WHITE PAPER | AUGUST 2020

MongoDB + Diamanti: A Hyper-Accelerated Story

AUTHORS:

Boris Kurktchiev, Diamanti
Jason Mimick, MongoDB

DIAMANTI mongoDB.

Contents

Overview

Organizations today are embracing containers and Kubernetes for rapid development and portability of stateful applications. While containers and Kubernetes can abstract away much of the infrastructure, it is still an important consideration when containerizing high-performance databases such as MongoDB. Enterprises need to ensure that the infrastructure can support multiple service level offerings (performance levels) and is self-managed across on-premises and cloud environments. In addition to this, these stateful applications contain business-critical information and demand advanced data services like data protection, monitoring, and fine-grained RBAC over a transactional data store with encryption at rest and in transit.

MongoDB has traditionally delivered excellent performance to value ratios. Typically, these deployments have been bare-metal or VM based. However, as cloud-native continues to grow and dominate enterprise software development and delivery practices, the ability to offer first-class enterprise-grade data services into containerized platforms such as Kubernetes is more urgent than ever before.

This document showcases the power of using MongoDB on Diamanti to drive I/O value as well as demonstrate the elasticity and usability driven by the adoption of Kubernetes and MongoDB Enterprise Advanced.

Goals of the Tests

1. Showcase a resilient MongoDB architecture on Kubernetes
2. Demonstrate the I/O capabilities of MongoDB and Diamanti Kubernetes Platform under different workload scenarios

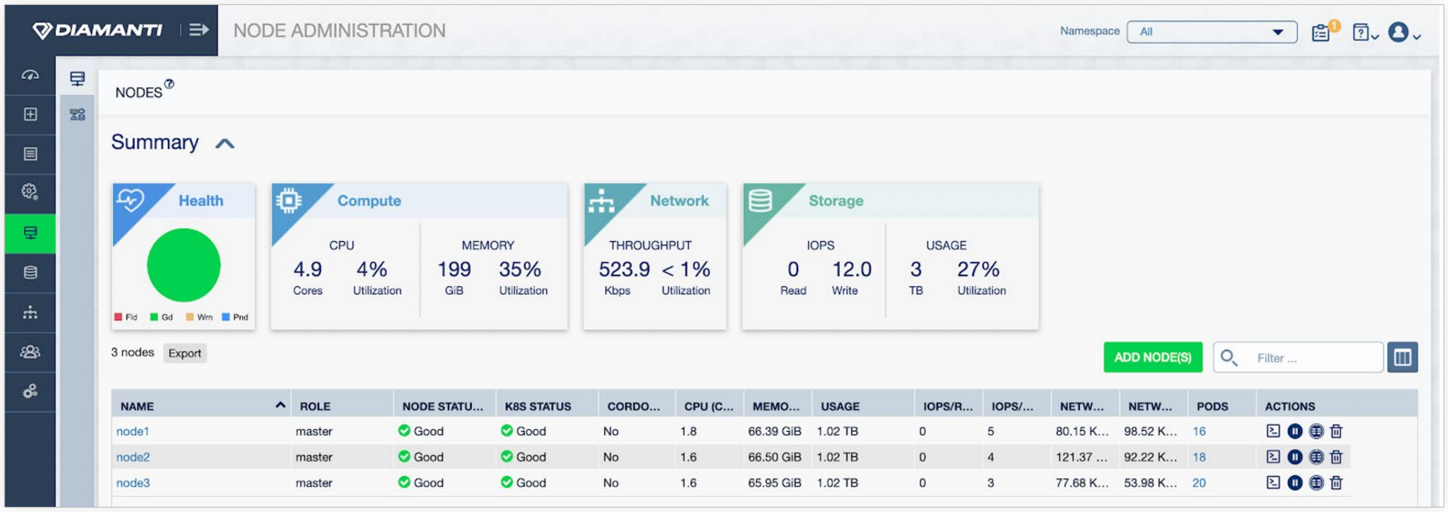
Specifications

For these tests we used 2 Diamanti D20 clusters each with 3 nodes, specifications described below, combined with MongoDB Kubernetes Operator and MongoDB Cloud Manager to showcase the benefits of a purpose-built platform. While we chose the Cloud Manager for this test case, MongoDB Ops Manager would be typically used for production scenarios. The entire example from the functional and Kubernetes perspective is identical using either cluster management solution.

Hardware Used

Diamanti D20 series of modern hyper-converged platforms are packaged with Diamanti Ultima I/O acceleration cards with varying configurations of Intel CPUs, memory, and NVMe storage. Diamanti Ultima is a pair of second generation PCIe based I/O acceleration cards that offload networking and storage traffic to deliver dramatically improved performance. They provide enterprise-class storage services for disaster recovery (DR) and data protection (DP) of mission critical applications, including mirroring, snapshots, multi-cluster asynchronous replication, and backup and recovery.

For these tests, we used the Diamanti D20 “Small” nodes to form the two 3 node clusters. The D20 “Small” has 20 physical CPU cores running at 2.2GHz each, 192 GB of RAM and Diamanti Ultima I/O acceleration for 40G networking, and 4 TB of ultra-fast, low latency NVMe Storage.



CPU		MEMORY		THROUGHPUT		IOPS		USAGE	
4.9	4%	199	35%	523.9	< 1%	0	12.0	3	27%
Cores	Utilization	GiB	Utilization	Kbps	Utilization	Read	Write	TB	Utilization

3 nodes

NAME	ROLE	NODE STATU...	K8S STATUS	CORDO...	CPU (C...	MEMO...	USAGE	IOPS/R...	IOPS/...	NETW...	NETW...	PODS	ACTIONS
node1	master	✓ Good	✓ Good	No	1.8	66.39 GiB	1.02 TB	0	5	80.15 K...	98.52 K...	16	
node2	master	✓ Good	✓ Good	No	1.6	66.50 GiB	1.02 TB	0	4	121.37 ...	92.22 K...	18	
node3	master	✓ Good	✓ Good	No	1.6	65.95 GiB	1.02 TB	0	3	77.68 K...	53.98 K...	20	

Figure 1: Easy to Manage and Monitor Diamanti D20 nodes

Software Used

Diamanti Spektra is the prevalidated, pre-packaged, and fully-featured software stack including CNCF certified Kubernetes, container runtime, operating system, enterprise-class DP/DR features, access controls, and Management UI. Our recent release of Diamanti Spektra 2.4 offers protection for data at rest with AES 256-bit volume encryption. With this capability, enterprise customers can prevent data theft and protect against security breaches while not incurring significant performance penalties.

The MongoDB database-as-a-service tier is implemented with MongoDB Ops Manager in conjunction with the MongoDB Enterprise Kubernetes Operator. MongoDB Ops Manager is an enterprise database management system. Ops Manager helps you automate database administration tasks such as deployment, upgrades, scaling events, and more through the platform UI and API. The MongoDB Kubernetes Operator introduces native custom resource definitions (CRDs) into your Kubernetes cluster enabling you to dynamically provision MongoDB clusters and MongoDB Ops Manager instances.

Yahoo! Cloud Serving Benchmark (YCSB) was utilized as a simple off-the-shelf synthetic test harness tool. While this tool connects to MongoDB just like any other application (with a secure connection string) we did optimize the software specifically for this project. As such, you may not get the same performance results and neither these tests nor results are valid for any other database which YCSB may happen to support. But you will be able to use the exact same infrastructure and platform tooling for any MongoDB application running on Kubernetes.

You can try out these demonstrations yourself with the example deployments and charts available in the total-cluster Github repository.

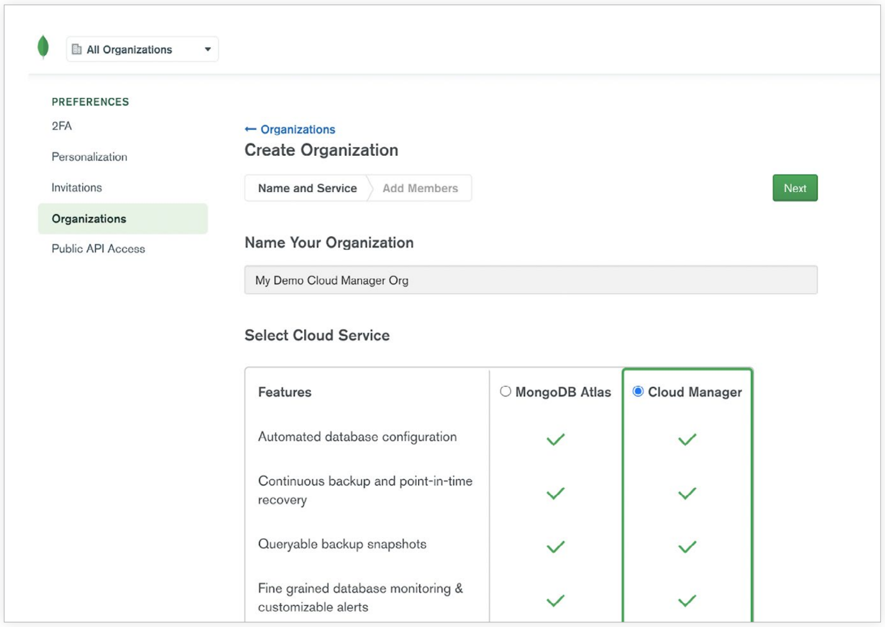
Test Results

There are various levels of 'sophistication' you can choose for your total-cluster. The easiest way to get started is with the extra-lite option available in the setups[1]. This option is "lightweight" in the sense that you'll be leveraging MongoDB Cloud Manager which is a hosted version of MongoDB Ops Manager. Using Cloud Manager perfectly positions your organization to incorporate fully hosted databases running on MongoDB Atlas, while still allowing you to utilize Diamanti hardware for your applications. More advanced configurations like this are possible through a number of cloud-native MongoDB options such as the MongoDB Atlas Service Broker, the MongoDB Atlas Terraform Provider, and the MongoDB Atlas CloudFormation Custom Resources.

1. https://github.com/jasonmimick/total-cluster/blob/master/README.md#setups

The first step is to head over to cloud.mongodb.com. Here you need to:

- Create an account (if you do not have one already)
- Create a Cloud Manager organization[2]- note to connect databases running in your own Kubernetes cluster it must be a Cloud Manager project



Name Your Organization

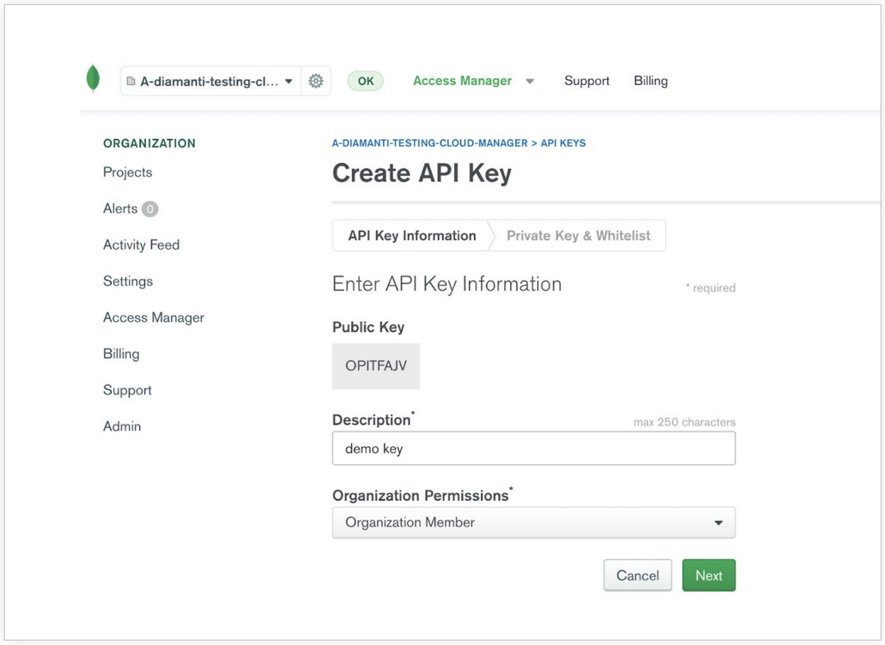
My Demo Cloud Manager Org

Select Cloud Service

Features	MongoDB Atlas	[✓ Cloud Manager]
Automated database configuration	✓	✓
Continuous backup and point-in-time recovery	✓	✓
Queryable backup snapshots	✓	✓
Fine grained database monitoring & customizable alerts	✓	✓

Figure 2: Create organization in the MongoDB Cloud Manager

- Add an Organization API Key[3] as shown in Figure 3 below:



Enter API Key Information * required

Public Key: OPITFAJV

Description* (max 250 characters): demo key

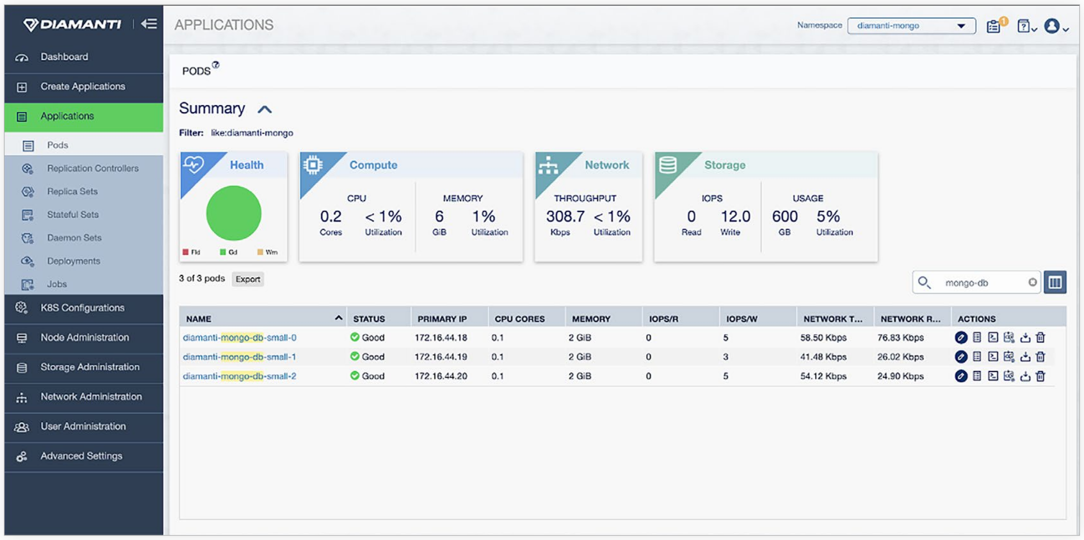
Organization Permissions*: Organization Member

Figure 3: Create an API Key in the organization

2. https://docs.cloudmanager.mongodb.com/tutorial/manage-organizations/#create-an-organization
3. https://docs.atlas.mongodb.com/configure-api-access/#manage-programmatic-access-to-an-organization

You can find all the details for installing this in your own clusters in the Diamanti-MongoDB YCSB Tests How to document over in our repository.

Below you can see the simple deployment of MongoDB on the Diamanti Spektra management console. For all testing, we utilized a **3-way ReplicaSet using 4 GB of RAM and 4 CPU cores per container**, along with making use of the Diamanti Spektra Quality of Service (QoS) capabilities to give the containers a guarantee of 20,000 IOPS and 500 MB/s of network throughput. The setup is shown below.



Summary

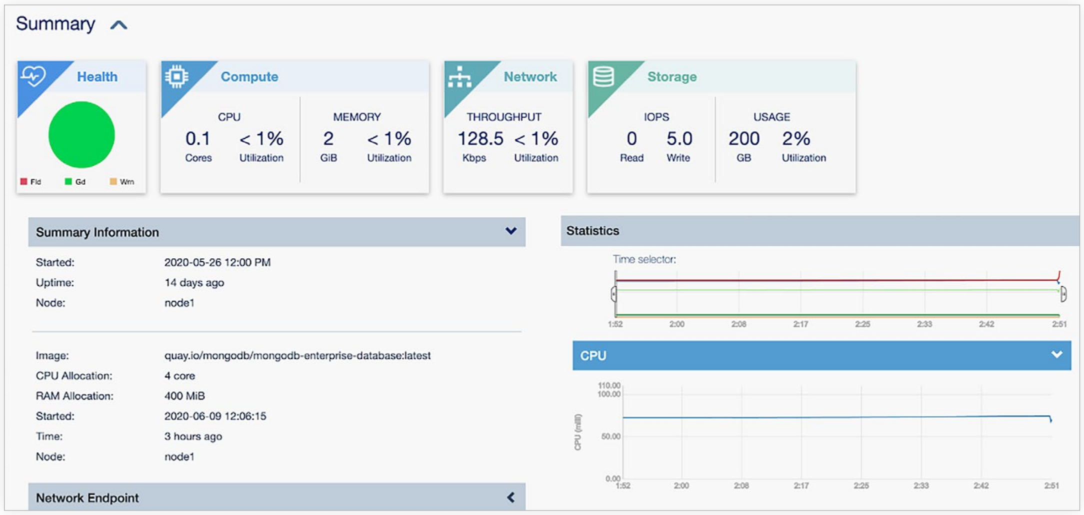
Filter: like:diamanti-mongo

Health	Compute		Network	Storage	
	CPU	MEMORY	THROUGHPUT	IOPS	USAGE
	0.2 Cores, < 1% Utilization	6 GiB, 1% Utilization	308.7 Kbps, < 1% Utilization	0 Read, 12.0 Write	600 GB, 5% Utilization

3 of 3 pods Export

NAME	STATUS	PRIMARY IP	CPU CORES	MEMORY	IOPS/R	IOPS/W	NETWORK T...	NETWORK R...	ACTIONS
diamanti-mongo-db-small-0	Good	172.16.44.18	0.1	2 GiB	0	5	58.50 Kbps	76.83 Kbps	
diamanti-mongo-db-small-1	Good	172.16.44.19	0.1	2 GiB	0	3	41.48 Kbps	26.02 Kbps	
diamanti-mongo-db-small-2	Good	172.16.44.20	0.1	2 GiB	0	5	54.12 Kbps	24.90 Kbps	

Figure 4: Diamanti MongoDB 3-way ReplicaSet overview



Summary

Health	Compute		Network	Storage	
	CPU	MEMORY	THROUGHPUT	IOPS	USAGE
	0.1 Cores, < 1% Utilization	2 GiB, < 1% Utilization	128.5 Kbps, < 1% Utilization	0 Read, 5.0 Write	200 GB, 2% Utilization

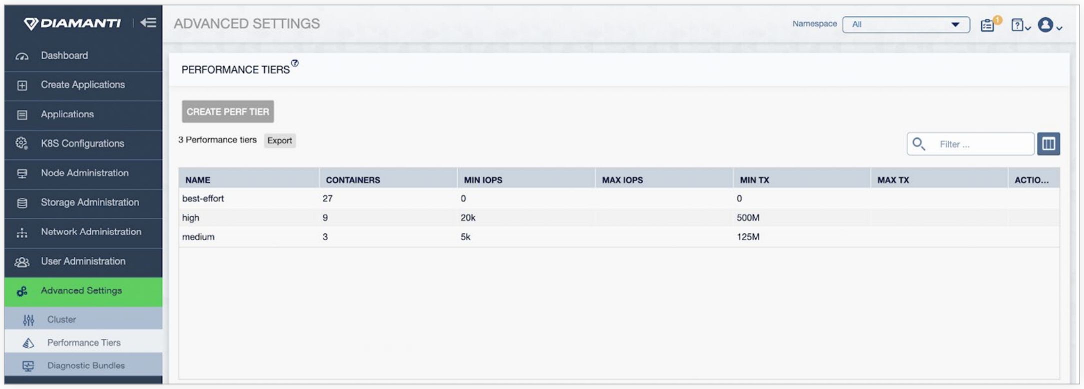
Summary Information

Started:	2020-05-26 12:00 PM
Uptime:	14 days ago
Node:	node1
Image:	quay.io/mongodb/mongodb-enterprise-database:latest
CPU Allocation:	4 core
RAM Allocation:	400 MiB
Started:	2020-06-09 12:06:15
Time:	3 hours ago
Node:	node1

Figure 5: Diamanti container level metrics

Figure 4 shows the Diamanti Spektra dashboard displaying the 3 deployed containers in the namespace **diamanti-mongo**. Figure 5 shows the detailed metrics of one of the deployed containers.

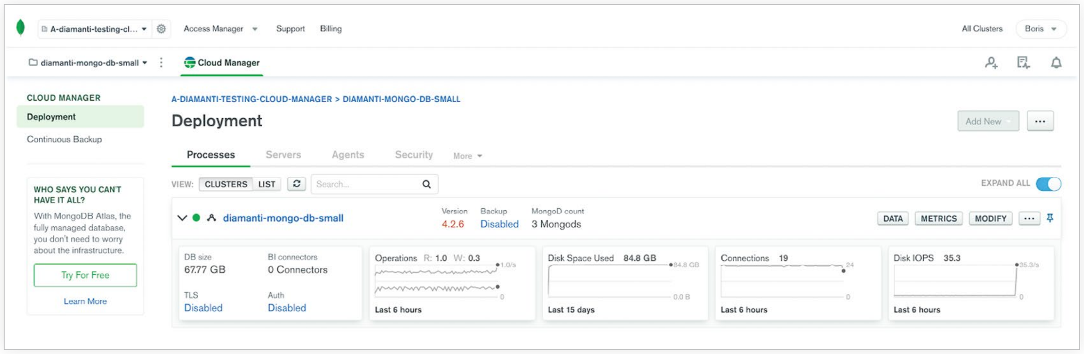
One of the built-in features of the Diamanti platform is its ability to augment Kubernetes by providing a way for users to define QoS tiers, the default set of which is shown in the figure below.



3 Performance tiers

NAME	CONTAINERS	MIN IOPS	MAX IOPS	MIN TX	MAX TX	ACTIO...
best-effort	27	0		0		
high	9	20k		500M		
medium	3	5k		125M		

Figure 6: Diamanti default set of performance tiers



A-DIAMANTI-TESTING-CLOUD-MANAGER > DIAMANTI-MONGO-DB-SMALL

Deployment

diamanti-mongo-db-small

Version 4.2.6 Backup Disabled MongoD count 3 Mongods

DB size 67.77 GB BI connectors 0 Connectors

TLS Disabled Auth Disabled

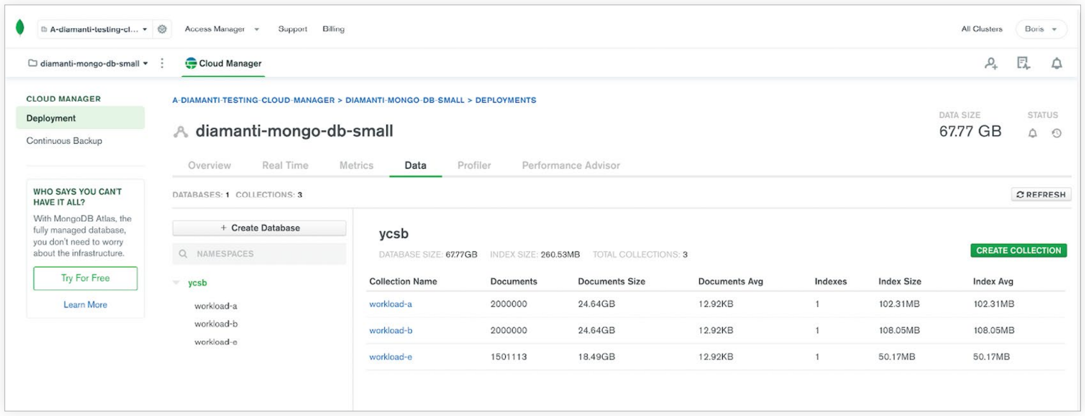
Operations R: 1.0 W: 0.3 Last 6 hours

Disk Space Used 84.8 GB Last 15 days

Connections 19 Last 6 hours

Disk IOPS 35.3 Last 6 hours

Figure 7: CloudManager view of the deployed cluster



A-DIAMANTI-TESTING-CLOUD-MANAGER > DIAMANTI-MONGO-DB-SMALL > DEPLOYMENTS

diamanti-mongo-db-small

DATA SIZE 67.77 GB STATUS

DATABASES: 1 COLLECTIONS: 3

ycsb

DATABASE SIZE: 67.77GB INDEX SIZE: 260.53MB TOTAL COLLECTIONS: 3

Collection Name	Documents	Documents Size	Documents Avg	Indexes	Index Size	Index Avg
workload-a	2000000	24.64GB	12.92KB	1	102.31MB	102.31MB
workload-b	2000000	24.64GB	12.92KB	1	108.05MB	108.05MB
workload-e	1501113	18.49GB	12.92KB	1	50.17MB	50.17MB

Figure 8: CloudManager view of the loaded data for the experiments

Tests I - Workload A (50/50 read/write)

The overall strategy for all of the completed tests was to use as much of the default workload configurations as possible. The notable changes are highlighted for each test. The most notable of these changes was the use of much larger **fieldcount** and **fieldlength** parameters than what is used by default. This was done in order to get the tests to behave closer to real-world usages of MongoDB.

We would like to note that these numbers are chosen specifically for approximating real world usages, however, each application and usage can have vastly different sizing requirements which will produce different outcomes.

```
recordcount=1000000
operationcount=10000000
workload=site.ycsb.workloads.
CoreWorkload
readallfields=true
readproportion=0.5
updateproportion=0.5
scanproportion=0
insertproportion=0
requestdistribution=zipfian
table=workload-a
fieldcount=50
fieldlength=250
threadcount=16
```

Tests II - Workload B (95/5 read/write)

```
recordcount=1000000
operationcount=10000000
workload=site.ycsb.workloads.
CoreWorkload
readallfields=true
readproportion=0.95it
updateproportion=0.05
scanproportion=0
insertproportion=0
requestdistribution=zipfian
table=workload-b
fieldcount=50
fieldlength=250
threadcount=16
```

Tests III - Workload E (95/5 Scan/Update)

```
recordcount=1000000
operationcount=10000000
workload=site.ycsb.
workloads.CoreWorkload
readallfields=true
readproportion=0
updateproportion=0
scanproportion=0.95
insertproportion=0.05
requestdistribution=zipfian
maxscanlength=100
scanlengthdistribution
=uniform
table=workload-e
fieldcount=50
fieldlength=250
threadcount=16
```

Findings

First and foremost, we would like to note that we are not providing outputs of the LOAD phase for these tests, as they are inconsequential and their sole function is to pre-load data to the cluster. The specifics of the runs can be seen in Figure 8 showing each individual table created for the runs we are executing, as well as individual table statistics and information.

Test I - Workload A

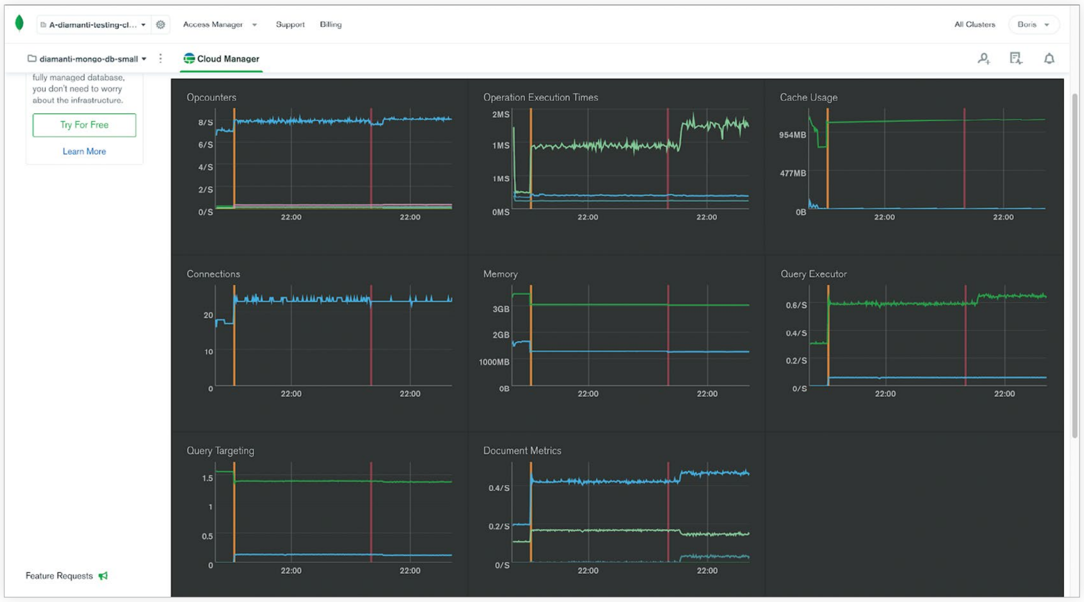
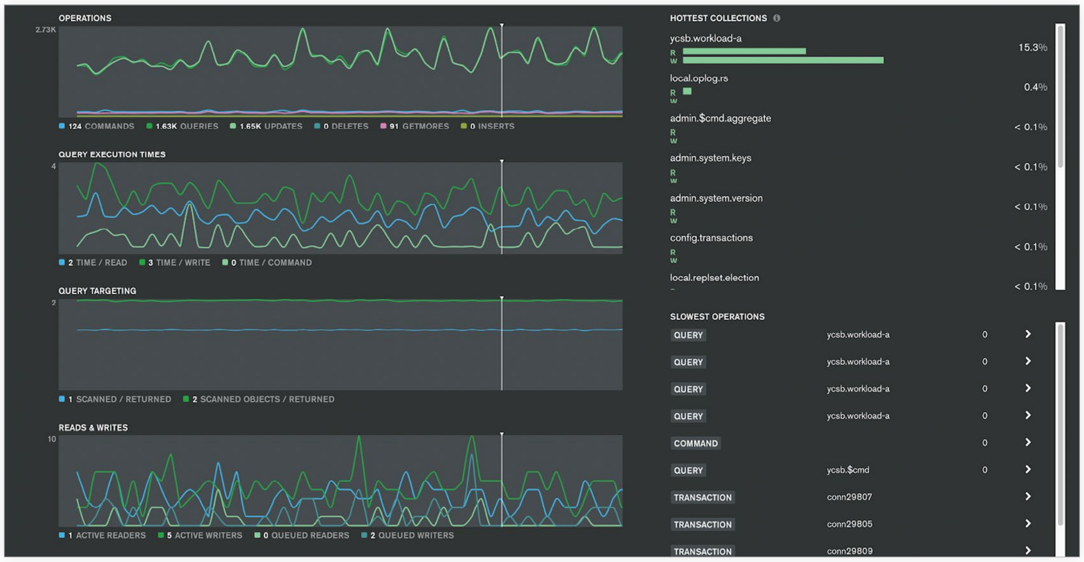


Figure 9: CloudManager view during Workload A run



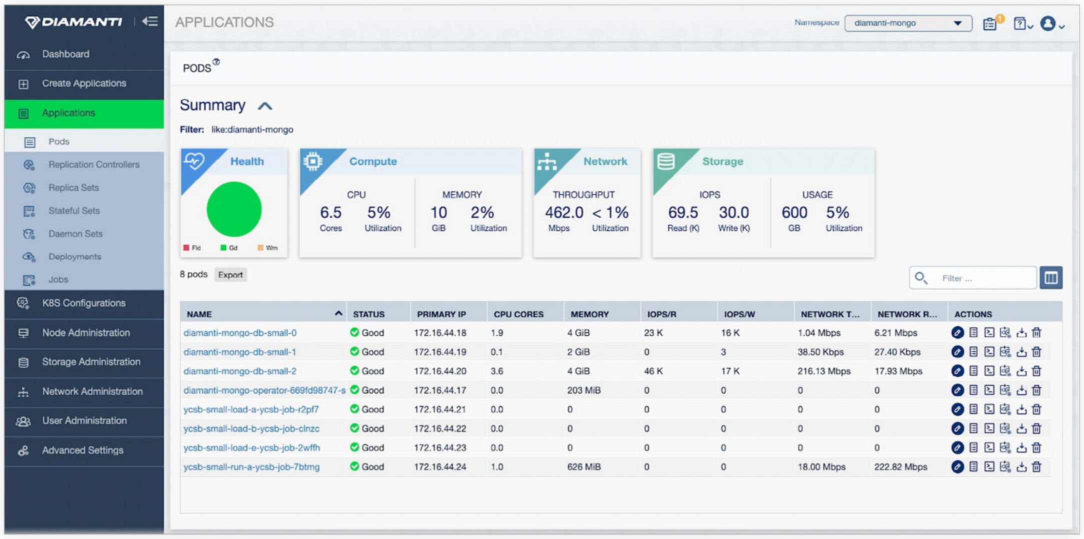
HOTTEST COLLECTIONS

ycsb.workload-a	15.3%
local.oplog.rs	0.4%
admin.$cmd.aggregate	< 0.1%
admin.system.keys	< 0.1%
admin.system.version	< 0.1%
config.transactions	< 0.1%
local.replset.election	< 0.1%

SLOWEST OPERATIONS

QUERY	ycsb.workload-a	0
QUERY	ycsb.workload-a	0
QUERY	ycsb.workload-a	0
QUERY	ycsb.workload-a	0
COMMAND		0
QUERY	ycsb.$cmd	0
TRANSACTION	conn29807	
TRANSACTION	conn29805	
TRANSACTION	conn29809	

Figure10: CloudManager Real-Time statistics view during the run



Filter: like:diamanti-mongo

Compute			
CPU		MEMORY	
6.5 Cores	5% Utilization	10 GiB	2% Utilization

Network	
THROUGHPUT	
462.0 Mbps	< 1% Utilization

Storage			
IOPS		USAGE	
69.5 Read (K)	30.0 Write (K)	600 GB	5% Utilization

8 pods

NAME	STATUS	PRIMARY IP	CPU CORES	MEMORY	IOPS/R	IOPS/W	NETWORK T...	NETWORK R...
diamanti-mongo-db-small-0	✓ Good	172.16.44.18	1.9	4 GiB	23 K	16 K	1.04 Mbps	6.21 Mbps
diamanti-mongo-db-small-1	✓ Good	172.16.44.19	0.1	2 GiB	0	3	38.50 Kbps	27.40 Kbps
diamanti-mongo-db-small-2	✓ Good	172.16.44.20	3.6	4 GiB	46 K	17 K	216.13 Mbps	17.93 Mbps
diamanti-mongo-operator-669fd98747-s	✓ Good	172.16.44.17	0.0	203 MiB	0	0	0	0
ycsb-small-load-a-ycsb-job-r2pf7	✓ Good	172.16.44.21	0.0	0	0	0	0	0
ycsb-small-load-b-ycsb-job-clnzc	✓ Good	172.16.44.22	0.0	0	0	0	0	0
ycsb-small-load-e-ycsb-job-2wffh	✓ Good	172.16.44.23	0.0	0	0	0	0	0
ycsb-small-run-a-ycsb-job-7btmg	✓ Good	172.16.44.24	1.0	626 MiB	0	0	18.00 Mbps	222.82 Mbps

Figure11: Diamanti view during the run phase

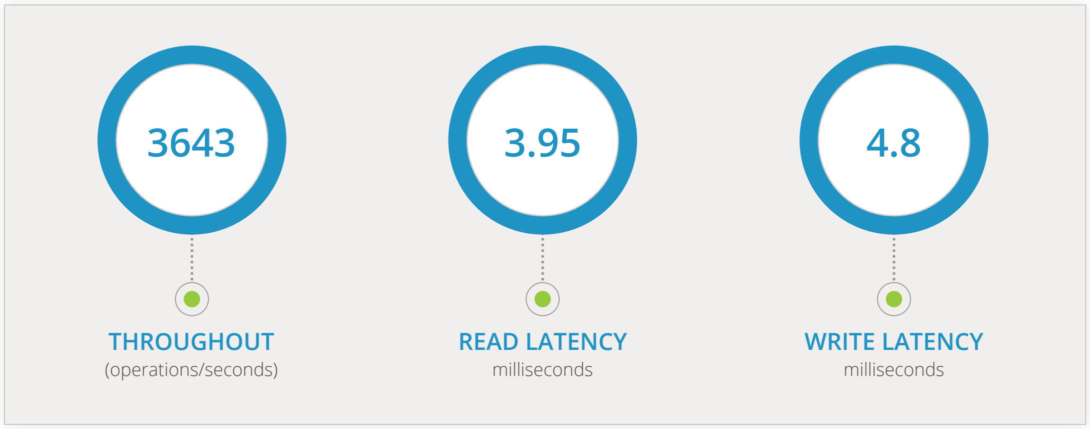
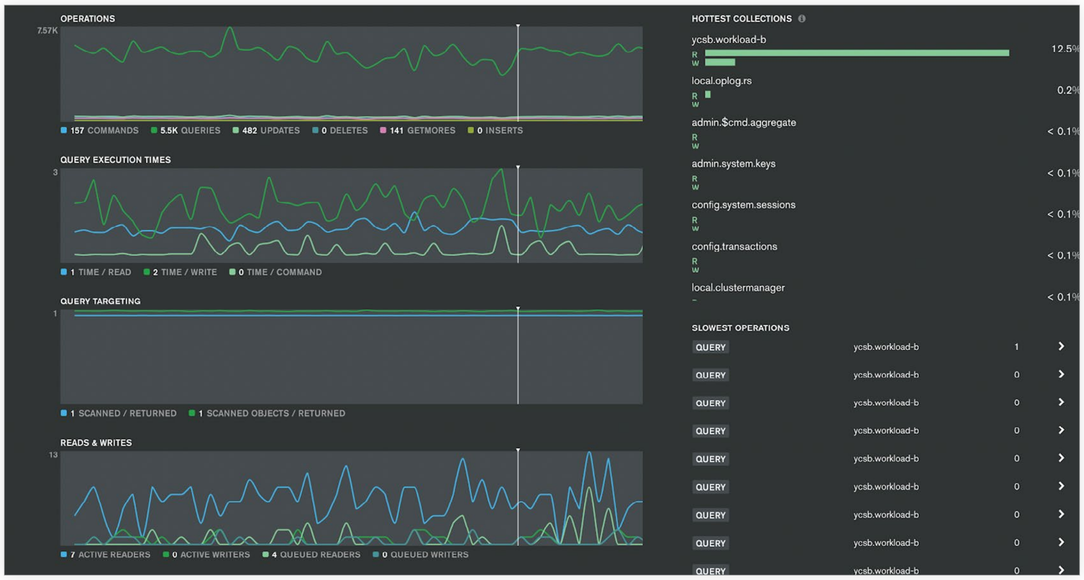


Figure12: Summary of the test run for Workload A

Raw Numbers for Workload A:

```
[OVERALL],RunTime(ms),2744332
[OVERALL],Throughput(ops/sec),
3643.8739919222603
[READ],Operations,4998747
[READ],AverageLatency(us),
3956.550892653699
[READ],MinLatency(us),280
[READ],MaxLatency(us),392959
[READ],95thPercentileLatency(us),2313
[READ], 99thPercentileLatency(us),75327
[CLEANUP],Operations,16
[CLEANUP],AverageLatency(us), 842.937
[CLEANUP],MinLatency(us),2
[CLEANUP],MaxLatency(us),13247
[CLEANUP], 95thPercentileLatency(us),204
[CLEANUP],
99thPercentileLatency(us),13247
[UPDATE],Operations,5001253
[UPDATE],AverageLatency(us),
4810.487657792957
[UPDATE],MinLatency(us),283
[UPDATE],MaxLatency(us),393727
[UPDATE], 95thPercentileLatency(us),4959
[UPDATE], 99thPercentileLatency(us),75903
```

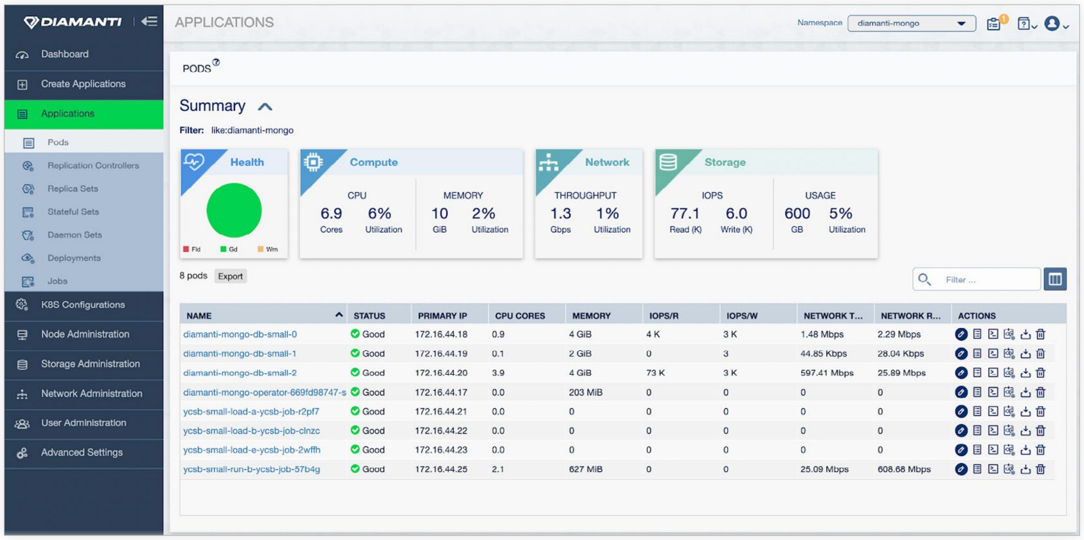
Test II - Workload B



SLOWEST OPERATIONS

QUERY	ycsb.workload-b	1
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0
QUERY	ycsb.workload-b	0

Figure13: CloudManager Real-Time statistics view during the run



Filter: like:diamanti-mongo

Compute				Network		Storage			
CPU		MEMORY		THROUGHPUT		IOPS		USAGE	
6.9 Cores	6% Utilization	10 GiB	2% Utilization	1.3 Gbps	1% Utilization	77.1 Read (K)	6.0 Write (K)	600 GB	5% Utilization

8 pods

NAME	STATUS	PRIMARY IP	CPU CORES	MEMORY	IOPS/R	IOPS/W	NETWORK T...	NETWORK R...	ACTIONS
diamanti-mongo-db-small-0	✓ Good	172.16.44.18	0.9	4 GiB	4 K	3 K	1.48 Mbps	2.29 Mbps	
diamanti-mongo-db-small-1	✓ Good	172.16.44.19	0.1	2 GiB	0	3	44.85 Kbps	28.04 Kbps	
diamanti-mongo-db-small-2	✓ Good	172.16.44.20	3.9	4 GiB	73 K	3 K	597.41 Mbps	25.89 Mbps	
diamanti-mongo-operator-669fd98747-s	✓ Good	172.16.44.17	0.0	203 MiB	0	0	0	0	
ycsb-small-load-a-ycsb-job-r2pf7	✓ Good	172.16.44.21	0.0	0	0	0	0	0	
ycsb-small-load-b-ycsb-job-clnzc	✓ Good	172.16.44.22	0.0	0	0	0	0	0	
ycsb-small-load-e-ycsb-job-2wffh	✓ Good	172.16.44.23	0.0	0	0	0	0	0	
ycsb-small-run-b-ycsb-job-57b4g	✓ Good	172.16.44.25	2.1	627 MiB	0	0	25.09 Mbps	608.68 Mbps	

Figure14: Diamanti view during the run

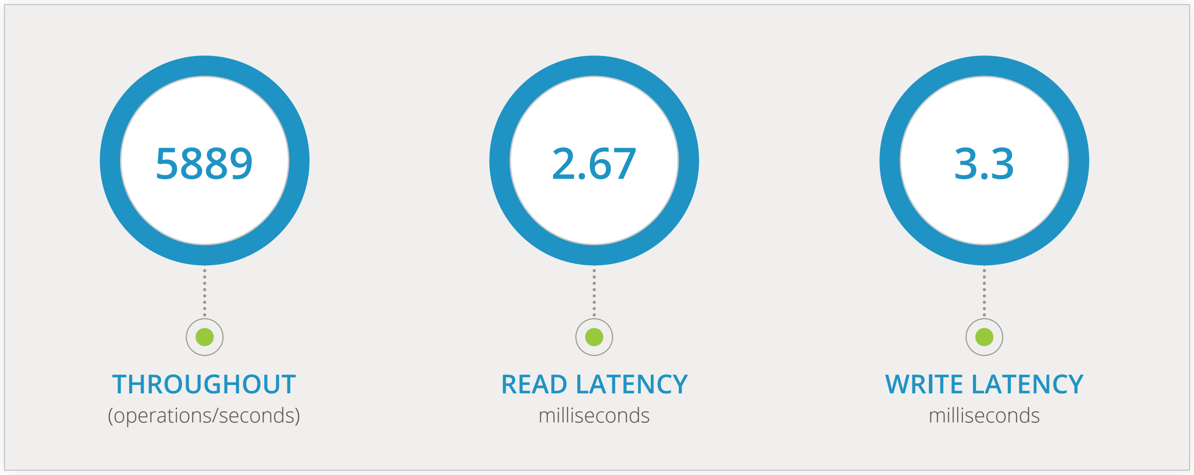


Figure15: Summary of the test run for Workload B

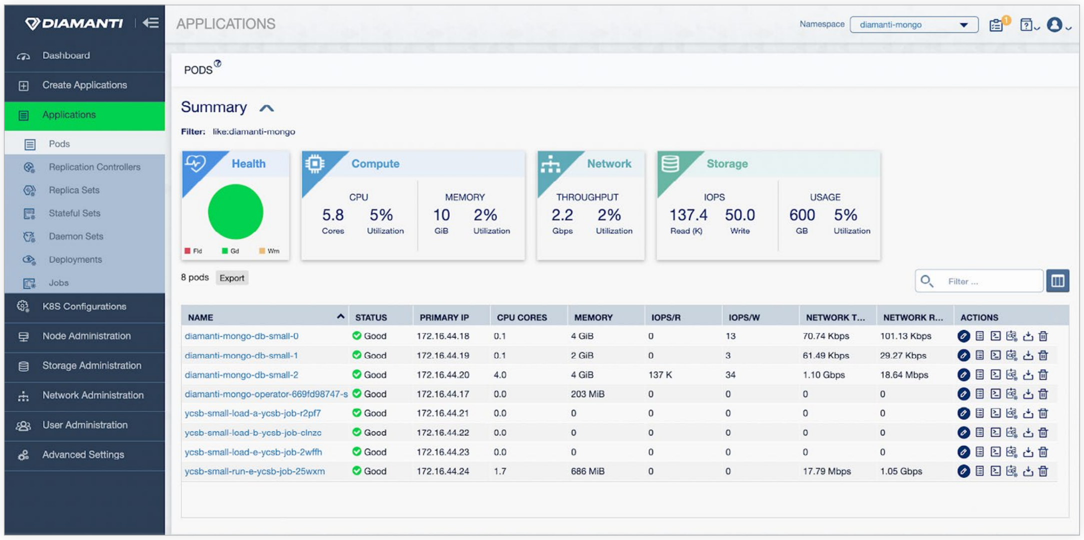
Raw Numbers for Workload B:

```
[OVERALL],RunTime(ms),1697923                     [CLEANUP],MinLatency(us),2
[OVERALL],Throughput(ops/sec),5889.5485837697     [CLEANUP],MaxLatency(us),12591
[READ],Operations,9500301                         [CLEANUP],95thPercentileLatency(us),27
[READ],AverageLatency(us),2678.458995772871       [CLEANUP],99thPercentileLatency(us),12591
[READ],MinLatency(us),222                         [UPDATE],Operations,499699
[READ],MaxLatency(us),133503                      [UPDATE],AverageLatency(us),3307.282377991551
[READ],95thPercentileLatency(us),1860             [UPDATE],MinLatency(us),319
[READ],99thPercentileLatency(us),61791            [UPDATE],MaxLatency(us),116031
[CLEANUP],Operations,16                           [UPDATE],95thPercentileLatency(us),2241
[CLEANUP],AverageLatency(us),790.75               [UPDATE],99thPercentileLatency(us),63487
```

Test III - Workload E

The starkest result here is that the scan heavy workload is producing much lower operations per second over the span of the test. This makes sense when you take into account what is happening in this test. In Figures 17 and 18, you can see all the YCSB scans are using the index and then returning the matching documents. So, if you take this plus the increase in document size - it means there are tons more I/O going on - not just one document over the wire but n-documents per query. This results in more I/O work on the backend. The important thing to note here is the fact that a 4 GB + 4 CPU core database is able to push 150K+ IOPS and do the work of a much larger instance in a substantially smaller footprint.

Note that we skip showing the CloudManager Real-time screens for this run as the database throughput and work is negligible. Instead, we are providing the CloudManager Profiler view as shown in Figures 17 and 18, which gives a better explanation of the scan operations executed by the workload. In this view you can see that a large swatch of documents gets scanned, producing the backend IOPS drive, and then returned, leading to high network throughput. But since each such scan takes over 100ms and returns an arbitrary number of documents, the actual transactions per second (TPS) are pretty low.



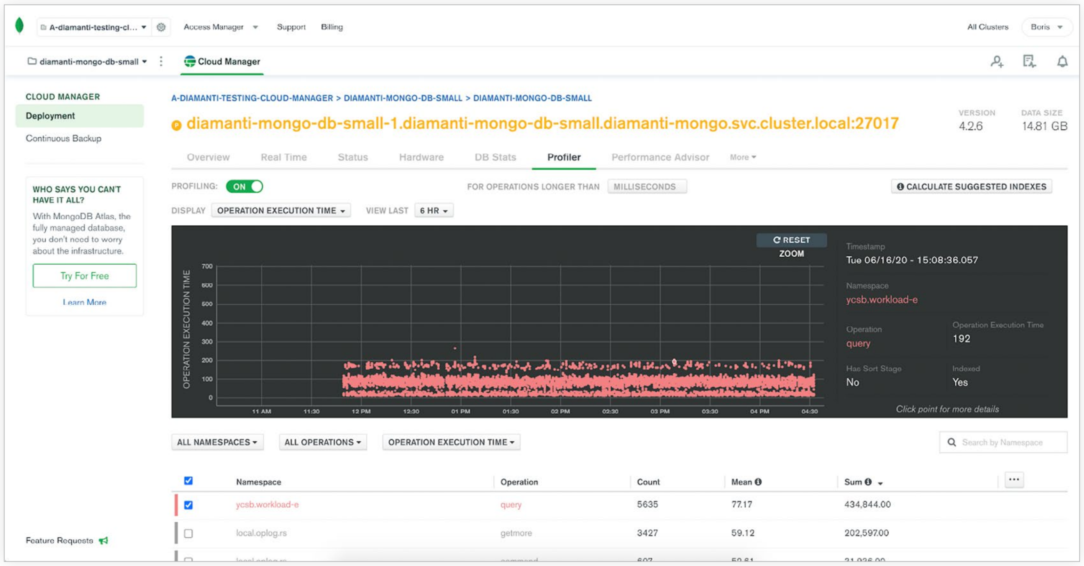
Filter: like:diamanti-mongo

Health	Compute: CPU	Compute: Memory	Network: Throughput	Storage: IOPS	Storage: Usage
🟥 Fld 🟩 Gd 🟧 Wrn	5.8 Cores, 5% Utilization	10 GiB, 2% Utilization	2.2 Gbps, 2% Utilization	137.4 Read (K), 50.0 Write	600 GB, 5% Utilization

8 pods

NAME	STATUS	PRIMARY IP	CPU CORES	MEMORY	IOPS/R	IOPS/W	NETWORK T...	NETWORK R...	ACTIONS
diamanti-mongo-db-small-0	Good	172.16.44.18	0.1	4 GiB	0	13	70.74 Kbps	101.13 Kbps	
diamanti-mongo-db-small-1	Good	172.16.44.19	0.1	2 GiB	0	3	61.49 Kbps	29.27 Kbps	
diamanti-mongo-db-small-2	Good	172.16.44.20	4.0	4 GiB	137 K	34	1.10 Gbps	18.64 Mbps	
diamanti-mongo-operator-669fd98747-s	Good	172.16.44.17	0.0	203 MiB	0	0	0	0	
ycsb-small-load-a-ycsb-job-r2pf7	Good	172.16.44.21	0.0	0	0	0	0	0	
ycsb-small-load-b-ycsb-job-clnzc	Good	172.16.44.22	0.0	0	0	0	0	0	
ycsb-small-load-e-ycsb-job-2wffh	Good	172.16.44.23	0.0	0	0	0	0	0	
ycsb-small-run-e-ycsb-job-25wxm	Good	172.16.44.24	1.7	686 MiB	0	0	17.79 Mbps	1.05 Gbps	

Figure16: Diamanti view during the Workload E run



A-DIAMANTI-TESTING-CLOUD-MANAGER > DIAMANTI-MONGO-DB-SMALL > DIAMANTI-MONGO-DB-SMALL

diamanti-mongo-db-small-1.diamanti-mongo-db-small.diamanti-mongo.svc.cluster.local:27017

VERSION 4.2.6 DATA SIZE 14.81 GB

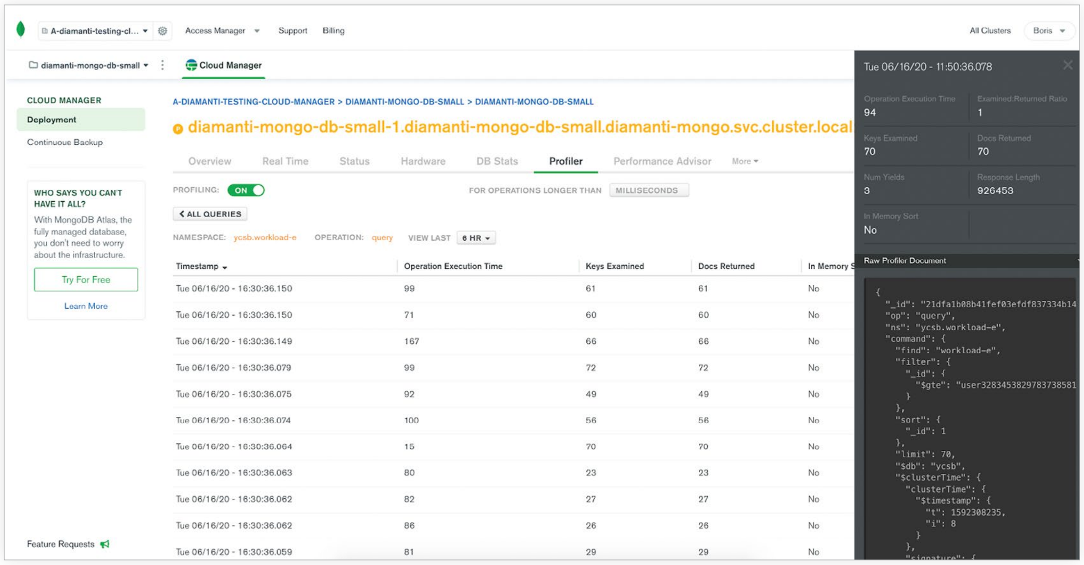
PROFILING: ON FOR OPERATIONS LONGER THAN MILLISECONDS

DISPLAY OPERATION EXECUTION TIME VIEW LAST 6 HR

Timestamp: Tue 06/16/20 - 15:08:36.057
Namespace: ycsb.workload-e
Operation: query
Operation Execution Time: 192
Has Sort Stage: No
Indexed: Yes

	Namespace	Operation	Count	Mean	Sum
✓	ycsb.workload-e	query	5635	77.17	434,844.00
☐	local.oplog.rs	getmore	3427	59.12	202,597.00
☐	local.oplog.rs	command	607	59.61	31,926.00

Figure17: CloudManager Profiler view, showing slow queries



A-DIAMANTI-TESTING-CLOUD-MANAGER > DIAMANTI-MONGO-DB-SMALL > DIAMANTI-MONGO-DB-SMALL

diamanti-mongo-db-small-1.diamanti-mongo-db-small.diamanti-mongo.svc.cluster.local

PROFILING: ON   FOR OPERATIONS LONGER THAN MILLISECONDS

NAMESPACE: ycsb.workload-e   OPERATION: query   VIEW LAST 6 HR

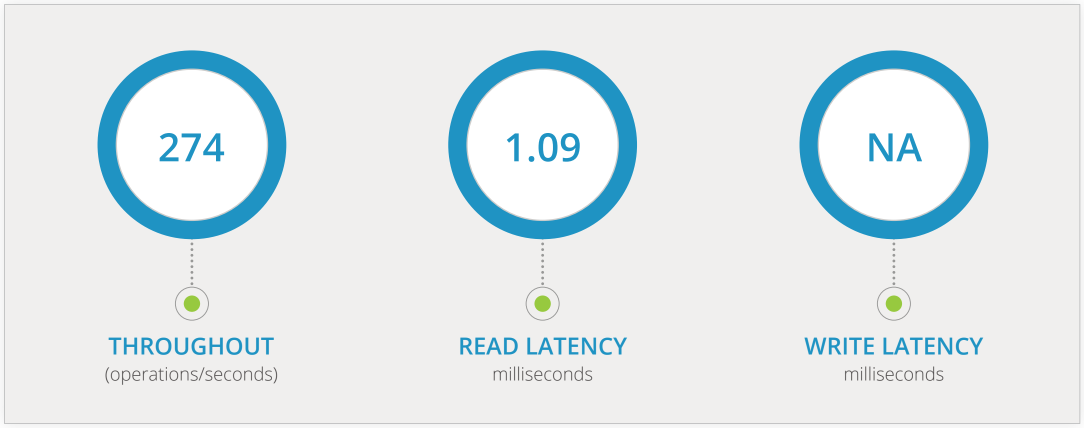
Timestamp	Operation Execution Time	Keys Examined	Docs Returned	In Memory S
Tue 06/16/20 - 16:30:36.150	99	61	61	No
Tue 06/16/20 - 16:30:36.150	71	60	60	No
Tue 06/16/20 - 16:30:36.149	167	66	66	No
Tue 06/16/20 - 16:30:36.079	99	72	72	No
Tue 06/16/20 - 16:30:36.075	92	49	49	No
Tue 06/16/20 - 16:30:36.074	100	56	56	No
Tue 06/16/20 - 16:30:36.064	15	70	70	No
Tue 06/16/20 - 16:30:36.063	80	23	23	No
Tue 06/16/20 - 16:30:36.062	82	27	27	No
Tue 06/16/20 - 16:30:36.062	86	26	26	No
Tue 06/16/20 - 16:30:36.059	81	29	29	No

Tue 06/16/20 - 11:50:36.078

Operation Execution Time	Examined:Returned Ratio
94	1
Keys Examined	**Docs Returned**
70	70
Num Yields	**Response Length**
3	926453
In Memory Sort	
No	

```
{
  "_id": "21dfa1b08b41fef03efdf837334b14
  "op": "query",
  "ns": "ycsb.workload-e",
  "command": {
    "find": "workload-e",
    "filter": {
      "_id": {
        "$gte": "user3283453829783738581
      }
    },
    "sort": {
      "_id": 1
    },
    "limit": 70,
    "$db": "ycsb",
    "$clusterTime": {
      "clusterTime": {
        "$timestamp": {
          "t": 1592308235,
          "i": 8
        }
      },
      "signature": {
```

Figure 18: CloudManager Profiler drill down view of a sample slow query



274 THROUGHOUT (operations/seconds)

1.09 READ LATENCY milliseconds

NA WRITE LATENCY milliseconds

Figure 19: Summary of the test run for Workload E

Raw Numbers for Workload E:

```
[OVERALL],RunTime(ms),36469749
[OVERALL],Throughput(ops/
sec),274.19985807963747
[CLEANUP],Operations,16
[CLEANUP],AverageLatency(us),1096.5
[CLEANUP],MinLatency(us),2
[CLEANUP],MaxLatency(us),17471
[CLEANUP],95thPercentileLatency(us),33
[CLEANUP],99thPercentileLatency(us),17471
[INSERT],Operations,501113
[INSERT],AverageLatency(us),8874.33323022951
[INSERT],MinLatency(us),343
[INSERT],MaxLatency(us),198527
[INSERT],95thPercentileLatency(us),50463
[INSERT],99thPercentileLatency(us),60447
[SCAN],Operations9498887
[SCAN],AverageLatency(us),60923.269726547966
[SCAN],MinLatency(us),270
[SCAN],MaxLatency(us),449791
[SCAN],95thPercentileLatency(us),122751
[SCAN],99thPercentileLatency(us),186879
```

Conclusion

Enterprises are beginning to adopt containers en masse, solidifying containers as the computing foundation for the next generation. Although Kubernetes is a powerful and modern platform for containerized workloads, it is full of complexity and difficult to deploy. Moreover, it is not a solution in and of itself, it requires many supporting elements to become a complete and operational platform fit for production use. Enterprises considering Kubernetes should look to remove as much complexity as possible through pre-built and commercially supported container platforms.

Early adopters used Kubernetes predominantly for stateless workloads due to its agility and simplicity. Stateful databases, such as MongoDB, are also increasingly being supported in Kubernetes, but require persistent storage and high performance I/O systems. While containers and Kubernetes can abstract away much of the infrastructure, the underlying physical I/O systems are still important, particularly for I/O heavy workloads like MongoDB. Thus, the underlying infrastructure is still an important consideration when containerizing high-performance workloads such as databases. Enterprises should also look to leverage Kubernetes operators, which will make provisioning and management of complex, stateful workloads easier.

What we have shown is that combining Kubernetes and MongoDB is now quite a valid and easy task, however, what is not so easy is the ability to drive performance out of it without having a good underlying infrastructure. We have proven through the results for Workload A and B that Diamanti is capable of delivering a big punch in a tiny package. This means you can increase density for many and more complex workloads while needing to utilize less infrastructure.

The interesting results we see from Workload E also follow the same paradigm, however, it is harder to illustrate as the coveted transactions per second performance is not able to tell the full story here. Thus, we have to dive into understanding that this workload's nature and overall idea is to scan through data, which drives high backend I/O, while only returning matching documents for that transaction. This back and forth is expensive as far as I/O is concerned (this can be seen in Figure 16). While the overall TPS was marginal, we can clearly see that the same 4 GB + 4 CPU core database is able to utilize the Diamanti Ultima advantage by processing the I/O at superfast speed.

Download the IDC lab validation report here to learn more about the key findings, including:

- A comparison of operations per second between a MongoDB cluster provisioned in the public cloud and on a Diamanti cluster
- The throughput and latency observed when running on the Diamanti platform
- The simplicity of managing MongoDB in a hybrid cloud with Diamanti