Couchbase

What's new in Couchbase 6.6

Enterprise-Class NoSQL Database

	CB 6.0	New in CB 6.5 & 6.6
ACID Transactions	Single-Document ACID	Multiple-Document ACID
Query Enhancements	SQL for JSON (Query & Analytics), Declarative, Visual Query Plan, Search	ANSI Window Functions, ANSI CTE, Query+Search, Cost-Based Optimizer (Preview) Index Advisor (Preview) (GA in 6.6)
Eventing	Server-side programming	CURL, Source Bucket Mutation, User-Defined Functions (Preview)
SDK	SDKs with support for all Couchbase services	Scala Support, Simplified Async Programming, Improved HA & Observability, Improved APIs
High App & Data Density	10 Buckets	30 Buckets, Collections (Preview), High Data Density (Private Preview)
HA/DR	Active-Active, Memory-Memory Streaming	XDCR - Quality of Service, Advanced Filtering, Robust Rebalance, RMS for GSI
Backup/Restore	Full & Incremental Backup/Restore	60% better throughput, reduce storage requirements by 2x-4x
Security & Compliance	X.509 Authentication, RBAC, Secure Transport, Auditing, Field-Level Encryption	LDAP Group, Node-Node Encryption, TLS Cipher Settings

1 Distributed ACID Transactions

Distributed ACID Transactions

Simplify application logic with distributed ACID transactions, providing all-or-nothing semantics for durably modifying multiple documents distributed on different nodes and across buckets.

Transition from RDBMS schema:

- De-normalization has limitations

Multi-Asset Coordination:

- Transfer from one user to another
- Reservation items such as flights

Business/Application-level transaction needs to modify multiple documents (all-or-nothing):

- Microservices SAGAs - Event based orchestration

```
transactions.run((txnctx) -> {
  // Insert a document
  JsonDocument doc1 =
   JsonDocument.create("newDoc",JsonObject.create());
  txnctx.insert(bucket, doc1);

  // Replace a document
  TransactionJsonDocument doc2 =
    txnctx.getOrError(bucket, "doc2");
  doc2.content().put( "name", "bob");
  txnctx.replace(doc2);

  // Commit transaction
  txnctx.commit();
});
```

ACID in Couchbase

Couchbase Server 6.5 provides strong ACID guarantees while balancing scalability, availability and performance.

A	Atomicity	Guarantees all-or-nothing semantics for **updating multiple documents** in more than one shards on different nodes.
C	Consistency	**Replicas** immediately consistent with the transaction commit. **Indexes and XDCR** eventually consistent with the transaction commit (*N1QL can enforce strong consistency upon read with request_plus*)
I	Isolation	**Read Committed** isolation for concurrent transactions
D	Durability	**Data protection under failures**: 3 different levels - *replicate to majority of the nodes; replicate to majority and persist to disk on primary; or persist to disk on majority of the nodes.*

Couchbase Transactions: Built on Synchronous Replication

Synchronous Replication ensures that a mutation is visible to readers only after it has met its replication/persistence criteria and can be rolled back if those criteria are not met.

- SyncWrite is **all-or-nothing**
- **Sequential Consistency model**
- **3 Durability** levels (per-write)
  - Replicate to Majority
  - Replicate to Majority and Persist to Master
  - Persist to Majority
- Applicable to **transactions and single document mutations**
- Couchbase and Ephemeral buckets
- In-house **Jepsen** testing

```
MutationResult result =
      collection.insert(id,
      new Person("Jon Henry", 34, null, null),
      insertOptions()
            .durabilityLevel(DurabilityLevel.
                        MAJORITY_AND_PERSIST_ON_MASTER)
                  .timeout(Duration.ofSeconds(30))
       );
```

Couchbase Transactions: ACID at Scale with High Availability

- **No SPOF or performance bottleneck**
  - Couchbase Smart Clients act as the Transaction Coordinators
    - No reliance on a global transaction coordinator
  - Chosen algorithms ensure progress only depends upon availability of shards involved in the transactions
    - Read Committed: no global scheduler
    - Built-in retries: no distributed lock manager
- **Pay the transaction overhead only when you use it**
  - Flexibility to use a transaction is available at the level of an operation
  - No need to unconditionally impose it for every operation

2 Query Enhancements

Query Enhancements

Window Functions

Common Table Expressions

SEARCH in N1QL

User Defined Functions
(preview)

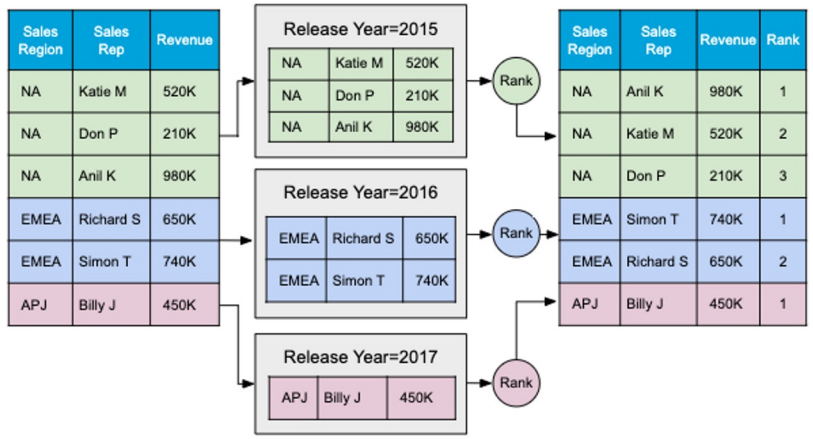
Index Advisor
(GA in 6.6)

Cost-Based Optimizer
(GA in 7.0)

ANSI Window Functions

With ANSI Window Functions, developers can
simplify financial and statistical aggregations in an easy and optimized way

- Answer common (but complex) business queries with minimal lines of code and optimized performance
- Use cases:
  - revenue growth month over month
  - top N sale districts by revenues for a given week
  - ranking of sales person by region based on revenue booked
- Couchbase is the first NoSQL Database to support ANSI Window Functions




```
SELECT sales_region,release_year, sales_rep,
 RANK() OVER (PARTITION BY sales_region,release_year
     ORDER BY SUM(revenue) DESC) as `rank`
FROM orders
GROUP BY sales_region, sales_rep, release_year
```

ANSI Common Table Expression (CTE)

With ANSI Common Table Expressions, developers have ease of maintenance and better readability by naming temporary SQL statements.

- CTE allows developer to isolate SQL statement into temporary named result set that can be referenced as a source table in the context of a larger query
- Offers the advantages of readability and ease of maintenance of complex queries without compromising performance.
- Couchbase is the first NoSQL Database to support ANSI CTE.

```
WITH current_period_task AS (
   SELECT DATE_TRUNC_STR(a.startDate,'month') AS month,
   COUNT(1) AS current_period_task_count
   FROM crm a
   WHERE a.type='activity' AND a.activityType = 'Task'
   AND DATE_PART_STR(a.startDate,'year') = 2018
   GROUP BY DATE_TRUNC_STR(a.startDate,'month')
),

last_period_task AS (
   SELECT x.month, x.current_period_task_count,
   LAG(x.current_period_task_count) OVER (ORDER BY x.month)
   AS last_period_task_count
   FROM current_period_task x
)
```

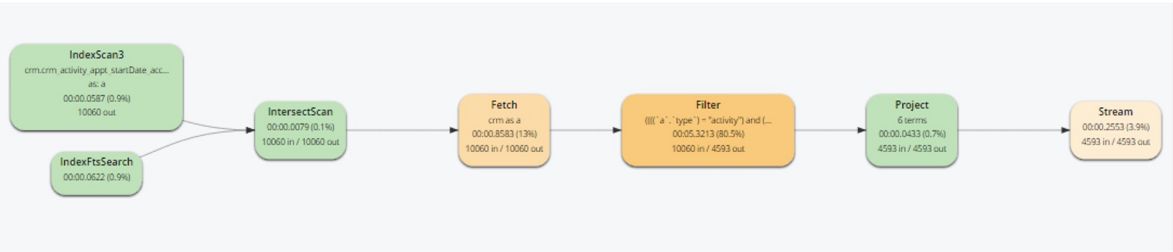
```
SELECT b.month,
       b.current_period_task_count,
       ROUND(((b.current_period_task_count - b.last_period_task_count ) /
                    b.last_period_task_count),2)
FROM last_period_task AS b
```

SEARCH Predicates in Query (Full Text Search)

With Search predicates in N1QL, developers can combine SQL and Search queries for powerful integration and better query performances

- Couchbase provides a comprehensive search capability in query beyond the simple LIKE() operator in most databases
- The SEARCH() operator supports keyword and fuzzy matchings across multiple document fields
- Developers do not have to write complex code to process and combine the results from separate SQL and search queries
- Better query performance with inverted indexes for search predicates instead of inefficient scans for LIKE()

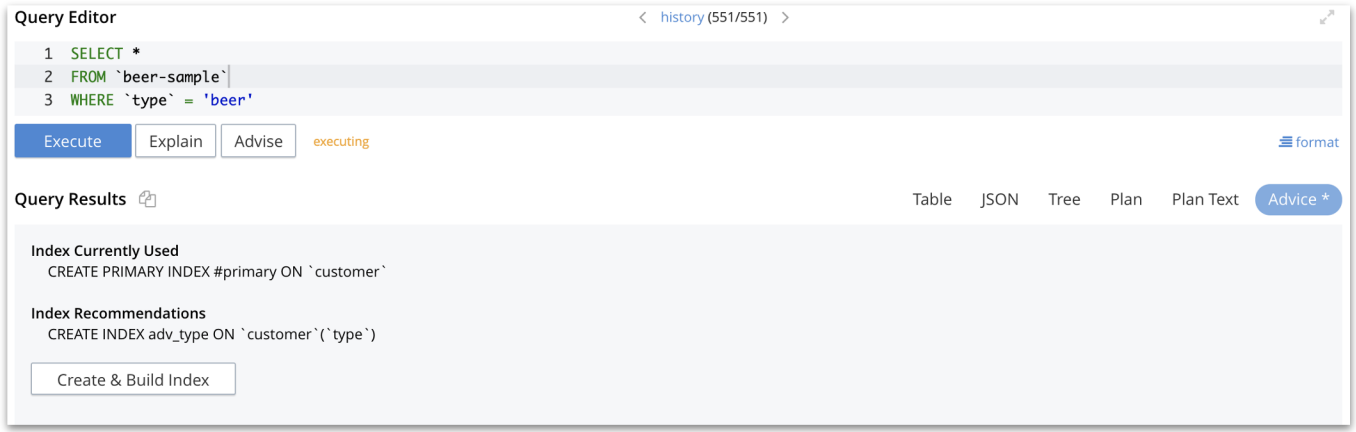
```
SELECT * FROM `beer-sample` b
WHERE SEARCH(b.desc, "fruity")
AND b.abv < 0.5;
```

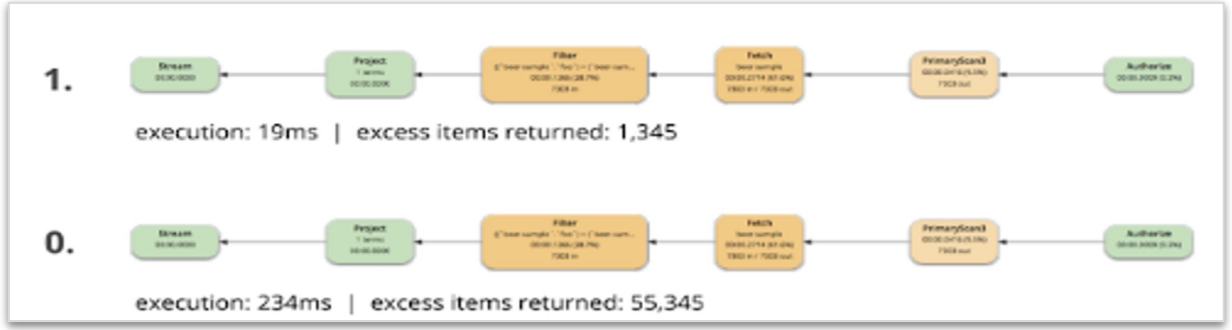


Index Advisor

With Index Advisor, developer can create better indexes based on the suggestions to speed up queries easily.

- The Index Advisor suggests appropriate indexes to speed up a given query, taking the guesswork out of query tuning
- The Index Advisor can also monitor and analyze the statistics collected from running a workload, and suggest the indexes that will speed up the queries in the workload
- The Index Advisor greatly reduces the complexity and efforts required for enterprise developers and operations engineers to determine the right indexes to speed up their queries





User-Defined Functions (Developer Preview)

With User-Defined Functions, developers define callable custom functions to ease development, simplify reusability and improve code performance.

- Allow developers to define custom functions in N1QL or Javascript (similar to PL/SQL) callable from N1QL queries. Interactive debugger to simplify development and testing
- Server-side logic that can be reused by any application and micro services; improve code maintenance and developer productivity
- Improve code performance by bringing application logic closer to the data

```
CREATE FUNCTION to_meters() { args[0] / 0.3048 };
```

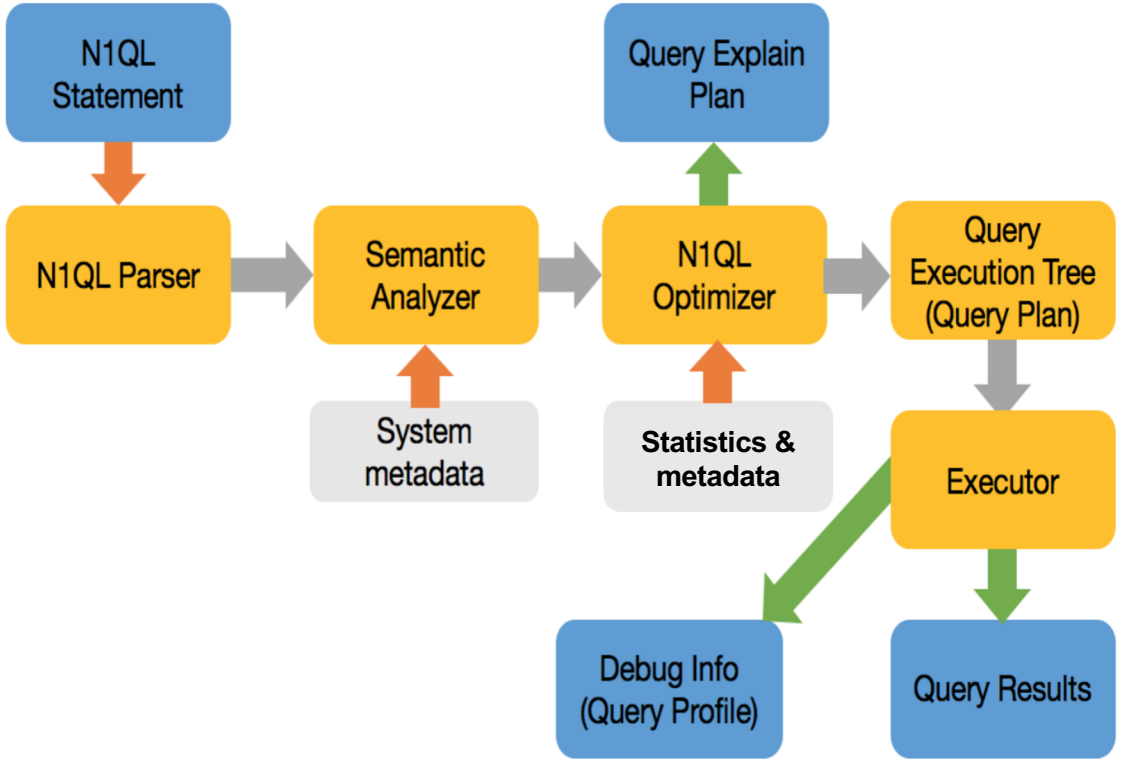
```
SELECT name, ROUND(to_meters(geo.alt)) AS mamsl
FROM `travel-sample`
WHERE type = "airport"
LIMIT 1;
```

```
{
  "airportname": "Calais Dunkerque",
  "mamsl": 39
},
```

Cost-Based Optimizer (Developer Preview)

With Cost-Based Optimizer, queries will run faster by using optimal access path without the need to provide optimizer hints.

- Cost-based optimizer that generates the optimal access path based on statistics collected on the data
- Eliminates time tweaking a query and providing optimizer hints to get the rule-based optimizer to pick the right query plan
- Leverages decades of research and experience in query optimization to collect and use statistics on JSON, arrays, and objects.
- Couchbase is the first NoSQL database with dynamic schema to support cost-based optimizer



Easy Migration from RDBMS to Couchbase

MySQL

```
SELECT ac.industry,
 SUM(CASE WHEN a.activitytype = 'Task'
    THEN 1 ELSE 0 END ) task,
 SUM(CASE WHEN a.activitytype ='Appointment'
    THEN 1 ELSE 0 END ) appts
FROM crm.activity a
  INNER JOIN crm.account ac
      ON (a.accid = ac.id)
WHERE a.startdate BETWEEN '2018-10-01'
           AND   '2018-12-31'
GROUP BY ac.industry
```

Couchbase - N1QL

```
SELECT ac.industry,
  SUM(CASE WHEN a.activityType = 'Task'
    THEN 1 ELSE 0 END) task,
  SUM(CASE WHEN a.activityType = 'Appointment'
    THEN 1 ELSE 0 END ) appts
FROM crm a
  INNER JOIN crm ac ON a.accid = ac.id
          AND ac.type='account'
WHERE a.type='activity'
  AND a.startDate BETWEEN '2018-10-01'
      AND '2018-12-31'
GROUP BY ac.industry
```

MongoDB Query Language

```
db.activity.aggregate(
 { $match: { startDate: { $gt: '2018-01-01',
                          $lt: '2018-12-31' } } },
 {
   $lookup: {
         from: "account",
         localField: "accid",
         foreignField: "id",
         as: "account_docs"
           }
 },
 { $match: { "account_docs": { $ne: [] } } },
 { $unwind: "$account_docs" },
 {
   $project: {
     item: 1,
     task: { $cond:{if:
       { $eq: ["$activityType", "Task"] }, then: 1,
            else: 0 } },
     appt: { $cond: { if:
      { $eq: ["$activityType", "Appointment"] }, then: 1,
            else: 0 } } }
  },
  {
      $group: {
           _id: "$account_docs.industry",
           tasks: { $sum: "$task" },
           appointments: { $sum: "$appt" }
  } }
);
```

3 SDK 3.0

SDK 3.0

Scala

Simplified Async Programming

Improved HA and Observability

Improved APIs

New Platform: Scala

- Native Scala APIs - get and upsert documents, run queries, perform analytics and full text searches
- Has built-in support for JSON and also integrates with popular Scala JSON libraries: Circe, µPickle / µJson, Json4s and Jawn
- Supports both Scala Futures and Project Reactor based reactive programming with **`Mono and Flux`**

```
…
cluster = Cluster.connect(env)
val bucket = cluster.bucket(config.bucketname)
…
val content = ujson.Obj("hello" -> "world")
val upsertResult = coll.upsert(docId, content)

upsertResult match {
 case Success(result) =>
   // do something interesting
 case Failure(err) =>
   // handle some errors
}
val id: String = insertDoc
for (i <- Range(0, 10)) {
 val options: QueryOptions = QueryOptions()
   .scanConsistency(ScanConsistency.RequestPlus())
   .adhoc(false)
   .namedParameters("id" -> id)
 val result: QueryResult = cluster.query("select `"
+ bucketName + "`.* from `" + bucketName + "` where
meta()" +
   ".id=$id", options).get
 val rows = result.allRowsAs[JsonObject].get
```

Simplified Async Programming

Simplifies asynchronous programming and avoids problems associated with callbacks, leveraging the latest async frameworks/concepts in key languages

- Node.js SDK - Supports Promises in JavaScript ES6
- Java SDK - Supports Reactor on top of reactive streams and supports Java 8 CompleteableFuture
- Scala SDK - Supports Reactor, Scala Native Futures, Akka Streams
  - Opens Couchbase up to the Akka developer ecosystem out of the box
- Python SDK - Supports Async IO in Python 3

Simplified Async Programming

```
function attemptChange(callback) {
  bucket.get('somedoc', function(err, res) {
    if (err) {
      callback(err);
      return;
    }

    if (res.value > 100) {
      res.value += 1;
    } else {
      res.value += 10;
    }

    bucket.replace('somedoc', res.value, {cas:
res.cas}, function(err, res) {
      if (err && err.code == 0x02) {
        // CAS missmatch on upsert, try again
        attemptChange(callback);
        return;
      } else if (err) {
        callback(err);
        return;
      }

      callback(null);
    })
  });
}

attemptChange(function(err) {
  if (err) {
    console.log('Change Errored', err);
  } else {
    console.log('Change Completed');
  }
});
```

Simplified Async Programming

- No more nested callbacks!
- Performs like asynchronous code, but looks simpler
- Simplified async error handling with await keyword from ECMAScript 6.x+

```
do {
  var res = await coll.get('somedoc');

    if (res.value > 100) {
      res.value += 1;
    } else {
      res.value += 10;
    }
    try {
      await coll.replace('somedoc', res.value, {cas: res.cas});
    } catch (e) {
      if (e instanceof couchbase.CasMismatchError) {
        continue;
      }

      // rethrow real errors.
      throw e;
    }
} while(false);
```

Improved HA and Observability

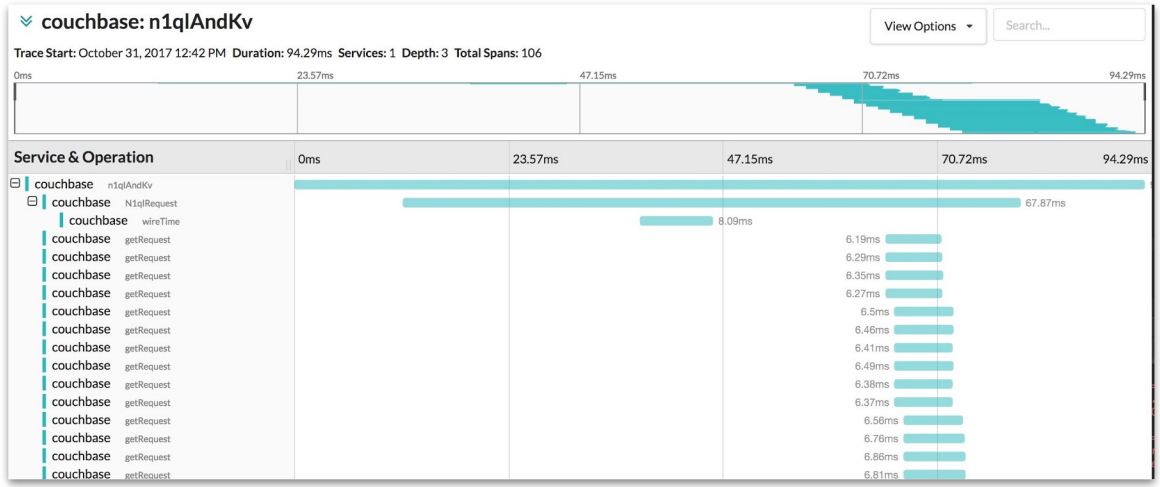
Circuit Breaker

- Improve system availability and prevent cascading failures by avoiding putting loads on a struggling resource
- Provides early warning on potential system issues when circuit breaker trips.

```
CircuitBreakerConfig.builder()
        .enabled(true)
        .errorThresholdPercentage(50)
        .rollingWindow(Duration.ofSeconds(60))
        .sleepWindow(Duration.ofSeconds(5))
        .volumeThreshold(20)
        .build();
```

OpenTelemetry

- Make it easy to get critical telemetry data from Couchbase applications in a robust and portable way
- OpenTelemetry is a CNCF sandbox project, and the next major version of the OpenTracing and OpenCensus projects



Improved APIs

Reduce Overloads

```
bucket.upsert
  upsert(D document)
  upsert(D document, PersistTo persistTo)
  upsert(D document, ReplicateTo replicateTo)
  upsert(D document, long timeout, TimeUnit timeUnit)
  upsert(D document, PersistTo persistTo, ReplicateTo, rep…
  upsert(D document, PersistTo persistTo, long timeout, …
  upsert(D document, ReplicateTo replicateTo, long time…)
  ...
```

```
MutationResult mutationResult = collection.upsert(Id, person,
  upsertOptions().expiry(Duration.ofDays(1))
);
```

Prepare for Collections

- Old: Cluster -> Bucket -> Document
- New: Cluster -> Bucket -> Scope -> Collections

```
1  //Open a Scope and then a Collection
2  var scope = await bucket.Scope("ACME");
3  var collection = await scope.Collection("tools");
4
5  //add a new documentawait
6  collection.Insert(key, new Tool(Name = "Hammer", Type="Ball peen"});
7
8  //retrieve the document
9  using (var result = await collection.Get(key)) {
10         var content = result.ContentAs<Tool>();
11         Console.WriteLine(content.Name);
12 }
```

4 High Density

High Application and Data Density Enhancements

30 Buckets

Collections
(preview)

HiDD
(private preview)

High Application Density - More Buckets per Cluster

User can create up to 30 buckets in a single Couchbase cluster and operate it seamlessly as long as sizing is done appropriately.

- Multi-tenancy support
  - Buckets provide physical, logical and access isolation
  - Improve resource utilization of hardware by co-locating multiple tenants
- Microservices consolidation
  - Buckets provide physical, logical and access isolation
  - Improve resource utilization of hardware by co-locating multiple microservices

- Previous limit was up to 10 buckets
- Sizing still critically important
  - Factor in raw overhead per bucket
  - Eventing service not supported with multiple functions per bucket if using large number of buckets

High Application Density - Collections (Dev Preview)

With Collections, application developers can organize and access control their data as separate logical containers within a bucket.

- Logical Data Container within a bucket
- Collections can be organized in Scopes
- Document IDs unique within a collection
- 1000s of collections in a bucket
- Security Isolation via RBAC
- Easier Migration from RDBMS
  - Closer mapping to relational schema
  - N1QL easier and more intuitive
- Improve hardware utilization
  - Consolidation of applications/microservices
  - Soft co-operative multi-tenancy

- **Available in 6.5 DP**:
  - Data Service functionality
  - Cluster manager functionality for collections including REST APIs, Rebalance and Failover handling
  - SDK APIs for collection access
- **GA in 7.0**:
  - RBAC at collection and scope level
  - UI awareness of collections
  - XDCR and Backup/restore at collection level
  - Indexing/Query/FTS/Analytics/Eventing at collection level

High Data Density - 30 TB per Node (GA in 7.1- Q1 2022)

With High Data Density support, enterprises can have dense data nodes with 30 TB per node with less than 1% memory residency for better TCO

- Fewer nodes in cluster
  - Support High Data Density per node - 30 TB data per node
    - Support with low memory residency <1% memory residency
- Faster disk access performance
  - Lower read response time
  - Higher write throughput
- Less SSD wear out
  - Lower write amplification
  - Compaction should not rewrite large parts of data
- Reduction in required disk capacity allocation
  - Lower space amplification

5 HA/DR/Backup

HA/DR Enhancements

XDCR filtering

XDCR QoS

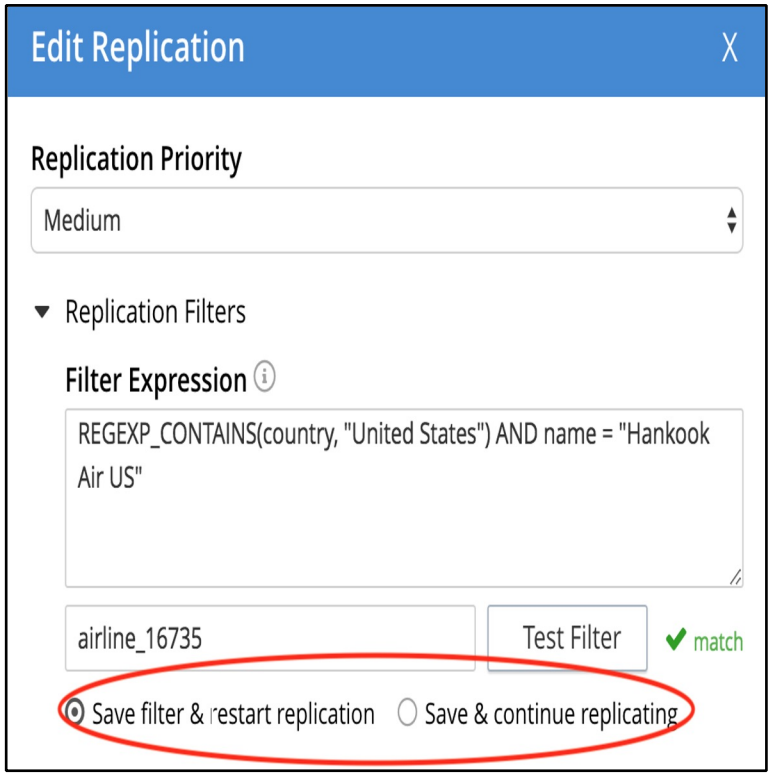
Backup Performance

Backup Size

XDCR - Advanced Filtering

XDCR advanced filtering enables customers to filter and replicate data based on keys, values and extended metadata to simplify partial replication between data centers.

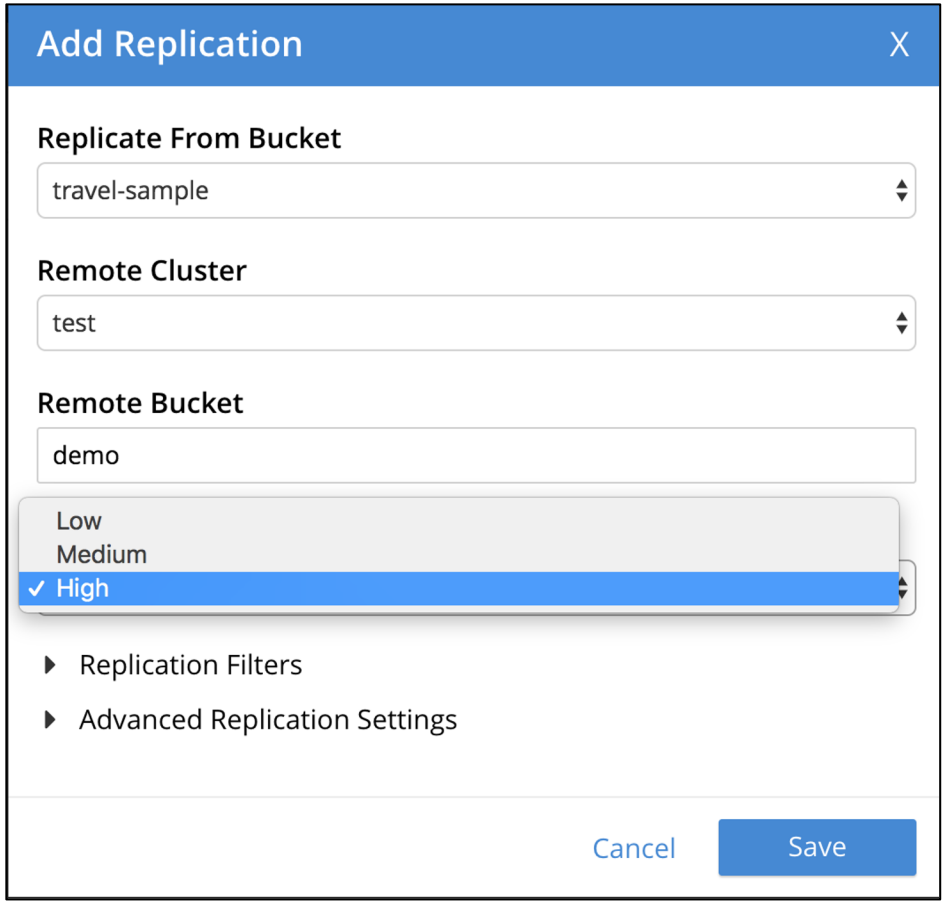
- Documents to be replicated can be filtered based on keys, values, meta-data or any combinations of those.
- Negative filters such as (?!regex),(?<!regex) will be supported which can be applied to filter all documents except a specific subset.
- Filtering out TTL, expirations and/or deletes to create hot/cold datasets.
- Modify filters at runtime.



XDCR - Quality of Service

XDCR quality of service allows control on replication streams by providing an option to prioritize ongoing replications over initial replications to improve uptime.

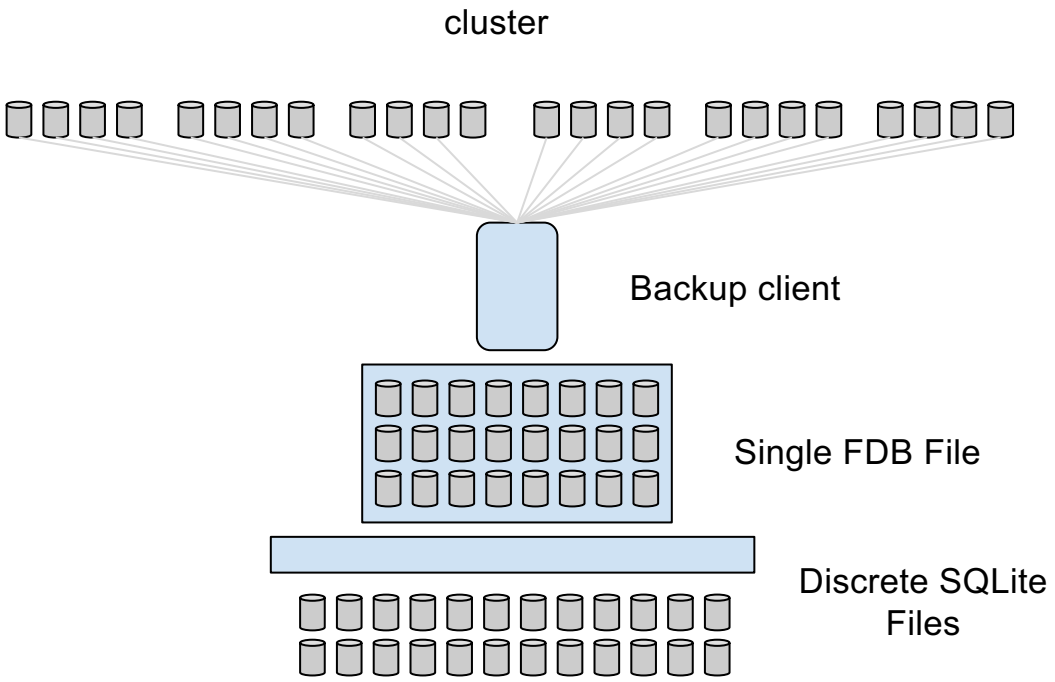
- Remove limitation of current behavior which always prioritizes new replication over existing ones, sometimes causing degradation of existing streams when new one is created.
- Now possible to classify a new or ongoing replication as “High, Medium, Low”, resources allocated accordingly.



Backup - Enhancements

Significantly reduced storage requirement for backup (~40% of original dataset) and performance improvement
of more than 60% for ease of administration and better TCO

- Moved from a single monolithic file to 100MB files, introduced DCP compression, replaced ForestDB with SQLite as storage engine and provided the ability to measure consistency.
- Improved List, Merge, Compact performance.
- Auto-bucket creation for restore.
- New Info command (developer preview).



6 Manageability Enhancements

Manageability Enhancements

Custom Statistics

Robust Rebalance

Custom UI Statistics

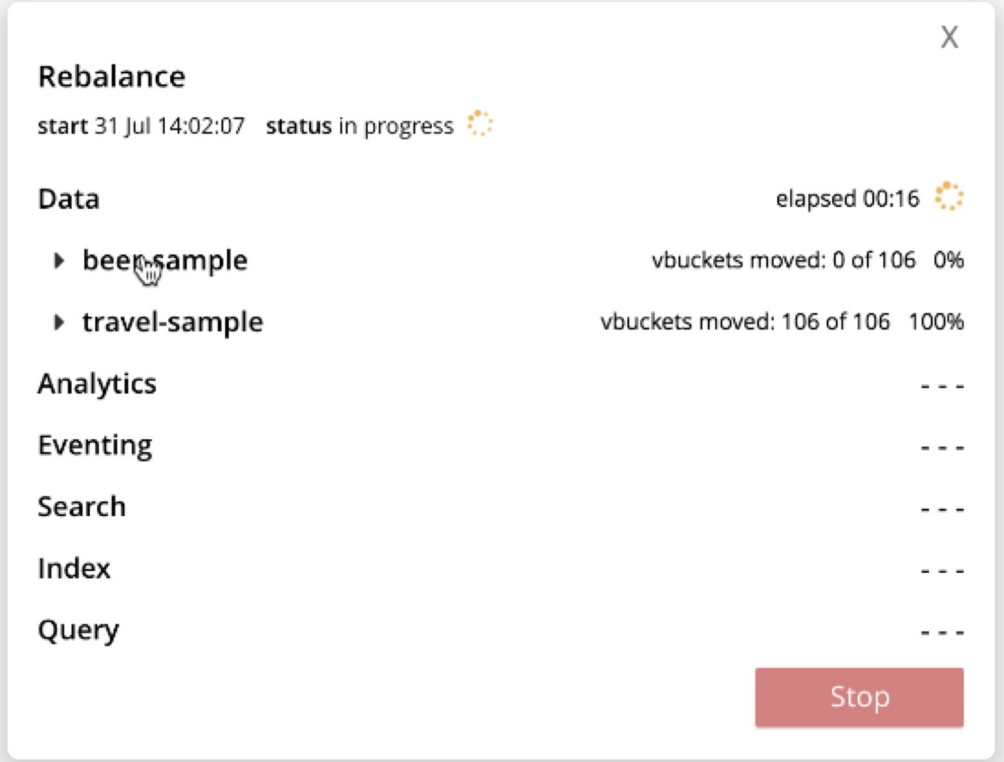
Customizable UI statistics to assist with troubleshooting and simplify monitoring of the whole cluster

- Annotated and labeled stats visuals, responsively sized for any screen
- Ability to use a click-and-drag focus tool to select a custom range of data — zooming in on areas of interest.
- Ability to aggregate stats from multiple nodes to compare and contrast.
- Service specific rendering.
- Ability to edit, group,drop, drag or add the charts
- Users can save the charts created by them.

Robust Rebalance

Rebalance is now more robust and resilient with automatic restart, better performance, and enhanced monitoring of rebalance progress.

- Auto-restart rebalance on failure
  - Configurable retry interval/count
- Auto-failover during rebalance
  - Prioritize availability
- Improved progress monitoring
  - Status for all services and stages
  - New look and feel
- Lower impact to front-end workloads
  - Scheduling improvements for lower resource utilization



7 Security Enhancements

Security Enhancements

LDAP Groups

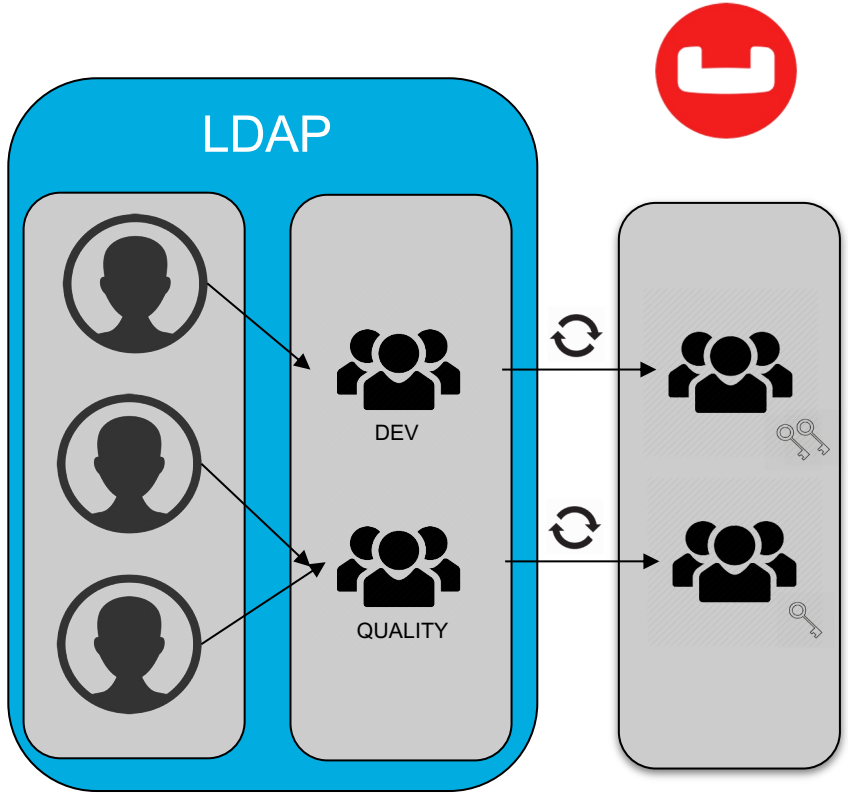
Node-Node Encryption

TLS Cipher Management

Security - Native LDAP Group Support

LDAP Group support simplifies LDAP integration and includes local group support

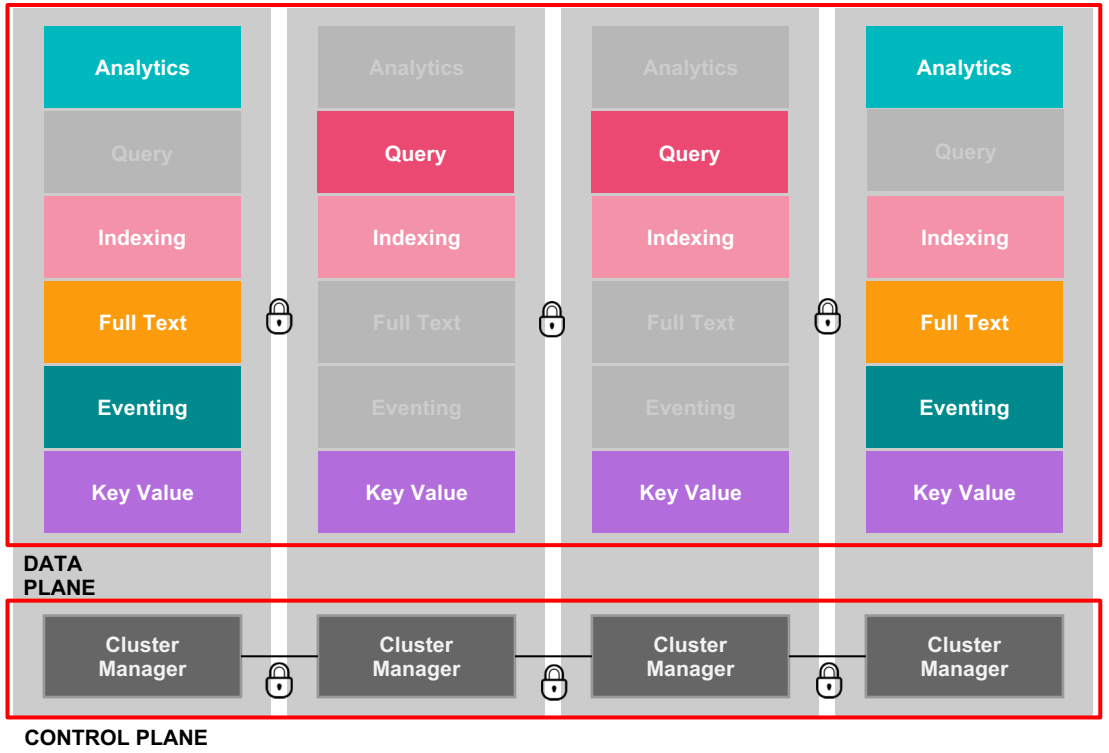
- LDAP Group Support enables easy integration with existing LDAP servers, and RBAC user management
- Native LDAP works without SASLauthd support, lighting up LDAP for Couchbase on Windows
- LDAP users and groups reside externally in LDAP
- Couchbase Groups map and sync to LDAP groups
- RBAC privileges assigned to Couchbase groups



Security - Node-to-Node Encryption

Node-to-node certificate-based encryption greatly and easily improves security within a cluster.

- On-the-wire encryption for data and control plane, across multiple nodes
- Certificate-based encryption management using existing node and cluster certificates
- Encryption can be controlled and enabled just for the control plane separately
- Using IPSec is not required
- Simple to setup and manage



Security - TLS Cipher Settings

TLS Cipher settings with centralized management and persistent settings to manage TLS security easily

- Centralized management of TLS cipher settings:
  - Get the set of supported ciphers supported across the different Couchbase services
  - Set the list of ciphers to be used consistently across services or per service
  - Persistent TLS cipher settings across cluster, node restarts and upgrades

```
curl localhost:8091/settings/security \
       -u Administrator:password \
          -d 'cipherSuites="[TLS_RSA_WITH_AES_128_CBC_SHA, \
                             TLS_RSA_WITH_AES_256_CBC_SHA, ...]"'
```

Thank You