TECHNICAL WHITEPAPER

YugaByte DB Enterprise Edition Evaluation Guide

September 2018

TABLE OF CONTENTS

1. Introduction

YugaByte DB

YugaByte DB is an Apache 2.0-licensed open source database for high performance applications that require ACID transactions and planet-scale data distribution. It is purpose-built for powering fast-growing online services on public, private and hybrid clouds with transactional data integrity, low latency, high throughput and multi-region scalability while also using popular NoSQL (Cassandra & Redis) and SQL (PostgreSQL) APIs. Enterprises gain more functional depth and development agility without any cloud lock-in when compared to proprietary cloud databases such as Amazon DynamoDB, Microsoft Azure Cosmos DB or Google Cloud Spanner. Enterprises also benefit from stronger data integrity guarantees and higher performance than those offered by legacy open source NoSQL databases such as MongoDB and Apache Cassandra.

This guide helps users gain a deeper technical understanding of YugaByte DB while also highlighting key criteria for evaluating any mission-critical database such as YugaByte DB. It is applicable to the version 1.1 of YugaByte DB Enterprise Edition.

2. Redefining Operational Databases

YugaByte DB is a Cassandra, Redis & PostgreSQL (beta) compatible database with sharding, replication and distributed transactions architecture similar to that of **Google Spanner**. It redefines operational databases by bringing together three must-have needs of user-facing cloud applications, namely high performance, ACID transactions and multi-region scalability, in a single operational database. It beats distributed NoSQL databases that offer high performance and multi-region scalability but give up on transactional guarantees (because of eventual consistency). It also comes ahead of monolithic SQL databases that offer transactions and performance but are unable to scale writes across multiple nodes or geographic regions.

Multi-API & Multi-Model Transactional App Development

Modern app architectures rely on data with different modeling constraints and access patterns. Polyglot persistence, first introduced in 2011, states that each such data model should be powered by an independent database that is purpose-built for that model. The original intent was to look beyond relational/SQL databases to the emerging world of NoSQL. However, polyglot persistence comes with the hidden cost of increased complexity across the board. Using multiple databases during development, testing, release and production can be overwhelming for many organizations. While app developers must learn efficient data modeling with various database APIs, they cannot simply ignore the underlying storage engine and replication architectures for each of the databases. Operations teams are now forced to understand scaling, fault-tolerance, backup/restore, software upgrades and hardware portability for multiple databases. And of course the onerous part of "operationalizing" these databases across multiple different cloud platforms. In order to reduce the above operationalization costs, enterprises have gravitated towards to proprietary but managed database services such as Amazon RDS/DynamoDB, Azure Cosmos DB and Google Cloud Spanner. However, the end result is cloud lock-in and astronomical cloud provider bills.

Multi-model databases are on the rise to free users from the complexity of polyglot persistence. YugaByte DB takes the multi-model approach one step further by being also multi-API. It is wire compatible with three widely popular database APIs - Cassandra Query Language (CQL), Redis and PostgreSQL. Database needs are future-proofed against changing application requirements since each of the YugaByte DB APIs solve a fundamental limitation that the original API implementation suffers from—it does so by adding ACID transactions & low latency secondary indexes to Cassandra, fault-tolerance & persistence to Redis and horizontal write scalability to PostgreSQL. YugaByte DB even extends some of these APIs to increase development agility. For example, it adds a native JSON data type to CQL and a native Time Series data type to Redis.

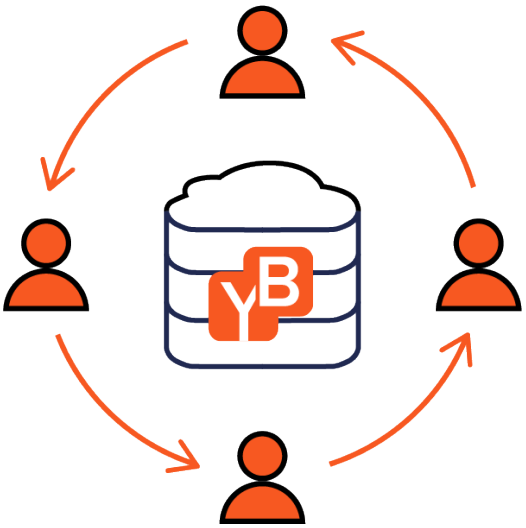
Multi-API & Multi-Model Transactional App Development

YugaByte DB is ideal for multi-zone, multi-region, hybrid cloud and multi-cloud deployment options. Users can change infrastructure choices as often as business demands—moving from single region to multi-region as well as moving from single cloud to hybrid cloud or multi-cloud are now zero downtime and zero data loss operations with no application impact. Enterprises are free to expand their business

globally anytime they choose without waiting for their database architecture to catch up.

Additionally, as a cloud native database built in the era of microservices, YugaByte DB lends itself extremely well to containerized deployments powered by Kubernetes orchestration. By automatically managing small data partitions and using strongly consistent replication to have multiple copies of such partitions readily available, a YugaByte DB cluster self heals in a few seconds in the event of any infrastructure failure (including container crashes and disk failures). This ensures that there is no noticeable impact on the overall cluster performance or availability.

Enterprise Edition vs. Community Edition



The open source YugaByte DB Community Edition (CE) is the best choice for the startup organizations with strong technical operations expertise. Such organizations would deploy YugaByte DB into production with traditional DevOps tools. The CE includes the core database engine as well as the support for all the three APIs (including the authentication/authorization models supported by these APIs). Data migration tools to help move data from existing databases are also included.

The commercial YugaByte DB Enterprise Edition (EE) includes all the features of the CE as well as additional features such read replicas, in-flight/at-rest encryption, distributed backups, multi-cloud orchestration, seamless cloud portability and comprehensive monitoring. Many of these features can be administered with the included YugaByte DB Admin Console. The EE is the simplest way to run YugaByte DB in mission-critical production environments with one or more regions (across both public cloud and on-premises datacenters).

3. How It Works?

Architecture

DocDB is YugaByte DB's Log Structured Merge tree (LSM) based storage engine and Raft is the distributed consensus protocol used for replication of data across the nodes in a cluster. A YugaByte DB cluster is comprised of two types of distributed services, YB-Master and YB-TServer. As described below, both these services are built on top of the DocDB storage engine and the Raft replication protocol. Note that sharding (aka horizontal data partitioning) and replication are automatic in YugaByte DB, so applications do not have to do anything special to leverage these features.

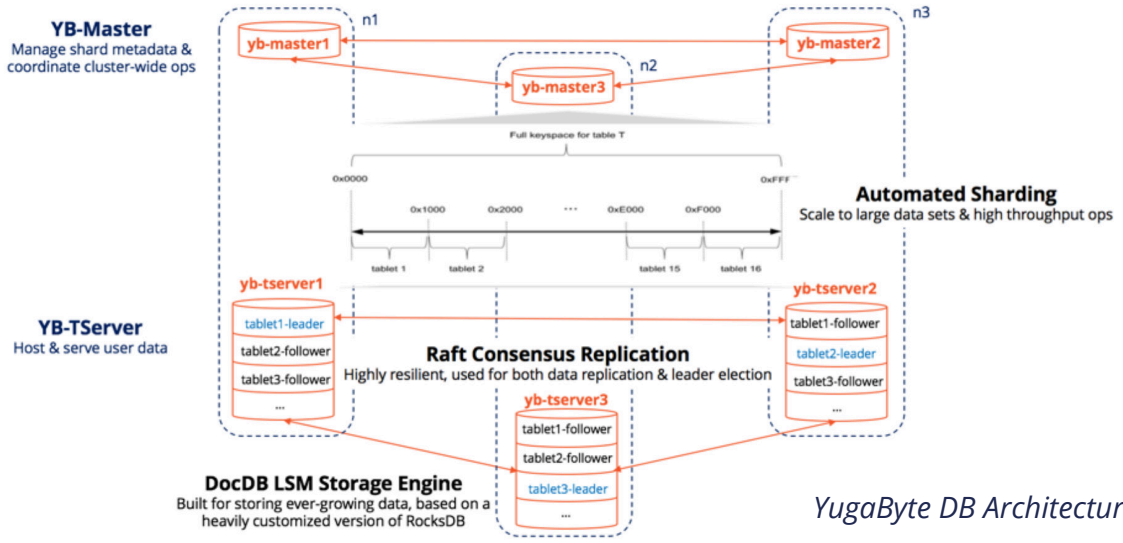
YB-Master

This service is responsible for keeping system metadata (such as shard-to-node mapping), coordinating system-wide operations (such as create/alter drop tables), and initiating maintenance operations (such as load-balancing). For increased fault tolerance, the number of YB-Masters equals the Replication Factor (RF) of the cluster. The minimum RF needed for fault tolerance is 3.

YB-TServer

This service represents the data nodes responsible for hosting/serving user data in shards (also known as tablets). The number of data nodes can be increased or decreased on-demand in a cluster.

Example of a 3 Node Cluster



YugaByte DB Architecture

The figure above highlights the architecture of a 3-node RF3 YugaByte DB cluster with 3 instances each of YB-Master and YB-TServer. User and system tables are auto-sharded into multiple tablets. Each tablet has a Raft group of its own (called as tablet-peers) with one leader and a number of followers equal to RF-1. Raft is used for leader election and data replication in both YB-Master and YB-TServer. Once data is replicated via Raft across a majority of the tablet-peers, it is applied to each tablet peer's local DocDB.

DocDB Storage Engine

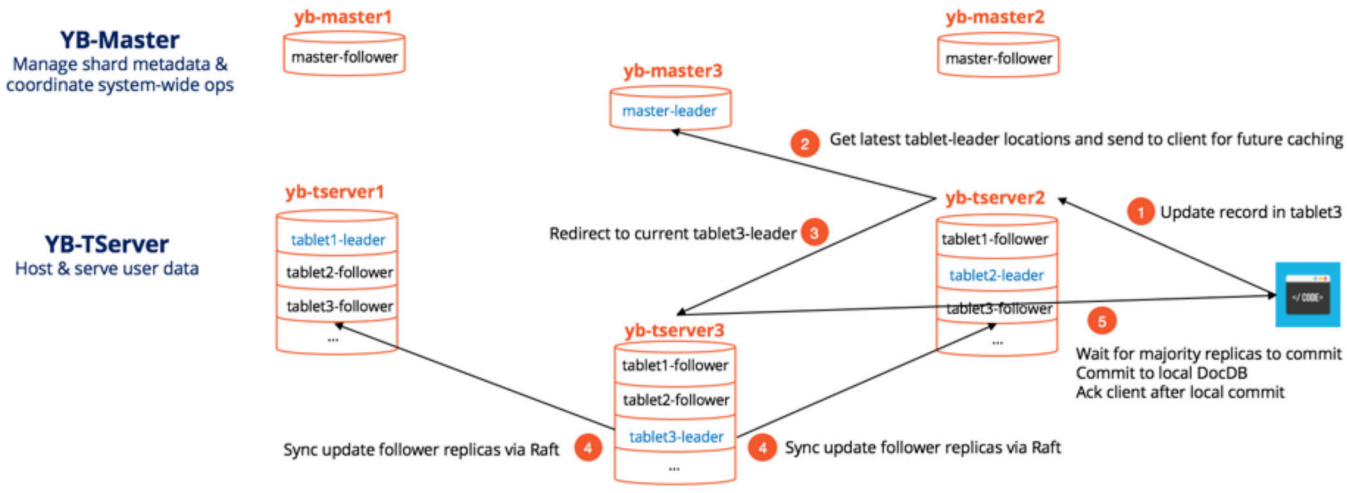
Built on a highly customized version of RocksDB, DocDB is a key-to-document storage engine that can store both primitive and complex data types for high volume data applications. The keys in DocDB are compound keys consisting of 1 or more hash organized components, followed by 0 or more ordered (range) components. These components are stored in their data type specific sort order; both ascending and descending sort order are supported for each ordered component of the key. This model allows multiple levels of nesting, and corresponds to a JSON-like format. More details about DocDB architecture can be found in YugaByte DB Docs.

Raft Distributed Consensus-Based Replication

As described in "How Does Consensus-Based Replication Work in Distributed Databases?", Raft has become the consensus replication algorithm of choice when it comes to building resilient, strongly consistent systems.

Strong Consistency with Zero Data Loss Writes

Raft enables single-row, single-operation ACID in YugaByte DB. Thereafter, YugaByte DB builds on this foundation to implement distributed ACID transactions (as described later).



Zero Data Loss Write Path

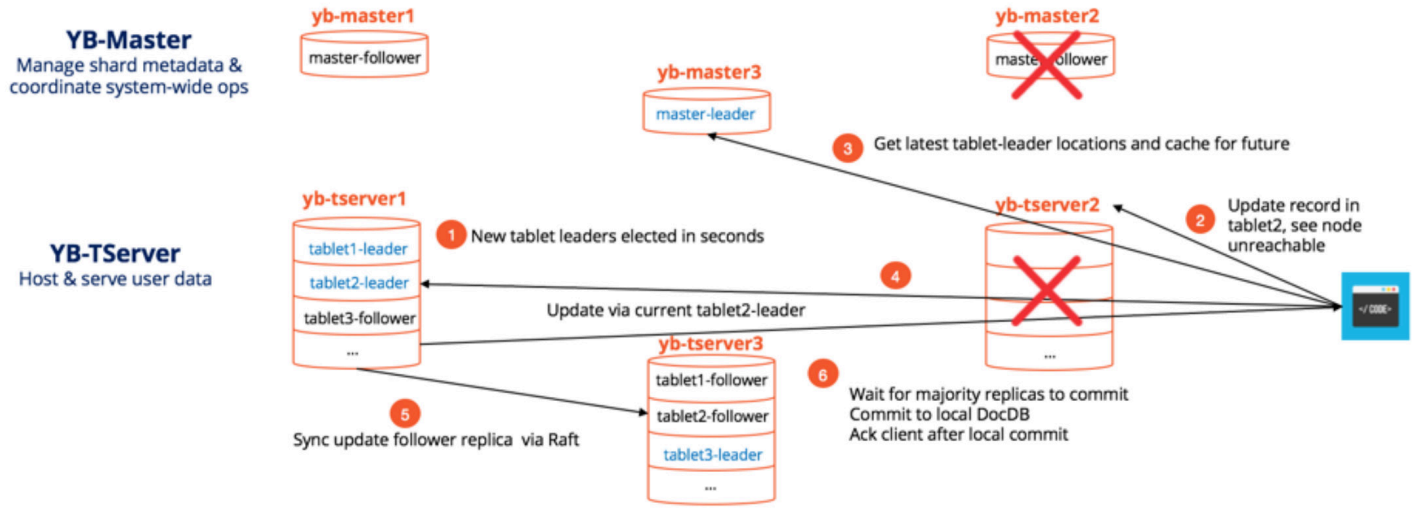
Here's how a single-row write operation works in YugaByte DB:

1. Let's say an application client wants to update a row in tablet3 and yb-tserver2 is where this request lands for the first time.
2. It gets automatically redirected to the current master-leader, yb-master3 to get the node location where tablet3's leader resides. This location is now cached in the client driver so that future requests do not involve the master-leader.
3. The actual request now gets redirected to the node that has tablet3-leader, yb-tserver3.
4. yb-tserver3 appends this update to its own Raft log and replicates it to the Raft log of the two followers. It waits for one follower to successfully commit in addition to itself.
5. yb-server3 now commits the update in its own local DocDB and acks the client noting the update as successful.

The end result is that the update above is now available for all future reads (strongly consistent reads from leader and timeline consistent reads from followers) and is also resilient against failure of one node or disk (see next section on Continuous Availability). Resilience against more failures is achieved by simply increasing the RF from three to five or seven.

Continuous Availability with Self-healing Leader Election

Raft enables single-row, single-operation ACID in YugaByte DB. Thereafter, YugaByte DB builds on this foundation to implement distributed ACID transactions (as described later).



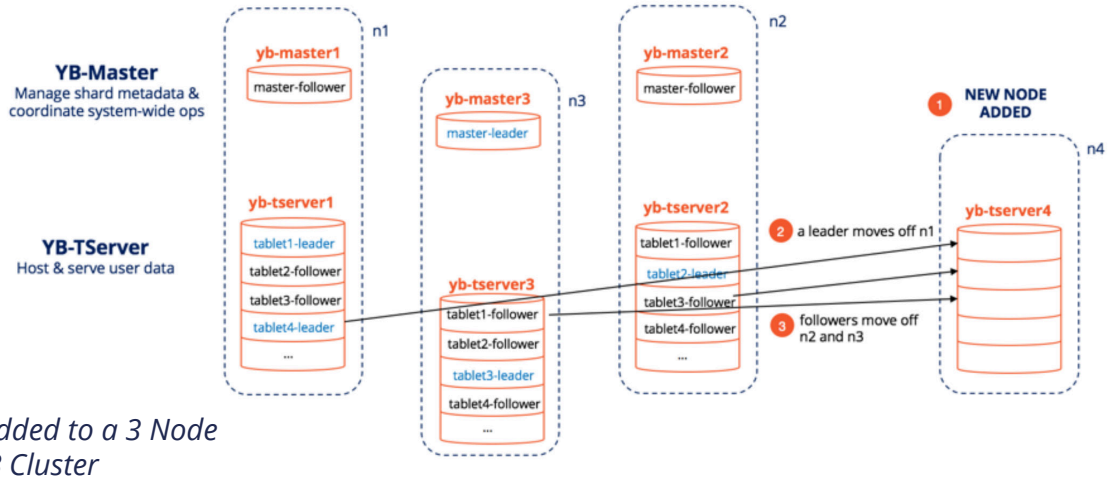
Continuous Availability Even Under Node/Disk/Network Failures

1. Assume node2 crashes and as a result we lose both yb-master2 and yb-tserver2. This means tablet2-leader, located on yb-tserver2, is also no longer available. This leads to the automatic leader election for tablet2 among the two remaining replicas. Let's assume that the tablet2 follower on node1 now becomes the tablet2-leader. The master-leader is updated of this change. Note that tablet1 and tablet3 that lost only followers because of node2 death had no impact on their availability.
2. App client now tries to update a record in tablet2 but finds yb-tserver2 unreachable.
3. It finds the latest location for tablet2-leader from master-leader and caches the information.
4. Update request is sent to tablet2-leader on node1.
5. tablet2-leader now synchronously replicates this update to tablet2-follower on node3, the only follower that's alive.
6. After tablet2-follower acknowledges the update, tablet2-follower updates its own record and acks the client noting the update as successful.

So the loss of one node in a RF3 cluster leads a very short write unavailability (of ~3 seconds) during which leader election for the impacted tablets takes place. This is by design since accepting new writes on those tablets can lead to data loss (since there are not enough replicas available for quorum). The system continues as normal after the leader election completes and tablets rebalance to the old node whenever it comes back.

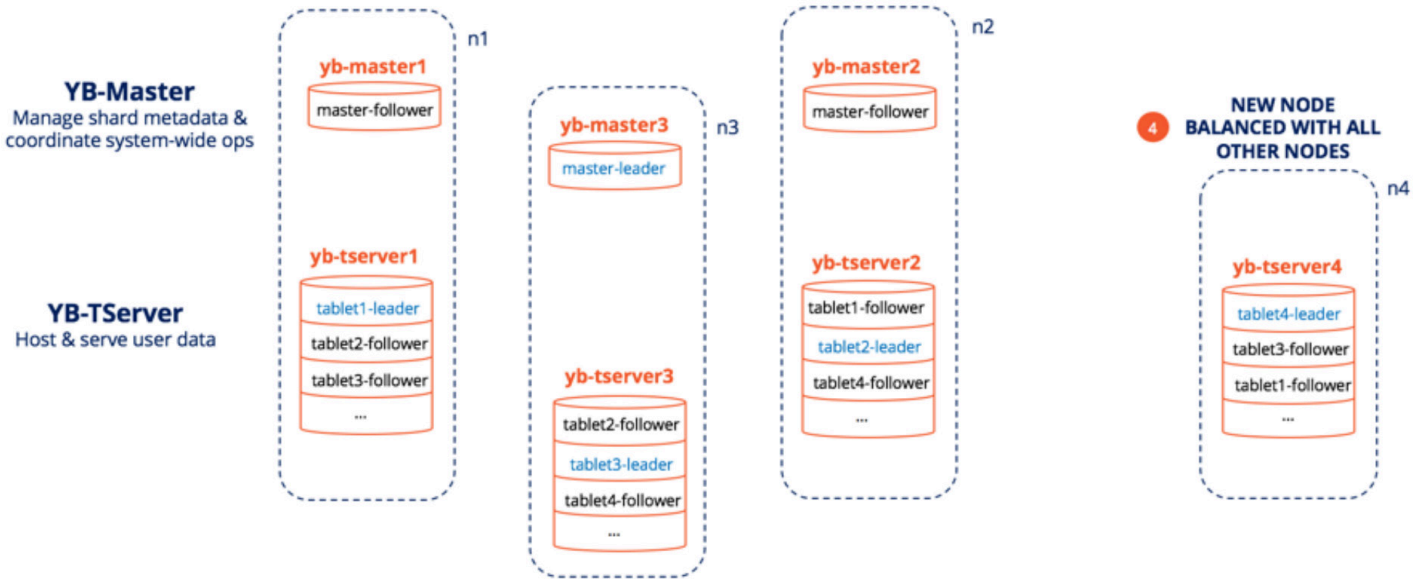
Rapid Scale-out and Scale-in with Auto-Rebalancing

Raft enables dynamic membership changes with ease. Removal of nodes boils down to re-running Raft leader election for only those tablets that are impacted followed by a YB-Master-led re-creation of replicas to achieve full replication. Addition of nodes becomes a leader-and-follower tablet rebalancing task initiated by the YB-Master. Let's review what exactly happens when a new node is added to a cluster that has three nodes with four tablets. The node n1 has two leaders, tablet1-leader and tablet4-leader.



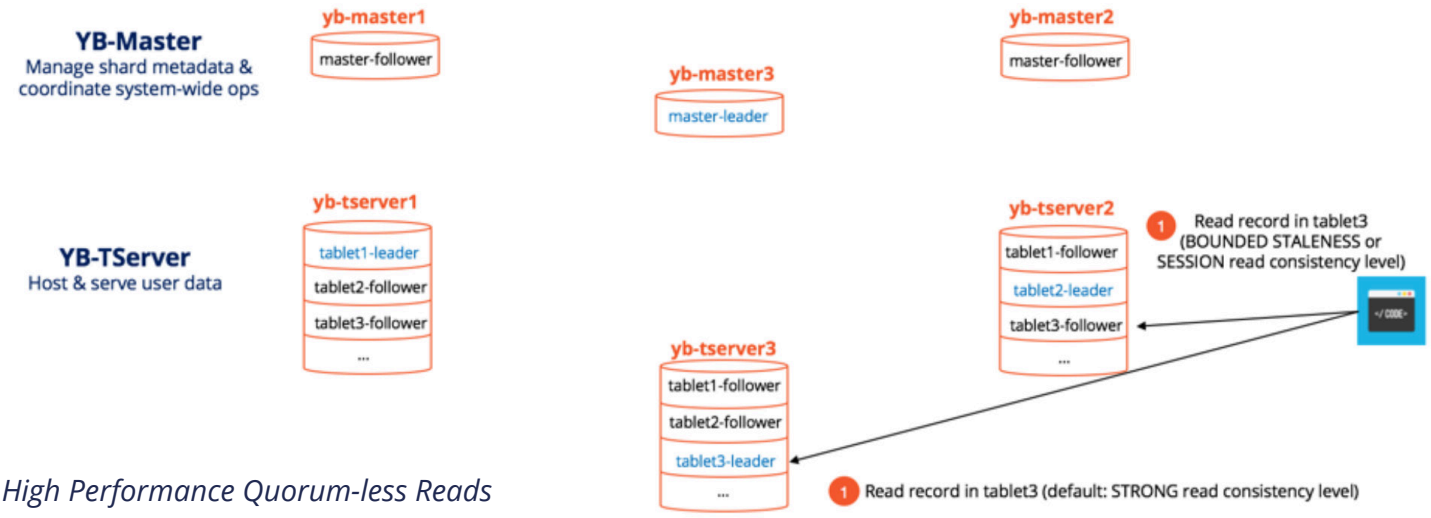
New Node Added to a 3 Node YugaByte DB Cluster

1. New node n4 is added to the cluster.
2. The master-leader is aware of this change to the cluster and initiates re-balancing operations which involves a leader (say tablet4-leader) moving off n1. Note that this move may involve tablet4-leader to temporarily give up its leadership to one of the other replicas and then taking it back on after the tablet4 becomes available at n4.
3. For a perfectly balanced cluster, some followers will also move from n2 and n3 to n4. Note that all the moves are undertaken in such a way that no single existing node bears the burden of populating the new node -- all nodes in the cluster chip in with their fair share and as a result the cluster never comes under stress.
4. The figure below shows the final, perfectly balanced cluster where each node has one leader and three followers.



A Perfectly Balanced 4 Node YugaByte DB Cluster

High Performance with Quorumless Reads and Tunable Latency



High Performance Quorum-less Reads

Given the difficult task of ensuring consistency is already completed at write time, YugaByte DB's Raft implementation ensures that read requests can be served with extremely low latency without using any quorum (where other replicas have to be consulted). Additionally, it allows the app client to choose from reading the leader (for strongly consistent reads) or from the followers (for timeline-consistent reads). YugaByte DB's Enterprise Edition even allows reading from Read Replicas which are asynchronously updated from the leaders/followers and do not participate in the write path. Reading from followers and read replicas allows the system to increase throughput significantly since there are more followers than leaders and can even reduce latency if the leader happens to be in a different region.

Distributed ACID Transactions

ACID transactions are a fundamental building block when developing business-critical, user-facing applications. They simplify the complex task of ensuring data integrity especially under highly concurrent workloads. While they are taken for granted in monolithic SQL/relational DBs, distributed NoSQL/non-relational DBs usually forsake them completely. MongoDB is an exception in the sense that it supports a restrictive single row/shard ACID option but does not support the fully distributed/multi-shard ACID option.

Implementing distributed ACID transactions in distributed databases requires the use of a transaction manager that can coordinate the various operations and then commit/rollback the transaction as needed. Popular NoSQL databases are designed to avoid this additional complexity in the fear that they will have to compromise on performance (in the form of increase in write latency and decrease in linear scalability). On the other hand, every YugaByte DB data node has a built-in transaction manager that is engineered to efficiently detect and optimally handle different scenarios involving single row, single shard and distributed ACID transactions without compromising performance. **Yes We Can! Distributed ACID Transactions with High Performance** details the various optimizations used to deliver low latency and high throughput transactions.

4. Key Evaluation Criteria

Following are the criteria that should be used to evaluate any mission-critical operational database including YugaByte DB Enterprise Edition. The remaining sections go into the details of each of these.

Cloud Native Operational Experience

YugaWare, the YugaByte DB Admin Console included in the Enterprise Edition, is a turnkey multi-cloud, multi-cluster management plane for YugaByte DB. It is deeply integrated with the infrastructure automation services of public clouds such as Amazon Web Services and Google Cloud (Microsoft Azure is on the roadmap). The net result is the ability to instantly spin up/down/expand YugaByte DB clusters as well as take distributed backups with a cloud native operational experience that was previously available only for proprietary cloud services. The Admin Console also offers integration for on-premises datacenter environments and Kubernetes deployments.

Performance Testing

YugaByte DB performance tests can be executed against either a suite of pre-built sample apps that ship in YugaByte DB or through the industry standard **YCSB benchmark**. The sample apps include multiple key-value workloads that are either read-heavy or write-heavy or batch writes. The YCSB benchmark ships with **6 core workloads** that cover the spectrum from write-heavy workloads to read-modify-write workloads.

Multi-Model Capabilities

In addition to the Cassandra compatible API, YugaByte DB also offers a Redis compatible API backed by all the same goodness of common database engine. This API essentially enables YugaByte DB to be used as a distributed transactional key-value store.

Cluster Expansion

YugaByte DB is architected to ensure that adding nodes to an existing cluster irrespective of the current size of the cluster should be a fully online operation with zero application impact.

Multi-Region Deployments

YugaByte DB ensures that requests for strongly consistent reads are served from the tablet leader while requests for follower reads are served from the followers in the nearest datacenter.

Since strongly consistent reads can have higher latency if the tablet leader is located in a different region, latency sensitive applications can choose to read from nearby followers at a lower latency by going for timeline consistency as opposed to strong consistency.

Preferred Region Placement and Continuous Balancing

YugaByte DB has the ability to place the tablet leaders only in specific AZs of specific regions. If a workload depends on strongly consistent reads from a preferred region, then YugaByte DB can ensure that those reads can be served with the absolute low latency. This configuration can be changed even on a running cluster thus accounting for the fact that workloads change their access pattern with time.

Node/Zone/Region Failure Testing

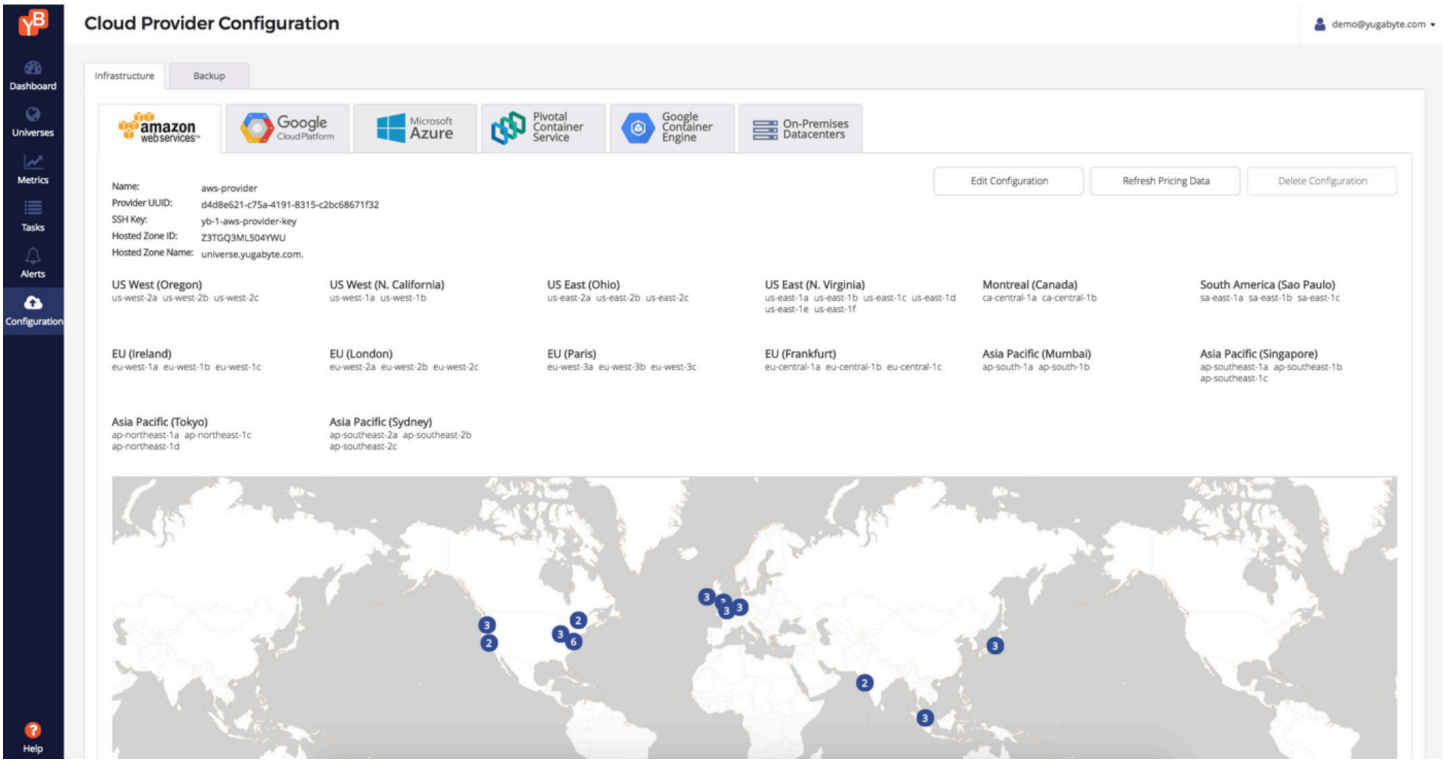
Infrastructure failures such as machine crashes, disk failures, network partitions and clock skews are bound to happen when using today's commodity hardware. As a highly fault tolerant database, YugaByte DB protects the cluster against data loss and self heals quickly to ensure continuous availability. Common tests in this area are testing a node/AZ failure in a single region deployment and testing a single region failure in a multi-region deployment.

5. Installing YugaByte DB Enterprise Edition

First, we will install the YugaByte DB Admin Console application using instructions from YugaByte DB Docs. We can then administer YugaByte DB EE clusters with extreme ease on any public or private cloud. A dedicated machine separate from the machines used for the DB cluster is needed to run this application. Both Internet-connected and airgapped installation options are supported for the Admin Console.

6. Configuring Multiple Cloud Providers

After the Admin Console is installed, we can configure any number of cloud providers so that YugaByte DB clusters can be created and dynamically managed on these cloud providers. As you can see below, major public cloud platforms as well as major Kubernetes distributions are supported. Even on-premises datacenters can be configured. The Admin Console does not manage the underlying VM/bare metal instances in case of on-premises datacenters and is simply responsible for managing the DB cluster on those instances.



7. Performance Testing

As a strongly consistent distributed database, a fault-tolerant YugaByte DB cluster starts with a minimum of 3 nodes and a Replication Factor of 3. These nodes can be running on machines with either Centos 7.x or RHEL 7.x. Either local or remote-attached disks should be mounted on these machines. A complete system configuration for performance testing is highlighted below.

Note that single node local clusters can be used to learn more about the database but are not recommended for any functional test or performance test environments. A complete checklist for deploying YugaByte DB is available here.

Recommended System Configuration

Amazon Web Services

Instance type: i3.4xlarge
Disks: 2 x 1.9 TB NVMe SSDs (comes preconfigured with the instance)

Google Cloud Platform

Instance type: n1-standard-16
Disks: 2 x 375GB SSDs

On-Premises Datacenter

Instance: 16 CPU cores
Disk size: 1 x 200 GB SSD (minimum)
RAM size: 30GB (minimum)

Recommended System Configuration

Login to the machine from which you are going to run your load-test.
ssh -i ~/.ssh/<your-key>.pem centos@<load-tester-ip>

Install java and wget then folow the instructions from https://docs.yugabyte.com/latest/quick-start/install/#download to download yugabyte.
Then, the sample apps will be in the java/yb-sample-apps.jar

Note: Assume $ENDPOINTS are the IP addresses (plus port) for the nodes, e.g. for YCQL:
ENDPOINTS="172.151.20.150:9042,172.151.23.97:9042,172.151.24.41:9042"
To get them go to the Resource Info widget on the main universe page:

Write-heavy KV workload

Run the key-value workload with higher number of write threads (representing write-heavy workload).

Load 1B keys of 256 bytes each across 256 writer threads

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--nouuid --value_size 256 --num_threads_read 0 --num_threads_write  256 --num_
unique_keys 1000000000
```

Expected Results

(Read) Ops/sec: ~90k
(Read) Latency: ~2.5/3.0 ms/op
CPU (User + Sys): 60%

Read-heavy KV workload

Run the key-value workload with higher number of read threads (representing read-heavy workload).

Load 1M keys of 256 bytes and access them with 256 reader threads

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--nouuid --value_size 256 --num_threads_read 256 --num_threads_write 0 --num_
unique_keys 1000000
```

Expected Results

(Write) Ops/sec: ~150k
(Write) Latency: ~1.66 ms/op
CPU (User + Sys): 60%

Batch Write-heavy KV workload

Run the key-value workload in batch mode and higher number of write threads (representing batched, write-heavy workload).

Load 1B keys of 256 bytes each across 64 writer threads in batches of 25 each

java -jar java/yb-sample-apps.jar --workload CassandraBatchKeyValue --nodes $ENDPOINTS --nouuid --batch_size 25 --value_size 256 --num_threads_read 0 --num_threads_write 64 --num_unique_keys 1000000000

Expected Results

(Batch Write) Ops/sec: ~140k
(Batch Write) Latency: ~9.0 ms/op
CPU (User + Sys): 80%

To view live metrics go to the Metrics tab on the Universe page:

Testing with YCSB Benchmark

Setup YCSB and configure it to use YugaByte-Cassandra Driver

Clone the YCSB repository

cd $HOME
git clone https://github.com/brianfrankcooper/YCSB.git
cd YCSB

Use YugaByte-Cassandra Driver

1. In pom.xml change the line:
<cassandra.cql.version>3.0.0</cassandra.cql.version>
to the latest version of the YugaByte-Cassandra driver:
<cassandra.cql.version>3.2.0-yb-17</cassandra.cql.version>

***Note**: You can (and probably should) always check **Maven** to find the latest version.*

2. In cassandra/pom.xml change the line:
<groupId>com.datastax.cassandra</groupId>
to
<groupId>com.yugabyte</groupId>

Build YCSB and Cassandra binds

mvn -pl com.yahoo.ycsb:cassandra-binding -am clean package -DskipTests

***Note**: For more information about YCSB see:*

1. *YCSB Wiki: **https://github.com/brianfrankcooper/YCSB/wiki***
2. *Workload info: **https://github.com/brianfrankcooper/YCSB/wiki/Core-Workloads***

Setup YugaByte-cqlsh

cd $HOME
git clone https://github.com/YugaByte/cqlsh

Run YCSB Tests

1. Create an executable file in the YCSB folder:

cd $HOME/YCSB

touch run-yb-ycsb.sh

chmod a+x run-yb-ycsb.sh

2. Copy the following contents in run-yb-ycsb.s:

***Note**: You may want to change the highlighted values below with the correct/intended ones for your setup.*

Run YCSB Tests

1. Create an executable file in the YCSB folder:

cd $HOME/YCSB

touch run-yb-ycsb.sh

chmod a+x run-yb-ycsb.sh

2. Copy the following contents in run-yb-ycsb.s:

***Note**: You may want to change the highlighted values below with the correct/intended ones for your setup.*

```
#!/bin/bash


# YB-CQL host (any of the yb-tserver hosts)
# (The other nodes should get automatically discovered by the driver).
hosts=127.0.0.1
ycsb=$HOME/YCSB/bin/ycsb
cqlsh=$HOME/cqlsh/bin/cqlsh
ycsb_setup_script=$HOME/YCSB/cassandra/src/test/resources/ycsb.cql
keyspace=ycsb
table=usertable
# See https://github.com/brianfrankcooper/YCSB/wiki/Core-Properties for param descriptions
params="-p recordcount=1000000 -p operationcount=10000000"

setup() {
  $cqlsh <<EOF
create keyspace $keyspace with replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
EOF
  $cqlsh -k $keyspace -f $ycsb_setup_script
}

cleanup() {
  $cqlsh <<EOF
drop table $keyspace.$table;
drop keyspace $keyspace;
EOF
}

delete_data() {
  $cqlsh -k $keyspace <<EOF
drop table usertable;
EOF
  $cqlsh -k $keyspace -f $ycsb_setup_script
}


run_workload() {
  local workload=$1
```

```
echo ========================== $workload ==========================
  $ycsb load cassandra-cql -p hosts=$hosts -P workloads/$workload $params \
    -p threadcount=40 | tee $workload-load.dat
  $ycsb run cassandra-cql -p hosts=$hosts -P workloads/$workload $params \
    -p cassandra.readconsistencylevel=QUORUM -p cassandra.writeconsistencylevel=QUORUM \
    -p threadcount=256 -p maxexecutiontime=180 | tee $workload-transaction.dat
  delete_data
}

setup

run_workload workloada
run_workload workloadb
run_workload workloadc
run_workload workloadd
run_workload workloade
run_workload workloadf

cleanup
```

We use YugaByte DB with **strongly consistent reads and writes**, which corresponds, in Cassandra, to using the QUORUM option for both cassandra.readconsistencylevel and cassandra.writeconsistencylevel (see the command above).

Run workload & check results

Simply run the script above:

./run-yb-ycsb.sh

***Note**: You may want to change the highlighted values below with the correct/intended ones for your setup.*

Results for each workload will be in workload[abcdef]-transaction.dat (e.g. workloada-transaction.dat)

8. Multi-Model Capabilities

We will run a Redis Key-Value workload on the same 3-node, 16-cpu cluster. Note that YugaByte DB's Redis compatible API called YEDIS is powered by the same unified database kernel/product and hence allows better durability, scalability and fault tolerance than the standalone open source Redis.

Note: Assume $ENDPOINTS are the IP addresses (plus port) for the nodes, e.g. for YEDIS:
ENDPOINTS="172.151.20.150:6379,172.151.23.97:6379,172.151.24.41:6379"
To get them go to the Resource Info widget on the main universe page:

Read-heavy workload

Run the key-value workload with higher number of write threads (representing write-heavy workload).

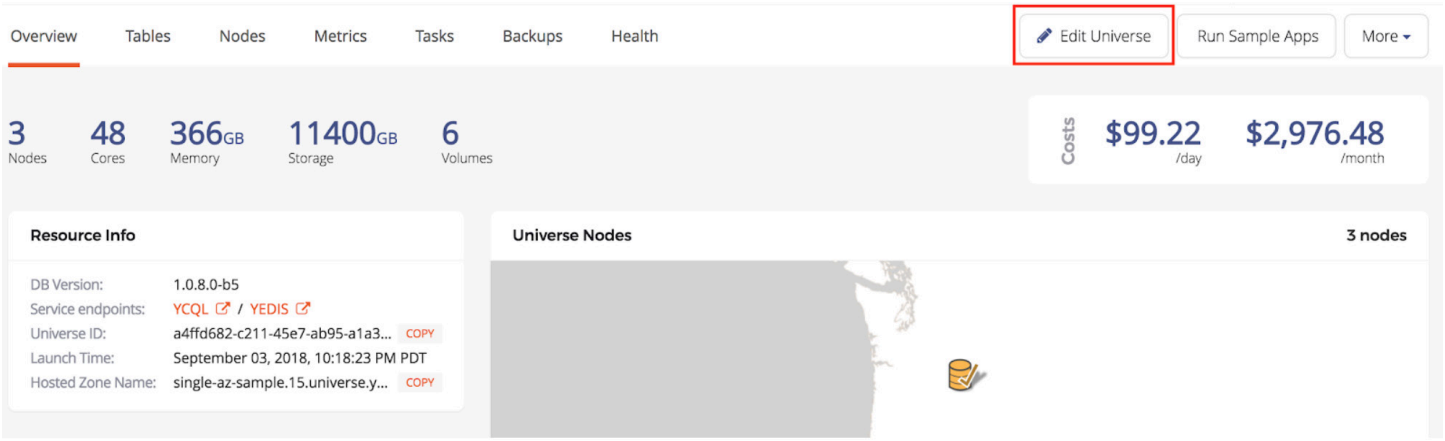
java -jar java/yb-sample-apps.jar --workload RedisKeyValue --nodes $ENDPOINTS
--num_threads_read 128 --num_threads_write 16

Expected Results
Read Ops/sec: ~100k
Read Latency: ~1.25 ms/op
Write Ops/sec: ~6k
Write Latency: ~2.70 ms/op
CPU (User + Sys): 65%

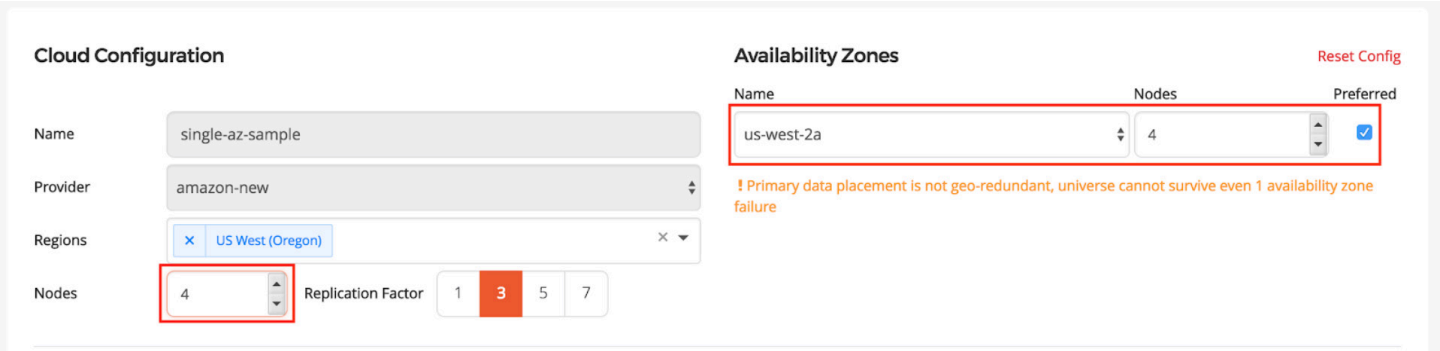
9. Cluster Expansion

Universe Expand Operation

Select “Edit Universe” and increase number of nodes to the desired value. Complete documentation for all aspects of Edit Universe are available in YugaByte DB Docs.

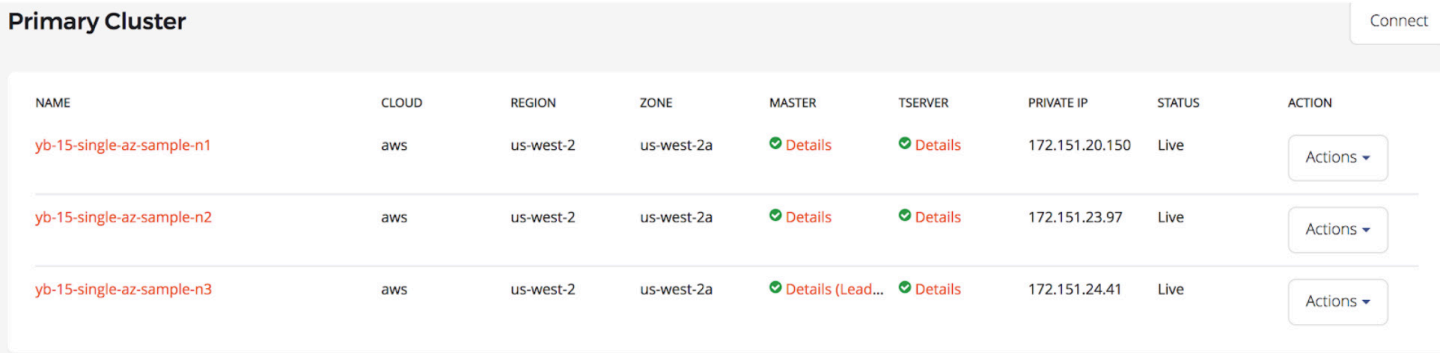


Under the "Cloud Configuration" section, increase the Nodes count as desired.

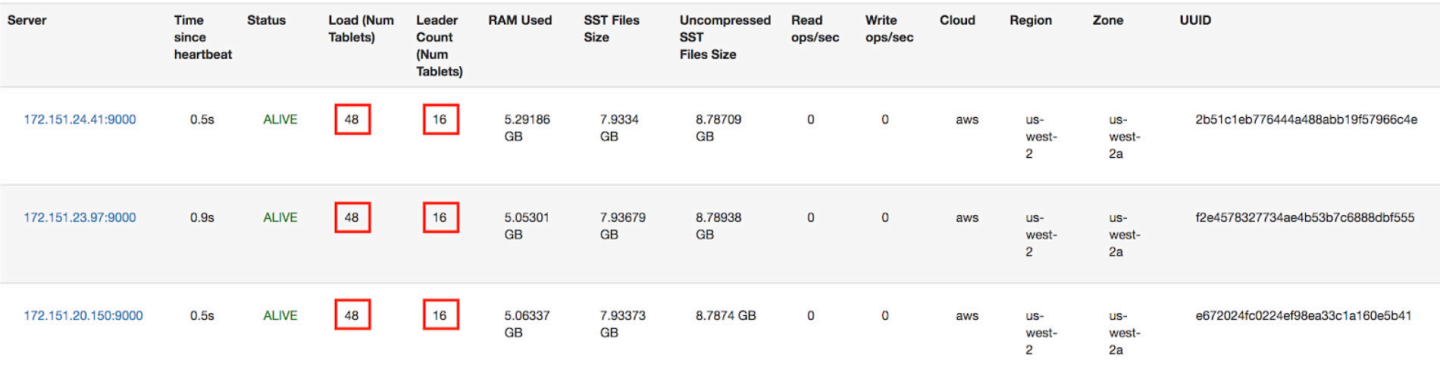


Double-check the node distribution in the "Availability Zones" section is the expected (for example testing a single-zone Universe).

Then, on the Universe page, go to the Nodes tab:



Click on the link for the Master leader -- "Details (Leader)" in the MASTER column -- and then on the Tablet Servers link in the master web UI.



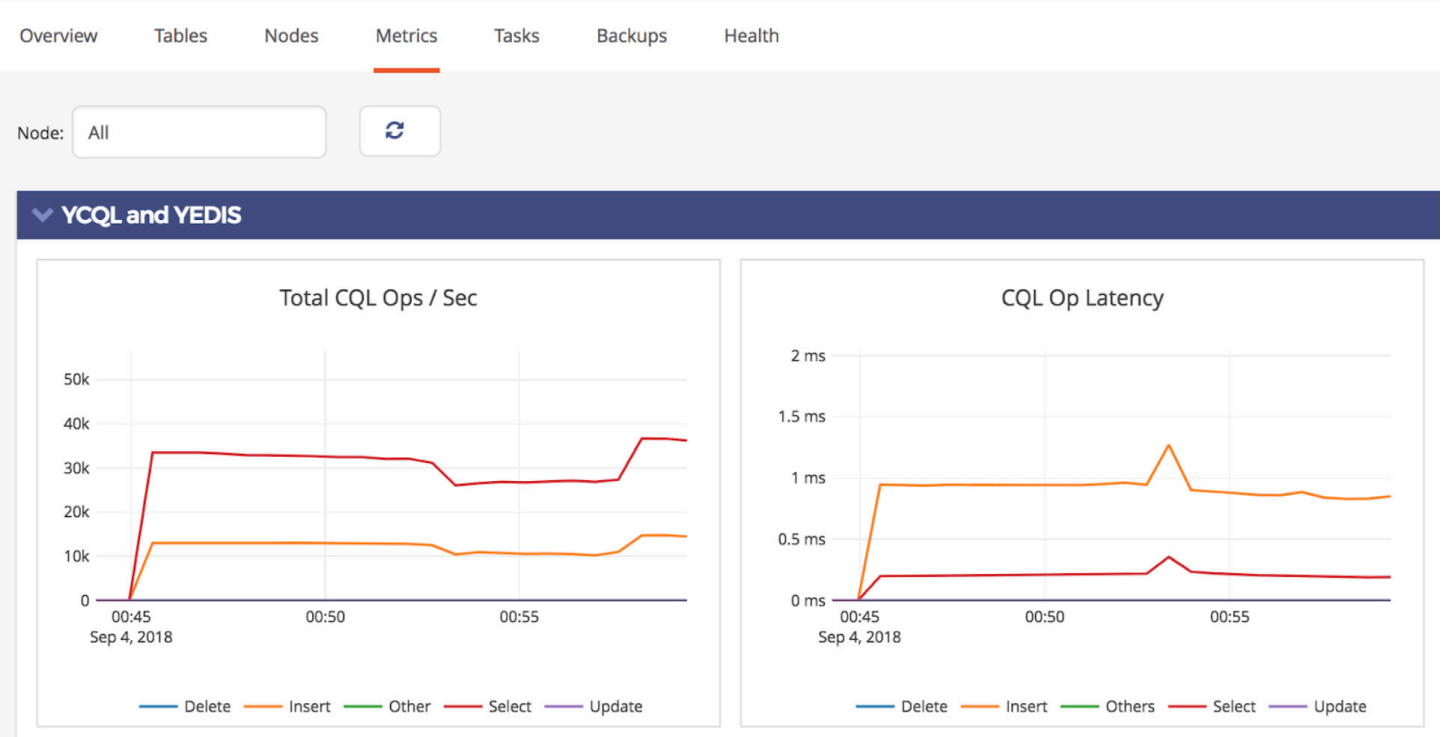
Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.24.41:9000	0.5s	ALIVE	48	16	5.29186 GB	7.9334 GB	8.78709 GB	0	0	aws	us-west-2	us-west-2a	2b51c1eb776444a488abb19f57966c4e
172.151.23.97:9000	0.9s	ALIVE	48	16	5.05301 GB	7.93679 GB	8.78938 GB	0	0	aws	us-west-2	us-west-2a	f2e4578327734ae4b53b7c6888dbf555
172.151.20.150:9000	0.5s	ALIVE	48	16	5.06337 GB	7.93373 GB	8.7874 GB	0	0	aws	us-west-2	us-west-2a	e672024fc0224ef98ea33c1a160e5b41

The example image above shows a 3-node universe with each TServer having 48 tablets or shards (48 * 3 = 144 tablets). This means 48 tablets (16 of which are leaders) per node. This corresponds to 36 tablets (and 12 leaders) per node in a 4-node universe.

Cluster During Expansion

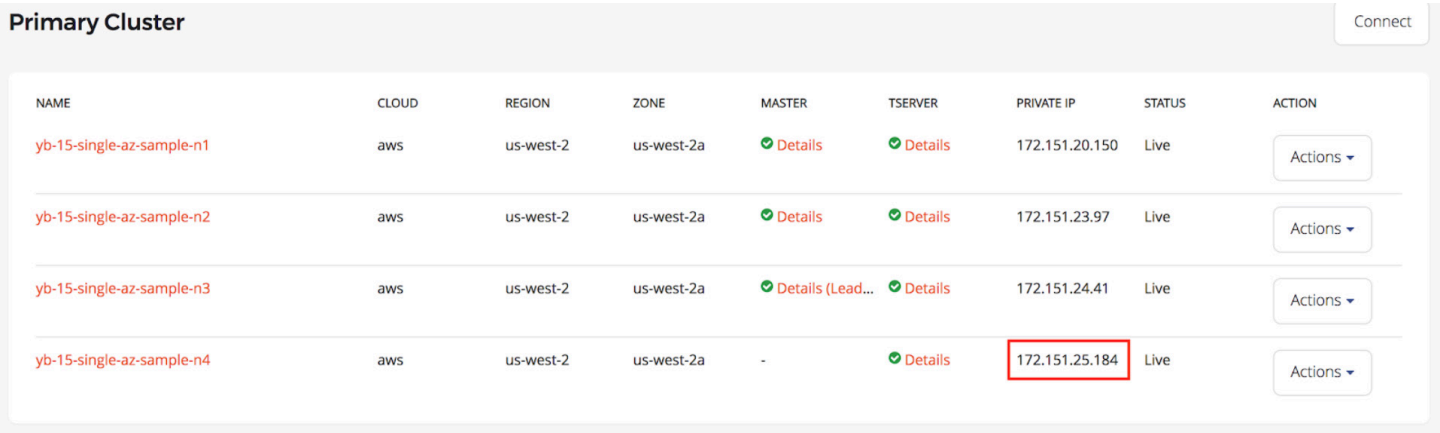
The expand operation should take under 10 minutes; but this is a fully online operation with the cluster still taking traffic. The progress of various steps is reported in the control plane. Once the node is provisioned, and software is installed and configured, the tablets are moved automatically in a throttled and safe manner all in an online manner.

In the image below you can see a slight dip in ops/sec during the expansion with the number increasing after the expand operation is done.



Cluster After Expansion

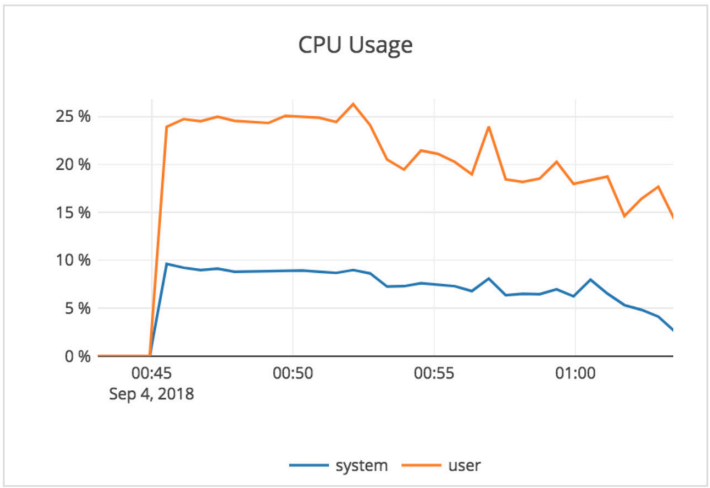
After expansion is done the new node should appear Live on the Nodes tab (note the IP of the new node):

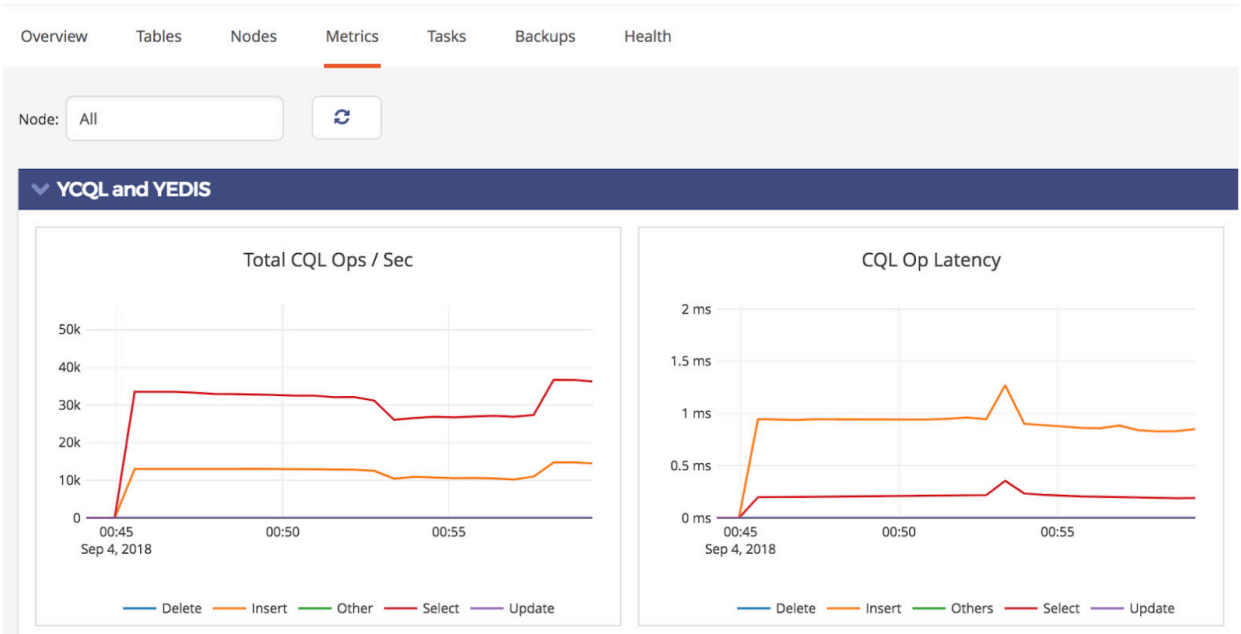


View of tablets (shards) on each tablet server shows that those 144 tablets are now spread evenly across the 4 nodes (so 36 tablets, 12 tablet leaders per node):

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.25.184:9000	0.2s	ALIVE	36	12	4.75689 GB	8.45235 GB	9.70047 GB	9151.04	3667.13	aws	us-west-2	us-west-2a	850d0e6f62b84eb1abce452ee6bca8f5
172.151.24.41:9000	0.7s	ALIVE	36	12	9.69899 GB	6.51135 GB	7.46701 GB	9135.48	3679.97	aws	us-west-2	us-west-2a	2b51c1eb776444a488abb19f57966c4e
172.151.23.97:9000	0.4s	ALIVE	36	12	8.15033 GB	6.22323 GB	7.23817 GB	9181.08	3695.89	aws	us-west-2	us-west-2a	f2e4578327734ae4b53b7c6888dbf555
172.151.20.150:9000	0.3s	ALIVE	36	12	8.78263 GB	5.91957 GB	6.89302 GB	9235.63	3703	aws	us-west-2	us-west-2a	e672024fc0224ef98ea33c1a160e5b41

As a result of the expand, you can see the drop in CPU-per-node & latency while the IOPS cluster wide increase:





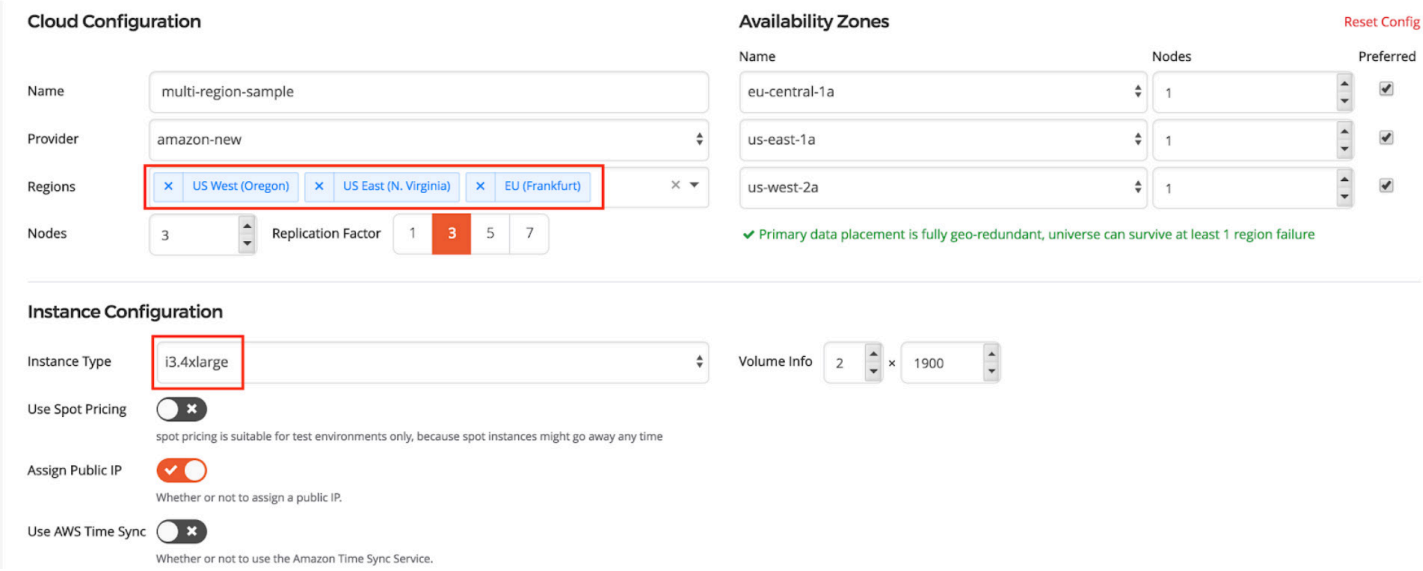
10. Multi-Region Deployments

We will now create a new cluster which has nodes spanning multiple regions. Complete documentation regarding multi-region deployments is available in YugaByte DB Docs.

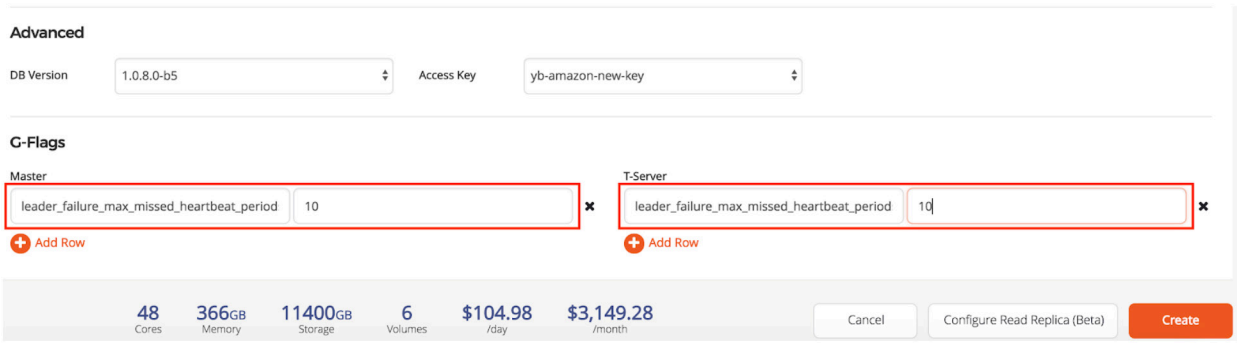
Following is the configuration of the multi-region universe we will create:

- Regions: (US West) Oregon, (US East) N. Virginia, (Europe) Frankfurt
- Nodes: 3
- Machine Type: i3.4xlarge

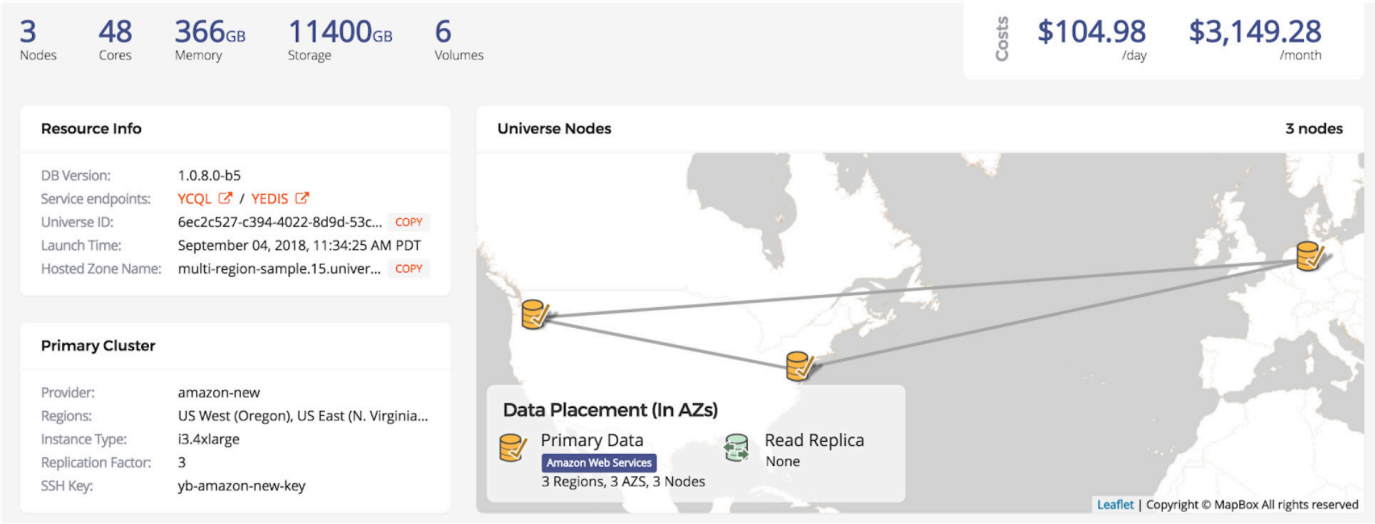
Create a Universe as before but add the 3 regions (US West) Oregon, (US East) N. Virginia, (Europe) Frankfurt in the Regions field under Cloud Configuration. We will use the same node type as before (i3.4xlarge).



Make sure to add the following G-Flag for both YB-Master and YB-TServer: leader_failure_max_missed_heartbeat_periods = 10. Since the the data is globally replicated, RPC latencies are higher. We use this flag to increase the failure detection interval in such a higher RPC latency deployment. See the screenshot below.



Then click Create as usual -- once the universe is created the overview page should look like below:



Workload Description

Running a workload representing users with their profile, modeled as a simple Cassandra KV table. This **workload is "read-heavy"** (representing users logging in, and hence need read access to the profile; it is fairly light in terms of writes - which correspond to someone updating their profile).

***Note**: Connect to the load-tester and get the (YCQL) ENDPOINTS exactly as in the single-az case above.*

Write-only workload

Load 1M keys (which will also be used by the read-heavy workloads below):

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--num_threads_write 256 --num_threads_read 0 --value_size 128 --num_unique_keys
1000000 --nouuid --with_local_dc us-west-2
```

Expected Results

(Write) Ops/Sec: 1219.93 ops/sec
(Write) Latency: 210.18 ms/op
CPU Usage <5% (Should be bottlenecked by latency).

The high write latencies are expected because of a multi-region setup. Such a setup naturally depends on the round-trip time between the regions since YugaByte DB uses distributed consensus on writes.

Strongly Consistent Reads Workload

Yugabyte reads are strongly consistent by default. In this case they, the leaders will be evenly distributed across the 3 nodes so read queries should be evenly distributed accordingly.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--num_threads_write 8 --num_threads_read 64 --value_size 128 --num_unique_keys
1000000 --nouuid --with_local_dc us-west-2
```

Expected Results

(Read) Ops/Sec: 851.34
(Read) Latency: 75.30 ms/op
(Write) Ops/Sec: 45.40 ops/sec
(Write) Latency: 175.94 ms/op
CPU Usage <5% (Should be bottlenecked by latency).

The high write latencies are expected because of a multi-region setup. Such a setup naturally depends on the round-trip time between the regions since YugaByte DB uses strongly consistent reads by default and the tablet leaders are distributed evenly over all nodes where we expect one third of the reads to go to each node (see also consistent prefix reads as well as preferred region feature below).

Consistent Prefix Reads Workload

We can set the read consistency to consistent prefix (allowing reads from follower tablets) by using the --local_reads flag. Since this a 3-node RF3 cluster, the local (US West) node will be able to answer all queries so we expect local AZ latency -- since the load tester is in the same AZ.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--num_threads_write 8 --num_threads_read 64 --value_size 128 --num_unique_keys
1000000 --nouuid --with_local_dc us-west-2 --local_reads
```

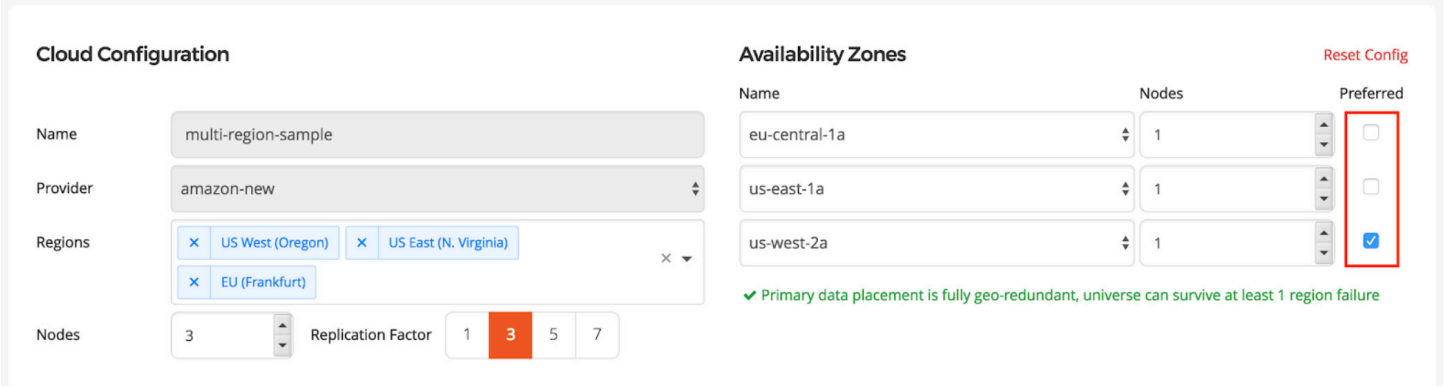
Expected Results
(Read) Ops/Sec: ~66k
(Read) Latency: 0.95 ms/op
(Write) Ops/Sec: ~32 ops/sec
(Write) Latency: ~240ms/op
CPU Usage ~22%

The high write latencies remain because writes are still strongly consistent (require quorum).

11. Preferred Region Placement and Continuous Balancing

In this workload, we are also using YugaByte "preferred region" capabilities where we can specify that "Oregon" (us-west-2) as the preferred region to keep the tablet leaders for various shards.

See below, how in the "universe specification" we have under the "Preferred" column next to us-west-2a a checkbox indicating that it is the preferred DC for this setup. This is an optional setting - but a nice feature to have if you know that most of the read/write access is going to come from one (or more) particular zones/region(s).



***Note**: Note: Before clicking Edit do the following two steps to monitor the cluster during the edit operation.*

Run a workload

java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS --num_threads_write 8 --num_threads_read 16 --value_size 128 --num_unique_keys 1000000 --nouuid --with_local_dc us-west-2

***Note**: Connect to the load-tester and get the (YCQL) ENDPOINTS exactly as in the single-az case above.*

Check the Tablet distribution

On the Master leader web UI (Tablet Servers page) check the tablet leader distribution before and after setting the preferred region.

(Now you can press Edit and keep monitoring a. and b. above)

Expected tablet distribution (Before > After)

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.27.87:9000	1.0s	ALIVE	48	48	873.126 MB	0 B	0 B	37346.1	75.1639	aws	us-west-2	us-west-2a	24c01d6186da46e397c4181139c04b22
172.152.21.212:9000	0.3s	ALIVE	48	0	853.362 MB	0 B	0 B	0	0	aws	us-east-1	us-east-1a	01262f00359849e7b75c12db6a51baec
172.158.20.93:9000	0.8s	ALIVE	48	0	851.947 MB	0 B	0 B	0	0	aws	eu-central-1	eu-central-1a	a8a014f6d5b749cea9a1b382ff864319

Primary Cluster UUID: 61cd09e9-d051-4ee6-ad49-7ab612b78671

Server	Time since heartbeat	Status	Load (Num Tablets)	Leader Count (Num Tablets)	RAM Used	SST Files Size	Uncompressed SST Files Size	Read ops/sec	Write ops/sec	Cloud	Region	Zone	UUID
172.151.27.87:9000	0.1s	ALIVE	48	16	872.278 MB	0 B	0 B	67.285	15.3548	aws	us-west-2	us-west-2a	24c01d6186da46e397c4181139c04b22
172.152.21.212:9000	0.6s	ALIVE	48	16	849.355 MB	0 B	0 B	56.9894	13.2362	aws	us-east-1	us-east-1a	01262f00359849e7b75c12db6a51baec
172.158.20.93:9000	0.9s	ALIVE	48	16	852.149 MB	0 B	0 B	71.9674	16.9923	aws	eu-central-1	eu-central-1a	a8a014f6d5b749cea9a1b382ff864319

Expected Results (Before > After)

(Read) Ops/Sec: ~200 > 37k (almost 200x increase)
(Read) Latency: ~85 > 0.5 ms/op
(Write) Ops/Sec: ~44 > 75 ops/sec
(Write) Latency: ~180 > 100 ms/ops
CPU Usage (User + Sys): <2% > ~11%

Note that the write latencies (and ops/sec) also improve because getting a quorum can now be faster; It does not need to involve the farther away Frankfurt node at all -- only the two closer ones Oregon (local) and N. Virginia.

12. Node/Zone/Region Failure Testing

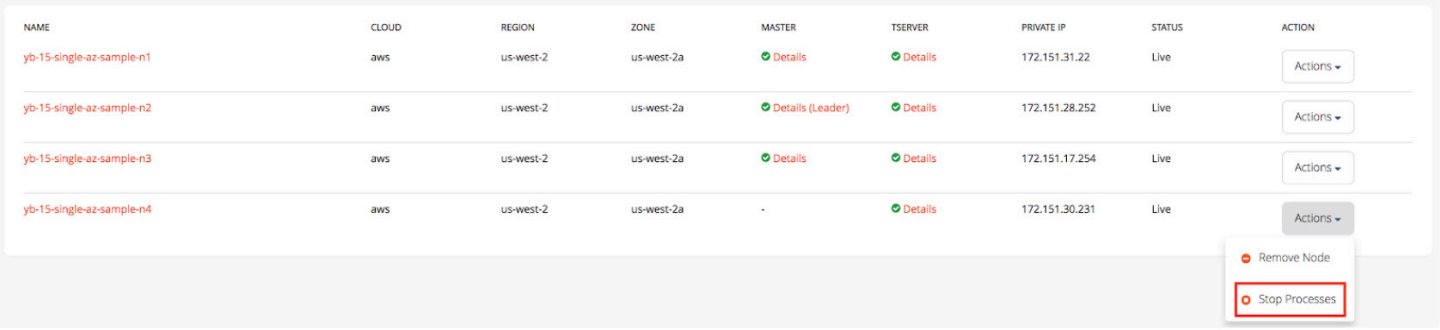
Note: Connect to the load-tester and get the (YCQL) ENDPOINTS of the single az universe exactly as before. Make sure to use the endpoints of the single-az universe for a) below and the endpoints of the multi-region universe for b) below.

Run a workload with 64 readers and 8 writer threads throughout the failure testing process.

```
java -jar java/yb-sample-apps.jar --workload CassandraKeyValue --nodes $ENDPOINTS
--nouuid --value_size 256 --num_threads_read 32 --num_threads_write  32 --num_
unique_keys 10000000
```

Testing single node failure in a single region deployment

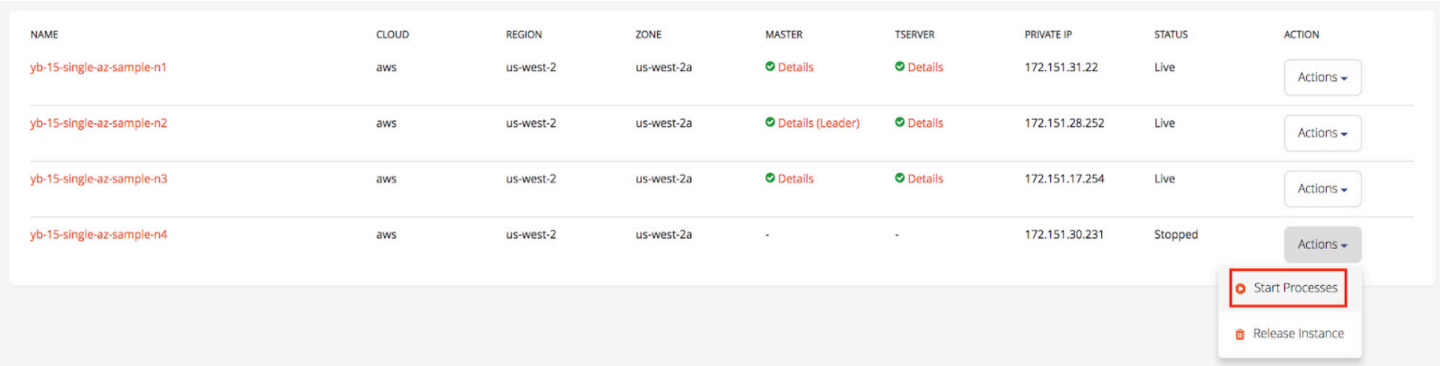
With load running, go to the "Nodes" tab on the universe page. On the "ACTION" column select "Stop processes" for the relevant node.



During the simulated node failure, you can check the following:

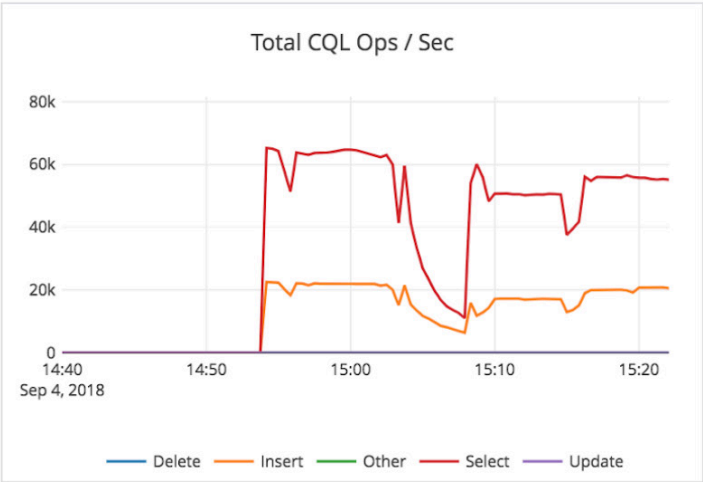
1. On the “Metrics” tab to check the ops/sec, latency, etc.
2. On the Master leader web UI (Tablet Servers page) the tablets/leaders distribution (on the nodes).

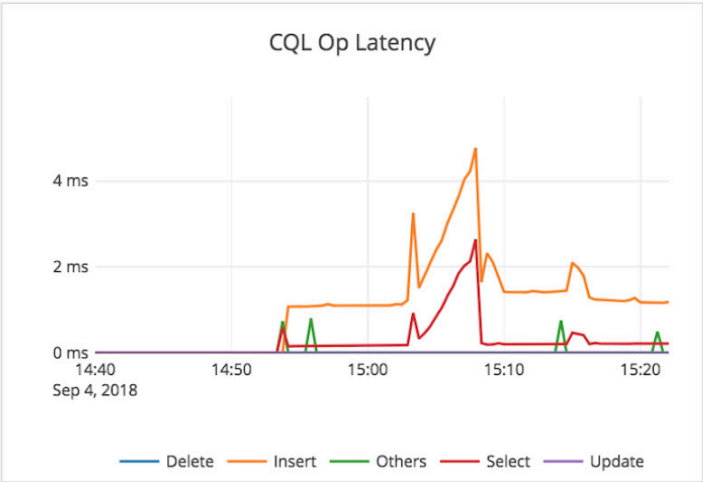
Finally, bring the node back up by doing “Start processes” on the “ACTION” column from the “Nodes” tab.



Expected Results

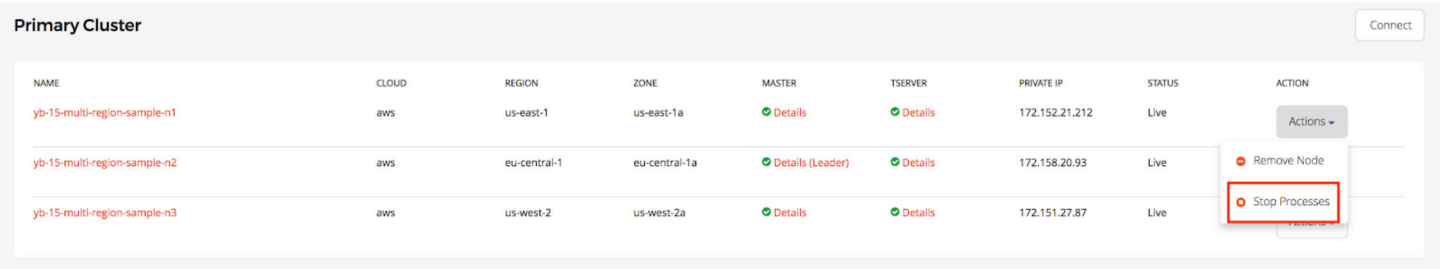
Metrics should slightly lower ops/sec (depending on overall load) while the one node is down but the cluster should still be able to take both reads and writes. Additionally there may be some temporary dips immediately after stopping or starting the processes are stopped/started (seen below at around 15:05 and, respectively, 15:15).





Testing a single region failure in the multi-region deployment

With load running, go to the “Nodes” tab on the universe page. On the “ACTION” column select “Stop processes” for the node(s) in the relevant region.



During the simulated region failure, you can check the following:
1. On the “Metrics” tab to check the ops/sec, latency, etc.
2. On the Master leader web UI (Tablet Servers page) the tablets/leaders distribution (on the nodes).

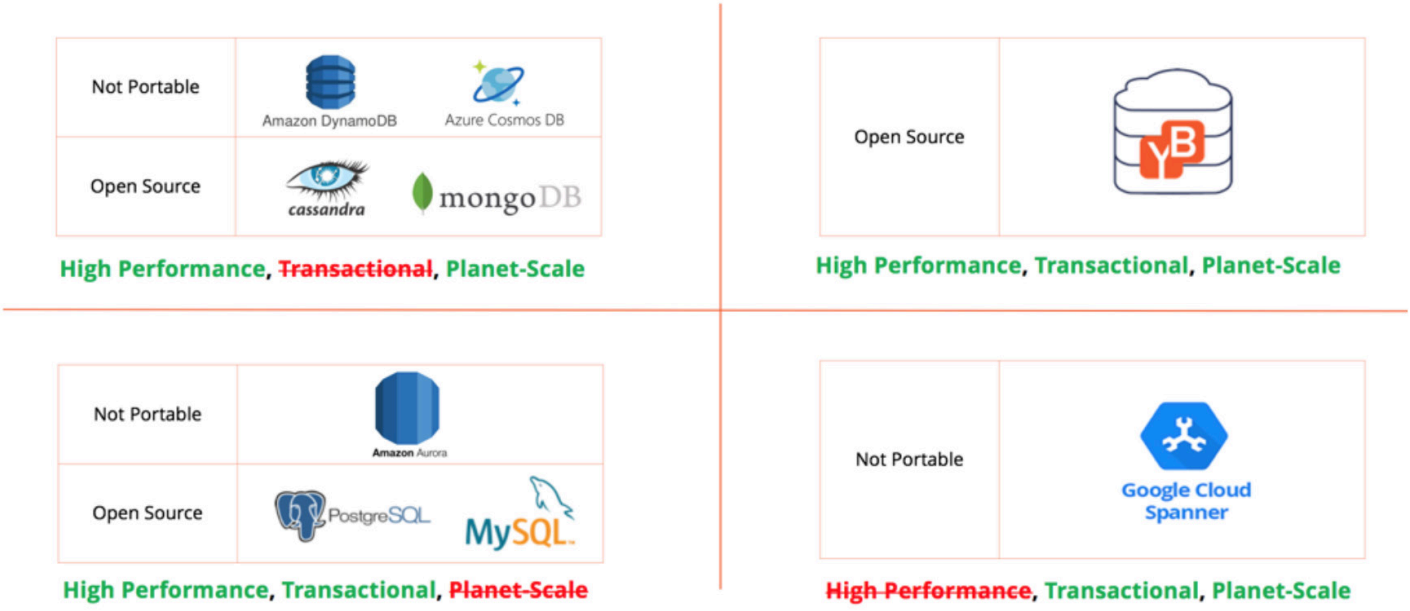
Finally, bring the node(s) back up by doing “Start processes” on the “ACTION” column from the “Nodes” tab.

Expected Results

Metrics should report a drop in ops/sec (depending on overall load) while the one node is down but the cluster should still be able to take both reads and writes. The exact values may vary significantly depending on settings (local or remote region is down, if using preferred region, if reads/write are concentrated in one region, etc).

13. Compare YugaByte DB to Other Databases

The image below highlights how YugaByte DB compares against other databases.



Monolithic SQL databases such as PostgreSQL, MySQL and even Amazon Aurora do not offer linear write scalability and hence cannot offer the various multi-region, hybrid cloud and multi-cloud deployment options that are increasingly becoming commonplace in enterprise architectures. Distributed NoSQL databases such as Apache Cassandra and MongoDB solve this write scalability problem but compromise on the transactional data integrity guarantees. Proprietary cloud databases such as Amazon DynamoDB and Azure Cosmos DB suffer from the same problem. Google Cloud Spanner entered the market in 2017 as a proprietary cloud SQL database that can scale across multiple regions while preserving global transaction guarantees. However, applications needing low latency and high throughput reads find Spanner cost prohibitive to use given the inability to use NoSQL APIs that understand the exact location of the data in a large cluster of nodes. YugaByte DB has higher average performance than Spanner -- it does so by having some tables serve single row/shard transaction workloads similar to high performing NoSQL but with strong consistency and some other tables serve distributed transaction workloads similar to a transactional SQL but with linear write scalability.

14. Conclusion

YugaByte DB is meant to be a system-of-record/authoritative database that distributed applications can rely on for correctness and availability. It allows applications to easily scale up and scale down across multiple regions in the public cloud, on-premises datacenters or across hybrid environments without creating operational complexity or increasing the risk of outages. With its cloud native operational experience and enterprise-grade features, YugaByte DB Enterprise Edition ensures that enterprises can develop, deploy and operate mission-critical clusters with unparalleled ease.

15. What's Next?

- Contact Sales
- Watch the explainer and demo videos
- Download the datasheet
- Download the Business-centric paper
- Download comparisons
- Get started with EE

YugaByte DB

YugaByte

For more info please visit **yugabyte.com**

YugaByte, Inc.
771 Vaqueros Ave
Sunnyvale, CA 94085

General inquiries:
info@yugabyte.com