

JAVA DEVELOPER WORKSHOP





AGENDA

- Lab Environment
- Introduction to Data Modeling
- Travel-sample data model
- Connecting to Couchbase
- Key/value and CRUD
- Working with Parts of a Document
- Querying for Documents with N1QL
- Full Text Search
- Field Level Encryption
- Threshold Logging
- Batching with RxJava
- Spring Boot & Spring Data



LAB ENVIRONMENT





Lab 0: Installing Couchbase

DOWNLOAD

<https://www.couchbase.com/downloads>

DEPLOYMENT OPTIONS

<https://docs.couchbase.com/server/6.0/install/get-started.html>

 On-Premises (Non-Cloud)	 Virtual Machines and Containers	 Public Cloud
Deploy Couchbase Server in a traditional data center on bare metal servers.	Deploy Couchbase Server within a virtualized or containerized environment, such as VMware or Docker.	Deploy Couchbase Server in a public cloud, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

Lab



Lab 0: Installing Couchbase

DOCKER

- Documentation: <https://docs.couchbase.com/server/6.0/install/getting-started-docker.html>
- Single server deployment for this workshop

```
docker run -d --name cbworkshop -p 8091-8096:8091-8096 -p 11210-11211:11210-11211 couchbase
```

PDF of the slide deck are available here: <https://github.com/BKaneAtWork/CouchbaseJavaWorkshop>

Lab



Lab 1: Initial Setup

Access UI Web Console: <http://localhost:8091/ui/index.html>

Click “Setup New Cluster”, Enter Cluster Name “CB Workshop”

Administration User: Administrator:password

Accept Terms, and Configure Disk, Memory, Services

Couchbase Server
Enterprise Edition 6.0.0 build 1693
[Setup New Cluster](#)
[Join Existing Cluster](#)

Couchbase > New Cluster

Cluster Name: CB Workshop

Create Admin Username: Administrator

Create Password: Confirm Password:

< Back [Next: Accept Terms](#)

Couchbase > New Cluster

Terms and Conditions Enterprise Edition
Couchbase Server must be licensed for use in production environments.

Couchbase Inc. Enterprise Subscription License Agreement

IMPORTANT-READ CAREFULLY: BY CLICKING THE "I ACCEPT" BOX OR INSTALLING, DOWNLOADING OR OTHERWISE USING THIS SOFTWARE AND ANY ASSOCIATED DOCUMENTATION, YOU, ON BEHALF OF YOURSELF AND AS AN AUTHORIZED REPRESENTATIVE ON BEHALF OF AN ENTITY ("LICENSEE") AGREE TO ALL THE TERMS OF THIS LICENSE AGREEMENT (THE "AGREEMENT") REGARDING YOUR AND LICENSEE'S USE OF THE SOFTWARE. YOU REPRESENT AND WARRANT THAT YOU HAVE FULL LEGAL AUTHORITY TO BIND THE LICENSEE TO THIS AGREEMENT. IF YOU DO NOT AGREE WITH ALL OF THESE

I accept the [terms & conditions](#)

< Back [Finish With Defaults](#) [Configure Disk, Memory, Services](#)

Lab



Lab 1: Initial Setup

- Memory Quotas:
 - Data: 1024 Mb
 - Index: 256 Mb
 - Search: 256 Mb
- Uncheck Analytics and Eventing
- Uncheck software updates
- Save & Finish

Couchbase > New Cluster / Configure

Host Name / IP Address Usually localhost or similar
127.0.0.1

Data Disk Path Path cannot be changed after setup
/opt/couchbase/var/lib/couchbase/data
Free: 52 GB

Indexes Disk Path Path cannot be changed after setup
/opt/couchbase/var/lib/couchbase/data
Free: 52 GB

Analytics Disk Paths Paths cannot be changed after setup
/opt/couchbase/var/lib/couchbase/data
Free: 52 GB + -

Java Runtime Path optional

Service Memory Quotas Per service / per node

Data 1024 MB

Index 256 MB

Search 256 MB

Query -----
256 MB

Eventing
1024 MB

Analytics

TOTAL QUOTA 1536MB

RAM Available 1998MB Max Allowed Quota 1598MB

Index Storage Setting
 Standard Global Secondary
 Memory-Optimized ⓘ

Enable software update notifications in the web console. ⓘ

< Back Save & Finish

Lab



Lab 1: Initial Setup - Install sample data set

Install travel-sample data set: Admin Console > Settings > Sample Buckets

Activity Documentation Support Administrator ▾

The screenshot shows the Couchbase Admin Console interface. At the top, there's a navigation bar with links for Activity, Documentation, Support, and Administrator. Below that is a blue header bar with the Couchbase logo and the text "CB Workshop > Settings". Underneath the header is a navigation menu with tabs: Cluster, Software Updates, Node Availability, Email Alerts, Auto-Compaction, and Sample Buckets (which is currently selected). The main content area has two sections: "Available Samples" and "Installed Samples". In "Available Samples", there are three checkboxes: beer-sample, gamesim-sample, and travel-sample, with travel-sample checked. In "Installed Samples", it says "none". At the bottom left is a blue button labeled "Load Sample Data".

Sample buckets contain example data and Couchbase views.

You can provision one or more sample buckets to help you discover the power of Couchbase Server.

Sample buckets (like all buckets in Couchbase Server 5.0+) can only be accessed by a user with privileges for that bucket.

Available Samples

- beer-sample
- gamesim-sample
- travel-sample

Load Sample Data

Installed Samples

none

Lab

Lab 1: Initial Setup - Create User



- Username: travel
- Password: password
- Roles:
 - Query Update[travel-sample]
 - Query Select[travel-sample]
 - Query Manage Index[travel-sample]
 - Query Insert[travel-sample]
 - Query Delete[travel-sample]
 - Search Reader[travel-sample]

CB Workshop > Security

username	full name	roles	auth domain	password set	
travel	travel	Query Update[travel-sample], Query Select[travel-sample], Query Manage Index[travel-sample], Query Insert[travel-sample], Query Delete[travel-sample], Search Reader[travel-sample]	Couchbase	03-25-2019	Cancel Add User

Add New User

X

Authentication Domain

Couchbase External

Username

travel

Full Name (optional)

travel

Password

.....

Verify Password

Roles

► Data Service

► Views

▼ Query and Index Services

Query Select ✓

Query Update ✓

Query Insert ✓

Query Delete ✓

Query Manage Index ✓

▼ Search Service

Search Admin

Search Reader ✓

Lab



Lab 1: Initial Setup - Clone repo

Download the workshop code from this repository:

```
git clone https://github.com/BKaneAtWork/CouchbaseJavaWorkshop
```

Build a simple executable jar:

```
cd CouchbaseJavaWorkshop  
mvn clean compile assembly:single
```

Run main class:

```
java -classpath target/CbDevWorkshop-0.0.1-SNAPSHOT-jar-with-dependencies.jar -  
Dcbworkshop.clusteraddress=localhost -Dcbworkshop.user=travel -  
Dcbworkshop.password=password -Dcbworkshop.bucket=travel-sample com.cbworkshop.MainLab
```

Scripts to run the java class (update env variables to match your environment):

- Mac: wrapper.sh
- Windows: wrapper.bat

Lab



INTRODUCTION TO DATA MODELING





The Plan – Introduction to Data Modeling

- Designing a Document Key
- Embedding Data as Complex JSON in Documents
- Create References Between Documents



Designing a Document Key

- Key design is as important as document design.
- There are three broad types of key:
 - Human readable, natural: e.g. an email address
 - Surrogate, computer generated, random: e.g. UUID
 - Compound: e.g. UUID with a natural portion



Human Readable Document Keys

```
key: nic@couchbase.com
```

```
{
  "name": "Nic Raboy",
  "email": "nic@couchbase.com",
  "address": "450 Mission Street",
  "city": "San Francisco",
  "state": "CA"
}
```



Generated Document Keys

```
key: f8d8d616-6ce0-4344-9793-4ec5a676c223
```

```
{
  "name": "Nic Raboy",
  "email": "nic@couchbase.com",
  "address": "450 Mission Street",
  "city": "San Francisco",
  "state": "CA"
}
```



Compound Document Keys

```
key: profile::f8d8d616-6ce0-4344-9793-4ec5a676c223
```

```
{  
  "name": "Nic Raboy",  
  "email": "nic@couchbase.com",  
  "address": "450 Mission Street",  
  "city": "San Francisco",  
  "state": "CA"  
}
```



Counter Document Keys

```
key: msg::66565
```

```
{
  "date": "2018-01-02 23:55:55",
  "user": "nic@couchbase.com",
  "msg": "This is a test message"
}
```

Counters documentation:

https://docs.couchbase.com/java-sdk/2.7/core-operations.html#devguide_kvcore_counter_generic

```
// Java
// Create the counter
bucket.counter("myCounter", 0, 20);

// Use counter for new key
String id = "msg::" + bucket.counter("myCounter", 1).content();
```



Mixed Document Key Example

```
key: profile::f8d8d616-6ce0-4344
```

```
{  
    "name": "Nic Raboy",  
    "email": "nic@couchbase.com",  
    "address": "450 Mission Street",  
    "city": "San Francisco",  
    "state": "CA"  
}
```

```
key: facebook::1234
```

```
{  
    "profile": "f8d8d616-6ce0-4344"  
}
```

```
key: twitter::1234
```

```
{  
    "profile": "f8d8d616-6ce0-4344"  
}
```



Relational Databases

Customers		
customer	firstname	lastname
2342	Nic	Raboy
Products		
product	name	
7182	Nintendo Switch	
Orders		
order	total	customer
1234	\$299.99	2342
Invoice		
order	product	quantity
1234	7182	2

```
SELECT
    c.firstname, c.lastname, p.name, o.total
FROM invoice AS i
JOIN order AS o ON i.order = o.order
JOIN product AS p ON i.product = p.product
JOIN customer AS c ON o.customer = c.customer
```



The Embedded Approach

Customers

customer	firstname	lastname
2342	Nic	Raboy

Products

product	name
7182	Nintendo Switch

Orders

order	total	customer
1234	\$299.99	2342

Invoice

order	product	quantity
1234	7182	2

```
{  
  "order": 1234,  
  "customer": {  
    "customer": "2342",  
    "firstname": "Nic",  
    "lastname": "Raboy"  
  },  
  "products": [  
    {  
      "product": 7182,  
      "name": "Nintendo Switch",  
      "quantity": 2  
    }  
  ],  
  "total": "$299.99"  
}
```



The Referenced Approach

Customers

customer	firstname	lastname
2342	Nic	Raboy

Products

product	name
7182	Nintendo Switch

Orders

order	total	customer
1234	\$299.99	2342

Invoice

order	product	quantity
1234	7182	2

```
{  
  "firstname": "Nic",  
  "lastname": "Raboy"  
}
```

```
{  
  "name": "Nintendo Switch"  
}
```

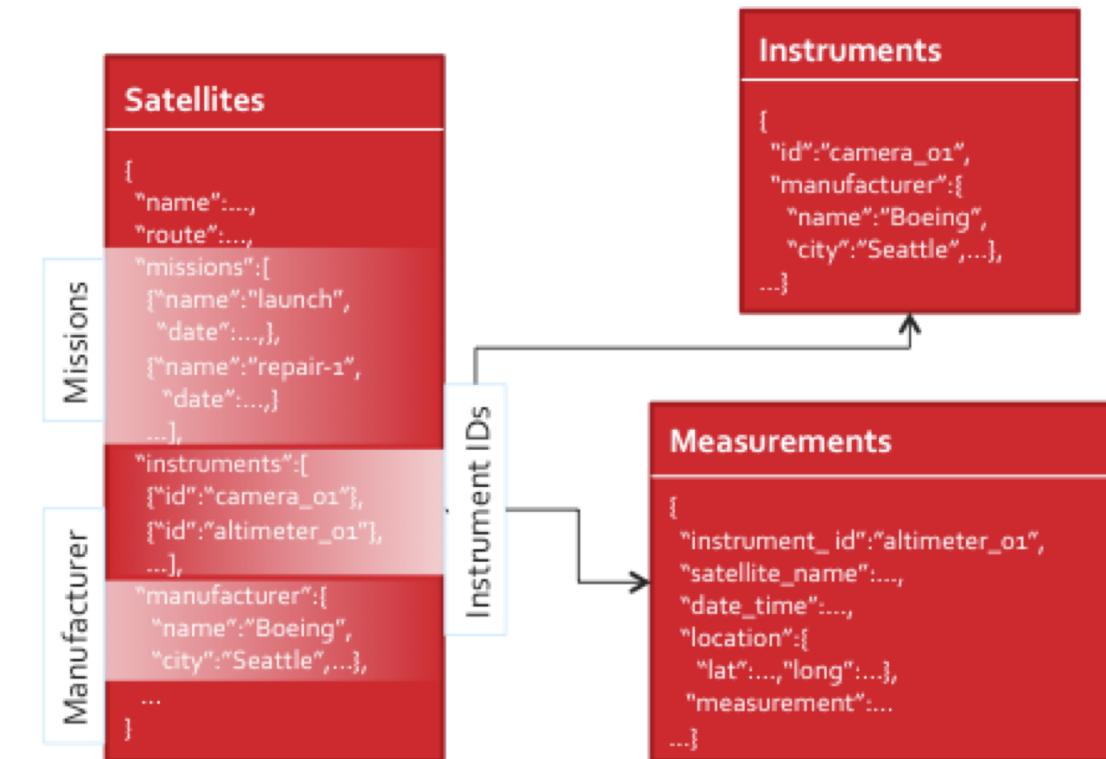
```
{  
  "customer": 2342  
  "total": "$299.99"  
}
```

```
{  
  "order": 1234,  
  "product": 7182,  
  "quantity": 2  
}
```



Embedding vs Referencing

- Embed When
 - Both docs **are** frequently accessed together
 - You need strict consistency between both docs
- Reference When
 - Both docs **aren't** frequently access together
 - You need strict consistency for the referenced information
 - Need to optimize for performance and referenced doc is large





Evolving Data Models

Change your code and you are good to go!

- Schema is App Driven in Couchbase Server
 - OK to maintain multiple versions of the schema
- Schema Changes with Couchbase Server
 - No Explicit Server Action Required by DBA
 - No Downtime required to update existing data to new shape



TRAVEL-SAMPLE DATA MODEL





Lab 2: Browse documents from the UI Console

- Console > Buckets > travel-sample > Documents
- Try Document ID: **route_10003**
- Try Where: **city = "Houston"**
- Create document: Add Document >
 - Document ID: **k_1**
 - Value: **{"name": "luis"}**
- Delete document ID **k_1**

Activity Documentation Support Administrator ▾

CB Workshop > Documents

CLASSIC EDITOR ADD DOCUMENT

Bucket: travel-sample Limit: 10 Offset: 0 Document ID: Where: e.g., 'meta().id = "some_id" and t

Retrieve Docs

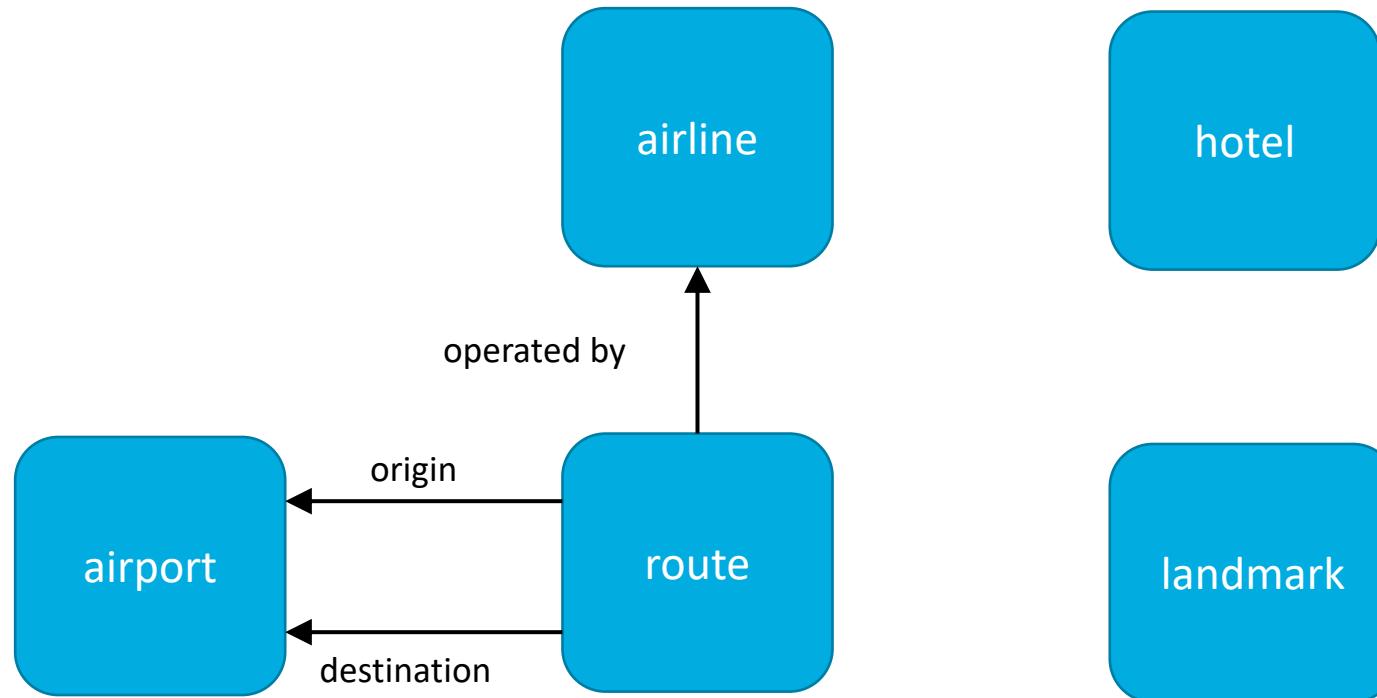
10 Results for travel-sample, limit: 10, offset: 0 simple spreadsheet < Prev Batch Next Batch >

id	airline_10	{"callsign": "MILE-AIR", "country": "United States", "iata": "Q5", "icao": "MLA", "id": 10, "name": "Mile Air", "type": "airline"}
airline_10123	{"callsign": "TXW", "country": "United States", "iata": "TQ", "icao": "TXW", "id": 10123, "name": "TXW", "type": "airline"}	
airline_10226	{"callsign": "atify", "country": "United States", "iata": "A1", "icao": "A1F", "id": 10226, "name": "atify", "type": "airline"}	
airline_10642	{"callsign": null, "country": "United Kingdom", "iata": null, "icao": "JRB", "id": 10642, "name": "JRB", "type": "airline"}	
airline_10748	{"callsign": "LOCAIR", "country": "United States", "iata": "ZQ", "icao": "LOC", "id": 10748, "name": "LOCAIR", "type": "airline"}	
airline_10765	{"callsign": "SASQUATCH", "country": "United States", "iata": "K5", "icao": "SQH", "id": 10765, "name": "SASQUATCH", "type": "airline"}	
airline_109	{"callsign": "ACE AIR", "country": "United States", "iata": "KO", "icao": "AER", "id": 109, "name": "ACE AIR", "type": "airline"}	
airline_112	{"callsign": "FLYSTAR", "country": "United Kingdom", "iata": "5W", "icao": "AEU", "id": 112, "name": "FLYSTAR", "type": "airline"}	
airline_1191	{"callsign": "REUNION", "country": "France", "iata": "UU", "icao": "REU", "id": 1191, "name": "REUNION", "type": "airline"}	
airline_1203	{"callsign": "AIRLINAIR", "country": "France", "iata": "A5", "icao": "RLA", "id": 1203, "name": "AIRLINAIR", "type": "airline"}	

Lab



Travel-sample data model





Travel-sample data model: airline

airline

- Sample key: airline_10123

```
{  
    "callsign": "TXW",  
    "country": "United States",  
    "iata": "TQ",  
    "icao": "TXW",  
    "id": 10123,  
    "name": "Texas Wings",  
    "type": "airline"  
}
```



Travel-sample data model: airport

- Sample key: airport_1256

```
{  
    "airportname": "Les Loges",  
    "city": "Nangis",  
    "country": "France",  
    "faa": null,  
    "geo": {  
        "alt": 428,  
        "lat": 48.596219,  
        "lon": 3.006786},  
    "icao": "LFAI",  
    "id": 1256,  
    "type": "airport",  
    "tz": "Europe/Paris"  
}
```



Travel-sample data model: route

- Sample key: route_10004

```
{  
  "airline": "AF",  
  "airlineid": "airline_137",  
  "destinationairport": "AMS",  
  "distance": 9442.50092891188,  
  "equipment": "777",  
  "id": 10004,  
  "sourceairport": "TPE",  
  "stops": 0,  
  "type": "route",  
}
```

```
"schedule": [  
  {  
    "day": 0,  
    "flight": "AF545",  
    "utc": "13:22:00"  
  },  
  {  
    "day": 0,  
    "flight": "AF350",  
    "utc": "01:21:00"  
  },  
  . . . ,  
  {  
    "day": 6,  
    "flight": "AF250",  
    "utc": "02:32:00"  
  } ]  
}
```



Travel-sample data model: hotel

- Sample key: hotel_10063

```
{  
    "address": "6 rue aux Juifs",  
    "alias": "Les Rouges Gorges",  
    "checkin": null,  
    "checkout": null,  
    "city": "Giverny",  
    "country": "France",  
    "description": "The rustic style ...  
",  
    "directions": "27620 Giverny",  
    "email": null,  
    "fax": null,  
    "free_breakfast": false,  
    "free_internet": true,  
    "free_parking": true,  
    "geo": {  
        "accuracy": "APPROXIMATE",  
        "lat": 49.078069,  
        "lon": 1.520866  
    },  
}
```

```
"id": 10063,  
"name": "The Robins",  
"pets_ok": false,  
"phone": null,  
"price": "60 / 70 euros",  
"public_likes": [],  
"reviews": [ ... ],  
"state": "Haute-Normandie",  
"title": "Giverny",  
"tollfree": null,  
"type": "hotel",  
"url":  
"http://givernyguesthouse.com/robin.h  
tm",  
    "vacancy": true  
}
```



Travel-sample data model: landmark

- Sample key: `landmark_10023`

```
{  
    "activity": "eat",  
    "address": "3 King Street, ME6 1EY",  
    "alt": null,  
    "city": "Gillingham",  
    "content": "Chinese restaurant just off the High Street.",  
    "country": "United Kingdom",  
    "directions": null,  
    "email": null,  
    "geo": {  
        "accuracy": "RANGE_INTERPOLATED",  
        "lat": 51.38697,  
        "lon": 0.54852  },  
    "hours": null,  
    "id": 10023,  
    "image": null,  
    "name": "Beijing Inn",  
    "phone": null,  
    "price": null,  
    "state": null,  
    "title": "Gillingham (Kent)",  
    "tollfree": null,  
    "type": "landmark",  
    "url": http://beijinginn.co.uk/div/  
}
```



CONNECTION BEST PRACTICES





Connection Basics

- Before you can perform KV CRUD, View and/or N1QL Queries
1. You first must connect your application to your cluster
 2. From the cluster connection you can obtain a reference to a Bucket
 - The Bucket is your entry point for the whole storage API
 - Each bucket instance connects to each server:
 - KV 11210/11211
 - Views 8092
 - N1QL 8093
 - Search 8094
 - 8091 is used on demand (fallback)
 - These connections are persistent
 3. Both Cluster and Bucket instances are **thread-safe** and **must** be reused across your application
 - **DO NOT** create new Cluster and Bucket instances per request
 - Impact of not following best practices:
 - Connection starvation
 - Performance issues



Connection Bootstrap Process

1. Connect to the cluster using the Cluster class
 - Default connection is made on 127.0.0.1
 - To connect to your cluster you need to specify at least one Couchbase endpoint
 - Endpoints can be IP addresses or FQDNs
 - **Best Practice:** Use FQDNs
 - **Best Practice:** Supply at least 3 Couchbase endpoints as a seed
 - Specify a **username** and **password** configured with access to desired **bucket** through **RBAC**
2. Using the cluster instance connect to a bucket
 - By default, the bucket connection will connect to the “default bucket” (on 11211)
 - Must supply the **bucket name**
 - The bucket will connect to one of the Couchbase servers in the seed list (1st one in the list)
3. The Couchbase server will return a **CLUSTER MAP**
 - Using the **CLUSTER MAP**, the bucket instance will create connections to each Couchbase server
 - Once the SDK has successfully connected to one node the bootstrap information will be discarded in favor of an up-to-date list of nodes from the cluster



Key/Value – API DML Methods (Retrieval)

- **get**-Retrieves a document or binary key/value.
- **getAndLock**-Lock the document or binary key/value on the server and retrieve it. When a document is locked, its CAS changes and subsequent operations on the document (without providing the current CAS) will fail until the lock is no longer held.
- **getFromReplica**-Get a document binary key/value from a replica server in your cluster.
- **search**- Returns an object which may be iterated over to retrieve search results.
- **unlock**-Unlock a previously locked document or binary key/value on within a bucket.



Key/Value – API DML Methods (Create/Delete/Update)

- **insert**-Insert a document or binary key/value. Fails if the item exists.
- **upsert**-Stores a document or binary key/value to the bucket, or updates if a document exists.
- **replace**-Replaces a document or binary key/value in a bucket. Fails if the item doesn't exist.
- **remove**-Deletes an item from the bucket. Fails if the item doesn't exist
- **append/prepend**-Appends or prepends in place the value of a binary k/v item. Does NOT work with documents
- **touch**-Updates the TTL of a document.
- **getAndTouch**-Retrieves a document or binary key/value and updates the expiry of the item at the same time.
- **counter**-Increments or decrements a key's numeric value.



JAVA SDK CONNECTING TO COUCHBASE





Including the SDK

For the labs, we'll use Maven. Edit the SDK version in the pom.xml:

```
<dependencies>
  <dependency>
    <groupId>com.couchbase.client</groupId>
    <artifactId>java-client</artifactId>
    <version>2.7.4</version>
  </dependency>
</dependencies>
```

TIP. Check latest version available here:

<https://docs.couchbase.com/java-sdk/2.7/start-using-sdk.html>



Connection Basics

```
import com.couchbase.client.java.Cluster;
import com.couchbase.client.java.CouchbaseCluster;
import com.couchbase.client.java.Bucket;
...
Cluster cluster = CouchbaseCluster.create("<host>");
cluster.authenticate("<username>", "<password>");
Bucket bucket = cluster.openBucket("<bucket-name>");
```

- Not like a JDBC Pool
- Reuse Bucket object
- Singleton pattern



Lab 3: Couchbase Connection

- Use source file: MainLab.java
- Implement method: initConnection()
 - Create a CouchbaseEnvironment and set non-default timeout values:

```
CouchbaseEnvironment env = DefaultCouchbaseEnvironment.builder()  
    .operationTracingEnabled(false)  
    .socketConnectTimeout(15000)  
    .connectTimeout(15000)  
    .kvTimeout(15000)  
    .continuousKeepAliveEnabled(true)  
    .keepAliveInterval(10000)  
    .build();
```

- Initialize the connection to the cluster and open the travel-sample bucket (using system properties):

```
Cluster cluster = CouchbaseCluster.create(env, clusterAddress);  
cluster.authenticate(user, password);  
bucket = cluster.openBucket(bucketName);
```

Lab



Lab 3: Couchbase Connection

- Build JAR
- Run application
- Check output:

```
INFO: CouchbaseEnvironment: {sslEnabled=false, sslKeystoreFile='null', sslTruststoreFile='null',
sslKeystorePassword=false, sslTruststorePassword=false, sslKeystore=null, sslTruststore=null, bootstrapHttpEnabled=true,
bootstrapCarrierEnabled=true, bootstrapHttpDirectPort=8091, bootstrapHttpSslPort=18091, bootstrapCarrierDirectPort=11210,
bootstrapCarrierSslPort=11207, ioPoolSize=8, computationPoolSize=8, responseBufferSize=16384, requestBufferSize=16384,
kvServiceEndpoints=1, viewServiceEndpoints=12, queryServiceEndpoints=12, searchServiceEndpoints=12,
configPollInterval=2500, configPollFloorInterval=50, ioPool=NioEventLoopGroup, kvIoPool=null, viewIoPool=null,
searchIoPool=null, queryIoPool=null, coreScheduler=CoreScheduler,
memcachedHashingStrategy=DefaultMemcachedHashingStrategy, eventBus=DefaultEventBus, packageNameAndVersion=couchbase-java-
client/2.7.4 (git: 2.7.4, core: 1.7.4), retryStrategy=BestEffort, maxRequestLifetime=75000,
retryDelay=ExponentialDelay{growBy 1.0 MICROSECONDS, powers of 2; lower=100, upper=100000},
reconnectDelay=ExponentialDelay{growBy 1.0 MILLISECONDS, powers of 2; lower=32, upper=4096},
observeIntervalDelay=ExponentialDelay{growBy 1.0 MICROSECONDS, powers of 2; lower=10, upper=100000},
keepAliveInterval=30000, continuousKeepAliveEnabled=true, keepAliveErrorThreshold=4, keepAliveTimeout=2500,
autoreleaseAfter=2000, bufferPoolingEnabled=true, tcpNodelayEnabled=true, mutationTokensEnabled=false,
socketConnectTimeout=15000, callbacksOnIoPool=false, disconnectTimeout=25000,
requestBufferWaitStrategy=com.couchbase.client.core.env.DefaultCoreEnvironment$2@5d624da6, certAuthEnabled=false,
coreSendHook=null, forceSaslPlain=false, queryTimeout=75000, viewTimeout=75000, searchTimeout=75000,
analyticsTimeout=75000, kvTimeout=15000, connectTimeout=15000, dnsSrvEnabled=false}
Jul 09, 2018 10:27:08 PM com.couchbase.client.core.node.CouchbaseNode signalConnected
INFO: Connected to Node 127.0.0.1/localhost
Jul 09, 2018 10:27:08 PM com.couchbase.client.core.config.DefaultConfigurationProvider$8 call
INFO: Opened bucket travel-sample
```

Lab



JAVA SDK KEY VALUE & CRUD





Creating Documents

- Data can be flat or complex
- Document keys can be custom, automatically generated, or incrementing
- The `insert` operator will create new documents if the key does not already exist
- The `upsert` operator will create or replace

```
JsonObject data = JsonObject.create()
    .put("firstname", "Nic")
    .put("lastname", "Raboy");
JsonArray address = JsonArray.create()
    .add(JsonObject.create().put("city", "Mountain View").put("state", "CA"))
    .add(JsonObject.create().put("city", "San Francisco").put("state", "CA"));
data.put("address", address);
bucket.insert(JsonDocument.create("person-1", data));
```



Retrieving Documents by Key

- Data can be retrieved using a key-value lookup or with a N1QL query
- Lookups are significantly faster than indexed queries with N1QL
- Java can be asynchronous (non-blocking) with RxJava or synchronous

```
// Java  
bucket.get("person-1").content();
```



Durability Requirements

- Durability requirements may be present as an extra option to mutation operations (insert/upsert).
The *level of durability* is configurable depending on the needs of the application.
- The level of durability may be expressed in two values:
 - The *replication factor* of the operation: how many replicas must this operation be propagated to. This is specified as the ReplicateTo option in most SDKs.
 - The *persistence* of the modification: how many persisted copies of the modified record must exist. Often found as a PersistTo option.
- Performance impact: operation times and latencies will increase when waiting on the speed of the network and the speed of the disk.



Lab 4: Create Object

- Implement method: `create(String[] words)`
- In this method, a JSON document for a message is composed, like this:

Key:

msg::some_text

```
{  
    "timestamp": 1511184840248,  
    "from": "luis",  
    "to": "daniel",  
    "type": "msg"  
}
```

- Add a type attribute with value “msg” to the JSON doc and insert/upsert the document to the bucket

```
json.put("type", "msg");  
bucket.insert(JsonDocument.create(key, json));  
//bucket.upsert(JsonDocument.create(key, json));
```

Lab



Lab 4: Create Object

- Build JAR
- Run application
- Try the create method (“create [key from to] ”) several times and see results in console
- Repeat with the same key (an error should appear)

```
# create msg1 Brian Bill
Document created with key: msg::msg1
# create msg2 Brian Bob
Document created with key: msg::msg2
# create msg2 Brian Bob
com.couchbase.client.java.error.DocumentAlreadyExistsException
```

- Repeat with bucket.upsert instead of bucket.insert and observe the results

```
# create msg2 Brian Brandi
Document created with key: msg::msg2
# create msg2 Brian Brandi
Document created with key: msg::msg2
```

Lab



Lab 5: Read Object

- Implement method: `read(String[] words)`
- Parameter for key is read from command line
- Read the document for the specified key from bucket into a `JsonDocument doc`, then uncomment the line to write it to STDOUT:

```
JsonDocument doc = bucket.get(key);
```

- Rebuild the JAR and run the application
- Test with values: `airline_10226, route_10009, hotel_10904`

```
# read airline_10226
{"country":"United States","iata":"A1","callsign":"atifly","name":"Atifly",
 "icao":"A1F","id":10226,"type":"airline"}
# read route_10009
{"schedule":[{"flight":"AF665","utc":"13:22:00","day":0}, {"flight":"AF232","utc":"16:10:00",
 "day":1}, {"flight":"AF137","utc":"09:29:00","day":1}, {"flight":"AF399",...}
```

- **Bonus:** implement code to output a friendly message when document is not found

Lab



Lab 6: Update Object

- Implement method: update(String[] words):
- Parameter for key is read from command line
- Read the document, modify attribute “name” to UPPERCASE the value, then use replace to modify the document:

```
JsonDocument doc = bucket.get(key);
String name = doc.content().getString("name");
doc.content().put("name", name.toUpperCase());
bucket.replace(doc);
```

- Rebuild the JAR and run the application
- Test with key airline_10642 by reading and updating, then re-reading:

```
# read airline_10642
{"country":"United Kingdom","iata":null,"callsign":null,"name":"Jc
royal.britannica","icao":"JRB","id":10642,"type":"airline"}
# update airline_10642
# read airline_10642
{"country":"United Kingdom","iata":null,"callsign":null,"name":"JC
ROYAL.BRITANNICA","icao":"JRB","id":10642,"type":"airline"}
```

Lab



Lab 7: Delete Object

- Implement method: delete(String[] words)
- Parameter for key is read from command line
- Delete the document:

```
bucket.remove(key);
```

- Rebuild the JAR and run the application
- To test, use create, then delete the same msg key:

```
# create 1001 Brian Doug
Document created with key: msg::1001
# read msg::1001
{"from":"Brian","to":"Doug","type":"msg","timestamp":1536513089236}
# delete msg::1001
# read msg::1001
java.lang.NullPointerException
        at com.cbworkshop.MainLab.read(MainLab.java:145)
        at com.cbworkshop.MainLab.process(MainLab.java:91)
        at com.cbworkshop.MainLab.main(MainLab.java:55)
#
```

Lab



JAVA SDK

WORKING WITH PARTS OF A DOCUMENT





The Plan – Working with Parts of a Document

- Get Parts of a JSON Document
- Update JSON Properties in a Document
- Batch Subdocument Operations Together



Large Documents

```
key: nraboy
```

```
{
  "profile": {
    "firstname": "Nic",
    "lastname": "Raboy"
  },
  "data": [
    // 20MB of data
  ]
}
```



Get Part of a Document

```
// Java
DocumentFragment<Lookup> result = bucket
    .lookupIn("nraboy")
    .get("profile")
    .execute();
```



Update Part of a Document

```
// Java
DocumentFragment<Mutation> result = bucket
    .mutateIn("nraboy")
    .replace("profile.firstname", "Nicolas")
    .execute();
```



Chain Subdocument Operations

```
DocumentFragment<Mutation> result = bucket
    .mutateIn("nraboy")
    .replace("profile.firstname", "Nic")
    .insert("profile.gender", "Male")
    .remove("data")
    .withDurability(PersistTo.MASTER, ReplicateTo.NONE)
    .execute();
```

```
DocumentFragment<Lookup> result = bucket
    .lookupIn("myKey")
    .get("sub.value")
    .exists("fruits")
    .execute();
```

```
String subValue = result.content("sub.value", String.class);
boolean fruitsExist = result.content("fruits", Boolean.class);
```



Lab 8: SubDocument API example

- Implement method: subdoc(String[] words)
- Parameter for key is read from command line
- Using SubDocument API:
 - Change the existing value of the “from” attribute to “Administrator”
 - Add a new attribute: “reviewed”, with value CurrentTimeMillis

```
DocumentFragment<Mutation> result = bucket
    .mutateIn(key)
    .replace("from", "Administrator")
    .insert("reviewed", System.currentTimeMillis())
    .execute();
```

Lab



Lab 8: SubDocument API example

- Rebuild the JAR and run the application
- To test, use create, then update the same msg key:

```
# create 1001 Brian Colin
Document created with key: msg::1001
# read msg::1001
{"from":"Brian","to":"Colin","type":"msg","timestamp":1536593956715}
# subdoc msg::1001
# read msg::1001
{"reviewed":1536593971600,"from":"Administrator","to":"Colin","type":"msg","timestamp":1
536593956715}
#
```

Lab



INTRODUCTION TO N1QL: SQL FOR JSON





The Plan – Querying for Documents with N1QL

- Understand the Power of N1QL
- Use Query Workbench and the Couchbase CLI
- Create Indexes for N1QL Queries
- Query for Data with N1QL
- Create Data with N1QL
- Remove Data with N1QL



What is N1QL?

- **Non-first (N1) Normal Form Query Language (QL)**
 - It is based on ANSI 92 SQL
 - Its query engine is optimized for modern, highly parallel multi-core execution
- SQL-like Query Language
 - Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data
- N1QL extends SQL to handle data that is:
 - **Nested:** Contains nested objects, arrays
 - **Heterogeneous:** Schema-optional, non-uniform
 - **Distributed:** Partitioned across a cluster



Power of SQL

Flexibility of
JSON



N1QL (EXPRESSIVE)

- Access to every part of JSON document
- Scalar & Aggregate functions
- Subqueries in the FROM clause
- Aggregation on arrays

Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data



N1QL (FAMILIAR)

- SELECT * FROM bucket WHERE ...
- INSERT single & multiple documents
- UPDATE any part of JSON document & use complex filter
- DELETE
- MERGE two sets of documents using traditional MERGE statement
- EXPLAIN to understand the query plan
 - EXPLAIN SELECT * FROM bucket WHERE ...

Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data



N1QL (FEATURE-RICH)

- Access to every part of JSON document
- Functions (Date, Pattern, Array, Conditional, etc)
 - <https://docs.couchbase.com/server/6.0/n1ql/n1ql-language-reference/functions.html>
- ANSI JOINS (new in 5.5)
- Covering Index
- Prepared Statements
- USE KEYS
- LIKE

Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data



N1QL – Example 1

The first example uses a JOIN clause to find the distinct airline details which have routes that start from SFO.

```
SELECT DISTINCT airline.name, airline.callsign, route.destinationairport,  
route.stops, route.airline  
FROM `travel-sample` route  
JOIN `travel-sample` airline ON KEYS route.airlineid  
WHERE route.type = "route"  
AND airline.type = "airline"  
AND route.sourceairport = "SFO";
```

```
[  
  {  
    "airline": "UA",  
    "callsign": "UNITED",  
    "destinationairport": "ACV",  
    "name": "United Airlines",  
    "stops": 0  
  },  
  {  
    "airline": "AA",  
    "callsign": "AMERICAN",  
    "destinationairport": "JFK",  
    "name": "American Airlines",  
    "stops": 0  
  }]
```



N1QL – Example 2

Which airlines fly from each airport? - ARRAY_AGG function

```
SELECT route.sourceairport, ARRAY_AGG(DISTINCT airline.name) as airlines
FROM `travel-sample` route
JOIN `travel-sample` airline ON KEYS route.airlineid
WHERE route.type="route"
AND airline.type="airline"
GROUP BY route.sourceairport
ORDER BY route.sourceairport;
```

```
[
  {
    "airlines": [
      "American Airlines",
      "US Airways"
    ],
    "sourceairport": "ABI"
  },
```



N1QL – Example 3

Which airports can I reach by flying from San Francisco? - ANSI Join

Required index:

```
CREATE INDEX route_airports ON `travel-sample`(sourceairport, destinationairport) WHERE type = "route";
```

```
SELECT DISTINCT route.destinationairport
FROM `travel-sample` airport JOIN `travel-sample` route
    ON airport.faa = route.sourceairport
WHERE airport.type = "airport"
    AND route.type = "route"
    AND airport.city = "San Francisco"
    AND airport.country = "United States";
```

```
[
  {
    "destinationairport": "ABQ"
  },
  {
    "destinationairport": "ACV"
  },
```



Query Workbench and the Couchbase CLI

- Query Workbench
 - <http://localhost:8091/ui/index.html#!/query/workbench>
- Couchbase CLI (Mac)
 - /Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbq
- Couchbase CLI (Windows)
 - C:\Program Files\CouchbaseServer\bin\cbq.exe
- Couchbase CLI (Linux)
 - /opt/couchbase/bin/cbq



CRUD IN N1QL

C

```
INSERT INTO `travel-sample`  
(KEY, VALUE)  
VALUES ("person-1", {  
    "firstname": "Nic",  
    "lastname": "Raboy"  
})
```

R

```
SELECT name FROM `travel-sample` WHERE type="airline" AND icao="JRB"
```

U

```
UPDATE `travel-sample` USE KEYS ["hotel_10138"]  
SET pets_ok=true, free_internet=true
```

D

```
DELETE FROM `travel-sample`  
WHERE META().id = 'person-1'
```



Why do we need indexes?

Indexes are used to
efficiently look up objects
meeting ***user specified criteria***
without having to
search ***every*** object
in the collection



Primary Index

- The primary index is the index on the document key on every document in a keyspace
- The primary index is used for:
 - Complete keyspace scan – equivalent of table scan
 - Query does not have any predicates (filters)
 - Query has predicate on document key
 - No other index or access path can be used
- Primary index is optional

```
CREATE PRIMARY INDEX ON `travel-sample`;  
  
SELECT *  
FROM `travel-sample`;  
  
SELECT *  
FROM `travel-sample`  
WHERE META().id LIKE "user::%";
```



Secondary Index

- Simple Index
- Composite Index
- Functional Index
- Partial Index
- Replica Index
- Covering Index



Indexes

CB Workshop > Indexes

Global Indexes ▾ Views

view by server node ▾ filter indexes... 🔍

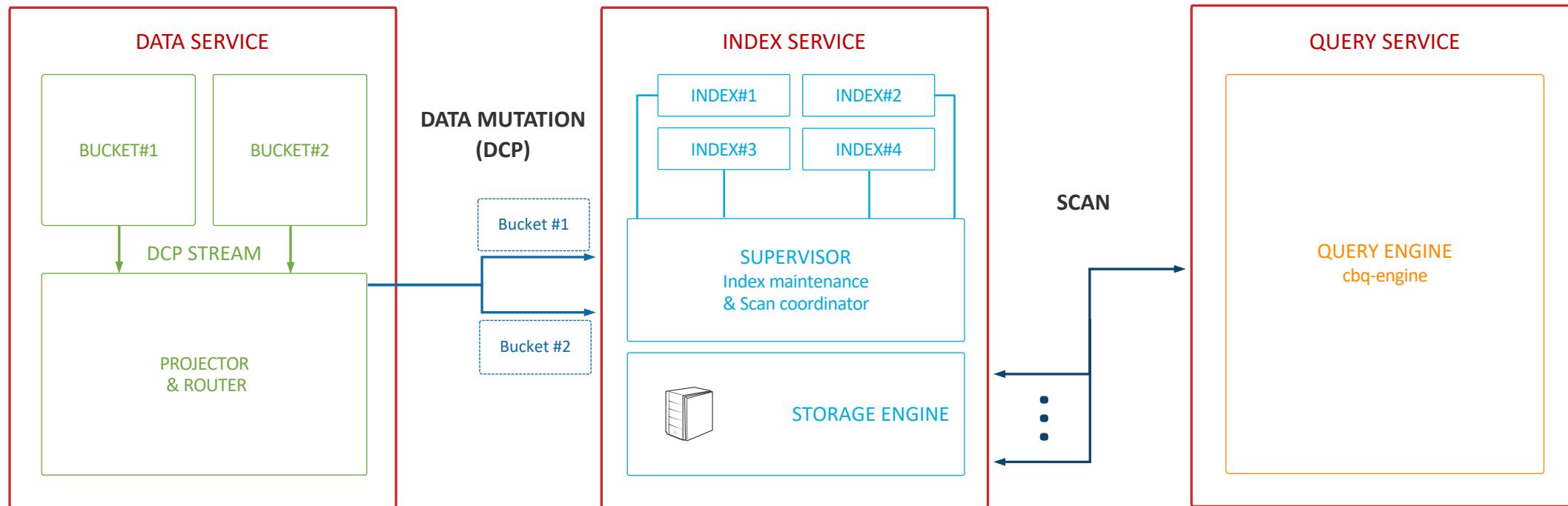
127.0.0.1:8091

index name ▾	bucket	storage type	build progress	
def_airportname	travel-sample	Standard GSI	100%	Drop
def_city	travel-sample	Standard GSI	100%	Drop
def_faa	travel-sample	Standard GSI	100%	Drop
def_icao	travel-sample	Standard GSI	100%	Drop
def_name_type	travel-sample	Standard GSI	100%	Drop
def_primary	travel-sample	Standard GSI	100%	Drop

Dashboard
Servers
Buckets
XDCR
Security
Settings
Logs
Documents
Query
Search
Analytics
Eventing
Indexes



Couchbase Services





JAVA SDK N1QL QUERIES





Raw Query String vs. DSL

Raw query string

```
N1qlQueryResult queryResult =  
bucket.query(N1qlQuery.simple("SELECT * FROM `travel-sample` LIMIT 10"));
```

Using the DSL

```
import static com.couchbase.client.java.query.Select.select;  
import static com.couchbase.client.java.query.dsl.Expression.*;  
  
...  
N1qlQueryResult queryResult =  
bucket.query(select("*").from("`travel-sample`").limit(10));
```

Iterate result

```
for(N1qlQueryRow row : queryResult){  
    System.out.println(row.value().toString());  
}
```



Lab 9: Simple Query

- Implement method: query ()
- Execute the query: "SELECT * FROM `travel-sample` LIMIT 10"
- Print the results to STDOUT
- Use both implementations, first raw and then DSL

```
N1qlQueryResult queryResult;
// Raw
queryResult = bucket.query(N1qlQuery.simple("SELECT * FROM `travel-sample` LIMIT 10"));
for(N1qlQueryRow row : queryResult) {
    System.out.println("Raw " + row.value().toString());
}
// DSL
queryResult = bucket.query(select("*").from("`travel-sample`").limit(10));
for(N1qlQueryRow row : queryResult) {
    System.out.println("DSL " + row.value().toString());
}
```

Lab



Lab 9: Simple Query

- Rebuild the JAR and run the application
- Test using the query command

```
# query
{"travel-sample": {"country": "United States", "iata": "Q5", "callsign": "MILE-AIR", "name": "40-Mile Air", "icao": "MLA", "id": 10, "type": "airline"} }
{"travel-sample": {"country": "United States", "iata": "TQ", "callsign": "TXW", "name": "Texas Wings", "icao": "TXW", "id": 10123, "type": "airline"} }
{"travel-sample": {"country": "United States", "iata": "A1", "callsign": "atifly", "name": "Atifly", "icao": "A1F", "id": 10226, "type": "airline"} }
{"travel-sample": {"country": "United Kingdom", "iata": null, "callsign": null, "name": "JC ROYAL.BRITANNICA", "icao": "JRB", "id": 10642, "type": "airline"} }
{"travel-sample": {"country": "United States", "iata": "ZQ", "callsign": "LOCAIR", "name": "Locair", "icao": "LOC", "id": 10748, "type": "airline"} }
{"travel-sample": {"country": "United States", "iata": "K5", "callsign": "SASQUATCH", "name": "SeaPort Airlines", "icao": "SQH", "id": 10765, "type": "airline"} }
{"travel-sample": {"country": "United States", "iata": "KO", "callsign": "ACE AIR", "name": "Alaska Central Express", "icao": "AER", "id": 109, "type": "airline"} }
{"travel-sample": {"country": "United Kingdom", "iata": "5W", "callsign": "FLYSTAR", "name": "Astraeus", "icao": "AEU", "id": 112, "type": "airline"} }
{"travel-sample": {"country": "France", "iata": "UU", "callsign": "REUNION", "name": "Air Austral", "icao": "REU", "id": 1191, "type": "airline"} }
{"travel-sample": {"country": "France", "iata": "A5", "callsign": "AIRLINAIR", "name": "Airlinair", "icao": "RLA", "id": 1203, "type": "airline"} }
#
```

Lab



Query with parameters

Create a new query with named parameters from a JsonObject.

Named parameters have the form of \$name, where the name represents the unique name.

```
JsonObject params = JsonObject.create()  
    .put("src", "JFK")  
    .put("dst", "LHR");
```

```
String queryStr = "SELECT a.name FROM `travel-sample` r JOIN `travel-sample` a ON  
KEYS r.airlineid WHERE r.type='route' AND r.sourceairport=$src AND  
r.destinationairport=$dst";
```

```
N1qlQuery query = N1qlQuery.parameterized(queryStr, params);
```



Lab 10: Query with parameters

- Implement method: queryAirports(String[] words)
- Parameters (sourceairport, destinationairport) are read from command line and added to the params JsonObject
- Write a parameterized query with a join to find airlines flying from sourceairport to destinationairport and print the results to STDOUT

```
String queryStr = "SELECT a.name FROM `travel-sample` r JOIN `travel-sample` a " +
    "ON KEYS r.airlineid WHERE r.type='route' AND " +
    "r.sourceairport=$src AND r.destinationairport=$dst";

N1qlQuery query = N1qlQuery.parameterized(queryStr, params);

N1qlQueryResult queryResult = bucket.query(query);
for(N1qlQueryRow row : queryResult) {
    System.out.println(row.value().toString());
}
```

Lab



Lab 10: Query with parameters

- Rebuild the JAR and run the application
- Test using the queryairports command and two airport codes
 - **TIP:** Highest traffic airport codes are ATL, ORD, LHR, CDG, LAX, DFW, JFK

```
# queryairports JFK LHR
{"name": "British Airways"}
{"name": "Delta Air Lines"}
{"name": "American Airlines"}
{"name": "US Airways"}
{"name": "Virgin Atlantic Airways"}
{"name": "Air France"}  
#
```

Lab



Query Consistency

- not_bounded
 - Fastest
 - Returns data that is currently indexed and accessible by the index or the view.
- at_plus
 - Fast
 - A query submitted with at_plus consistency level requires all mutations, up to the moment of the scan_vector (the logical timestamp passed in with at_plus), to be processed before the query execution can start.
- request_plus
 - Slowest, but still fast
 - Requires all mutations, up to the moment of the query request, to be processed before the query execution can start.



Query Consistency

Sample code

```
// Java
N1qlQueryResult result =
    bucket.query(
        N1qlQuery.simple(
            statement,
            N1qlParams.build().consistency(ScanConsistency.REQUEST_PLUS)
        )
    );
```



JAVA SDK FULL TEXT SEARCH





Full Text Search vs N1QL

- Wildcard Queries are Slow
- What happens to variations of the wildcard term in N1QL / SQL?
 - '%golang%', does 'Golang' or 'GoLang' pop up?
 - '%golang%', do spelling errors like 'goang' pop up?
- FTS is for Natural Language Querying



Full Text Search – Query Types

Couchbase Full Text Search supports multiple types of search queries:

- Simple Queries
 - Match
 - Match Phrase
 - Fuzzy
 - Prefix
 - Regexp
 - Wildcard Query
 - Boolean Field
- Compound, Range, Query String, Geospatial, Non-analytic, Special

Full list: <https://docs.couchbase.com/server/6.0/fts/fts-query-types.html>



Lab 11: FTS – Create index

- Build an FTS index on travel-sample for hotels (type="hotel") on the attribute "description":
- Index name: sidx_hotel_desc
- Bucket: travel-sample
- Type Mappings: uncheck default
- Add Type Mapping:
 - type name: hotel
 - Check "only index specified fields"
 - Click "ok"
- + to insert child field: description; click "ok"
- Click "Create Index"

CB Workshop > Full Text Search > Add Index

Dashboard **Servers** **Buckets** **XDCR** **Security** **Settings** **Logs**
Documents **Query** **Search** **Analytics** **Eventing** **Indexes**

Name: sidx_hotel_desc **Bucket**: travel-sample

Type Identifier:
① JSON type field: type
② Doc ID up to separator: delimiter
③ Doc ID with regex: regular expression

Type Mappings:
+ Add Type Mapping
hotel | only index specified fields
field: description | ok | cancel
type: text
searchable as: description
analyzer: inherit
index: checked | store: unchecked | include in_all field: checked | include term vectors: checked
default | disabled | dynamic

▶ Analyzers
▶ Custom Filters
▶ Date/Time Parsers
▶ Advanced

Index Replicas: 0 **Index Type**: Version 6.0 (Scorch)

Create Index Cancel

Lab



Lab 11: FTS – Create index

- Search for “farm”

CB Workshop > Full Text Search > Search Results

farm show advanced query settings

full text query syntax help

Results for sidx_hotel_desc

Show Scoring 7 results (2ms server-side)

Rank	Document ID
1.	hotel_40177
2.	hotel_15134
3.	hotel_27626
4.	hotel_1507
5.	hotel_4399
6.	hotel_28259
7.	hotel_5846

Documents

Query

Search

Analytics

Eventing

Indexes

Dashboard

Servers

Buckets

XDCR

Security

Settings

Logs

Advanced

Default Analyzer:

select “en”,

Update Index

Test:

- “farms”
- “farming”

Lab



Full Text Search: Java example

```
String term = ... ;  
  
MatchQuery fts = SearchQuery.match(term);  
SearchQueryResult result = bucket.query(new SearchQuery("sidx_hotel_desc", fts));  
for (SearchQueryRow row : result) {  
    System.out.println(row);  
}
```



Lab 12: FTS – Java example

- Implement method: search(String[] words)
- Parameter for the search term is read from command line
- Write code to search using the index sidx_hotel_desc for the search term and print results to STDOUT

```
MatchQuery fts = SearchQuery.match(term);
SearchQueryResult result = bucket.query(new SearchQuery("sidx_hotel_desc", fts));
for (SearchQueryRow row : result) {
    System.out.println(row);
}
```

Lab



Lab 12: FTS – Java example

- Rebuild the JAR and run the application
- Test using the search command with a search term, e.g. “farm”

```
# search farm
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_13aa53f3', id='hotel_40177', score=2.961906336277922,
explanation={}, locations=DefaultHitLocations{size=0, locations=[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_aa574717', id='hotel_15134', score=2.31408575442229,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_54820232', id='hotel_27626', score=1.956240330221747,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_18572d87', id='hotel_1507', score=1.9547615867497852,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_54820232', id='hotel_4397', score=1.6941537614950384,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_aa574717', id='hotel_4399', score=1.5968708706127248,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_f4e0a48a', id='hotel_28259', score=1.4280208176277713,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
DefaultSearchQueryRow{index='sidx_hotel_desc_66ab355a2a9392b8_6ddbfb54', id='hotel_5846', score=1.002838035409521,
explanation={}, locations=DefaultHitLocations{size=0, locations[], fragments={}, fields={}}
```

Lab



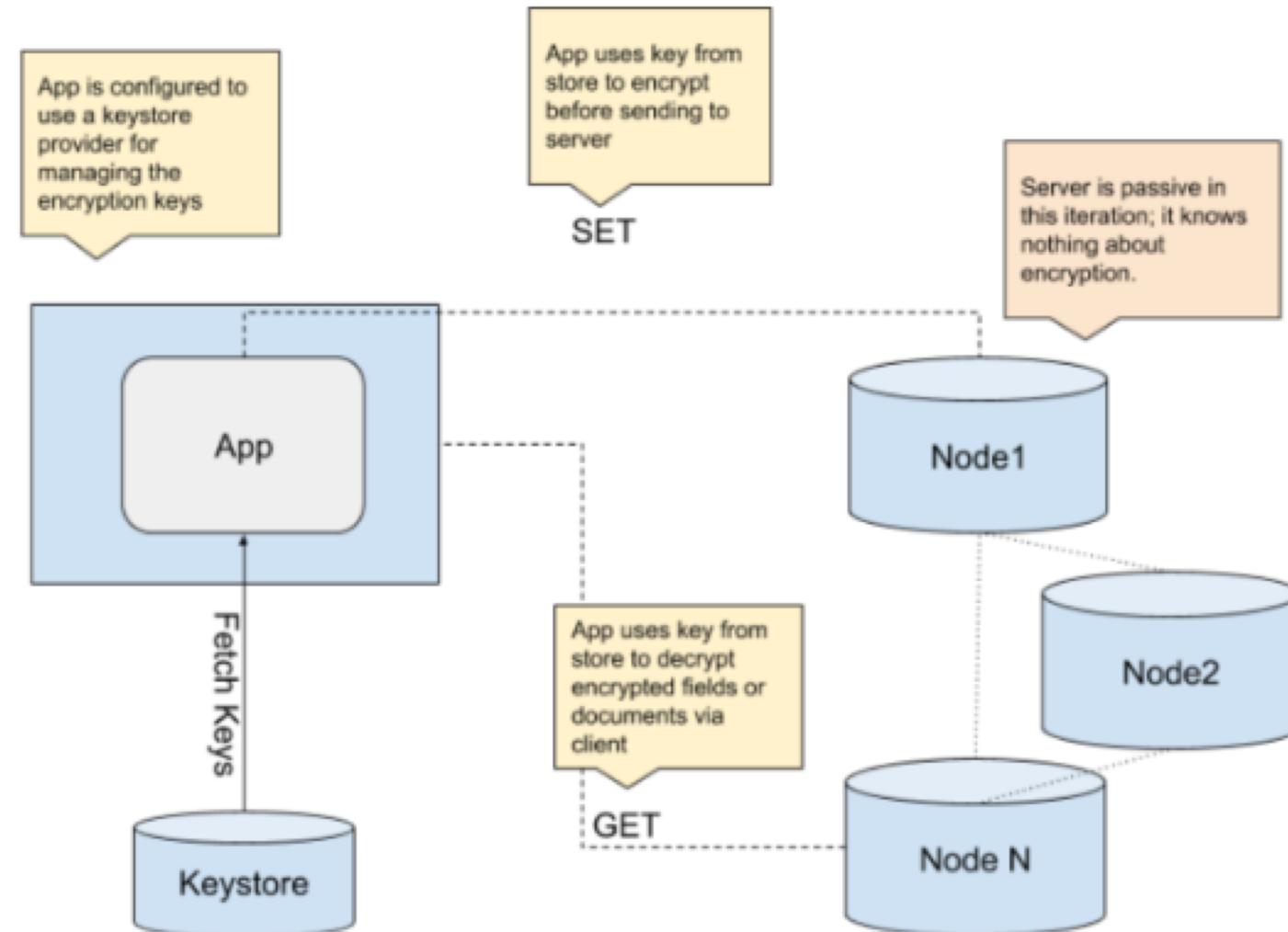
JAVA SDK FIELD LEVEL ENCRYPTION





Field Level Encryption

- We support field-level encryption in our SDKs to protect sensitive user data.
- Fields within a document can be securely encrypted by the SDK, to support FIPS-140-2 compliance.
- This is a client-side implementation, with key management and initialization of data encryption done during configuration of the SDK, which then exposes the API during runtime, for normal read/write operations.
- In Couchbase Data Platform, AES-256, AES-128, and RSA are all supported. Native Keystores (including Java Key Store and Windows Certificate Store) are supported, as well as an in-memory keystore for development and testing.





Field Level Encryption

- Behind the API, the format displayed here is used internally to encompass both the temporary field name used to hold the encrypted value, plus the additional metadata associated with the algorithm used and the public key:
- This is how Couchbase stores the JSON internally, but it is exposed to the developer as a public API.
- It's important that encrypted fields be treated as "special" fields and not mutated by APIs other than through the Field Level Encryption (FLE) API. For example, for AES-HMAC-SHA256 the entire temporary field is signed; if any changes are made to any fields ("alg", "kid", "ciphertext", "sig" or "iv") then the decryption will fail because the signature has changed.
- Field Level Encryption for all SDKs is a separate package from the Couchbase SDK itself; Additional Maven dependency.

```
__[PREFIX]_[FIELDNAME] : {  
    "kid" : "[KEY_IDENTIFIER]",  
    "alg": "[ALGORITHM"]],  
    "ciphertext": "[BASE64_ENCRYPTED_DATA]",  
    "sig": "[BASE64_HMAC_SIGNATURE]",  
    "iv" : ["INITIALIZATION_VECTOR"]  
}
```



Field Level Encryption: Java example

Create a configuration to connect to your Couchbase cluster and configure the crypto manager in the Couchbase environment with the encryption crypto algorithm and key store providers.

To apply encryption to a repository entity, use the `EncryptedField` annotation with provider name as registered on the crypto manager:

```
@EncryptedField(provider = "AES")
public String password;
```

Encryption can also be applied to `JsonObjects`:

```
JsonObject.create().putAndEncrypt("foo", "bar", "AES");
```

Encrypted fields in entities are decrypted based on the annotations.

To decrypt use fields in `JsonObject`, use `getAndDecrypt` methods:

```
JsonDocument stored = bucket.get("mydoc");
stored.content().getAndDecrypt("foo", "AES")
```



Lab 13: Field Level Encryption

- Modify method: initConnection()
 - Setup key store and encryption configuration manager:

```
JceksKeyStoreProvider kp = new JceksKeyStoreProvider("secret");
kp.storeKey("SecretKey", "aabbccddqqnnmmeerryybbff3322kk99".getBytes());
kp.storeKey("HMACsecret", "myauthsecret".getBytes());
kp.publicKeyName("secretkey");
kp.signingKeyName("hmacsecret");
AES256CryptoProvider aes256CryptoProvider = new AES256CryptoProvider(kp);
CryptoManager cryptomanager = new CryptoManager();
cryptomanager.registerProvider("AES", aes256CryptoProvider);
```

- Include the tracer in the Environment builder:

```
env = DefaultCouchbaseEnvironment.builder()
    .cryptoManager(cryptomanager)
    ...
```

Lab



Lab 13: Field Level Encryption

- 13A: Implement method: encryptPut(String[] words)
- Parameters (key, ccNum) are read from command line and added to a JsonObject
- Using Field Level Encryption API:
 - Encrypt the credit card number (ccNum) with the AES encryption provider and add to JsonObject
 - Add a timestamp attribute: “reviewed”, with value of currentTimeMillis
 - Create a JsonDocument from the key and JsonObject, and upsert the doc into the bucket

```
JsonObject jObj = JsonObject.create()
    .putAndEncrypt("ccNum", ccNum, "AES")
    .put("timestamp", System.currentTimeMillis());
JsonDocument doc = JsonDocument.create(key, jObj);

bucket.upsert(doc);
```

Lab



Lab 13: Field Level Encryption

- 13B: Implement method: encryptGet(String[] words)
- Parameter for the key is read from command line
- Using Field Level Encryption API:
 - Get the document by key
 - Within a try/catch block, decrypt the credit card number field (ccNum) with the AES encryption provider
 - Write the value to STDOUT

```
try {  
    String decryptedString =  
        bucket.get(key).content().toDecryptedString("AES");  
    System.out.println(decryptedString);  
} catch (Exception e) {  
    System.out.println(e);  
}
```

Lab



Lab 13: Field Level Encryption

- Rebuild the JAR and run the application
- Test using the fleput command to create a doc with an encrypted field, then use the read and f eget commands to see the encrypted and unencrypted field values

```
# fleput myKey 1234-5678-9012-3456
Running encrypt put on field ccNum for key myKey
# read myKey
{"__crypt_ccNum":{"sig":"x7JnVpkj4Y8hydltN3icg+pLi2SJMPA9O9H4dxU+20s=","ciphertext":"nqJeB8hvvlT1oKQtu0KSYwm0EXHXMOCIYeKhFpx3TDY=","alg":"AES-256-HMAC-SHA256","iv":"qixS8dxQ6Bud/Qe7zDH91w==","kid":"secretkey"},"timestamp":1553664508421}
# f eget myKey
Running encrypt get on doc myKey
{"ccNum":"1234-5678-9012-3456","timestamp":1553664508421}
#
```

Lab



JAVA SDK THRESHOLD LOGGING





Threshold Logging

- Response Time Observability, introduced in Couchbase 5.5, is implemented as Threshold Logging in Java SDK from release 2.6.0.
- Provides telemetry for monitoring how well your Couchbase cluster is meeting your application SLAs.
 - In your application you can set thresholds to allow you to log operations based on performance of your requests.
 - You can correlate timing information from client through to server and back to quickly triage performance issues and help identify the root cause of performance issues.
- The performance of operations are all exposed via methods you are already familiar with in the Couchbase SDK.
- Couchbase SDK team selected OpenTracing as their tracing API:
 - Separate the data that is being collected from the mechanism that collects and does something with that data
 - Couchbase SDKs provide a Tracer implementation out of the box called the ThresholdLoggingTracer
 - Use any OpenTracing compatible tracer implementation: open source (e.g. jaeger), or one from APM vendors



Threshold Logging

Output fields in detail:

- **total_us**: The total time it took to perform the full operation: here around 1.5 milliseconds.
- **server_us**: The server reported that its work performed took 23 microseconds (this does not include network time or time in the buffer before picked up at the cluster).
- **decode_us**: Decoding the response took the client 1.2 milliseconds.
- **last_dispatch_us**: The time when the client sent the request and got the response took around 1 millisecond.
- **last_local_address**: The local socket used for this operation.
- **last_remote_address**: The remote socket on the server used for this operation. Useful to figure out which node is affected.
- **last_operation_id**: A combination of type of operation and id (in this case the opaque value), useful for diagnosing and troubleshooting in combination with the last_local_id.
- **last_local_id**: With Server 5.5 and later, this id is negotiated with the server and can be used to correlate logging information on both sides in a simpler fashion.

```
{  
    "server_us" : 23,  
    "last_local_id" : "41837B87B9B1C5D1/000000004746B9AA",  
    "last_local_address" : "127.0.0.1:55011",  
    "last_operation_id" : "get:0x1",  
    "decode_us" : 1203,  
    "last_dispatch_us" : 947,  
    "last_remote_address" : "127.0.0.1:11210",  
    "total_us" : 1525  
}
```



Threshold Logging: Java example

You can customize Tracer settings through the `CouchbaseEnvironment.Builder`.

Here is an example of how to customize our default tracer to:

- reduce the threshold for KV operations from 500 ms to 100 ms
- reduce the time interval when the information gets logged from 1 minute to 10 seconds

```
Tracer tracer = ThresholdLogTracer.create(ThresholdLogReporter.builder()
    .kvThreshold(100, TimeUnit.MILLISECONDS) // reduce default 500ms threshold
    .logInterval(10, TimeUnit.SECONDS)        // log every 10 seconds
    .build());
```

```
CouchbaseEnvironment env = DefaultCouchbaseEnvironment.builder()
    .tracer(tracer)
    .build();
```

```
Cluster cluster = CouchbaseCluster.create(env, "127.0.0.1");
cluster.authenticate("username", "password");
```

```
Bucket bucket = cluster.openBucket("travel-sample");
```



Lab 14: Threshold Logging

- Modify method: `initConnection()`
 - Create a create a tracer with a KV threshold of 1 microsecond:

```
Tracer tracer = ThresholdLogTracer.create(ThresholdLogReporter.builder()  
    .kvThreshold(1, TimeUnit.MICROSECONDS) // 1 microsecond  
    .logInterval(1, TimeUnit.SECONDS) // log every second  
    .sampleSize(Integer.MAX_VALUE)  
    .pretty(true) // pretty print the json output in the logs  
    .build());
```

- Include the tracer in the Environment builder, and enable tracing:

```
env = DefaultCouchbaseEnvironment.builder()  
    .tracer(tracer)  
    .operationTracingEnabled(true)  
    ...
```

Lab



Lab 14: Threshold Logging

- Rebuild the JAR and run the application
- Test using the threshold command (or read or update)

```
# threshold
Mar 27, 2019 6:30:59 AM com.couchbase.client.core.tracing.ThresholdLogReporter logOverThreshold
WARNING: Operations over threshold: [ {
    "top" : [ {
        "operation_name" : "get",
        "server_us" : 2320,
        "last_local_id" : "0A2BBE8639B1BCAD/000000007B702834",
        "last_local_address" : "127.0.0.1:61928",
        "last_remote_address" : "127.0.0.1:11210",
        "last_dispatch_us" : 5099,
        "decode_us" : 8144,
        "last_operation_id" : "0x2",
        "total_us" : 18600
    }, {
        "operation_name" : "upsert",
        "server_us" : 108,
        "last_local_id" : "0A2BBE8639B1BCAD/000000007B702834",
        "encode_us" : 3442,
        "last_local_address" : "127.0.0.1:61928",
        "last_remote_address" : "127.0.0.1:11210",
        "last_dispatch_us" : 2647,
        "last_operation_id" : "0x3",
        "total_us" : 10597
    } ],
    "service" : "kv",
    "count" : 2
} ]
```

Lab



JAVA SDK BATCHING WITH RXJAVA





Batching with RxJava

Implicit batching is performed by utilizing a few operators:

- **Observable.just()** or **Observable.from()** to generate an Observable that contains the data you want to batch on.
- **flatMap()** to send those events against the Couchbase Java SDK and merge the results asynchronously.
- **last()** if you want to wait until the last element of the batch is received.
- **toList()** if you care about the responses and want to aggregate them easily.



Batching with RxJava

- The following example creates an observable stream of 5 keys to load in a batch,
- asynchronously fires off get() requests against the SDK,
- waits until the last result has arrived,
- and then converts the result into a list and blocks at the very end

```
Cluster cluster = CouchbaseCluster.create();
Bucket bucket = cluster.openBucket();

List<JsonDocument> foundDocs = Observable
    .just("key1", "key2", "key3", "key4", "key5")
    .flatMap(new Func1<String, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(String id) {
            return bucket.async().get(id);
        }
    })
    .toList()
    .toBlocking()
    .single();
```



Batching with RxJava - Batching mutations

- The following code generates a number of fake documents and inserts them in one batch.
- Note that you can decide to either collect the results with **toList()** as shown in the previous slide or just use **last()** as shown here to wait until the last document is properly inserted.

```
// Generate a number of dummy JSON documents
int docsToCreate = 100;
List<JsonDocument> documents = new ArrayList<JsonDocument>();
for (int i = 0; i < docsToCreate; i++) {
    JsonObject content = JsonObject.create()
        .put("counter", i)
        .put("name", "Foo Bar");
    documents.add(JsonDocument.create("doc-"+i, content));
}

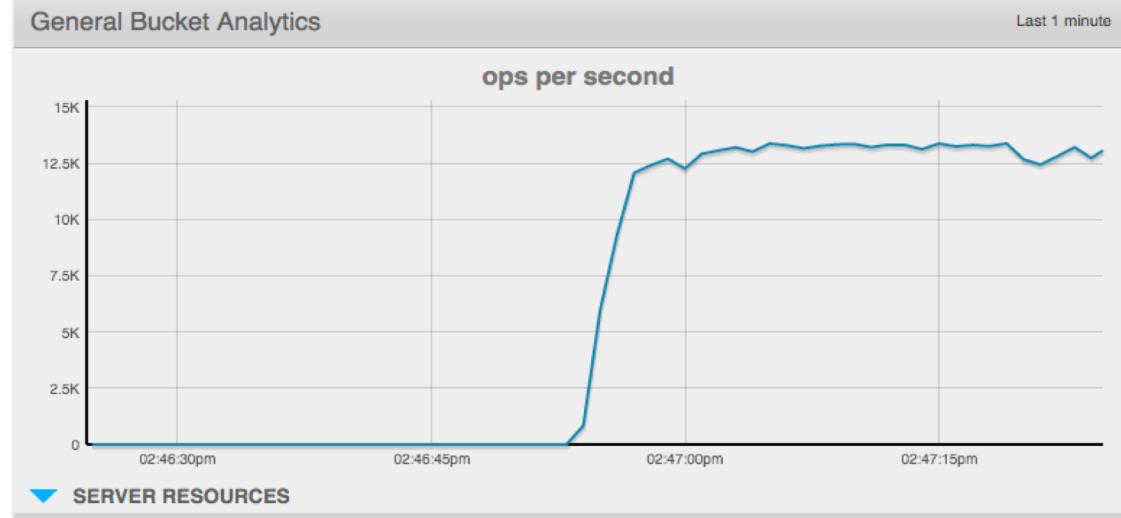
// Insert them in one batch, waiting until the last one is done.
Observable
    .from(documents)
    .flatMap(new Func1<JsonDocument, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(final JsonDocument docToInsert) {
            return bucket.async().insert(docToInsert);
        }
    })
    .last()
    .toBlocking()
    .single();
```



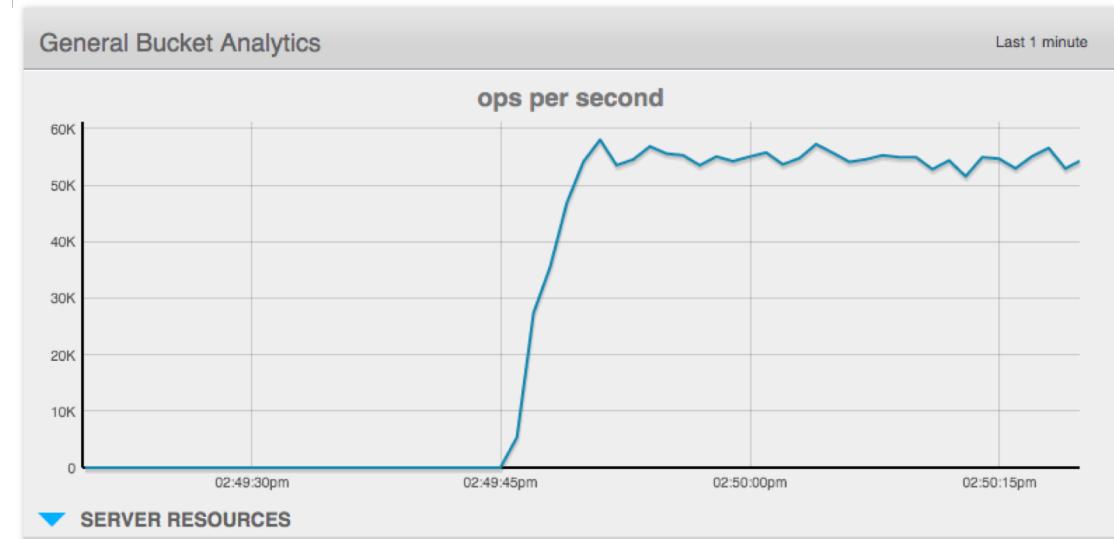
Sync Vs. ASync

- ✓ Here are two code samples that showcase serialized and batched loading of documents.

```
// Serialized workload of loading documents
while(true) {
    List<JsonDocument> loaded = new ArrayList<JsonDocument>();
    int docsToLoad = 10;
    for (int i = 0; i < docsToLoad; i++) {
        JsonDocument doc = bucket.get("doc-" + i);
        if (doc != null) {
            loaded.add(doc);
        }
    }
}
```



```
// Same workload, utilizing batching effects
while(true) {
    int docsToLoad = 10;
    Observable
        .range(0, docsToLoad)
        .flatMap(new Func1<Integer, Observable<JsonDocument>>() {
            @Override
            public Observable<JsonDocument> call(Integer i) {
                return bucket.async().get("doc-"+i);
            }
        })
        .toList()
        .toBlocking()
        .single();
}
```





Bulk Read Async

```
List<String> priceKeys;  
List<JsonObject> res = Observable  
    .from(priceKeys)  
    .flatMap(k -> bucket.async().get(k))  
    .map(doc -> doc.content())  
    .toList()  
    .toBlocking()  
    .single();
```



Bulk Write

Async version:

```
List<JsonDocument> docs = ...;  
Observable  
    .from(docs)  
    .flatMap(doc -> bucket.async().insert(doc))  
    .last()  
    .toBlocking()  
    .single();
```

Sync version:

```
for(JsonDocument doc : docs) {  
    bucket.insert(doc);  
}
```



Lab 15: Bulk Write Performance

- Implement 2 methods:
 - 15A: bulkWrite (String[] words) : Async version
 - 15B: bulkWriteSync (String[] words) : Sync version
- Parameter for the number of messages to insert is read from command line
- Delete all messages in the bucket:

```
DELETE FROM `travel-sample` WHERE type='msg'
```

- An array of JsonDocuments of messages is created (count is number parameter)
- Insert the messages into the bucket...

15A: asynchronously	15B: synchronously
<pre>Observable .from(docs) .flatMap(doc -> bucket.async().insert(doc)) .last() .toBlocking() .single();</pre>	<pre>for(JsonDocument doc : docs) { bucket.insert(doc); }</pre>

Lab

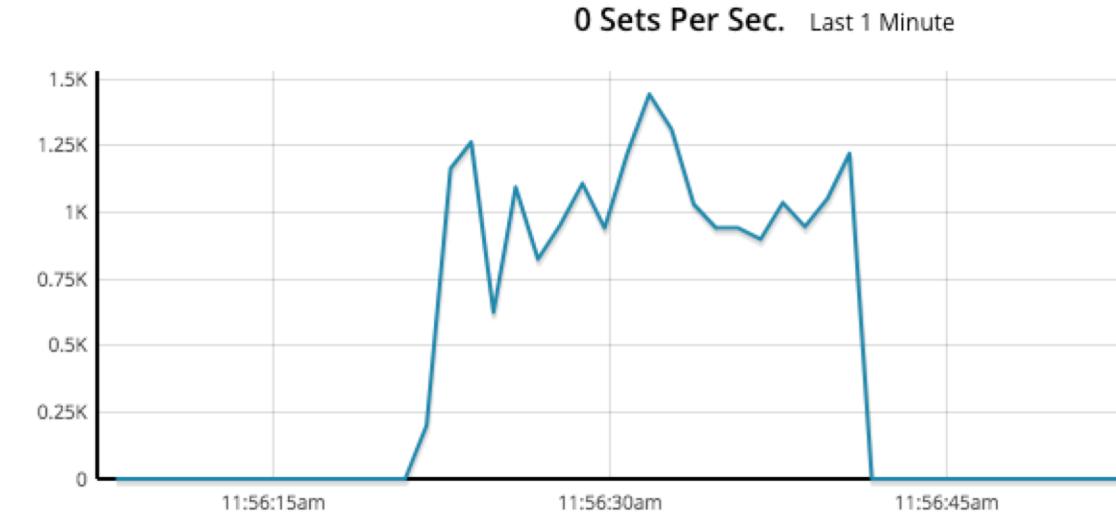


Lab 15: Bulk Write Performance

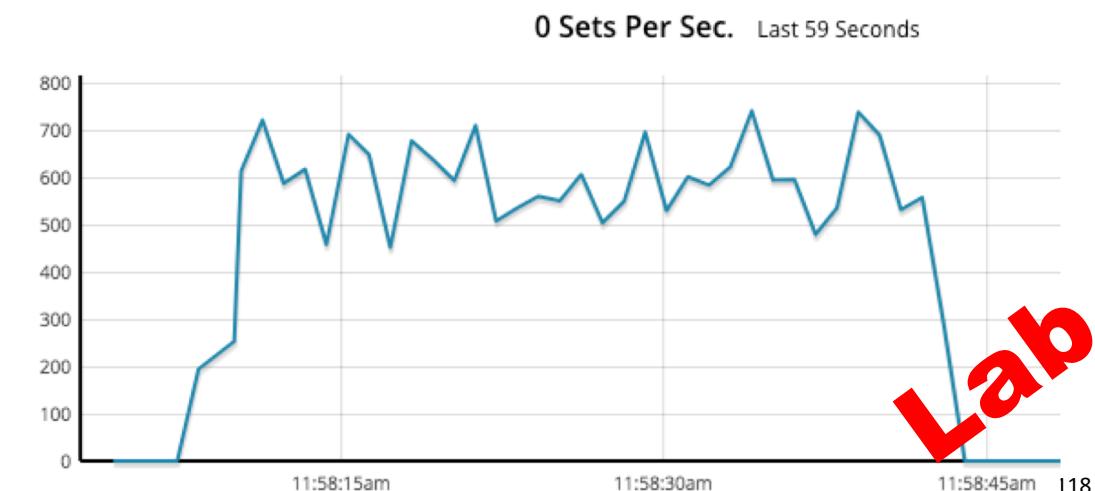
- Print the time elapsed to STDOUT
- Note: in initConnection, set env.keepAliveInterval(60000) to minimize keepAlive warnings
- Rebuild the JAR and run the application
- Test both methods creating 20000 messages
- Compare results sync vs. async: check both time and operations per second in the console

```
# bulkwrite 20000
Deleting messages ...
Writing 20000 messages
Time elapsed 17508 ms
# bulkwritesync 20000
Deleting messages ...
Writing 20000 messages
Time elapsed 64430 ms
#
```

Async



Sync





Query Async mode

```
bucket.async()
    .query(select("*").from(`travel-sample`).limit(5))
    .flatMap(result -> result.errors()
        .flatMap(e -> Observable
            .<AsyncN1qlQueryRow>error(new CouchbaseException("N1QL Error/Warning: " + e)))
        .switchIfEmpty(result.rows()))
    )
    .map(AsyncN1qlQueryRow::value)
    .subscribe(
        rowContent -> System.out.println(rowContent),
        runtimeError -> runtimeError.printStackTrace()
    );
}
```



Lab 16: Simple Query – Async version

- Implement method: `queryAsync (String[] words)`
- Execute the query: `"SELECT * FROM `travel-sample` LIMIT 5"` and print the results to STDOUT using asynchronous implementation

```
bucket.async()
    .query(N1qlQuery.simple(select("*").from("`travel-sample`").limit(5)))
    .subscribe(result -> {
        result.errors()
            .subscribe(
                e -> System.err.println("N1QL Error/Warning: " + e),
                runtimeError -> runtimeError.printStackTrace()
            );
        result.rows()
            .map(row -> row.value())
            .subscribe(
                rowContent -> System.out.println(rowContent),
                runtimeError -> runtimeError.printStackTrace()
            );
    });
});
```

Lab



Lab 16: Simple Query – Async version

- Rebuild the JAR and run the application
- Test using the queryasync command

```
# queryasync
# {"travel-sample":{"country":"United States","iata":"Q5","callsign":"MILE-AIR","name":"40-Mile
Air","icao":"MLA","id":10,"type":"airline"}}
{"travel-sample":{"country":"United States","iata":"TQ","callsign":"TXW","name":"Texas
Wings","icao":"TXW","id":10123,"type":"airline"}}
 {"travel-sample":{"country":"United
States","iata":"A1","callsign":"atifly","name":"Atifly","icao":"A1F","id":10226,"type":"airline"} }
 {"travel-sample":{"country":"United Kingdom","iata":null,"callsign":null,"name":"JC
ROYAL.BRITANNICA","icao":"JRB","id":10642,"type":"airline"} }
 {"travel-sample":{"country":"United
States","iata":"ZQ","callsign":"LOCAIR","name":"Locair","icao":"LOC","id":10748,"type":"airline"} }
```

Lab



SPRING & COUCHBASE



THANK YOU

