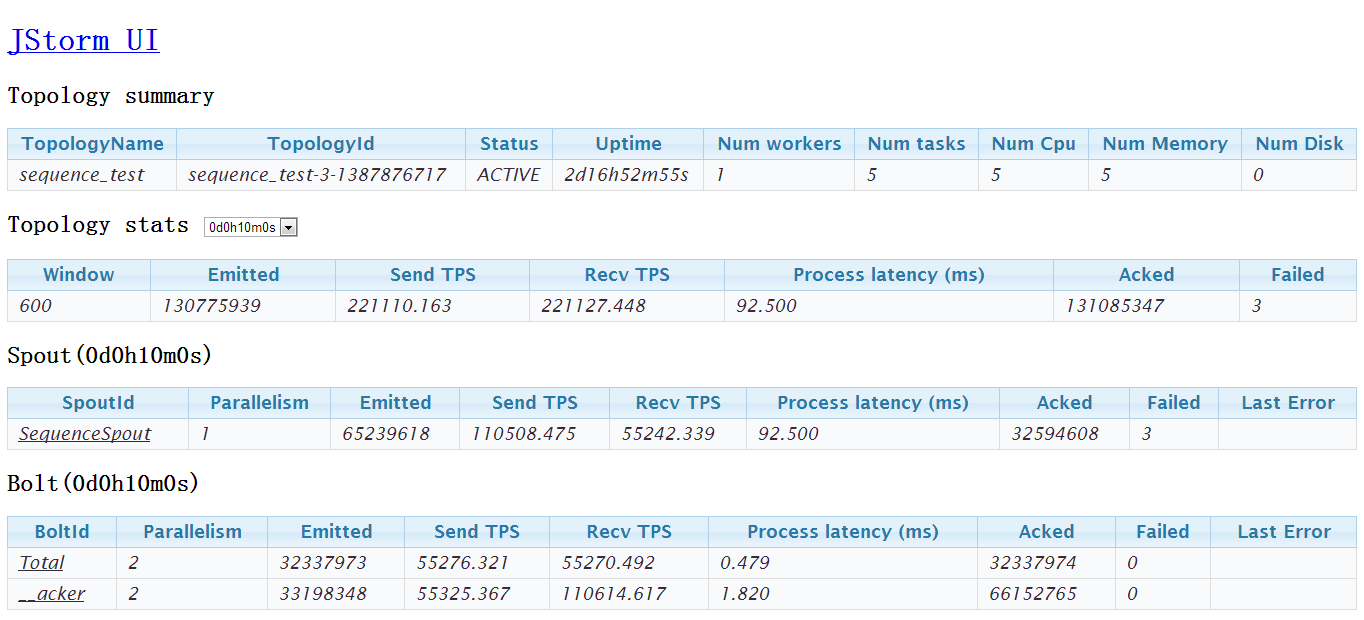
1. Submit the topology to JStorm and storm
2. Compare the two cluster’s result

**Test snapshots:**

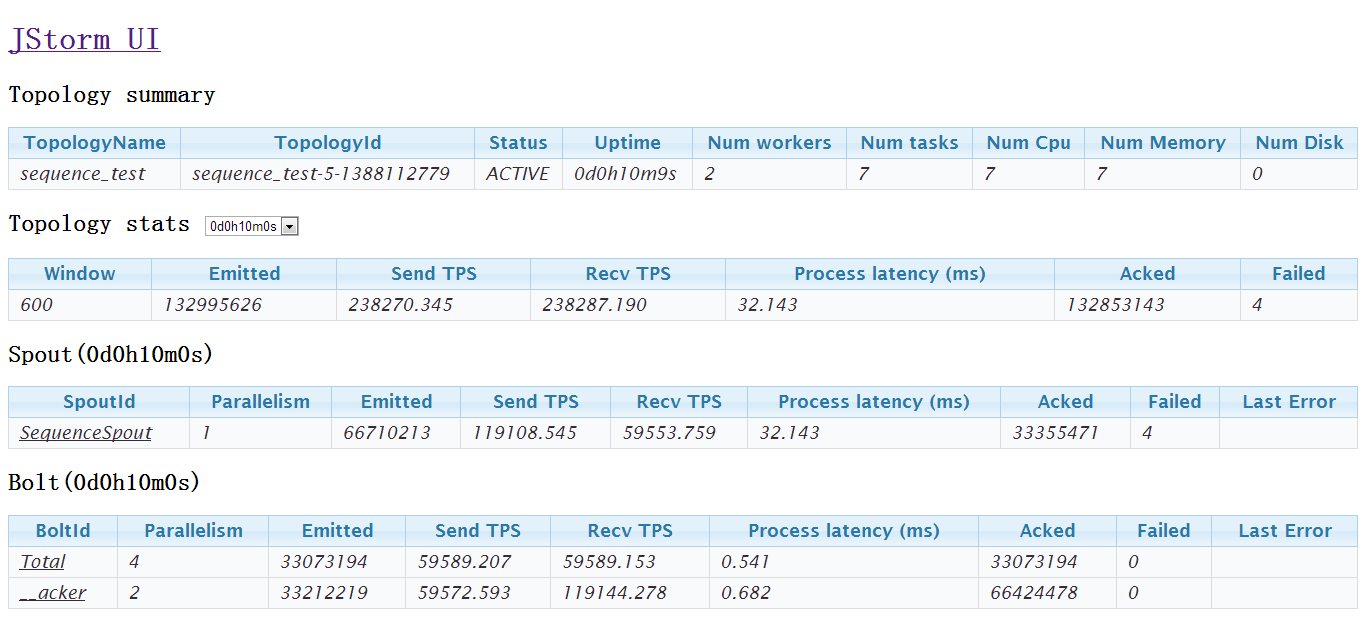
Prepare:

1. They are the same machine
2. Use the same topology
3. The topology spout generate tuple, bolt aggregate the value

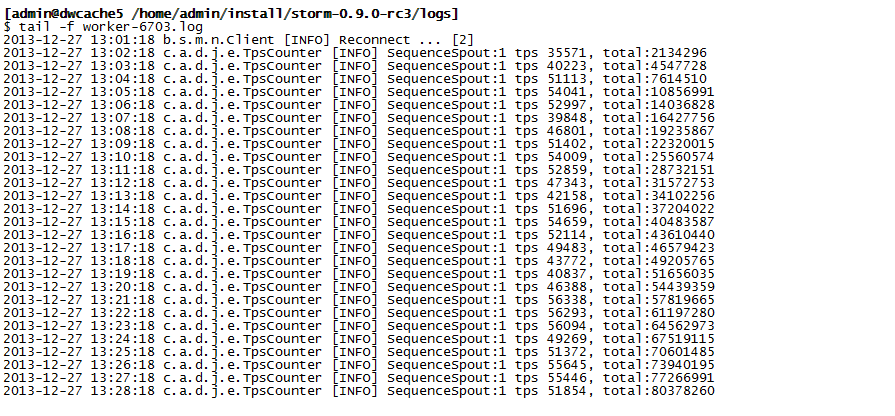
JStorm with netty:



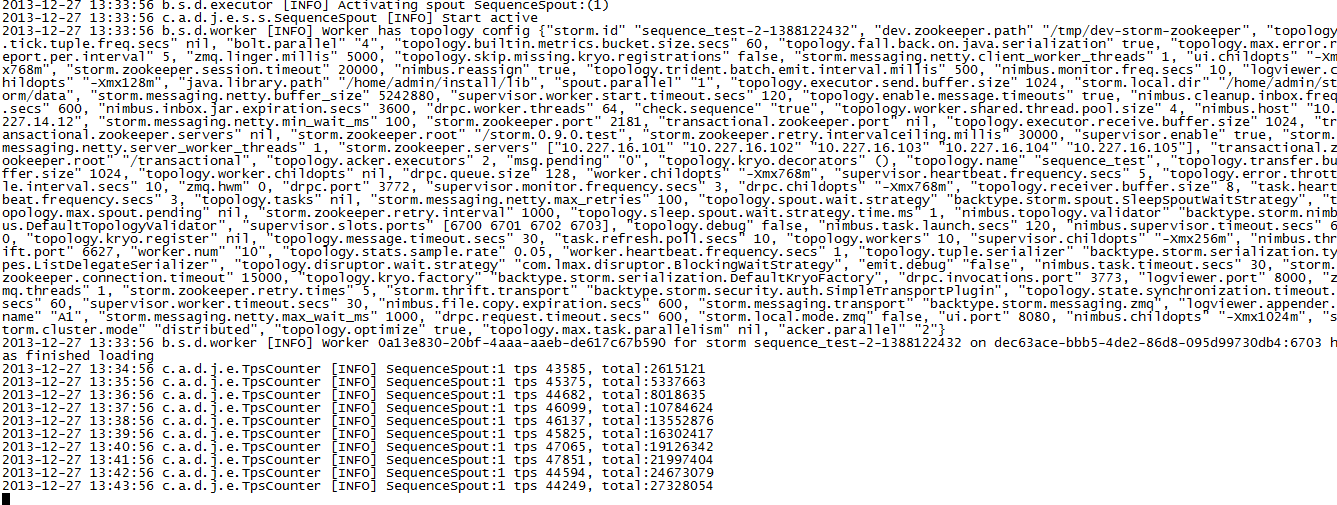
JStorm with zeroMQ:



Storm with netty



Storm with zeroMQ:



## Program correctness

### Simplest test

**Test purpose:**

Check whether JStorm work correctly?

**Test result:**

JStorm work correctly

**Test prepare**:

Topology sequence\_test

In the spout, it will random generate TradCustomer Tuple, then summary the new TradeCustomer.customer.value to local variable, do the same thing for TradeCustomer.trade.value, at last send the tuple

TradeCustomer{

Pair trade;

Pair customer;

}

Pair{

**protected** String key;

**protected** Long value;

}

In the total bolt, it just summary tuple’s TradeCustomer.customer.value to local variable, do the same job for TradeCustomer.trade.value

Then kill the topology

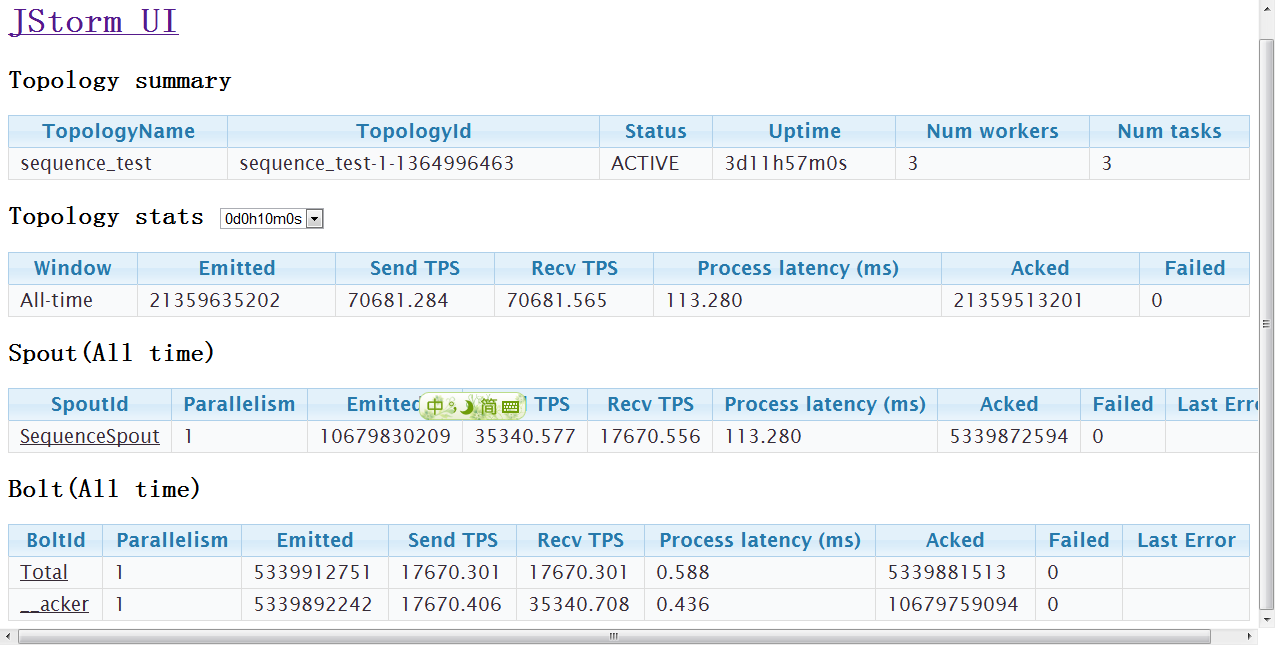
Then compare between the spout’s summary result and the total bolt’s result

**Test process:**

1. Submit the topology to JStorm
2. Wait 3 days
3. Kill the topology
4. Check spout’s summary result and bolt’s summary result

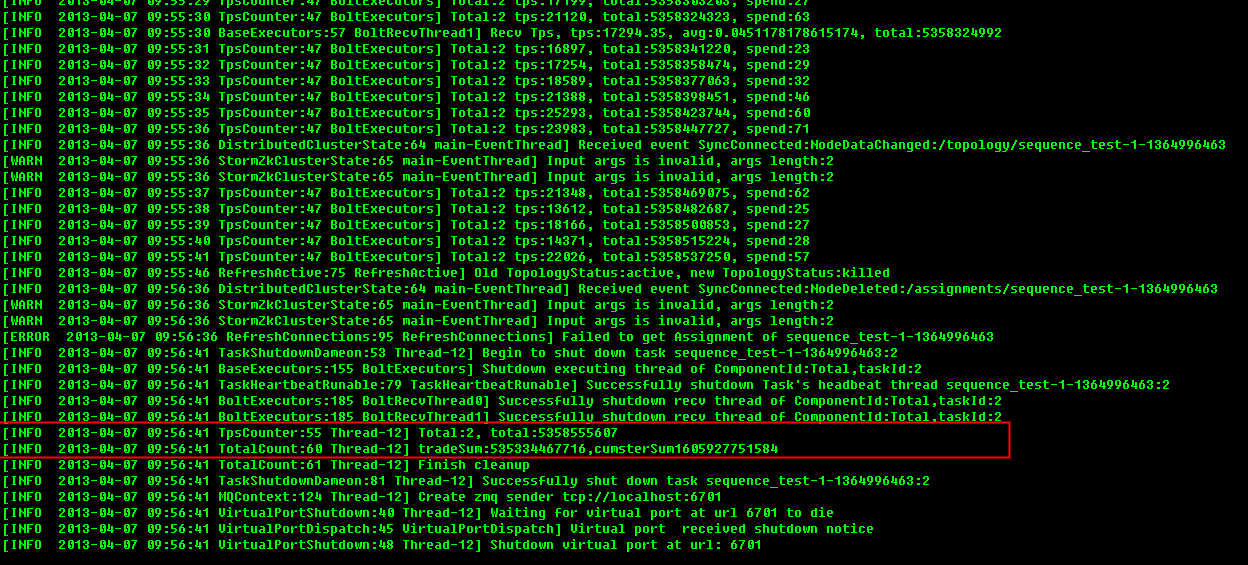
**Test snapshots:**

Running status:



Compare result:

Total Bolt’s result



Spout’s result:

