Administration & Operations

Planning / Testing

AEROSPIKE

Objectives

At the end of this module, you will be able to:

- Have a basic sense of capacity planning.
- Load test data into an Aerospike database using the Aerospike Java Benchmark tool.
- Run benchmark tests.

Capacity Planning

Capacity Planning

For a exact, detailed planning guide, go to:

http://www.aerospike.com/docs/operations/plan/capacity/

For a quick estimate (no secondary indexes) you will need the following:

- number of records
- replication factor
- size of records
- number of bins
- average set size
- type of storage (RAM or flash/SSD)
- Finally, overprovision calculated storage capacity to not hit the high-water-mark.

Capacity Planning – Quick Estimate

Area	How stored	Formula	Note
Primary Index	RAM	n * r * 64	The amount of RAM needed for the primary index is fixed at 64 bytes.
Data storage	RAM	n * r * (2 + (17 * b) + v)	Every objects needs 2 bytes for overhead, 17 bytes per bin, and the actual data
Data storage	Flash/SSD	n * r * p Where p is (64 + (9 + s) + (28 + 2 or 5)*b + v) -> round up to nearest 128 bytes	Every object needs to store the index (64 bytes), set overhead (9 +s bytes) , general overhead (28 bytes), type info (2(int) or 5(str) bytes per bin), and the actual data. Because Aerospike stores data in 128 byte blocks, you must round up to the nearest 128 byte amount.

- n = number of records
- r = replication factor
- v = average size of records
- b = number of bins
- s = avg set name length

Benchmarks

Benchmarking

Goal: **Saturate the cluster**

- Start with single server – BEWARE: “writes” values are not representative
- Change **1 thing** at a time
    - Move toward production configuration
- Are you **bottlenecked** on the
    - Client
    - Network

Note:

- Don’t expect high performance with **VMs**. You must be very careful when testing in a public cloud such as Amazon.
- Don’t expect realistic number with client and server on the **same host**.

The Java Benchmark Tool

- Multi-platform
- Portable
- Easy to use
- Total throughput easily seen from database nodes

- Poor performance with objects larger than 4 KB
- Run multiple instances on multiple servers to get to high throughput

Java Benchmark Syntax

```
>./run_benchmarks [options] [./run_benchmarks –u (for usage)]
```

Flag	Description
`-b, --bins <arg>`	The number of bins (like columns) of data.
`-g, --throughput <arg>`	The target total throughput. This will include all operations.
`-h, --host <arg>`	A seed node (any node in the cluster). The tool will learn about the other nodes from this one.
`-p, --port <arg>`	The port on the Aerospike node to connect to (default 3000)
`-k, --keys <arg>`	This is the number of keys/records to use. If you select a number too small, you will get write contention. Too large and you may run out of memory.
`-s, --set <arg>`	The set where the data will be loaded.
`-latency <arg>`	Produce a table of latencies as measured from the client. This can be compared with the latencies on the server to see where any bottleneck may lie. Recommended value "-latency 7,1"
`-n, --namespace <arg>`	The namespace to test against. This is similar to a "database" in a relational database.
`-o, --objectSpec <arg>`	The type of object to use: "I" is an integer, "S:<size>" is the size of the object. Is is recommended to start with small objects to remove the network
`-w, --workload <arg>`	Type of workload: `I` - Insert only. test stops when the final record is loaded (based on –k above) `RU,<percent>` - The read percentage. "90" means a 90% read rate.
`-z, --threads <arg>`	The number of threads used by the benchmark. For bare metal, 64 is a good value. For VMs, you should use 4-8 threads/core.

The Java Benchmark Tool

The Aerospike benchmark tool can be found within the full Aerospike Java Client.

Untar, go to the appropriate directory and compile it:

```
cd /home/aerotraining/packages/aerospike
tar xvf aerospike-client-java.tgz
cd aerospike-client-java-*/benchmarks
mvn package
```

Loading Test Data

Loading Data Example

Prior to running any tests, you should load data into the database.

```
./run_benchmarks     -h 127.0.0.1 \    # Local DB server
                     -p 3000 \         # Local port
                     -n test \     # Namespace “test”
                     -k 100000 \       # Load 100,000 records
                     -s testset \      # Use the set "testset"
                     -latency "7,1" \ # Show latency numbers
                     -o S:100 \    # Use 100 byte string
                     -w I \            # Insert only
                     -z 8              # Use 8 threads
```

- ./run_benchmarks -h 127.0.0.1 -p 3000 -n test -s testset -k 100000 -latency "7,1" -S 1 -o S:100 -w I -z 8

When the script has loaded all the data, it will exit.

Loading Data Example Output

```
$./run_benchmarks -h 127.0.0.1 -p 3000 -n test -s testset -k 100000 -latency "7,1" -S 1 -o
S:100 -w I -z 8
```

```
Benchmark: 127.0.0.1:3000, namespace: test, set: testset, threads: 8, workload: INITIALIZE
keys: 100000, start key: 1, key length: 10, bins: 1, throughput: unlimited, debug: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2014-08-27 22:12:52.091 INFO Thread 1 Add node BB9ED14150B0022 127.0.0.1:3000
2014-08-27 22:12:52.137 write(count=0 tps=0 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write     0%    0%    0%    0%    0%     0%     0%
2014-08-27 22:12:53.141 write(count=2432 tps=2432 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write    90%   10%    9%    8%    7%     5%     1%
2014-08-27 22:12:54.170 write(count=6492 tps=4060 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write    92%    8%    7%    6%    5%     3%     1%
2014-08-27 22:12:55.194 write(count=12705 tps=6213 timeouts=0 errors=0)
...
```

While this is running, check on the AMC.

Latency measurement

Latency parameter determines how latencies are displayed.

Format:

- "-latency N,E[,us]"
- N is the number of buckets
- E is the number of powers of 2 between each bucket
- us to get microsecond resolution on timings (Default is ms)

Buckets show percent transaction greater than that number (except 1st one)

-latency "7,1"

<=1ms	>1ms	>2ms	>4ms	>8ms	>16ms	>32ms	>64ms	>128ms	...
58%	42%	27%	15%	3%	1%	0%	0%	0%	

-latency "5,2"

<=1ms	>1ms	>2ms	>4ms	>8ms	>16ms	>32ms	>64ms	>128ms	...
58%	42%	27%	15%	3%	1%	0%	0%	0%	

Running Benchmarks

Running Balanced Workload

The first test to run is on a balanced workload. This will test database operations that are **50% reads 50% writes** (and updates).

```
./run_benchmarks    -h 127.0.0.1 \    # Local DB server
                    -p 3000 \         # Local port
                    -n test \      # Namespace “test”
                    -k 100000 \       # Load 100,000 records
                    -s testset \      # Use the set "testset"
                    -latency "7,1" \ # Show latency numbers
                    -o S:100 \     # Use 100 byte string
                    -w RU,50 \        # Read at 50%
                    -z 8              # Use 8 threads
```

This script will continue running until it is stopped.

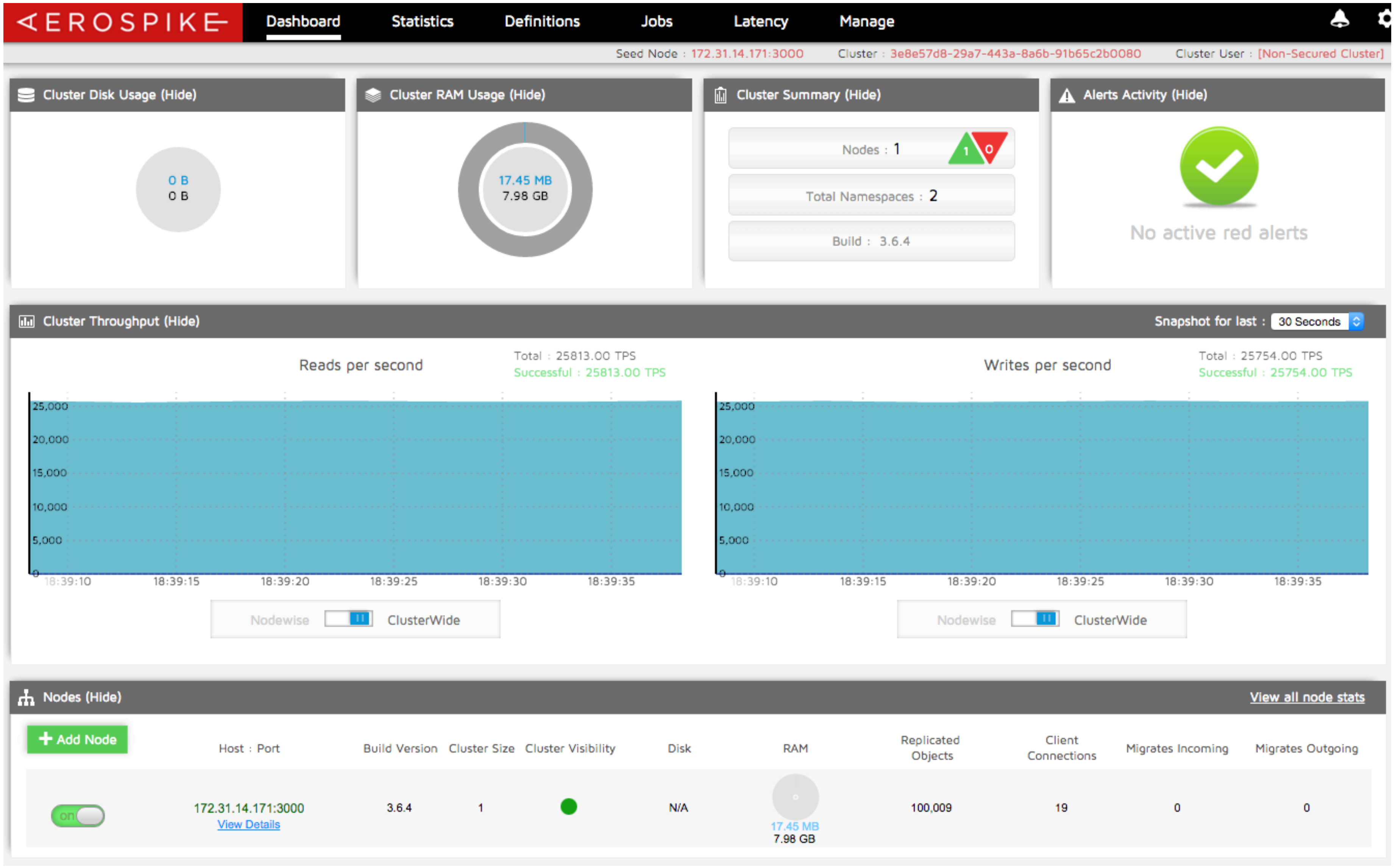
Running Balanced Load Example Output

```
$./run_benchmarks -h 127.0.0.1 -p 3000 -n test -s testset -k 100000 -latency "7,1" -o S:100 -
w RU,50 -z 8

Benchmark: 127.0.0.1:3000, namespace: test, set: testset, threads: 8, workload: READ_UPDATE
read: 50% (all bins: 100%, single bin: 0%), write: 50% (all bins: 100%, single bin: 0%)
keys: 100000, start key: 0, key length: 10, bins: 1, throughput: unlimited, debug: false
read policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500, reportNotFound: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2014-08-27 22:14:56.269 INFO Thread 1 Add node BB9ED14150B0022 127.0.0.1:3000
2014-08-27 22:14:56.329 write(tps=2 timeouts=0 errors=0) read(tps=0 timeouts=0 errors=0)
total(tps=2 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   89%  11%  11%  11%  11%     5%     2%
read    86%  14%  13%  13%  13%     9%     2%
2014-08-27 22:14:57.577 write(tps=1458 timeouts=0 errors=0) read(tps=1402 timeouts=0
errors=0) total(tps=2860 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   86%  14%  13%  12%  11%     6%     1%
read    89%  11%  10%   9%   8%     5%     2%
2014-08-27 22:14:58.585 write(tps=1994 timeouts=0 errors=0) read(tps=1934 timeouts=0
errors=0) total(tps=3928 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   93%   7%   7%   6%   5%     3%     1%
read    94%   6%   5%   5%   4%     2%     1%
...
```

While this is running, check on the AMC.

AMC running benchmark



Seed Node : 172.31.14.171:3000 Cluster : 3e8e57d8-29a7-443a-8a6b-91b65c2b0080 Cluster User : [Non-Secured Cluster]

Cluster Disk Usage (Hide): 0 B / 0 B

Cluster RAM Usage (Hide): 17.45 MB / 7.98 GB

Cluster Summary (Hide)

Nodes : 1

Total Namespaces : 2

Build : 3.6.4

Alerts Activity (Hide): No active red alerts

Cluster Throughput (Hide) — Snapshot for last : 30 Seconds

Reads per second — Total : 25813.00 TPS, Successful : 25813.00 TPS

Writes per second — Total : 25754.00 TPS, Successful : 25754.00 TPS

Nodes (Hide) — View all node stats

	Host : Port	Build Version	Cluster Size	Cluster Visibility	Disk	RAM	Replicated Objects	Client Connections	Migrates Incoming	Migrates Outgoing
on	172.31.14.171:3000 View Details	3.6.4	1	●	N/A	17.45 MB 7.98 GB	100,009	19	0	0

Running High-read Workload

You can then run a workload based on high read rates. This will test database operations that are **95% reads 5% writes**

```
./run_benchmarks    -h 127.0.0.1 \    # Local DB server
                    -p 3000 \         # Local port
                    -n test \      # Namespace “test”
                    -k 100000 \       # Load 100,000 records
                    -s testset \      # Use the set "testset"
                    -latency "7,1" \ # Show latency numbers
                    -o S:100 \     # Use 100 byte string
                    -w RU,95 \        # Read at 95%
                    -z 8              # Use 8 threads
```

This script will continue running until it is stopped.

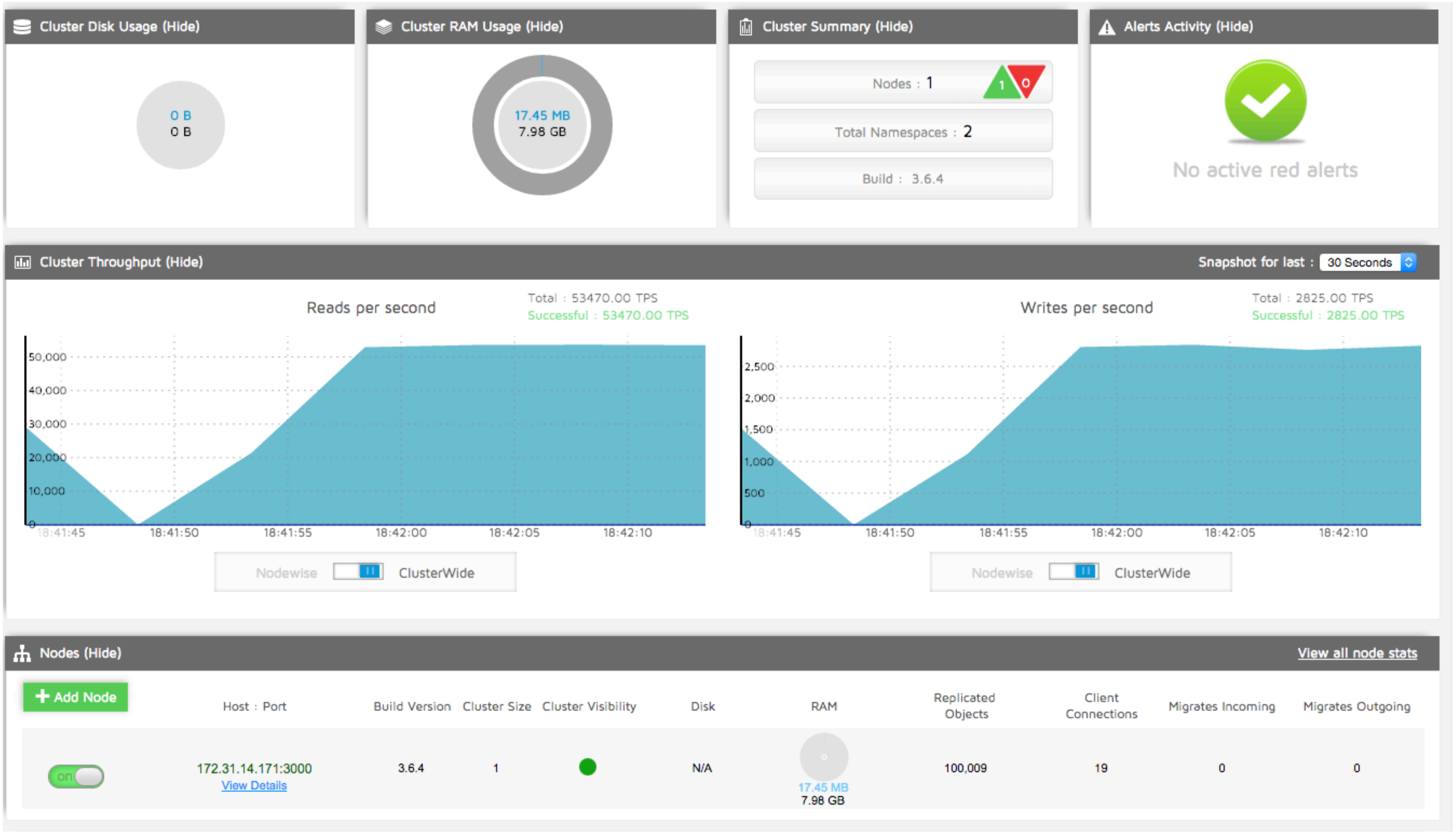
Running High-read workload Example Output

```
$./run_benchmarks -h 127.0.0.1 -p 3000 -n test -s testset -k 100000 -latency "7,1" -o S:100 -w
RU,95 -z 8
```

```
Benchmark: 127.0.0.1:3000, namespace: test, set: <empty>, threads: 8, workload: READ_UPDATE
read: 95% (all bins: 100%, single bin: 0%), write: 5% (all bins: 100%, single bin: 0%)
keys: 100000, start key: 0, key length: 10, bins: 1, throughput: unlimited, debug: false
read policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500, reportNotFound: false
write policy: timeout: 0, maxRetries: 2, sleepBetweenRetries: 500
bin type 1: string[100]
2015-05-28 18:58:09.946 INFO Thread 1 Add node BB94FB6A4647106 127.0.0.1:3000
…
2015-05-28 18:58:17.356 write(tps=488 timeouts=0 errors=0) read(tps=8374 timeouts=0 errors=0)
total(tps=8862 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   96%   4%   3%   2%   2%    1%    1%
read    97%   3%   3%   2%   2%    1%    1%
2015-05-28 18:58:18.391 write(tps=468 timeouts=0 errors=0) read(tps=9569 timeouts=0 errors=0)
total(tps=10037 timeouts=0 errors=0)
      <=1ms >1ms >2ms >4ms >8ms >16ms >32ms
write   95%   5%   4%   3%   2%    1%    0%
read    97%   3%   2%   2%   2%    1%    0%
...
```

While this is running, check on the AMC.

AMC running benchmark



Cluster Disk Usage (Hide)

0 B
0 B

Cluster RAM Usage (Hide)

17.45 MB
7.98 GB

Cluster Summary (Hide)

Nodes : 1 (1 ▲, 0 ▼)

Total Namespaces : 2

Build : 3.6.4

Alerts Activity (Hide)

No active red alerts

Cluster Throughput (Hide)

Snapshot for last : 30 Seconds

Reads per second
Total : 53470.00 TPS
Successful : 53470.00 TPS

Writes per second
Total : 2825.00 TPS
Successful : 2825.00 TPS

Nodes (Hide)

View all node stats

+ Add Node

	Host : Port	Build Version	Cluster Size	Cluster Visibility	Disk	RAM	Replicated Objects	Client Connections	Migrates Incoming	Migrates Outgoing
on	172.31.14.171:3000 View Details	3.6.4	1	●	N/A	17.45 MB 7.98 GB	100,009	19	0	0

Summary

What we have covered:

- Have a basic sense of capacity planning.
- Load test data into an Aerospike database using the Aerospike Java Benchmark tool.
- Run benchmark tests.