kaminario.

WHITE PAPER

Kaminario K2 Reference Architecture for OpenStack and MongoDB

April 2017

TABLE OF CONTENTS

Executive Summary

Enterprises' constant growth and activity drive data usage. The on-demand approach of modern businesses and applications, where customers and different business units demand agility and simplicity, is pushing the envelope of capacity and performance more than ever before. Businesses will rise and fall on their ability to deliver market needs in the fastest way possible.

OpenStack, an open source platform for cloud management is the leading Infrastructure-as-a-service (IaaS) platform that powers many of these business-critical applications. As an IaaS platform, OpenStack environments are very likely to generate a random blend of I/O requests and create heavy workloads that can swamp storage systems.

MongoDB is the leading NoSQL database that is widely used by a wide range of businesses to support the infrastructure for modern applications and environments where relational databases cannot deliver the agility, flexibility and scalability of various types of data.

The Kaminario K2 all-flash array is the perfect storage platform for hosting and powering as-a-Service and cloud-scale environments. Running VisionOS™, the K2 array has a unique scalable architecture that can independently scale performance and capacity, matching the demanding and challenging requirements of cloud-scale infrastructure. K2 delivers a consistent level of throughput, IOPS and low latencies needed to support the demanding blend of storage I/O workloads of any environment thanks to its adaptive block size algorithm.

This reference architecture covers K2's benefits for OpenStack environments running virtualized servers, with MongoDB as the hosted application infrastructure. Detailed performance and scalability test results are provided for a range of OpenStack operations, showing that K2 performs well under different types of MongoDB workloads with no degradation in performance or efficiency. Best Practices and high-level installation steps are also covered in the report, showing the process to deploy such an environment.

Introduction to K2

Kaminario is leading the revolution of enterprise flash storage by creating the industry's most scalable, intelligent and cost-effective all-flash storage array in the market. Built from the ground up to take advantage of the most modern flash SSD capabilities, the K2 all-flash storage array is the only product to feature a true scale-out and scale-up architecture that allows organizations to grow capacity and performance based on their needs. This architecture ensures both data availability and a consistent level of high throughput, IOPS and low latencies needed to support the demanding random I/O generated by business-critical systems including mixed-workloads such as OpenStack.

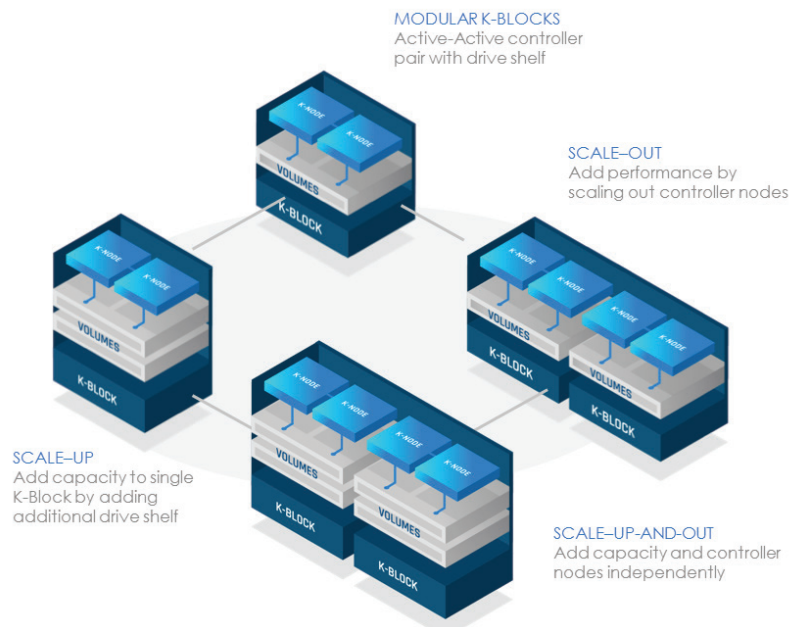

Figure 1: Kaminario K2 Scalable Architecture

The K2 all-flash array is an appliance which is a combination of tested and packaged hardware, software and services. K2's Gen6 hardware platform is based on leading off-the-shelf enterprise components that enable K2's software-defined architecture and software stack. The K2 runs Kaminario VisionOS™, the next-generation flash operating system stack, that provides the core software and advanced data services framework.

VisionOS enables modular components and services that demonstrate a superior value proposition across a real scale-out storage platform, both in innovation and in ease of use:

- **DataShrink** – Data reduction features and capabilities are mandatory for economics of flash storage. With differentiating inline, global, adaptive and selective deduplication, together with inline byte aligned compression, thin provisioning and zero detection, Kaminario is able to establish itself a cost-efficiency leader of flash storage.

- **DataProtect** – Kaminario values its customers' data more than anything. Native array based snapshots and replication allow for returning to any point in time in any site. Data-at-rest AES256 encryption makes sure that data is kept private and safe at all times. A highly resilient design of no single point of failure, non-disruptive upgrades (NDU) and a robust RAID scheme facilitate 99.999% of data availability.
- **DataManage** –The K2 can be managed by various means. Internal management includes an intuitive web-based GUI, a scriptable CLI and a fully programmable RESTful API platform.
- **DataConnect** –K2's RESTful API allows for external applications of the IT eco-system to easily integrate and seamlessly manage the K2. This eco-system is constantly growing and includes: VMware vSphere, Microsoft VSS, OpenStack, Flocker (containers) and Cisco UCS director.

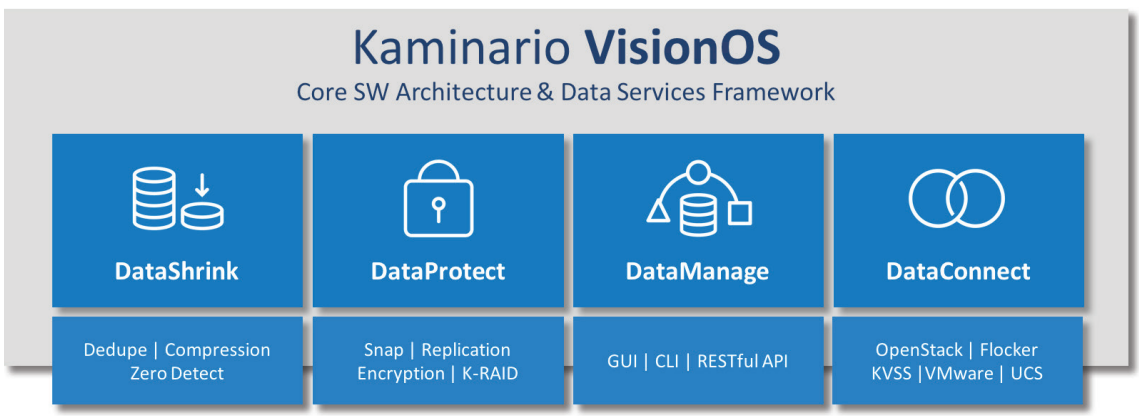

Figure 2: Kaminario VisionOS

What is OpenStack

General Overview

Established at 2010, OpenStack is a free and open source platform that collects many smaller projects together to enable cloud computing management in an Infrastructure-as-a-service (IaaS) manner.

The core of OpenStack is built from several projects which are integrated within a single platform. Every project is responsible for managing a different type of resource (Compute, Networking, Storage, etc.) and all together build OpenStack. This modular design gives OpenStack its power of scalability and flexibility.

Just like the Kaminario K2, OpenStack enables enterprises and service providers to start small and scale by adding resources as they are required.

OpenStack can be deployed as a private cloud on-premises; it could also be deployed as a public cloud for cloud computing service providers, or, as a combination of both – a hybrid cloud.

Modular Design

The modular design of OpenStack is based on many projects, with the more relevant listed below:

- **Nova** - provides compute capability for OpenStack by managing virtual machines upon demand.
- **Neutron** – provides advanced network services and management.
- **Cinder** - provides persistent block storage services deployed instances. The Kaminario K2 Cinder Driver is the key function that enables the Cinder module the option to manage the Kaminario K2.
- **Glance** – Provides a repository for disk images.
- **Keystone** – Provides identity service used by OpenStack for authentication and authorization.
- **Horizon** – Provides a web-based management UI for OpenStack.

The Kaminario K2 Cinder Driver

The Kaminario K2 Cinder Driver provides integration between OpenStack and The Kaminario K2 by using the OpenStack Cinder service. With this integration, users can deploy virtual machines (Instances) and services on top of Kaminario K2 volumes, enabling high performance to applications and services with full-redundancy and high-availability.

The Kaminario K2 Cinder Driver communicates with K2 via the K2 RESTful API service. Using the K2 RESTful API is a fast and reliable method to trigger different CRUD operations remotely.

Kaminario K2 and OpenStack

Kaminario's VisionOS is the perfect companion for OpenStack. In environments like OpenStack where resources requirements are not easy to predict, the Kaminario K2 is a perfect fit. Given its ability to scale-up and scale-out, the Kaminario K2 can always grow along an OpenStack deployment and handle the dynamic demands from the storage layer.

The Kaminario K2 is an optimal solution for OpenStack due to the following:

Kaminario K2 Data reduction

- **Deduplication** – Whenever you deploy an instance in OpenStack, you can always scale it to a larger number of instances. For example, you can start with a single instance of MongoDB and then later convert it into a Replica-Set to improve high-availability and redundancy. The Kaminario K2 deduplication mechanism can eliminate duplicated data in the most efficient way by utilizing its adaptive size deduplication mechanism.
- **Compression** – Whether your instances are running databases, web-applications or any other services, all data is compressed by the Kaminario K2 always-on compression mechanism.

Kaminario K2 Performance

- Optimized for mixed workloads: a global adaptive block algorithm allows the system to automatically adapt to support OLTP, OLAP, virtualized and VDI environments without compromising IOPS or bandwidth.

Kaminario K2 Scalability

- K2 is the only storage array on the market today that supports both scale-up and scale-out, enabling organizations to start small and grow seamlessly. This flexibility eliminates the need to compromise on how to non-disruptively expand and grow the array based on the customer requirements.

Kaminario K2 iSCSI VLAN Support

- Networking in OpenStack can be sophisticated and may include many VLANs. The Kaminario K2 support up to 256 unique VLANs.

Kaminario K2 LAG support

- Each K2 storage controller (K-Node) has two 25Gb Ethernet ports, and these two ports (which must be within the same K-Node) can be logically aggregated into a single logical port, otherwise known as a Link Aggregation Group (LAG), port channel or port bonding. A LAG makes management easier and improves connectivity redundancy.

Solution Overview

To demonstrate the integration of the Kaminario K2 Cinder driver with OpenStack, we will deploy a MongoDB 3.2 based solution.

Each of the instances we cover in this reference architecture is running on top of a Kaminario K2 volume. The instances' K2 volumes are managed by the Kaminario K2 Cinder driver which is responsible to communicate with K2 and to create volumes, map volumes, delete volumes, create snapshots and more.

At first, we will deploy a standalone MongoDB instance and load some data into it. Even at this step, right at the beginning of the process, we will show how the K2 data-reduction mechanism kicks in and provides major capacity saving.

Then, we will convert the single instance into a Replica-Set of three (3) instances for redundancy and high-availability improvements. At this step, we show how the Kaminario K2 data reduction has improved even better after data was replicated into the two new instances. At this stage, we have three (3) copies of the same data. As a next step, we will convert the Replica-Set into a Sharded Cluster to improve performance, especially throughput.

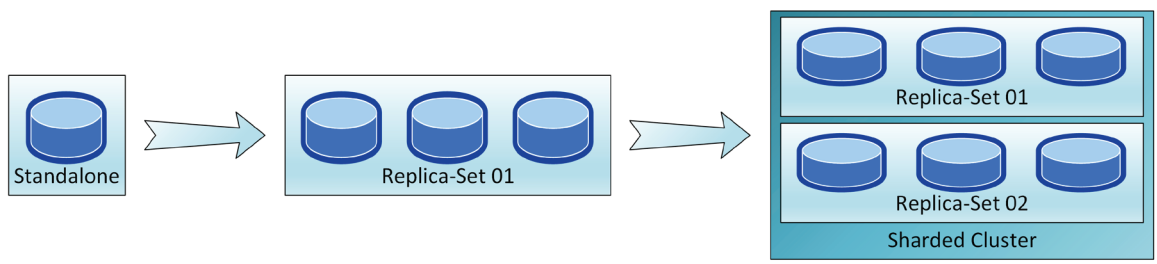

Figure 3: Instance layout

At each configuration – Standalone, Replica-Set and Sharded Cluster, we will run a workload that performs both reads and writes in variable ratios. The benchmark is searching for documents based on a filter criteria and update the results with some random data.

Furthermore, the solution covers some best practices and configuration recommendations for MongoDB, based on Kaminario's white paper: MongoDB on Kaminario K2.

The OpenStack flavor used in this solution is Ubuntu's OpenStack which makes use of Juju. Later in this reference architecture you may find some information on installing the Kaminario K2 Cinder Juju Charm and deploying Ubuntu's OpenStack.

Solution Environment

The environment used in this solution is based on Ubuntu's flavor of Mitaka OpenStack and was built as described in the following diagram:

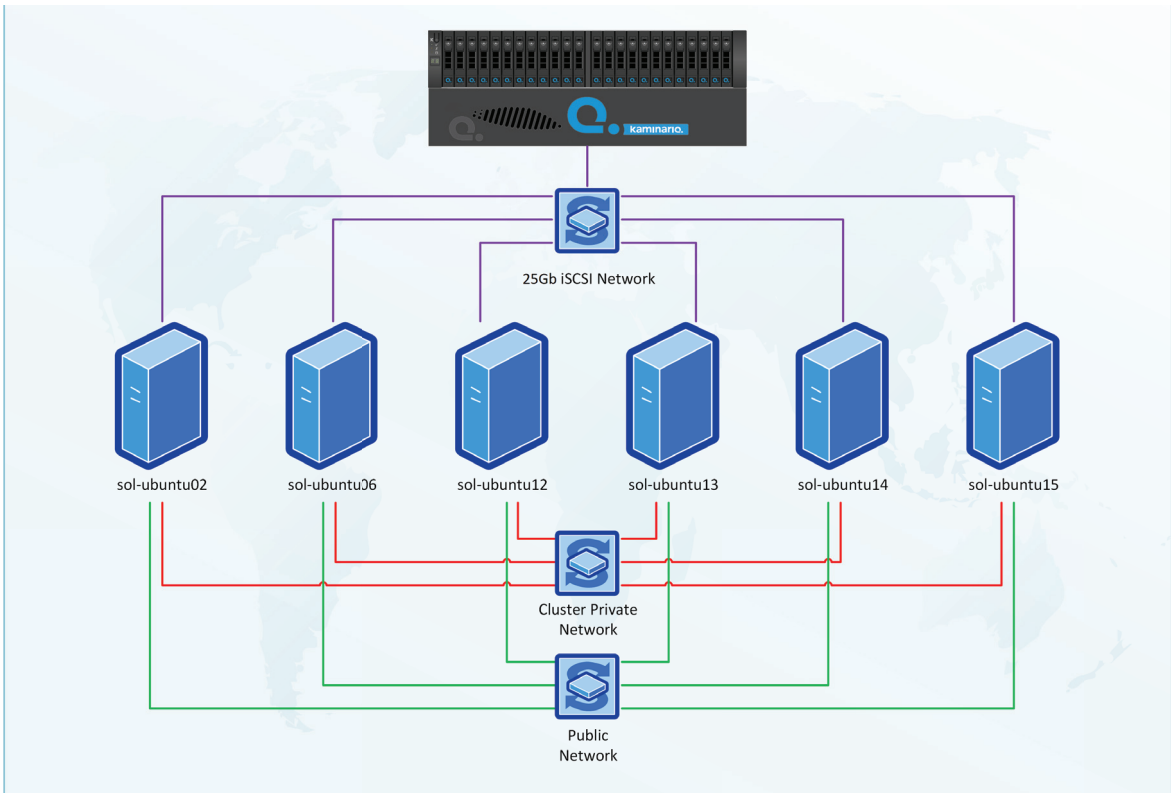

Figure 4: Solution's diagram

The Kaminario K2 system that was used in this environment is a single K-Block system with SSDs of 480GB – the entry point of the Kaminario K2.

Each of the six (6) physical nodes has the following spec:

- 2 x Intel Xeon E5649 CPU @ 2.53Ghz
- 96GB of RAM
- Ubuntu 14.04.05 LTS x64 (Trusty Tahr)

More information on deploying Ubuntu's OpenStack and installing the Kaminario K2 Cinder driver can be found at *Appendix A – Deploying Ubuntu OpenStack*.

K2 Preparations for OpenStack

Creating a Kaminario K2 user

It is highly recommended to create a dedicated user for the Cinder driver to access with to the K2. Login to the Kaminario K2 using the security user and add a user with the mgmt-app role:

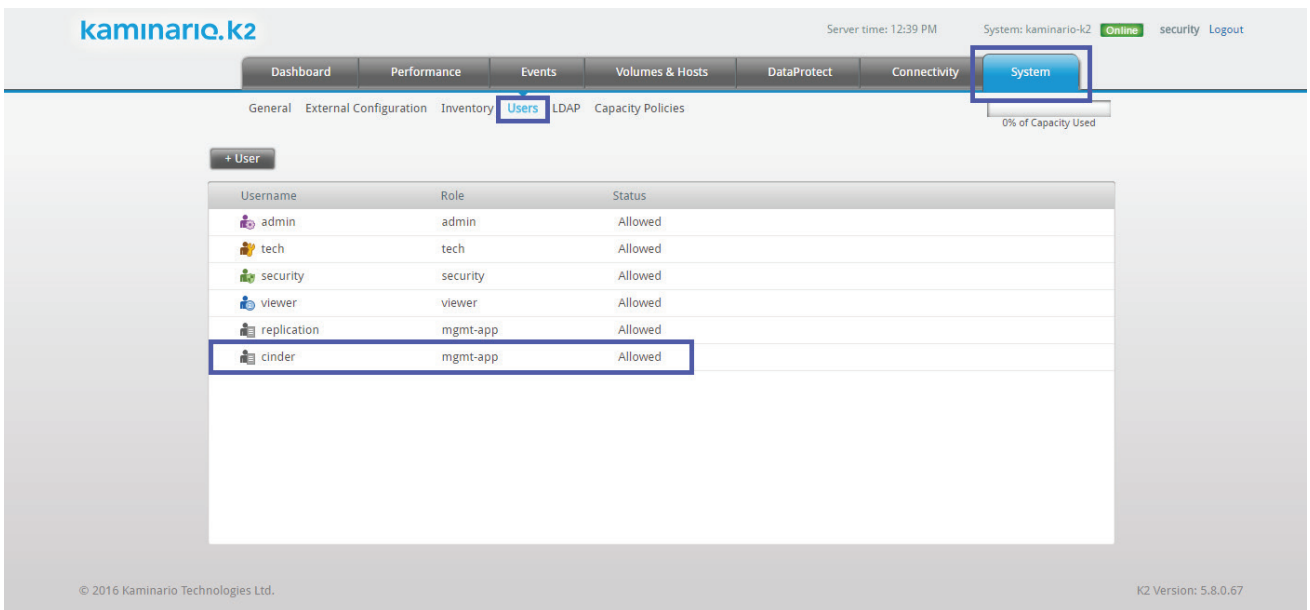

Figure 5: Creating a Cinder user in the K2

For further information on creating users in the K2 please see the *Kaminario K2 User Guide* document on the Kaminario Customer Portal.

Network configuration - Creating LAGs

Link aggregation, also referred to as port bonding, port teaming, Ethernet trunking, and link bundling, applies to methods of combining multiple Ethernet ports into a single logical port. When LAG is configured on K2, the two physical 25GbE ports in a K-Node act as a single logical interface.

Creating and using LAGs makes management simpler and improves high-availability.

To create a LAG, follow the next steps:

1. Login to the K2 web GUI and select the Connectivity tab.
2. Select the IP Data Ports option from the sub-menu.
3. Click the +LAG button to open the new LAG screen.
4. Set the MTU to 9000.
5. Set the mode to Adaptive Load Balancing.
6. Repeat steps 1-5 for the second LAG.

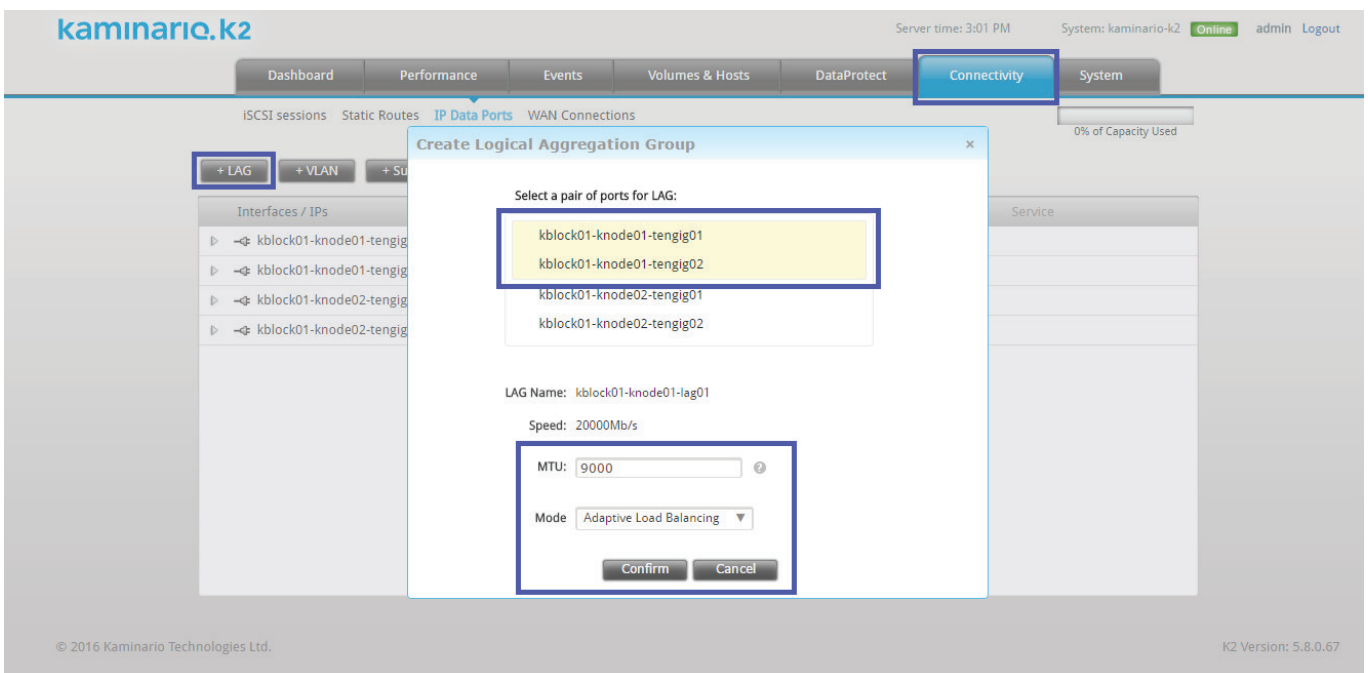

Figure 6: K2 LAG Configuration

Getting Ready for Deployment

Add Support for UNMAP (TRIM)

To support UNMAP (TRIM) in virtual guests, the Glance image from which the VMs are deployed should be configured with a SCSI Controller and a SCSI disk. To do so, use the following procedure:

1. Get the ID of the relevant Glance Image:
   ```
   ubuntu@sol-ubuntu15:~$ glance image-list
   ```
2. Set the Image's controller and disk bus to SCSI:
   ```
   ubuntu@sol-ubuntu15:~$ glance image-update --property hw_scsi_model=virtio-scsi
   --property hw_disk_bus=scsi <image_id>
   ```

Deploying the First Instance

First, we will start with a standalone MongoDB instance and insert some data to it.

Using the OpenStack Horizon GUI and the 'Launch Instance' wizard, we will create a new instance on a K2 volume, based on an Ubuntu 14.04 image.

The first step of the wizard asks us to give a name for the new instance. Although the current instance is a standalone, it will become a part of a replica-set, thus we will name the new instance as 'mongo-rs01-n01'.

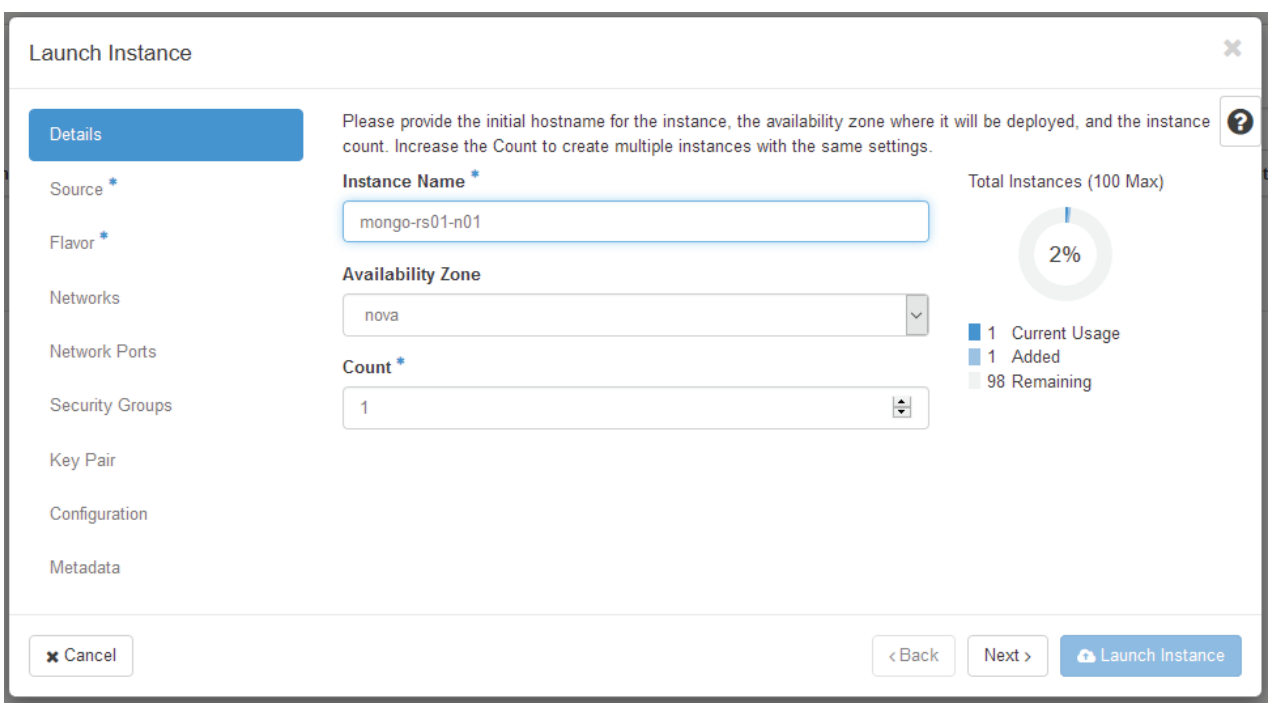

Figure 7: Launching a new instance in OpenStack

In the next screen of the wizard we can configure the new instance source for deployment. In this demonstration, we set the new instance with the following settings:

- Set the Boot Source to 'Image' so the new instance will be created from an Ubuntu 14.04 image.
- Set 'Create New volume' to a value of 'Yes' so the new instance would be deployed on a K2 external persistent volume.
- Set the K2 volume size to 100GB.
- Optionally, we can select to delete the K2 volume automatically once the instance is deleted.

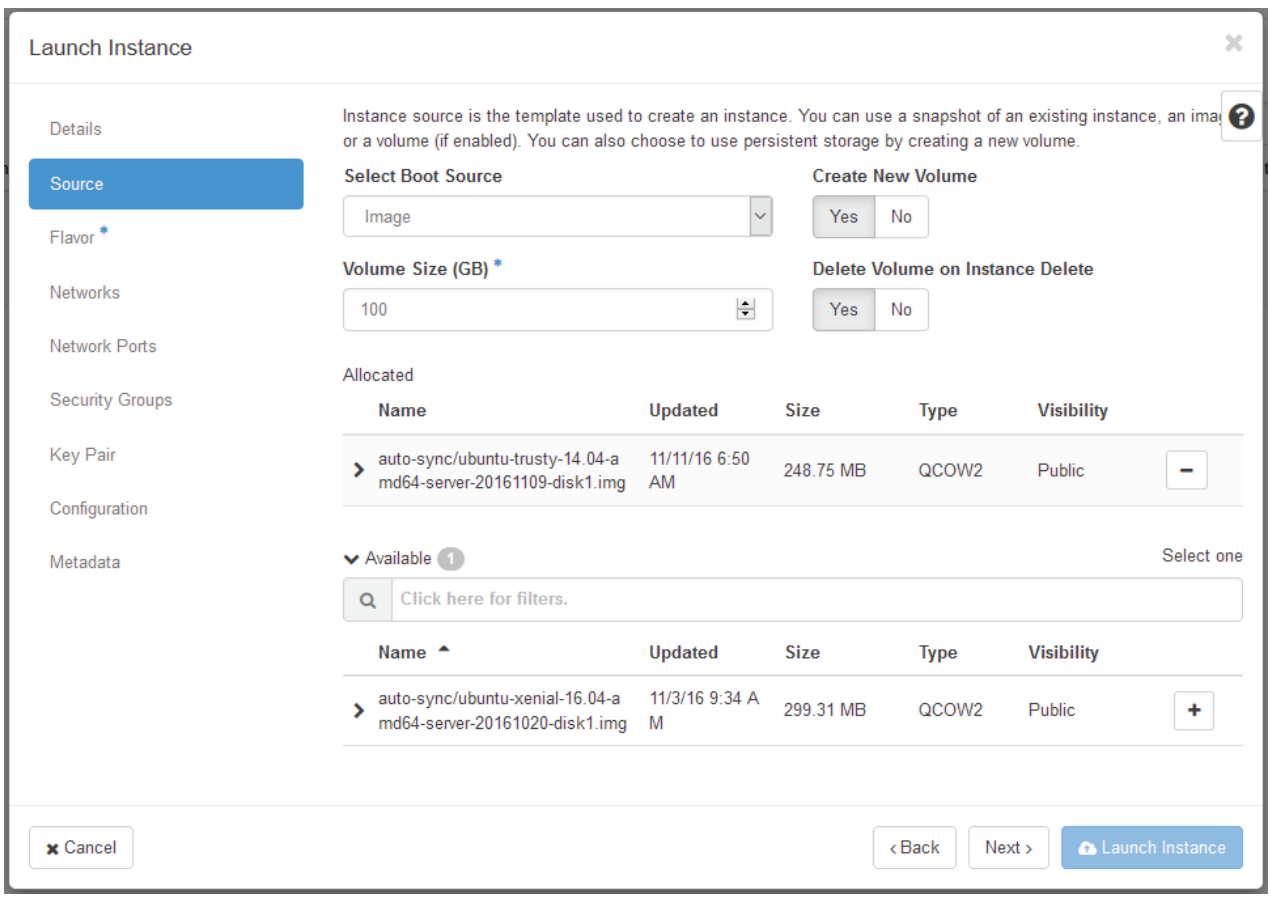

Figure 8: Setting the instance source in OpenStack

At the last mandatory step of the wizard we need to select the flavor of the new instance. A flavor defines the amount of resources available to the new instance. In this demonstration, we set all instances with the 'medium' flavor of 2 vCPUs and 4GB RAM.

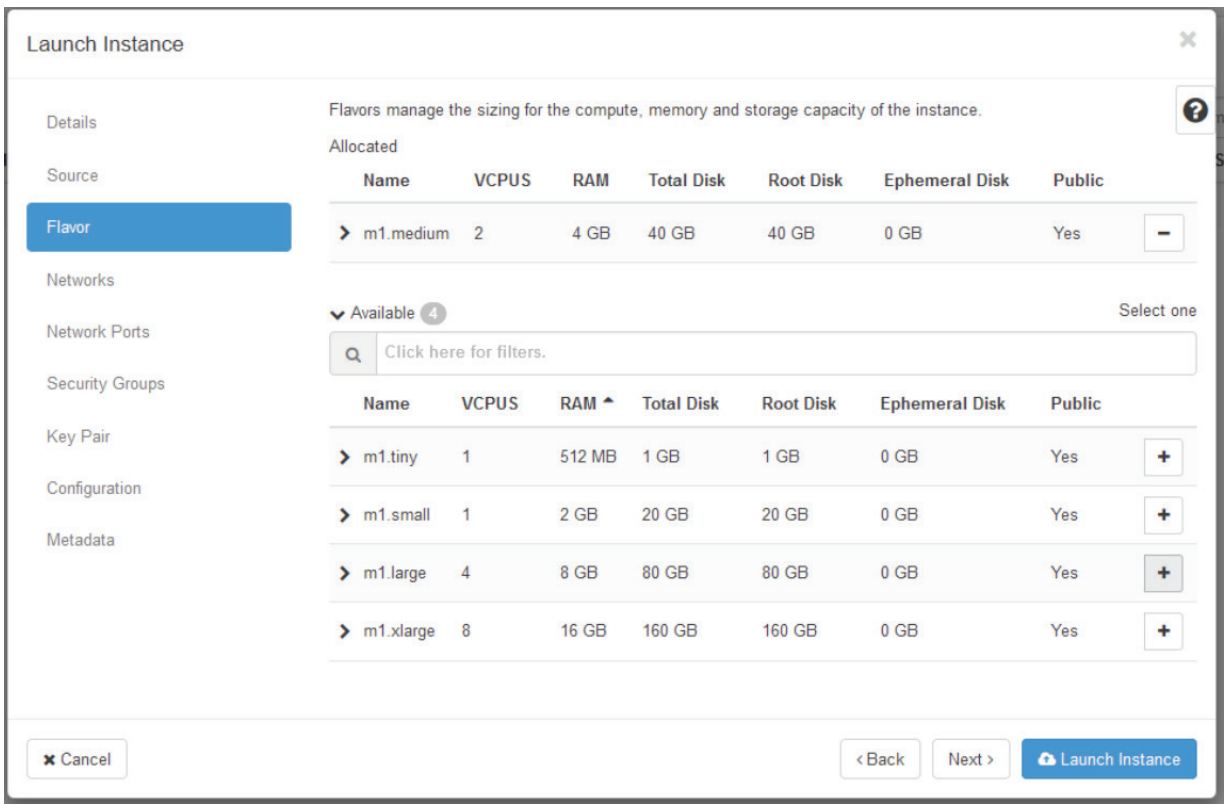

Figure 9: Setting the instance flavor in OpenStack

Once the wizard has completed, we can see that a new volume was created in the K2 and was mapped to host sol-ubutnu13 – the Cinder node in our environment. The new volume is mapped to the Cinder node to download the selected image (Ubuntu 14.04) to the K2 volume.

After the image was completely downloaded to the volume, the volume may be unmapped from the Cinder node and remapped to another node – the Compute node to which the instance was allocated to. The remapping process happens automatically if needed. In case the Cinder node is also the Compute node to which the instance was allocated to, this movement would not take place.

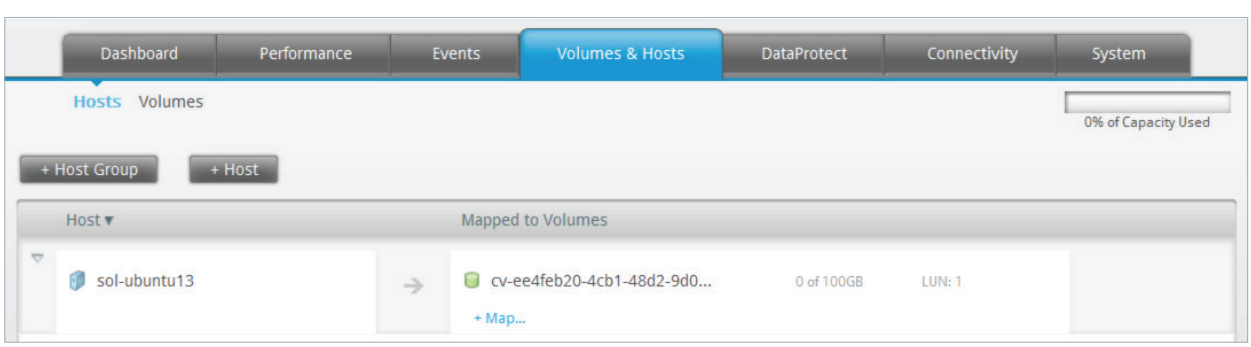

Figure 10: A K2 volume auto-created by Kaminario's Cinder Driver

The Kaminario K2 system which was used in this solution was totally empty with no data in it before creating the first instance. By creating the first and only instance, we have 2.2GB of data allocated (Used by the instance), but only 1.2GB were physically used by the K2 after compression and deduplication, thus saving 45% of capacity which equals to data reduction rate of 1.8:1 in this case.

Figure 11: The K2 dashboard shows the system's data-reduction rate with a single instance deployed

Installing a Standalone MongoDB Server

Once we have the first instance up-and-running we can allocate it a floating IP for external access. Using SSH, we connect to the floating IP and use the Key Pair we generated previously in advance to authenticate with the new Instance. Following the instructions from MongoDB website, we install MongoDB Enterprise edition for Ubuntu 14.04 as a standalone database.

This two-phase task of deploying an instance and then installing Enterprise MongoDB can be merged into a single task by creating a custom image. The custom image would include an operating system with MongoDB pre-installed, so an instance deployment would cover both phases.

Having the MongoDB Enterprise installed and ready for use, we make sure that the mongod service is stopped so we can change the MongoDB /etc/mongod.conf configuration file. Most settings are default with changes marked in ***bold and italics:***

```
ubuntu@mongo-rs01-n0-1:~$ cat /etc/mongod.conf
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /var/lib/mongodb
  journal:
    enabled: true
#  engine:
#  mmapv1:
  syncPeriodSecs: 1
  wiredTiger:
    collectionConfig:
      blockCompressor: none

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1,10.10.0.126

#processManagement:

#security:

#operationProfiling:

replication:
  replSetName: rs01

#sharding:

## Enterprise-Only Options:

#auditLog:

#snmp:
```

Configuration Details

The following section details the parameters configured in the suggested solution.

syncPeriodSecs: 1

Setting the `storage.syncPeriodSecs` parameter to 1 means that MongoDB would flush data to disk using the fsync command every second. With traditional storage arrays, setting this parameter to flush data every second can overload the storage system and may lead to disk bottleneck.

When using the Kaminario K2 as a storage layer, the rapid flushes are quickly-served so disk bottleneck is not an issue. Having the `storage.syncPeriodSecs` parameter set to 1 minimize the amount of dirty cache, allowing better utilization of RAM.

```
wiredTiger:
  collectionConfig:
    blockCompressor: none
```

Setting the `storage.wiredTiger.collectionConfig.blockCompressor parameter` to none disables the MongoDB built-in compression engine.

Disabling the MongoDB built-in compression allows the host to better utilize the CPU for database workload and not for compression workload. By that, we offload the compression workload to the Kaminario K2 which has an always-on inline compression.

bindIp: 127.0.0.1,10.10.0.126

Configures the IP address that *mongod* binds to in order to listen for connections. In our configuration, we added the internal IP address of the instance so it can communicate with other instances in the private network.

replSetName: rs01

The `replication.replSetName` parameter configures the name of the replica-set that the instance is part of. In the first part of the solution, we use MongoDB as a standalone instance although the replica-set is configured. In the second part of the solution we convert the standalone instance into a replica-set using MongoDB commands. Only at the second step the `replication.replSetName` parameter becomes active.

Inserting Data to MongoDB

To insert new data to MongoDB, a JavaScript code was written to generate random JSON documents to be inserted into a collection.

The inserted data is built as follows:

- A single database named 'myDB1'
- 50 Collections named 'myCol1' to 'myCol50'
- 15,000 documents per collection.
  - Average size of each document is ~16.5KB.
- Total of 750,000 documents in the database.
- Total size of dataset is about 12GB.

The JavaScript code was wrapped with a simple Bash loop to create all 50 collections in parallel as follows:

```
[root@sol-lx01 MongoDB]# cat insertData.sh
#!/bin/bash
MONGOHOST="172.16.16.64"
PORT=27017
DBNAME="myDB1"
NUMBER _ OF _ COLLECTIONS=50
JS _ FILENAME=createCollection.js

for (( i=1 ; i <= ${NUMBER _ OF _ COLLECTIONS} ; i++ )); do
    mongo $MONGOHOST:$PORT/$DBNAME --eval "arg1=${i}" $JS _ FILENAME & done
```

After having MongoDB installed and populated with data as described above, the K2 data reduction for that single instance has improved even better. We can see that 18.9GB were allocated and virtually used by the instance, but only 7.9GB of capacity were physically used by the K2 after compression and deduplication, thus saving 58% of capacity which equals to data reduction rate of 2.4:1 in this case.

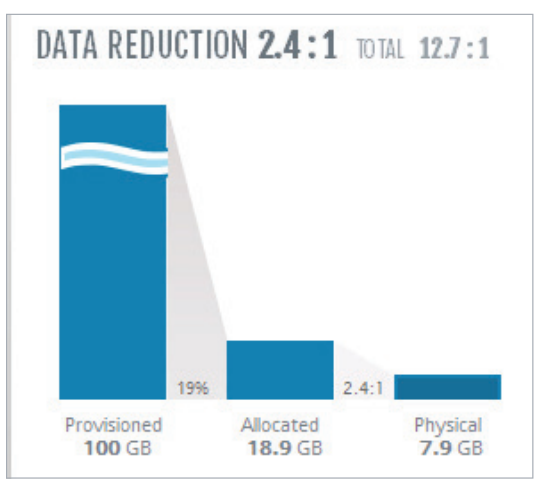

Figure 12: K2 data-reduction rate for a single instance with data populated

Create Indexes for all Collections

To maximize performance and optimize MongoDB, we will create an index based on the mNumber field in each collection. We will use the following command to create an index for each collection automatically:

```
ubuntu@mongo-rs01-n01:~$ for ((i=1;i<=50;i++)); do mongo myDB1 --eval 'db.myCol'${i}'.createIndex(
{ mNumber : 1 } )' ; done
```

Convert a Standalone to a Replica Set

Following MongoDB guidelines, we converted the single instance into a 3-node Replica-Set. A Replica-Set is a cluster of MongoDB servers with a master-slave replication and automated failover. A Replica-Set improves high-availability and redundancy of MongoDB, but requires a larger amount of storage capacity as data is replicated between the cluster nodes. In this case, we are creating a 3-node Replica-Set, thus total capacity needed has grown x3.

To overcome this capacity increase, the Kaminario K2 provides both compression and deduplication and reduces the needed physical capacity.

To deploy the two extra instances, we use the same OpenStack procedure we followed for the first instance. The new instances will automatically get new internal IP addresses so they can communicate with the rest of the MongoDB instances.

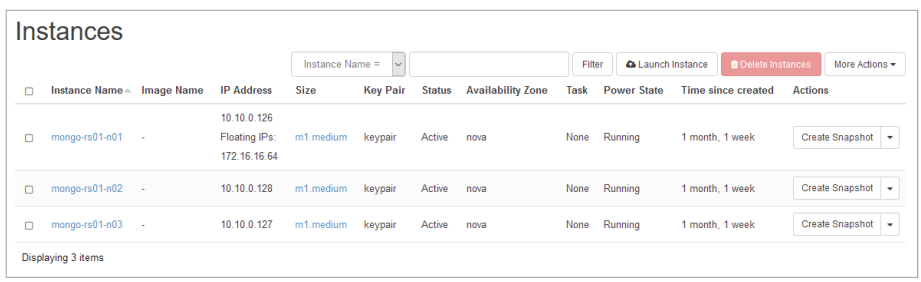

Figure 13: Running three OpenStack MongoDB instances

Once the new Replica-Set has completed syncing all nodes, we can see that 54.5GB were allocated and virtually used by the whole Replica-Set (three instances), but only 14.5GB of capacity were physically used by the K2 after compression and deduplication, thus saving 73% of capacity which equals to a data reduction rate of 3.8:1 in that case:

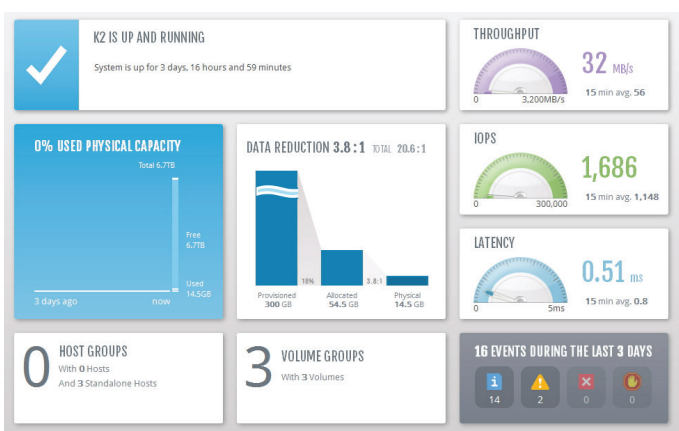

Figure 14: K2 data-reduction rate for three instances deployed with data populated

Convert a Replica-Set to a Sharded Cluster

Following the **instructions provided by MongoDB**, we convert the Replica-Set to a Sharded Cluster. Sharding is a method for distributing data across multiple servers. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

When using a Sharded Cluster, MongoDB distributes the read and write workload across the shards in the Sharded Cluster, allowing each shard to process a subset of cluster operations.

When sharding a collection, data is moving around from the original Replica-Set to the new Replica-Set added to the cluster. The K2, as a block-level storage array, is not aware that capacity in the old Replica-Set was freed up, so to update the K2 with that information, we should UNMAP (using the fstrim command line) the nodes in the original Replica-Set:

```
ubuntu@mongo-rs01-n01:~$ sudo fstrim / -v
/: 89550068121 bytes were trimmed
```

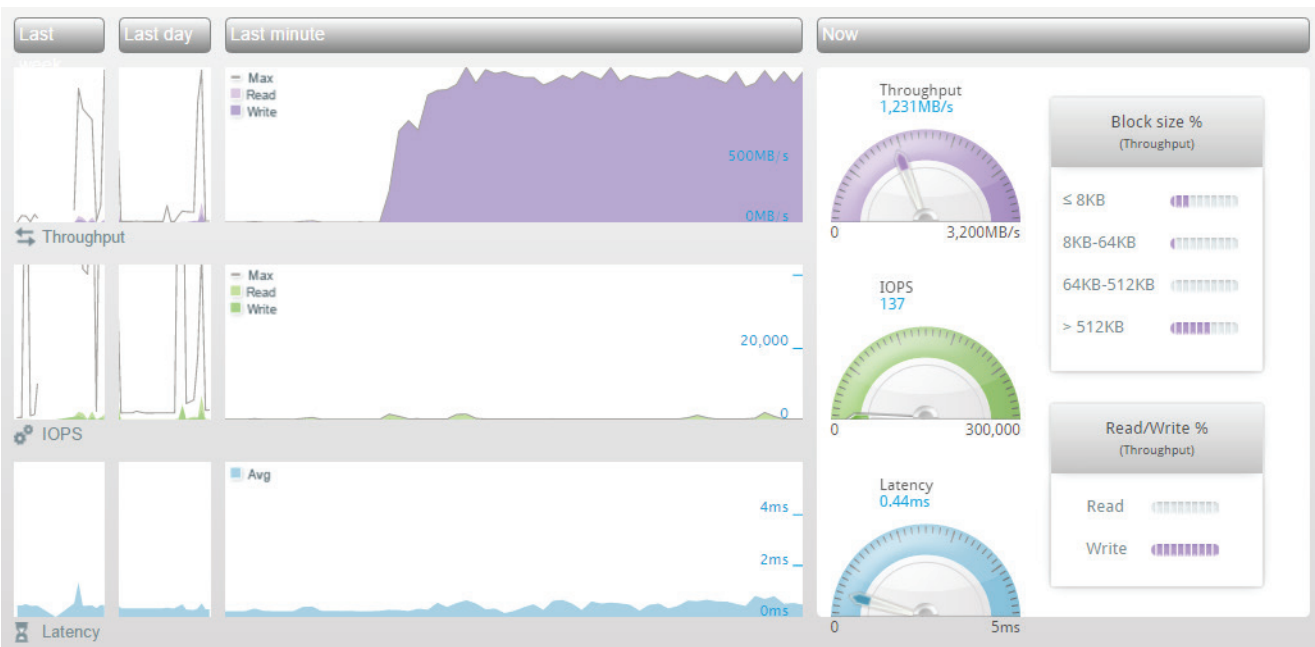

Figure 15: K2 GUI when running fstrim from a MongoDB instance in OpenStack

Benchmark Procedure

To benchmark MongoDB in the most realistic way we tried to simulate a real user operation – search for a specific data and update it with some new data. To do so, we created a Python script which uses the PyMongo library to work with MongoDB. To optimize the script and utilize all available resources, the script has support for multiprocessing and multithreading.

The random queries search for documents with a specific *mNumber* value and a random value for the 'state' key out of 57 possible values. Records that comply the filter are then updated with a random new value for the 'state' key out of the 57 possible values for that key.

To compare the different configurations, we separately benchmark the Standalone MongoDB, the Replica-Set and the Sharded Cluster. In each of the configurations we run 10 iterations of the benchmark and results are inspected from two points of view:

- MongoDB point of view – in this perspective, we inspect the number of operations per second, number of updated documents per second, and number of active connections. To monitor these metrics, we use the MongoDB Cloud Manager platform for easier and cleaner visuals.

- Kaminario K2 point of view - in this perspective, we inspect the amount of IOPS, Throughput and disk latency as provided by the Kaminario K2 using its GUI.

The two points of view can reveal some interesting information when combined. We inspect these results in the following section – Benchmark Results.

Benchmark Results

Standalone MongoDB Results

In a standalone MongoDB, we have a single node running the database with no replication mechanism or any other overheads. The standalone benchmark can be used as a baseline for the different configurations.

MongoDB Results

When looking at the operation counters chart, we see that the standalone MongoDB reached an average of about 190 commands per second and an average of about 90 update operations per second. These metrics indicate the number of MongoDB database operations, and not the number of updated documents per second as can be seen in the second chart below.

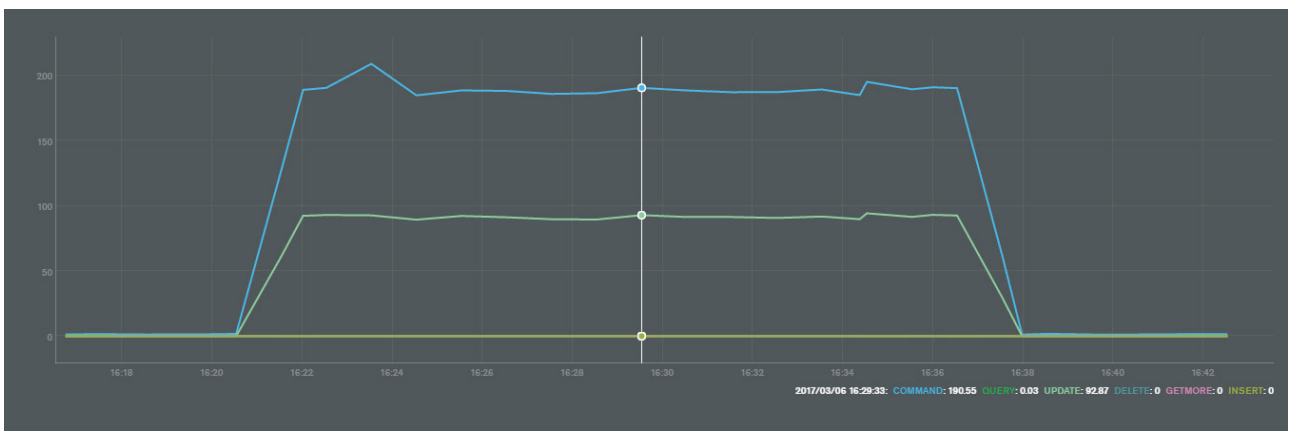

Figure 16: Standalone MongoDB Operation Counters

Next, we inspect the number of updated documents per second, and as can be seen in the next chart, the standalone MongoDB updates about 4,400 documents per second in average.

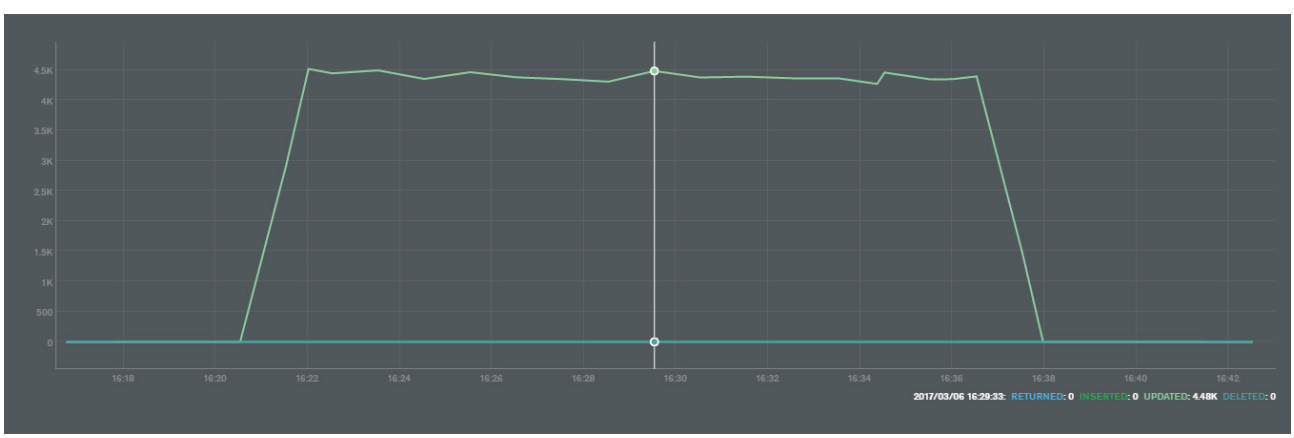

Figure 17: Standalone MongoDB Documents Metrics

Kaminario K2 Results

The different MongoDB counters inspected above gives us some indication about the pace of updates and MongoDB operations per second, but how does this translates into storage performance? The next chart show us that a standalone MongoDB can generate an impressive workload in the K2 with a consistent latency of 0.45 milliseconds throughout the whole 10 iterations of the benchmark.

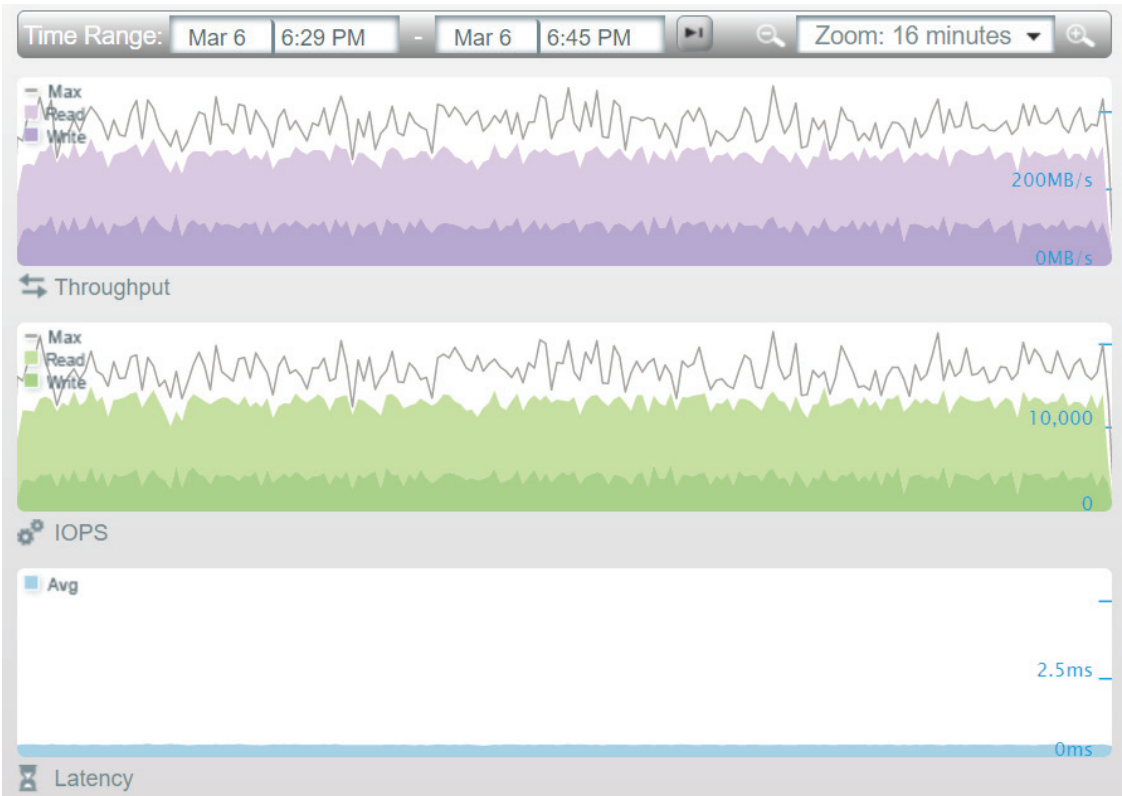

Figure 18: Standalone MongoDB K2 Results

The data in the chart above is presented in the following table:

Parameter	Average	Maximum
IOPS	16,739	21,304
Throughput (MB/s)	365	462
Latency (milliseconds)	0.44	0.47

Replica-Set MongoDB Results

In a Replica-Set, new data coming into the database is replicated between the different nodes in the Replica-Set to achieve high-availability and redundancy – the main goal of Replica-Sets. The Replica-Set replication mechanism indeed improves high-availability, but is also a source for overhead to the normal operation of the database, so it is important to keep in mind that it may impact on the database performance.

MongoDB Results

For a Replica-Set, the MongoDB charts are divided to the primary node and the two secondary nodes. The primary node receives all traffic from clients, so the number of updated documents per second is reflected only in the primary node charts.

The next chart shows that the number of update operation per second has decreased to 76, but the number of commands per second has grown to above 300, due the MongoDB Replica-Set replication overhead.

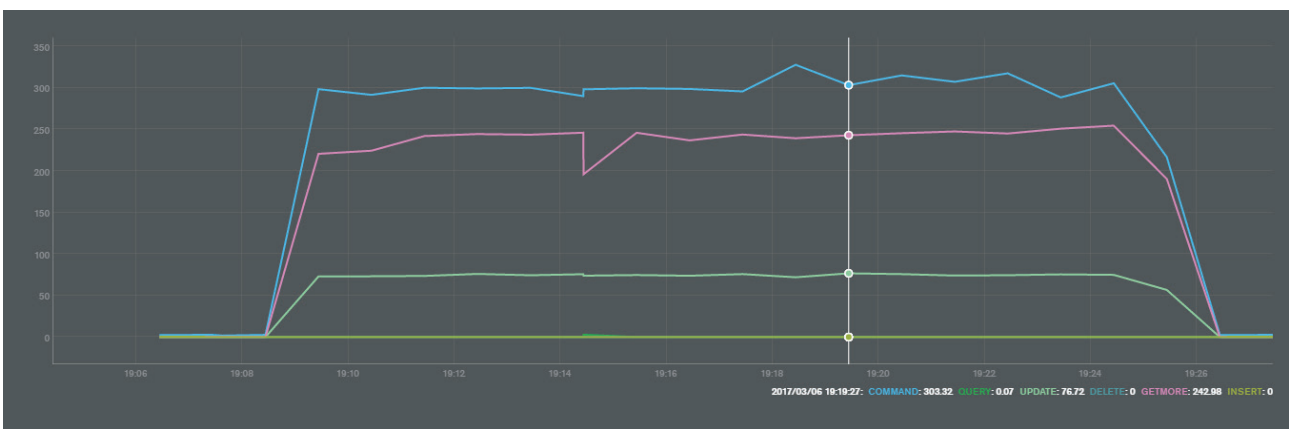

Figure 19: Replica-Set, Primary node, Operation Counters

Similarly, we can see that the number of updated documents per second has decreased to 3,400 documents per second.

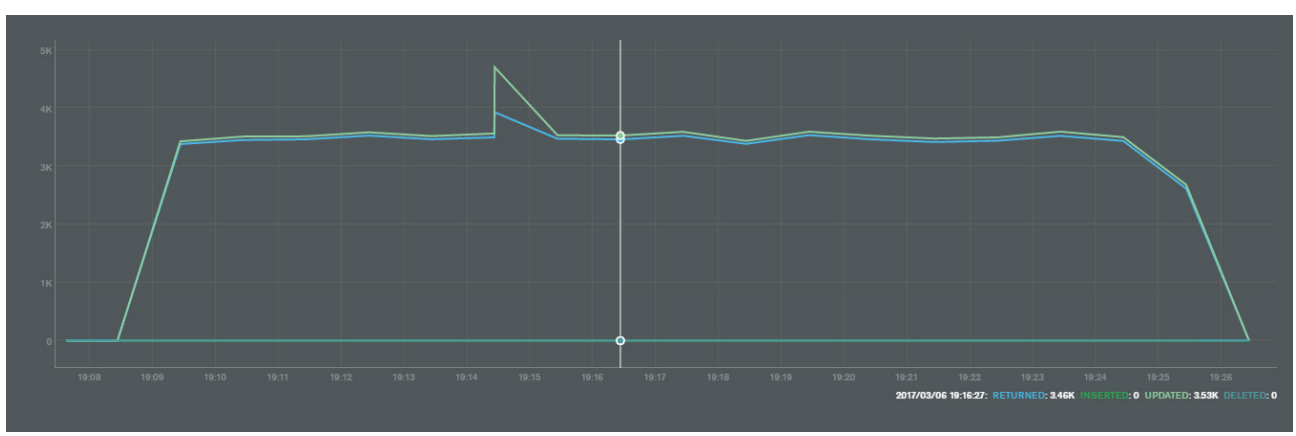

Figure 20: Replica-Set, Primary node, Documents Metrics

When looking at the secondary nodes in a Replica-Set, the main thing to note is the replication operation counters as the next two charts demonstrate. In our case, and as the charts below shows, the majority (if not only) of the replication operations are update operations, just like our benchmark.

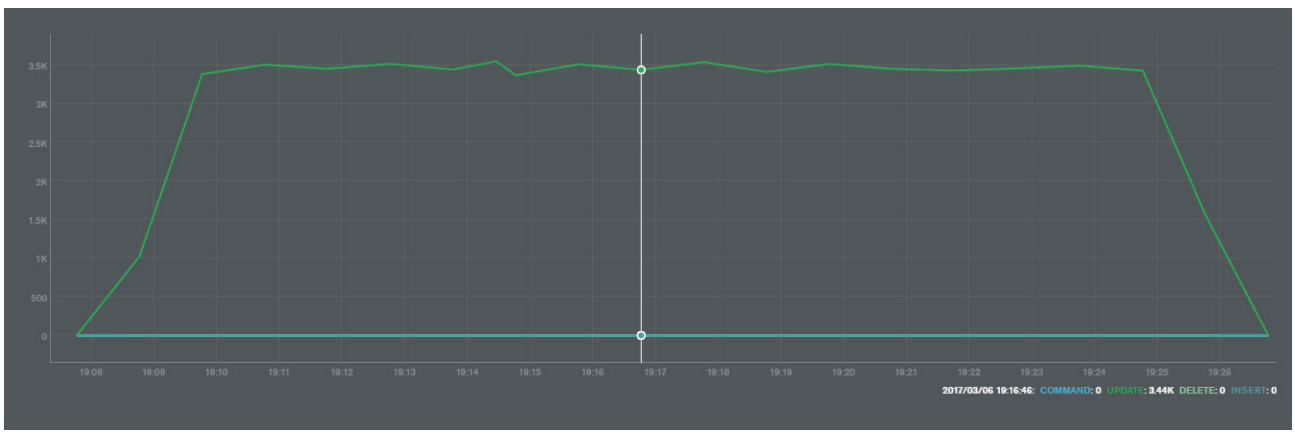

Figure 21: Replica-Set, Secondary node #1, Replication Operations Counters

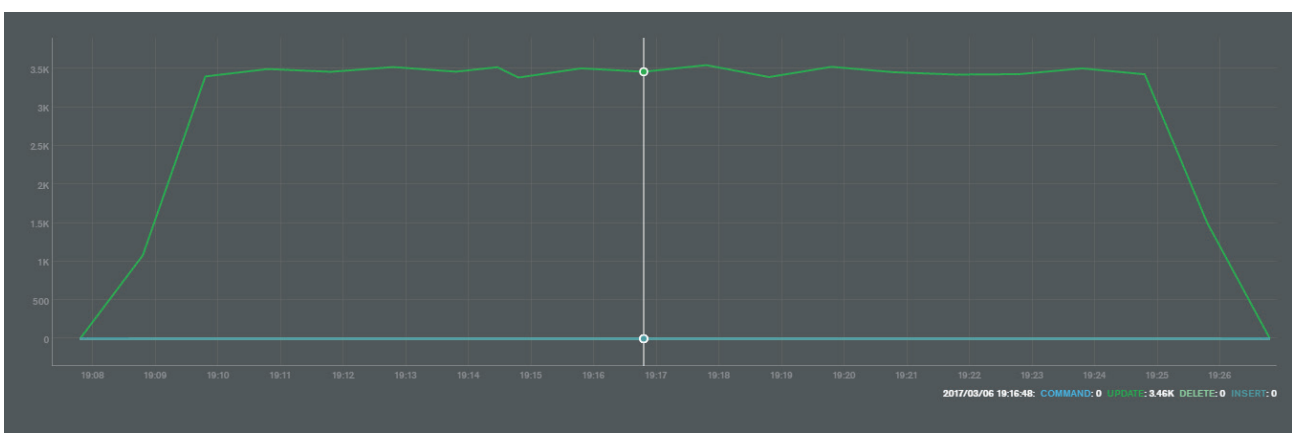

Figure 22: Replica-Set, Secondary node #2, Replication Operations Counters

Kaminario K2 Results

Although a Replica-Set is not intended to improve the overall performance of a database, we can easily see that the workload at the storage level has increased compared to the standalone storage performance.

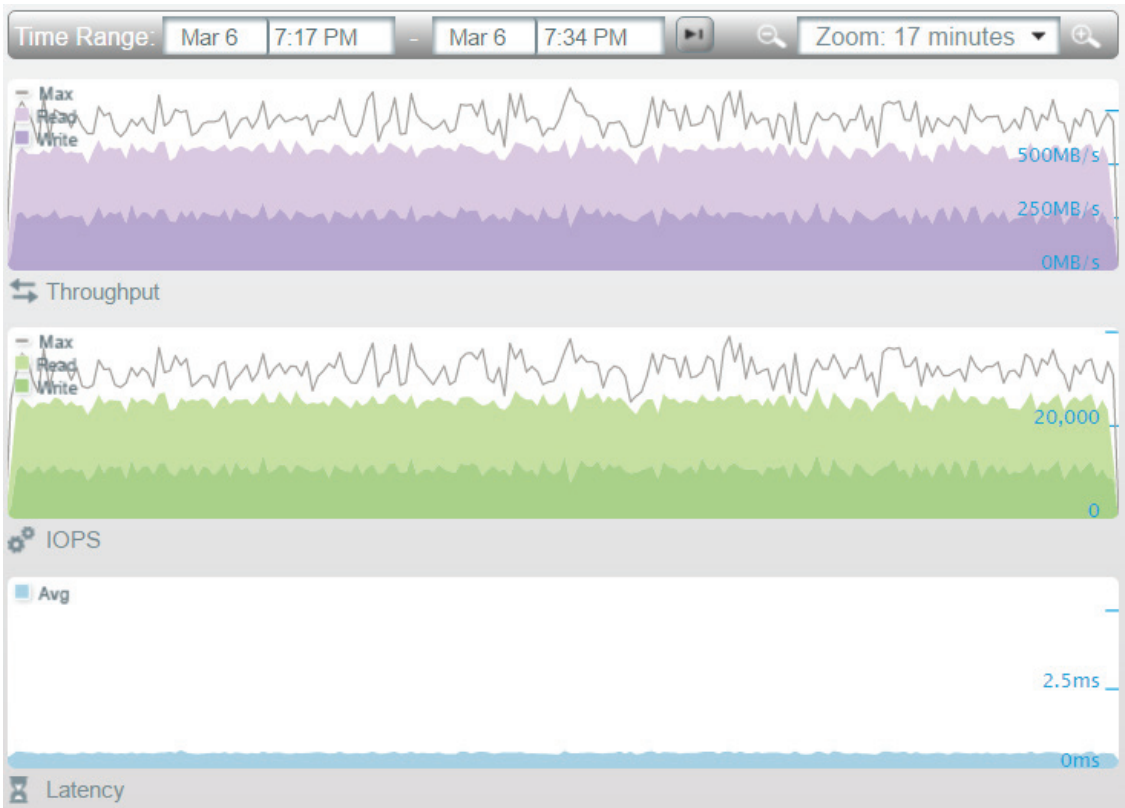

Figure 23: Replica-Set MongoDB K2 Results

The data in the chart above is presented in the following table:

Parameter	Average	Maximum
IOPS	31,792	38,912
Throughput (MB/s)	703	851
Latency (milliseconds)	0.54	0.63

Sharded MongoDB Results

As described earlier in this document, the goal of a Sharded Cluser is to improve the performance, especially throughput, of a Replica-Set while preserving high-availability and redundancy. The results below show that a Sharded Cluster indeed improves throughput over a Replica-Set.

MongoDB Results

Like in the Replica-Set results, once again we inspect each server with its relevant metrics.

Starting with the first Replica-Set in the Sharded Cluster, we look at the primary node's operation counters and documents operations. The average amount of update operations is 46, much lower than what we have seen so far – but keep in mind that this is only part of the overall workload as it is spread across the cluster's nodes.

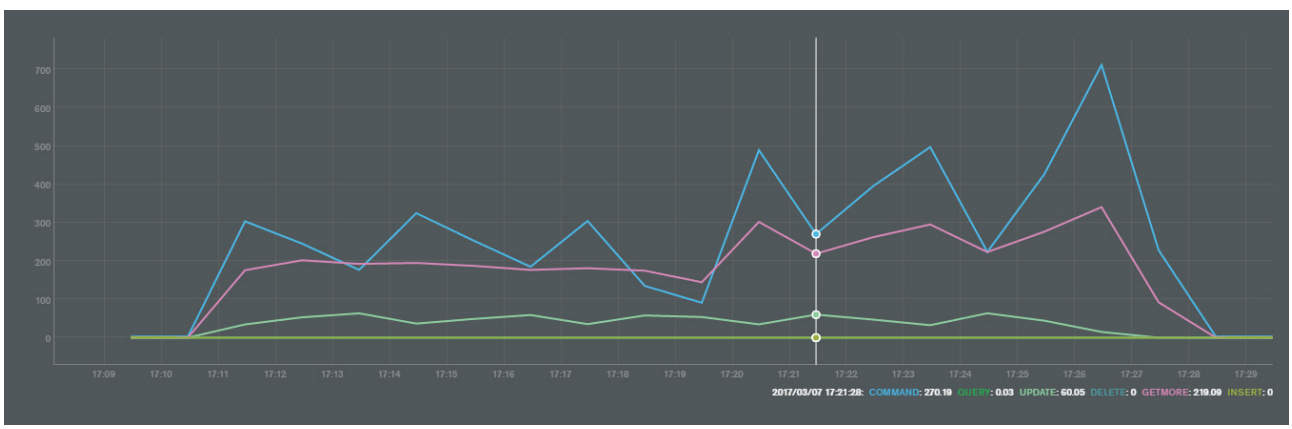

Figure 24: Sharded Cluster, Replica-Set #1, Primary node, Operation Counters

Looking at the documents metrics, we can see an average of 2,150 documents updated per second. Again, this value is lower than the Standalone and Replica-Set configuration, but the same applies here – it is only part of the overall workload.

Figure 25: Sharded Cluster, Replica-Set #1, Primary node, Documents Metrics

Next, we inspect the first Replica-Set's secondary nodes. Unlike the Replica-Set benchmark where all queries were served by the primary node, we can see that secondary node #1 returned an average of 2,200 documents per second, meaning it was used for read operations along being a replication target.

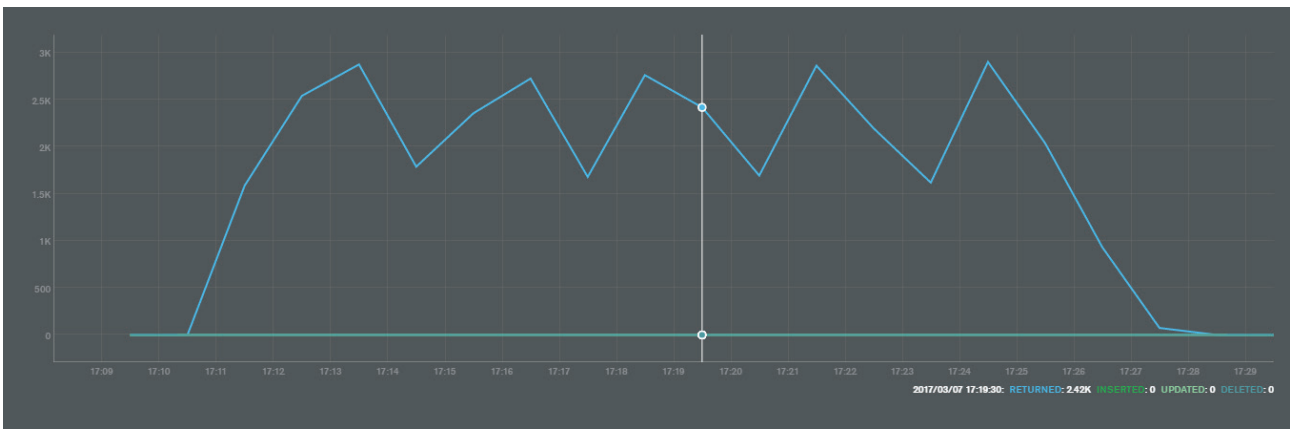

Figure 26: Sharded Cluster, Replica-Set #1, Secondary node #1, Documents Metrics

Like in the Replica-Set benchmark, we can see that the majority of the replication operations are update operations.

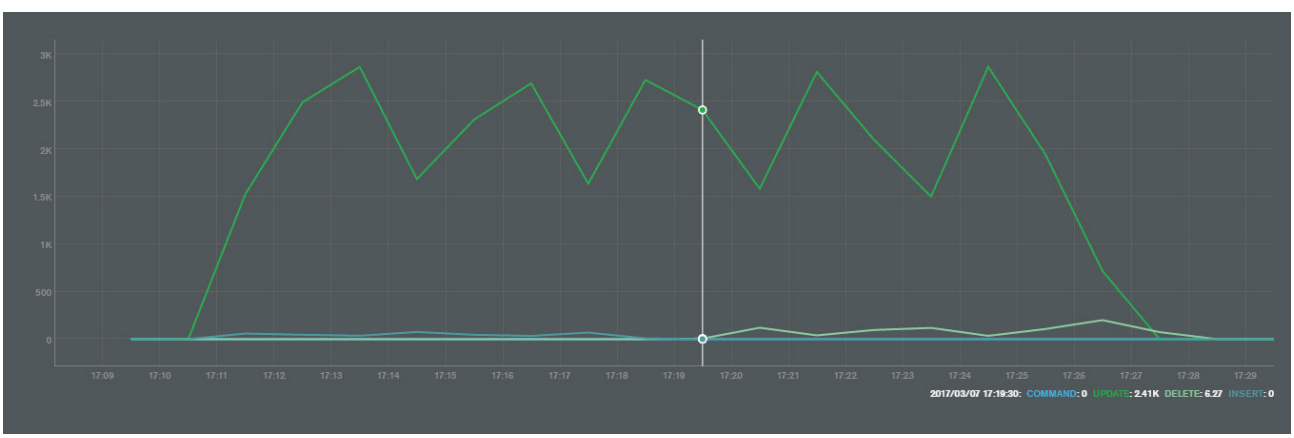

Figure 27: Sharded Cluster, Replica-Set #1, Secondary node #1, Replication Operations Counters

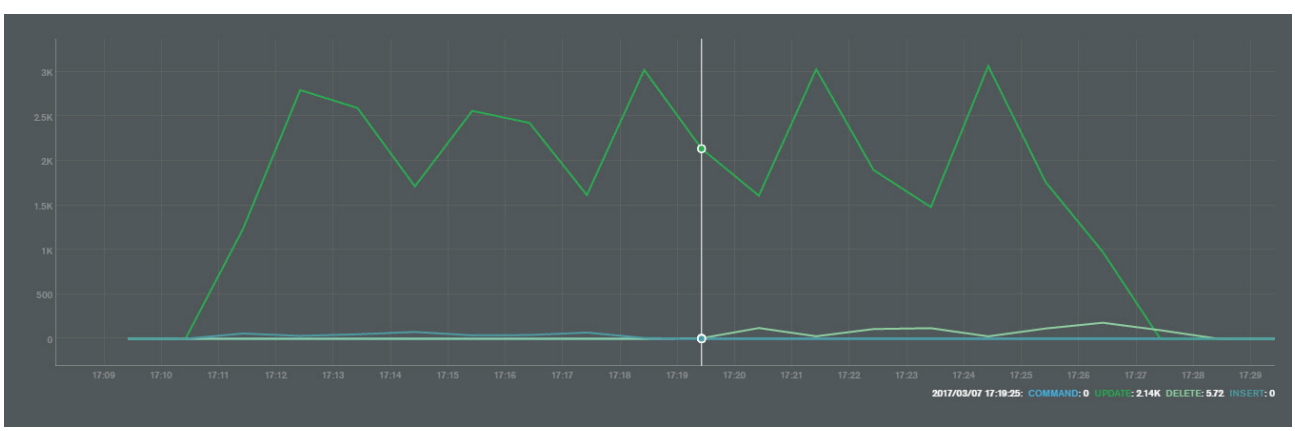

Figure 28: Sharded Cluster, Replica-Set #1, Secondary node #2, Replication Operations Counters

The following are similar charts of the second Replica-Set in the Sharded Cluster.

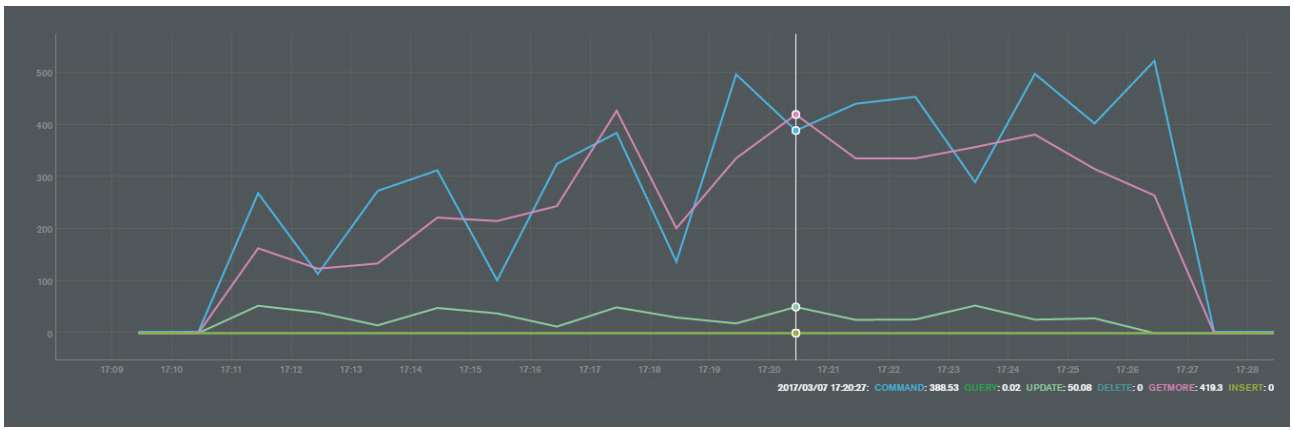

Figure 29: Sharded Cluster, Replica-Set #2, Primary node, Operation Counters

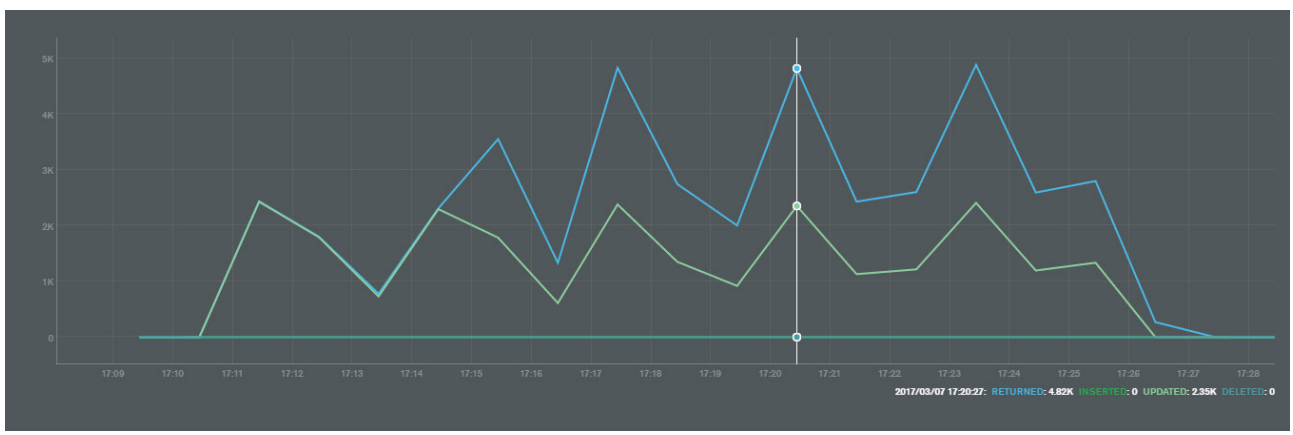

Figure 30: Sharded Cluster, Replica-Set #2, Primary node, Documents Metrics

Figure 31: Sharded Cluster, Replica-Set #2, Secondary node #1, Replication Operations Counters

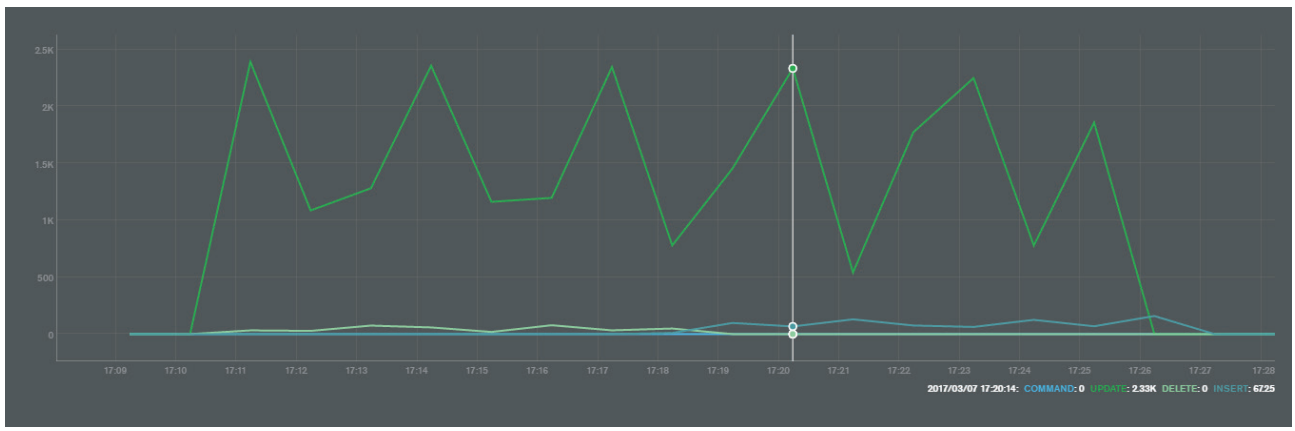

Figure 32: Sharded Cluster, Replica-Set #2, Secondary node #2, Replication Operations Counters

Summing up the two Replica-Sets in the Sharded Cluster, we achieved an average of 81 update operations per second and an average of 3,750 documents updates per second.

Kaminario K2 Results

As expected, we can see higher throughput for the Sharded Cluster with a consistent sub-1ms latency.

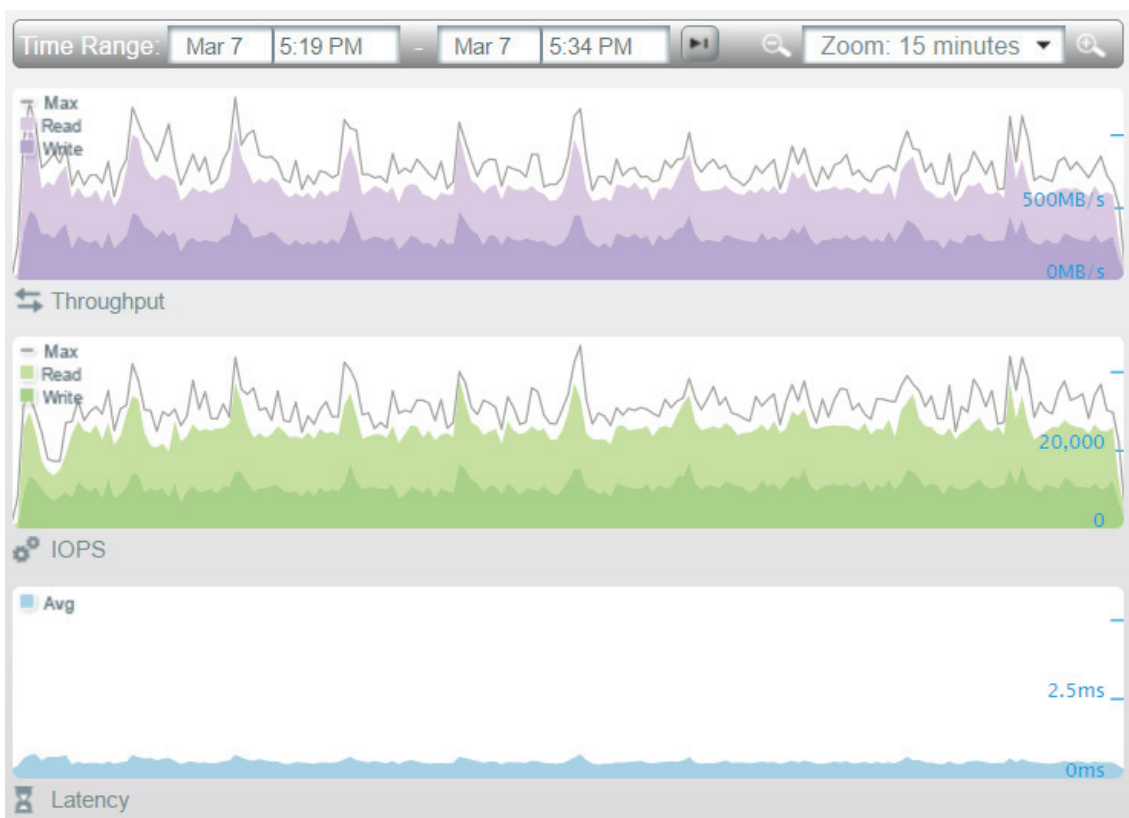

Figure 33: Sharded Cluster MongoDB K2 Results

The data in the chart above is presented in the following table:

Parameter	Average	Maximum
IOPS	31,534	46,731
Throughput (MB/s)	799	1,260
Latency (milliseconds)	0.60	0.92

Benchmark Conclusions

The following table compares the averages of the relevant metrics, both of K2 and MongoDB.

	Metric	Standalone		Replica-Set		Sharded Cluster	
MongoDB	Update Operations	90		76		81	
	Document Operations	4,400		3,400		3,750	
		Average	**Maximum**	**Average**	**Maximum**	**Average**	**Maximum**
Kaminario K2	IOPS	16,739	21,304	31,792	38,912	31,534	46,731
	Throughput	365	462	703	851	799	1,260
	Latency	0.44	0.47	0.54	0.63	0.60	0.92

From the table above, it seems like the standalone has the best performance from all configurations tested, probably because the Standalone configuration has no overhead at all (no replication to secondary nodes). On the other hand – such a configuration has no redundancy whatsoever – a failure means the whole database is unavailable.

The number of updated documents per second in our tests was in the range of 3,400 – 4,400. It is important to notice that some other public benchmarks may reach up to 80,000 operations per second. This gap is a derivative of how Kaminario conducted its benchmark:

1. Test a real-life scenario of large documents with an average size of 16.5KB per document, where other benchmarks might choose to test relatively small documents, as small as 100 bytes, to portray hero numbers that cannot be reached in a real environment.
2. Use a small footprint of RAM within the server to "force" the database to access the storage layer instead of using the RAM for most operations. After all, this benchmark is focused on the storage layer and not on MongoDB performance or fine tuning.

All in all, we can see that the Sharded Cluster indeed reached a very high throughput (above 1.2GB/s) as it is intended to. Also, we assume that fine tuning to the data structure of the documents in our database and optimizing the benchmark script could possibly improve the performance even further.

OpenStack, Kaminario K2 and MongoDB are all products that are designed for scalability at their base. Both OpenStack and MongoDB require a storage infrastructure layer that can meet their flexibility and agility to scale, however this storage layer must also provide the data services and cost benefits of shared storage.

K2 Cinder Driver Installation and Configuration

OpenStack Supported Features

The Kaminario K2 Cinder driver support the following features:

Volumes

- Create and delete volumes
- Attach/Detach volumes to/from instances
- Clone a volume
- Extend a volume
- Manage/Unmanage non-OpenStack volumes
- Volume retype

Replication

- Native K2 replication support

Snapshots

- Create and delete snapshots
- Create a volume from a snapshot

Images

- Copy an image to a volume
- Copy a volume to an image

Installing the K2 Cinder Driver

The Kaminario K2 Cinder driver is pre-packaged with OpenStack starting with the Newton release.

For Mitaka and Liberty Ubuntu based OpenStack releases, a Kaminario Cinder Juju Charm can be self-deployed as described in the next section.

For non-Ubuntu based Mitaka and Liberty OpenStack environments please contact Kaminario Support team for assistance with the Kaminario Cinder Driver installation.

Kaminario Cinder Juju Charm

The Kaminario Cinder Juju Charm can be deployed as detailed in the following link.

Installing in Ubuntu OpenStack

To deploy the Kaminario Cinder Charm you must be connected to the correct Juju environment which contains the OpenStack components like Keystone, Nova and Cinder.

This environment is auto-installed by Landscape during the deployment process.

From the host where the command openstack-install was issued (MAAS Server), issue the following command:

Set JUJU_HOME variable:

```
export JUJU_HOME=~/.cloud-install/juju
```

Make sure you're using the maas environment

```
ubuntu@sol-ubuntu15:~$ juju env
maas
```

Connect to the landscape machine

```
ubuntu@sol-ubuntu15:~$ juju ssh landscape-server/0
```

sudo to root

```
ubuntu@juju-machine-0-lxc-1:~$ sudo su -
```

Set the JUJU_HOME variable

```
root@juju-machine-0-lxc-1:~# export JUJU_HOME=/var/lib/landscape/juju-homes/$(ls -tr /var/lib/
landscape/juju-homes/ | tail -1)
```

Deploy the Kaminario Cinder Charme

```
root@juju-machine-0-lxc-1:~# juju deploy cs:~hareeshk/kaminario-cinder-3
Added charm "cs:~hareeshk/trusty/kaminario-cinder-3" to the environment.
```

Set the Charme settings. 'backend_name' will be used later:

```
root@juju-machine-0-lxc-1:~# juju service set kaminario-cinder san_user="cinder" san_
password="cinder" san_ip="192.168.5.22" backend_name="kaminario-iscsi" protocol="iscsi"
```

Add relations to cinder and nova-compute

```
root@juju-machine-0-lxc-1:~# juju add-relation kaminario-cinder cinder
root@juju-machine-0-lxc-1:~# juju add-relation kaminario-cinder nova-compute-kvm
```

Optional
Get the available configuration settings for the Kaminario Cinder charm:

```
root@juju-machine-0-lxc-1:~# juju service get kaminario-cinder
```

Optional
How to set a parameter:

```
root@juju-machine-0-lxc-1:~# juju service set kaminario-cinder is_multipath_enabled=true
```

Upgrading the Juju Charm

```
root@juju-machine-0-lxc-1:~# juju upgrade-charm kaminario-cinder
```

Configuring the K2 Volume Types

OpenStack features a mechanism called Volume Types which assists an administrator to categorize the available volumes for users. For example, in the case of the Kaminario K2, we would like to set a volume type of Compression and Deduplication enabled, and a second volume type of Compression only with no deduplication. Using this feature allows an administrator to create different tiers of storage services for better management and resource allocation.

Using the Admin OpenStack RC file, create a cinder volume type for Kaminario volumes with compression and deduplication enabled. At this step, we create a new volume type without setting its configuration, we will set the volume type settings in the next step.

```
ubuntu@sol-ubuntu13:~$ cinder type-create kaminario-iscsi-d
```

ID	Name	Description	Is_Public
898d947a-b27b-4b26-b3e6-05fda7f78b	kaminario-iscsi-d	–	True

Next, we configure the volume type settings. To do so, set the 'volume_backend_name' parameter to the value of 'backend_name' from the previous section (Kaminario Cinder Juju Charm) for the newly created volume type (the 'backend_name' is a parameter of Kaminario's Cinder Driver). If you are not sure what the value of 'backend_name' is, you can verify it using the following command:

```
root@juju-machine-0-lxc-1:~# juju service get kaminario-cinder | grep backend_name -A3
  backend_name:
    description: Kaminario Storage backend name
    type: string
    value: kaminario-iscsi
```

Alternatively, you can look at the cinder configuration file at the Cinder host:

```
ubuntu@sol-ubuntu13:~$ sudo cat /etc/cinder/cinder.conf | grep backend_name
volume_backend_name = kaminario-iscsi
```

Once the backend_name is known, use it as a value for the 'volume_backend_name' parameter in the following command:

```
ubuntu@sol-ubuntu13:~$ cinder type-key kaminario-iscsi-d set volume_backend_name=kaminario-iscsi
```

Next, we will create a second volume-type for non-dedupe volumes:

```
ubuntu@sol-ubuntu13:~$ cinder type-create kaminario-iscsi-nd
```

ID	Name	Description	Is_Public
a26aed32-d358-45aa-83c0-cb8d2b455dc4	kaminario-iscsi-nd	–	True

Set the backend_name the same as before. This time, there's an extra parameter for disabling deduplication for the non-dedup volume-type:

```
ubuntu@sol-ubuntu13:~$ cinder type-key kaminario-iscsi-nd set volume_backend_name=kaminario-
iscsi kaminario:thin_prov_type=nodedup
```

Appendix A - Deploying Ubuntu OpenStack

OpenStack deployment can be a bit complicated, depending on type of deployment. Since our focus is the Kaminario K2 integration with OpenStack, we chose to use Ubuntu OpenStack as its installation process is relatively easier than other methods

Ubuntu OpenStack Deployment

The following describes the steps to deploy an Ubuntu OpenStack environment in a very high-level. For more detailed installation steps visit Ubuntu's website.

Installing MAAS

MAAS – Metal-as-a-service. A solution provided by Ubuntu to manage and control physical servers to automate operating system installation and system settings configuration.

MAAS can be managed using GUI, CLI and RESTful. We will use the GUI and CLI in this example.

1. Install Ubuntu 14.04 LTS on a physical node. That node would eventually be the MAAS server in your environment. MAAS (Metal-as-a-service) is a platform for managing physical servers in an easy and automated way.
2. SSH into the newly installed Ubuntu server and install basic requirements and repositories:
   sudo apt-get install python-software-properties

   ```
   sudo add-apt-repository ppa:juju/stable
   sudo add-apt-repository ppa:maas/stable
   sudo add-apt-repository ppa:cloud-installer/stable
   sudo apt-get update
   ```

3. Install MAAS:
   ```
   sudo apt-get install maas
   ```
4. Once MAAS is installed an admin account has to be created:
   ```
   ssudo maas-region-admin createadmin
   ```
5. Configure IPtables masquerade
   ```
   sudo iptables -t nat -A POSTROUTING -s 192.168.2.0/24 ! -d 192.168.2.0/24 -j MASQUERADE
   ```
6. Run the following commands and make sure the correct IP addresses are listed:

   ```
   sudo dpkg-reconfigure maas-cluster-controller
   sudo dpkg-reconfigure maas-region-controller
   ```

7. Login to the MAAS server web GUI at http://<MAAS-IP-ADDRESS>/MAAS. with the credentials created before at step 4 above.

MAAS Configuration

1. From the MAAS web GUI, select the Images tab and import the Ubuntu 14.04 LTS and Ubuntu 16.04 LTS images.

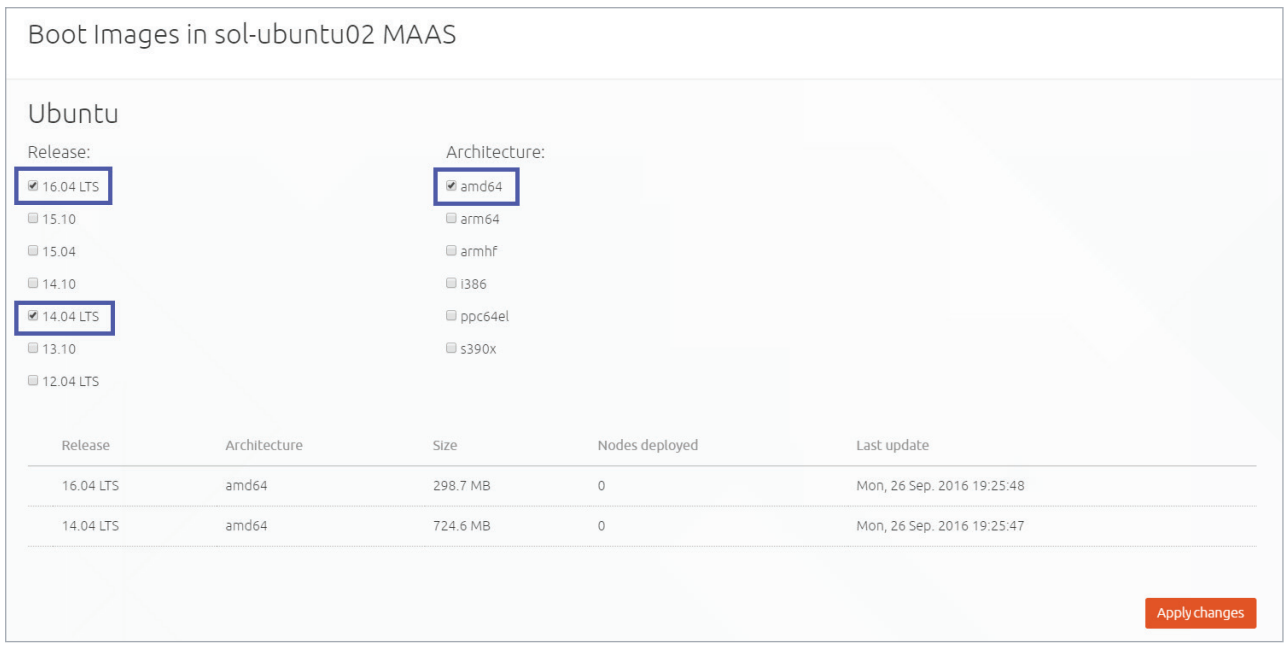

Figure 34: Adding Boot Images to MAAS

2. From the MAAS web GUI, select the Subnets tab and configure all networks with required CIDR, default gateway and DHCP addresses allocation for the DHCP and PXE interface.

3. Boot nodes in PXE mode and wait for them to enlist in the MAAS web GUI. Once all nodes are listed in MAAS web GUI, you can edit their randomly generated names into some more meaningful names:

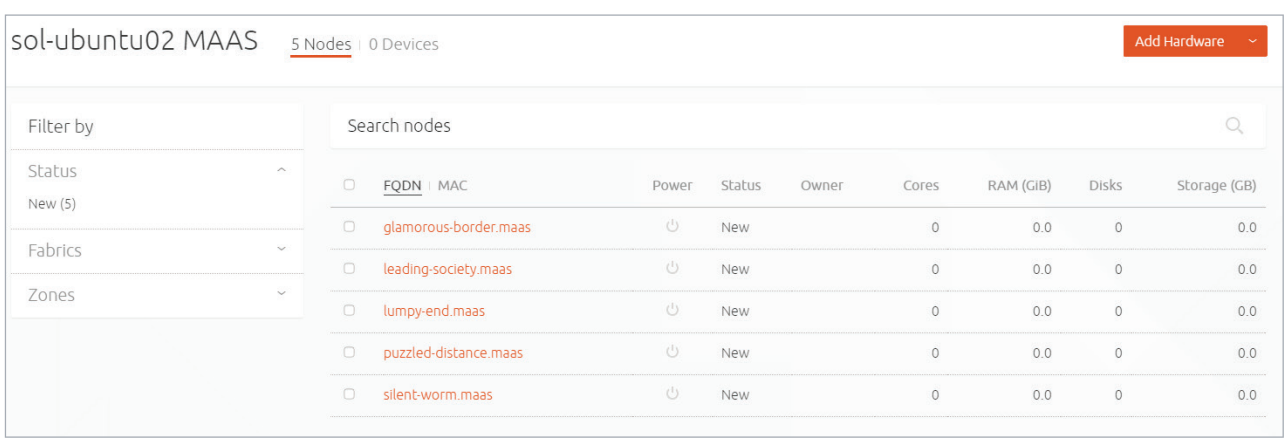

Figure 35: New Hosts were added to MAAS

After changing the nodes' names:

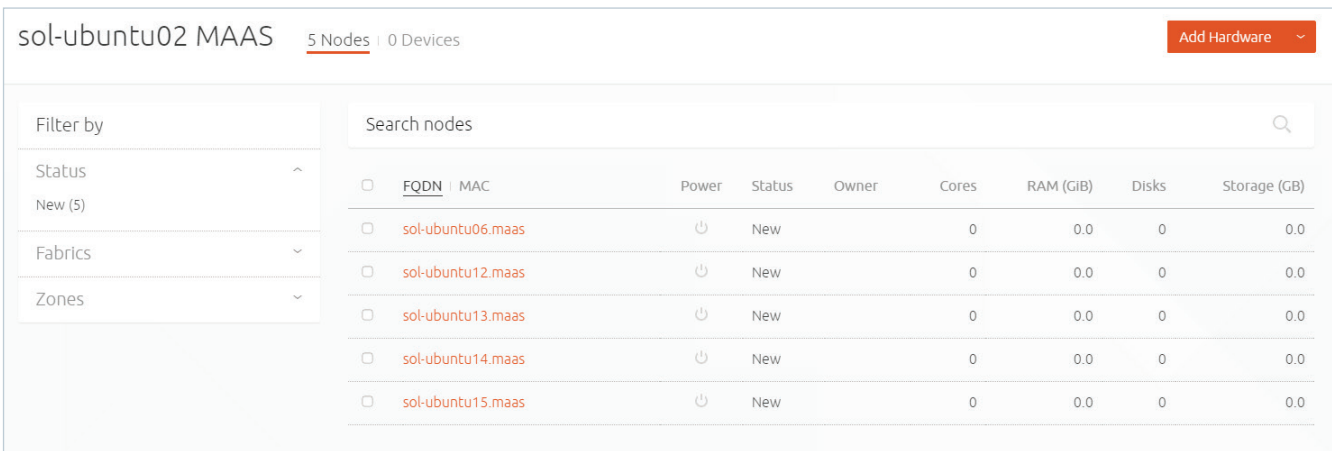

Figure 36: Hosts in MAAS with Updated Names

4. From the MAAS web GUI, Select the Nodes tab and check the checkbox next to all listed nodes. Select 'Commission' from the drop-down list at the top right corner of the screen. Commissioning the nodes boot the nodes into an image loaded from PXE to discover the nodes' system details.

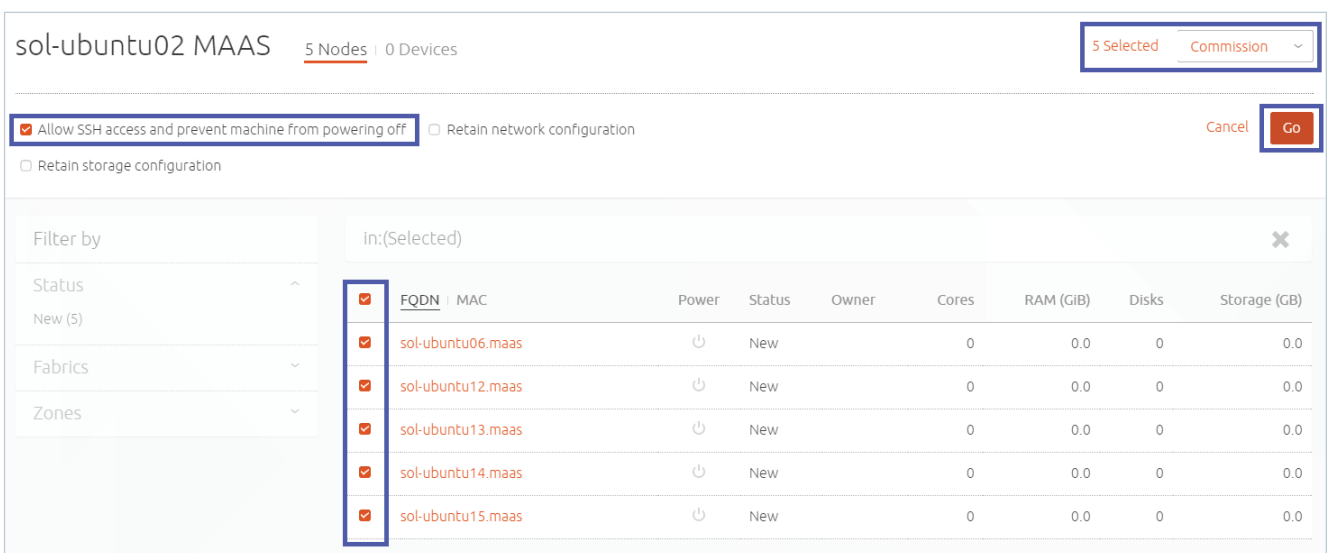

Figure 37: Commissioning Hosts in MAAS

5. After commission is done, edit each node network settings to comply your needs.

Ubuntu Autopilot Installer

1. From 1. From the MAAS server CLI, install the Ubuntu OpenStack package and launch it:
   ```
   sudo apt-get install openstack
   sudo openstack-install
   ```
2. Follow the installer steps by filling all required information:
   a. Choose the 'Landscape OpenStack Autopilot' option.
   b. Create a new OpenStack password.
   c. Fill in your:
      i. Admin email address.
      ii. Admin name.
      iii. MAAS server IP address
      iv. MAAS user API key
   d. Once installation is done, the installer will show a link and credentials:

Figure 38: Landscape OpenStack Autopilot Installation Completed

3. The next step of the process is to deploy the rest of the nodes in MAAS by deploying the Juju OpenStack charm. Login to the URL given in the previous step and follow the wizard on screen. Make sure you have all OpenStack services wherever they should be deployed:

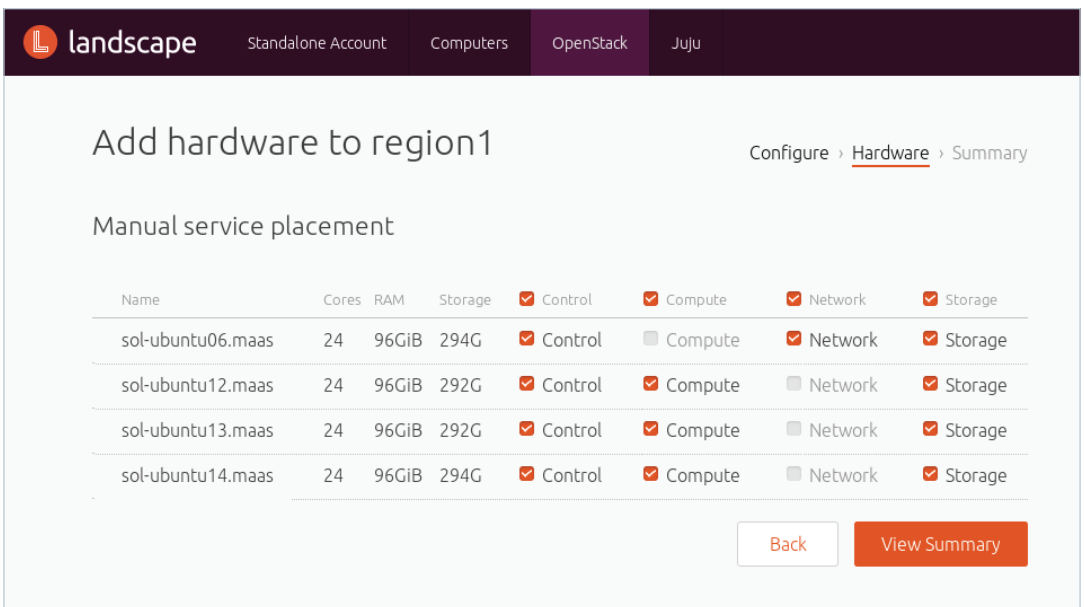

Figure 39: OpenStack Services Placement

Procced with the wizard until installation complete:

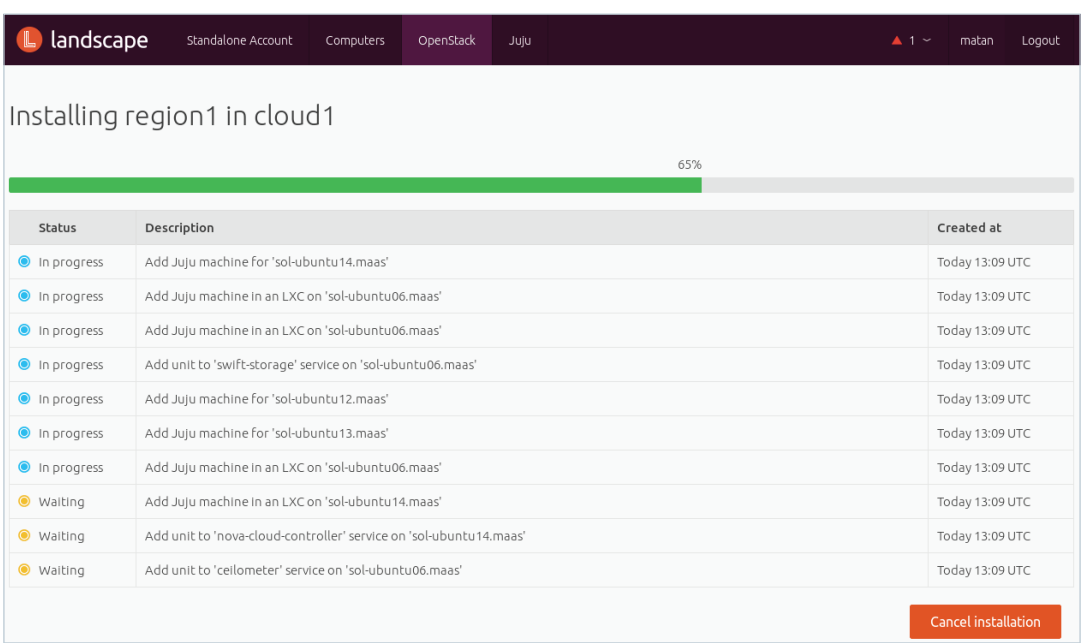

Figure 40: Landscape Installation in Progress

When installation is complete a message would appear on screen:

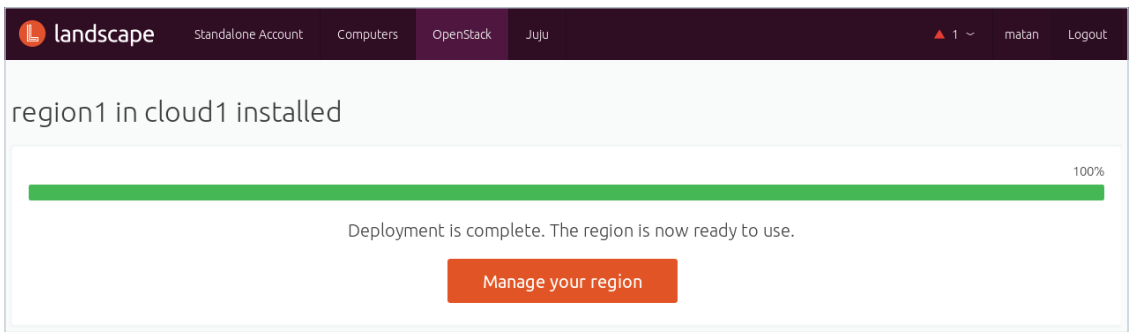

Figure 41: Landscape Installation Completed

By clicking the 'Manage your region' you will be forwarded to the Landscape Dashboard. It is recommended to download all RC files for future use:

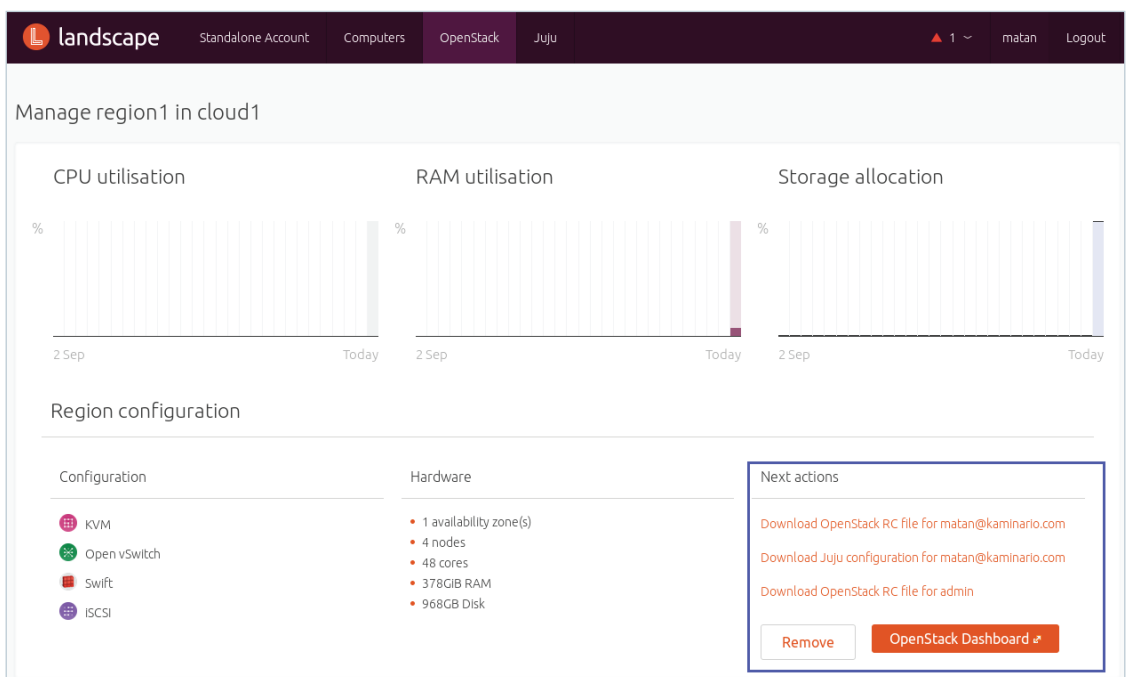

Figure 42: Landscape OpenStack Dashboard

kaminario.

Contact

Contact a business development representative to answer any questions you may have.

Schedule a Demo

Schedule a demo with an engineer and learn if Kaminario's solution works for you.

Request a Quote

Request a quote for your application from our business development team.

About Kaminario

Kaminario, the leading all-flash storage company, is redefining the future of modern data centers. Its unique solution enables organizations to succeed in today's on-demand world and prepares them to seamlessly handle tomorrow's innovations. Only Kaminario K2 delivers the agility, scalability, performance and economics a data center requires to deal with today's cloud-first, dynamic world and provide real-time data access -- anywhere, anytime. Hundreds of customers rely on the Kaminario K2 all-flash array to power their mission critical applications and safeguard their digital ecosystem. Headquartered in Needham, MA, Kaminario works with an extensive network of resellers and distributors, globally.

For more information, visit www.kaminario.com