

# 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

Batching with RXJava

Full Text Search

Spring Boot



# LAB ENVIRONMENT





# Lab 0: Installing Couchbase

## DOWNLOAD

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

## DEPLOYMENT OPTIONS

<https://docs.couchbase.com/server/5.5/install/get-started.html>

 Non-cloud On-Premises	 Virtualized/Containerized Environment	 Public Cloud
Run Couchbase within your bare metal data center.	Run Couchbase within a virtualized or containerized environment such as VMware or Docker.	Deploy and manage Couchbase in a public cloud such as Amazon Web Services(AWS), Microsoft Azure, or Google Cloud Platform.

Lab



# Lab 0: Installing Couchbase

## DOCKER

- Documentation: <https://docs.couchbase.com/server/5.5/install/getting-started-docker.html>
- Single server deployment for this lab
- TIP: create local directory (`/[FULL_PATH]/cb1`) and map with `-v` to enable persistence upon recreating containers

```
docker run -d -p 8091-8094:8091-8094 -p 11210:11210 --name cb1 -v  
/[FULL_PATH]/cb1:/opt/couchbase/var couchbase:enterprise-5.5.1
```

Lab



# Lab 1: Initial Setup

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

Click “Setup New Cluster”, Enter Cluster Name “CB5 Docker”

Administration User: Administrator:password

Accept Terms, Configure Disk, Memory, Services

Couchbase Server  
Enterprise Edition 5.5.1 build 3511  
[Setup New Cluster](#)  
[Join Existing Cluster](#)

### Couchbase > New Cluster

Cluster Name  
CB5 Docker

Create Admin Username  
Administrator

Create Password  
6 characters minimum

Confirm Password  
6 characters minimum

< 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
- Leave Analytics and Eventing unchecked
- 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: 321 GB

Indexes Disk Path  
Path cannot be changed after setup  
/opt/couchbase/var/lib/couchbase/data  
Free: 321 GB

Analytics Disk Paths  
Paths cannot be changed after setup  
/opt/couchbase/var/lib/couchbase/data  
Free: 321 GB

Service Memory Quotas  
Per service / per node

Service	Quota (MB)
Data	1024
Index	256
Search	256

Analytics i 1024 MB

Query -----

Eventing 256 MB

TOTAL QUOTA 1536MB

RAM Available 1998MB Max Allowed Quota 1598MB

Index Storage Setting  
 Standard Global Secondary  
 Memory-Optimized i

Enable software update notifications in the web console. i

< Back Save & Finish

Lab



# Lab 1: Initial Setup - Install sample data set

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

The screenshot shows the 'CB5 Docker > Settings' interface. The top navigation bar includes links for Cluster, Software Updates, Node Availability, Email Alerts, Auto-Compaction, and Sample Buckets. The Sample Buckets link is currently selected, indicated by a dropdown arrow. Below the navigation, a descriptive text block states: '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.' A note in red text below reads: 'Sample buckets (like all buckets in Couchbase Server 5.0+) can only be accessed by a user with privileges for that bucket.' On the left, under 'Available Samples', there is a list of three buckets: 'beer-sample', 'gamesim-sample', and 'travel-sample'. The 'travel-sample' checkbox is checked. On the right, under '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

**Installed Samples**

none

Load Sample Data

Lab



# Lab 1: Initial Setup - Create User

- Username: travel
- Password: password
- Roles:
  - Application Access[travel-sample]
  - 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]

CB5 Docker > Security

Users ▾ Root Certificate Client Certificate Audit Log Redaction Session

external authentication is not enabled

username ▾	full name	roles	auth domain
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], Application Access[travel-sample]	Couchbase

[Cancel](#) [Add User](#)

### Add New User

Authentication Domain  
 Couchbase  External

Username: travel

Full Name (optional): travel

Password: .....

Verify Password: .....

Roles

travel-sample

- Bucket Admin
- Application Access
- XDCR Inbound
- Data Service
- Views

Query and Index Services

- Query Select
- Query Update
- Query Insert
- Query Delete
- Query Manage Index

[Cancel](#) [Add User](#)

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/deterministic: e.g. an email address
  - Computer generated/random: e.g. UUID
  - Compound: e.g. UUID with a deterministic 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](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: 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"  
    }  
  ],  
  "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"  
}
```

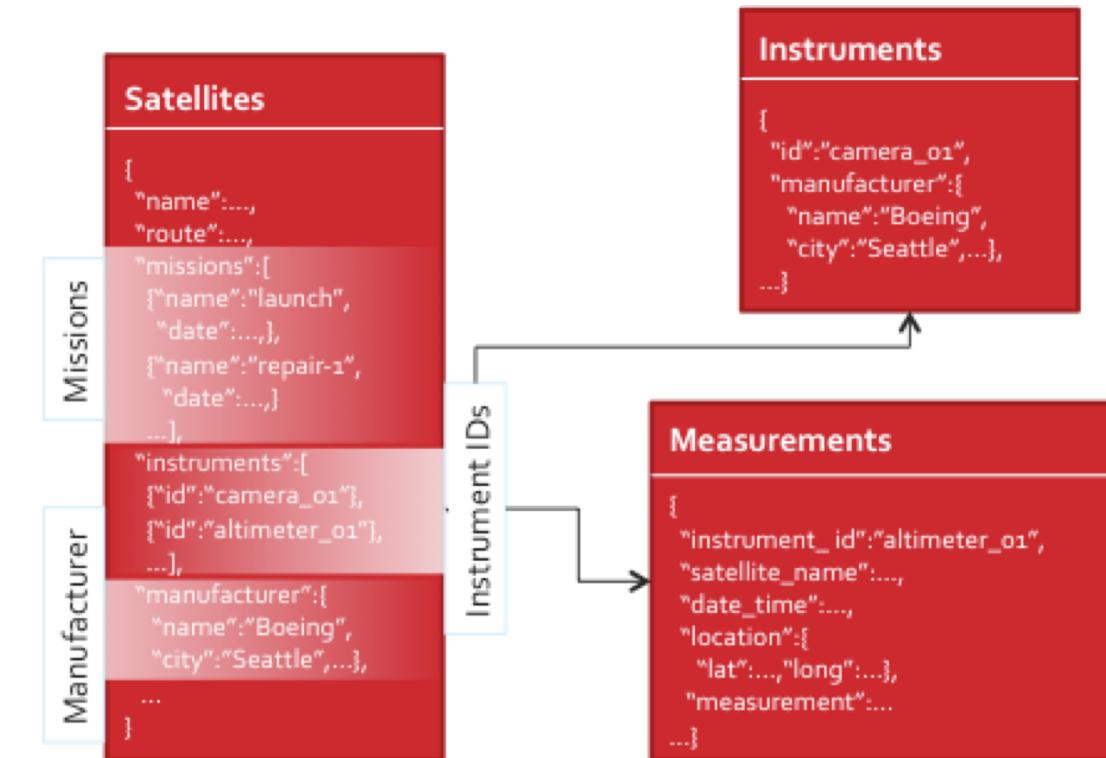
```
{  
  "customer": 2342  
  "total": "$299.99"  
}
```

```
{  
  "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 filter > start with > r
- Try Lookup ID: **route\_10003**
- Create document. Add Document >
  - Document ID: k\_1
  - Value {"name": "luis"}
- Delete document. ID: k\_1

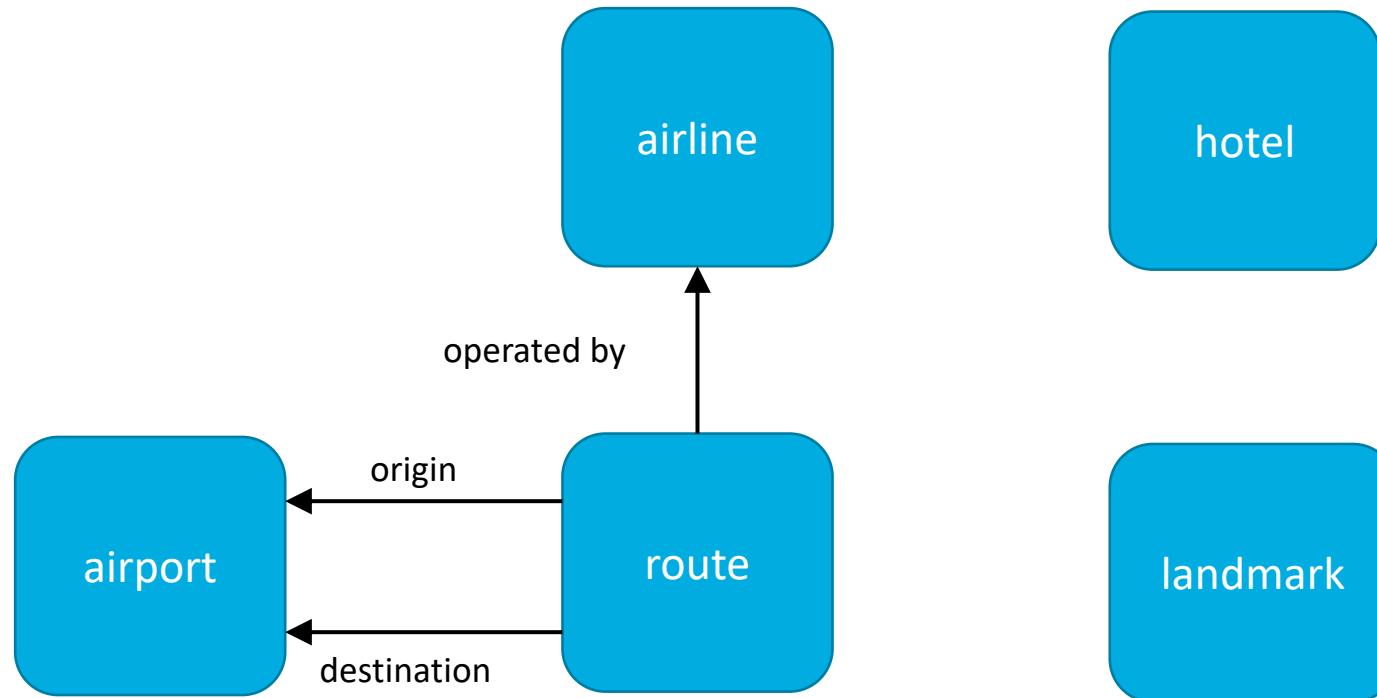
The screenshot shows the Couchbase UI interface. The top navigation bar includes links for HipChat, ToRead, Couchbase, VECI, Varios, and GitTalent. The main title is 'CB5 Docker > Buckets > Documents'. On the left, a sidebar menu lists: Dashboard, Servers, Buckets (selected), Indexes, Search, Query, XDCR, Security, Settings, and Logs. The main content area displays a table of documents in the 'travel-sample' bucket. The table columns are 'ID' and 'content sample'. Each row shows an ID (e.g., airline\_10, airline\_10123, airline\_10226, airline\_10642, airline\_10748, airline\_10765) and its corresponding JSON document. To the right of each document are 'Delete' and 'Edit' buttons. Above the table, there is a dropdown for 'Bucket' set to 'travel-sample', a 'filter' input field containing '?skip=0&include\_docs=true&limit=11', and a 'Document ID' search bar with a 'Look Up ID' button.

ID	content sample
airline_10	{"callsign":"MILE-AIR","country":"United States","iata":"Q5","icao":"MLA","id":10,"name":"40-Mile Air","type":"airline"}
airline_10123	{"callsign":"TXW","country":"United States","iata":"TQ","icao":"TXW","id":10123,"name":"Texas Wings","type":"airline"}
airline_10226	{"callsign":"atifly","country":"United States","iata":"A1","icao":"A1F","id":10226,"name":"Atifly","type":"airline"}
airline_10642	{"callsign":null,"country":"United Kingdom","iata":null,"icao":"JRB","id":10642,"name":"Jc royal.britannica","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":"SeaPort Airlines","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

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

route

```
{  
  "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  
    },  
}
```

hotel

```
"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.htm",  
"vacancy": true  
}
```



# Travel-sample data model: landmark

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
      - 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
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** and the **bucket password** obtained through **RBAC**
  - The bucket will connect to one of the Couchbase servers in the seed list (1<sup>st</sup> one in the list)
3. The Couchbase server will return a **CLUSTER MAP**
  - Using the **CLUSTER MAP**, the bucket instance will create 1 to 3 connections per 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>2.7.0</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/2.7.0 (git: 2.7.0, core: 1.7.0), 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();
```



## 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/5.5/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. This example JOINS the document of type "route" with documents of type "airline" using the KEY "airlineid". Documents of type "route" are on the left side of JOIN, and documents of type "airline" are on the right side of JOIN. The documents of type "route" (on the left) contain the foreign key "airlineid" of documents of type "airline" (on the right).

```
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"  
LIMIT 2;
```



## N1QL – Example 2

Which airlines fly from each airport?

```
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;
```



# 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 `default`  
(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 `default`  
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

Activity Documentation Support Administrator

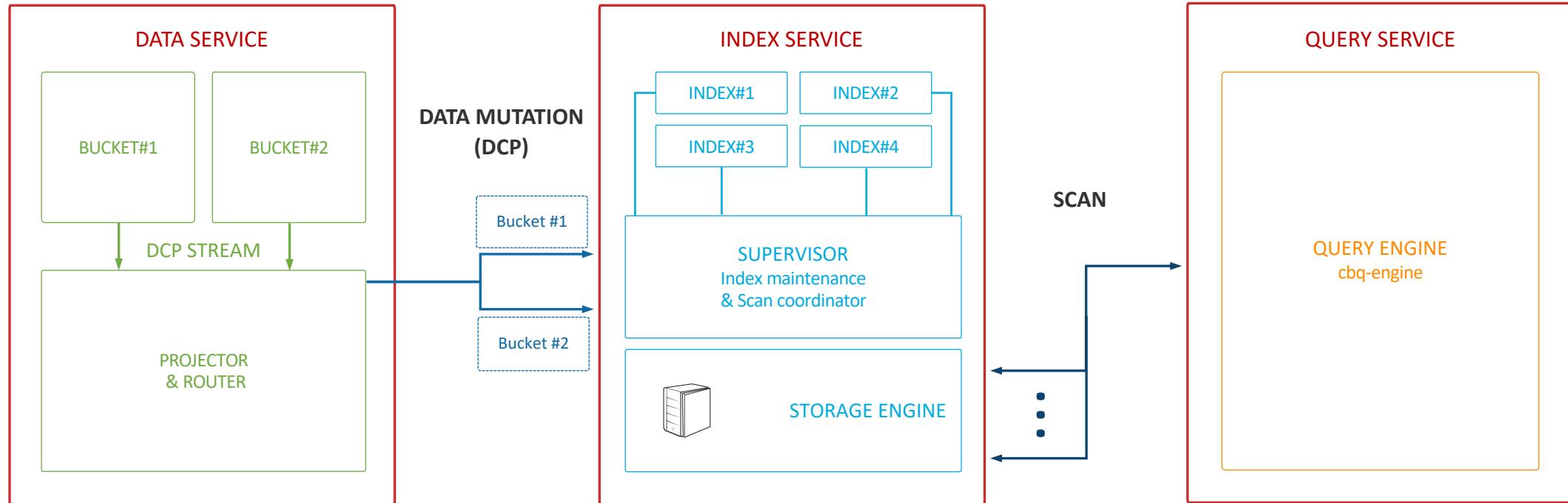
## CB5 Docker > Indexes

Global Indexes ▾ Views

bucket ▾	node	index name	storage type	status	build progress
sample	127.0.0.1:8091	#primary	Memory Optimized GSI	ready	100%
travel-sample	127.0.0.1:8091	def_airportname	Memory Optimized GSI	ready	100%
<b>Definition</b> CREATE INDEX `def_airportname` ON `travel-sample`(`airportname`) WITH { "defer_build":true }					
travel-sample	127.0.0.1:8091	def_city	Memory Optimized GSI	ready	100%
travel-sample	127.0.0.1:8091	def_faa	Memory Optimized GSI	ready	100%
travel-sample	127.0.0.1:8091	def_icao	Memory Optimized GSI	ready	100%
travel-sample	127.0.0.1:8091	def_name_type	Memory Optimized GSI	ready	100%
travel-sample	127.0.0.1:8091	def_primary	Memory Optimized GSI	ready	100%



# 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 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));
}

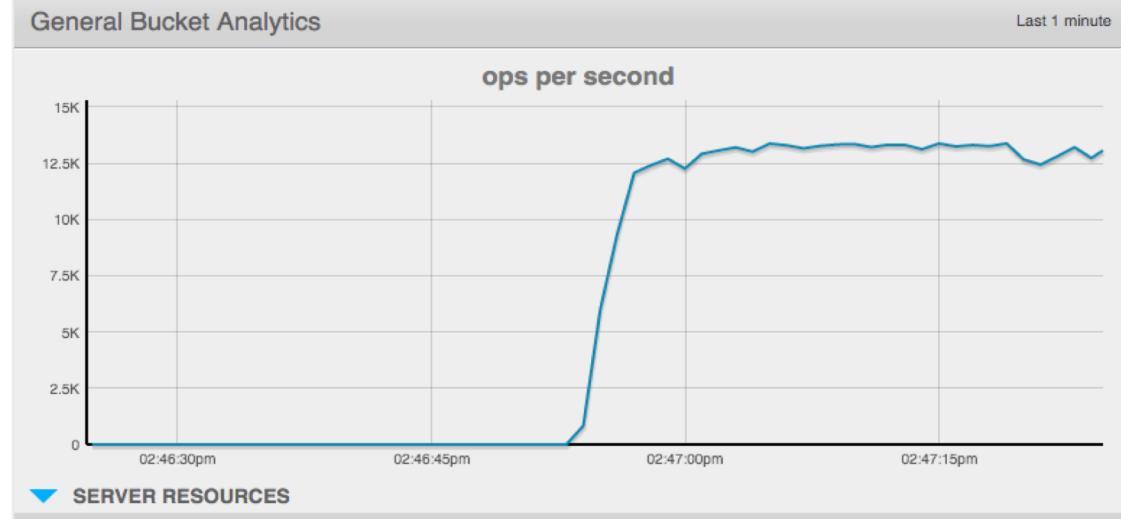
// Insert them in one batch, waiting until the last one is done.
Observable
    .from(documents)
    .flatMap(new Func1<JsonDocument, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(final JsonDocument docToInsert) {
            return bucket.async().insert(docToInsert);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



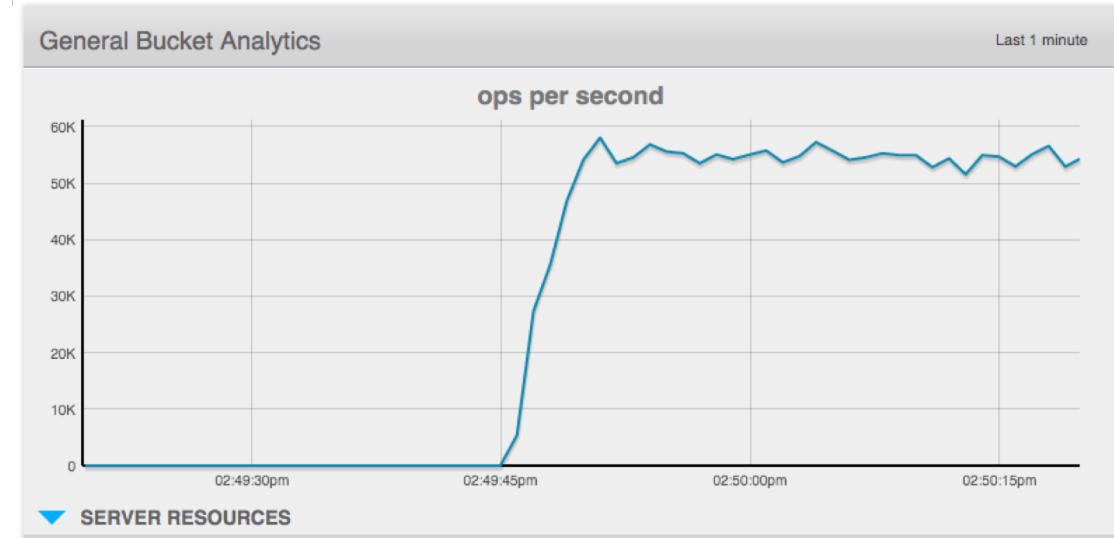
# Sync Vs. ASync

- ✓ 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 11: Bulk Write Performance

- Implement 2 methods:
  - 11A: bulkWrite (String[] words) : Async version
  - 11B: 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...

11A: asynchronously	11B: synchronously
<pre>Observable     .from(docs)     .flatMap(doc -&gt; bucket.async().insert(doc))     .last()     .toBlocking()     .single();</pre>	<pre>for(JsonDocument doc : docs) {     bucket.insert(doc); }</pre>

Lab

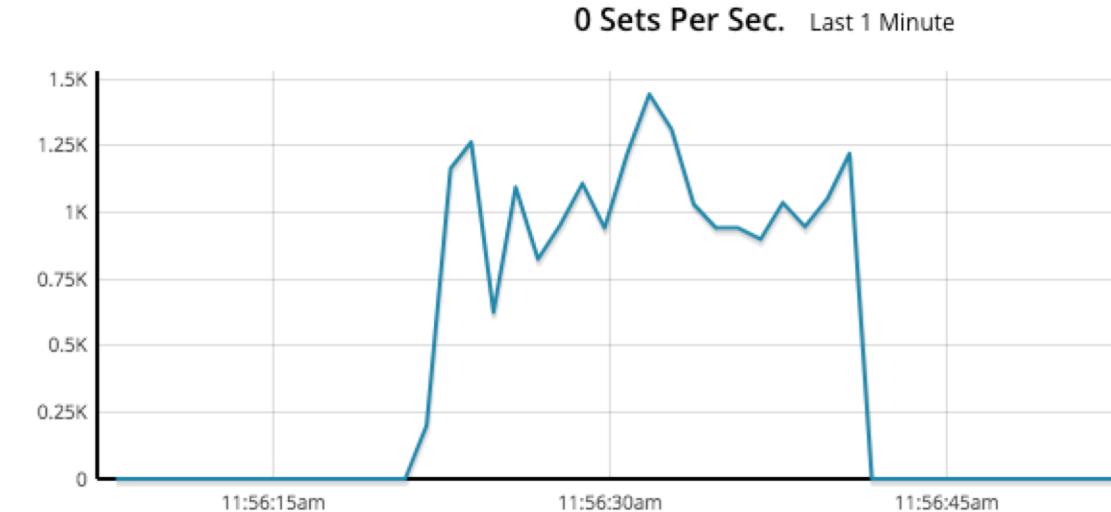


# Lab 11: Bulk Write Performance

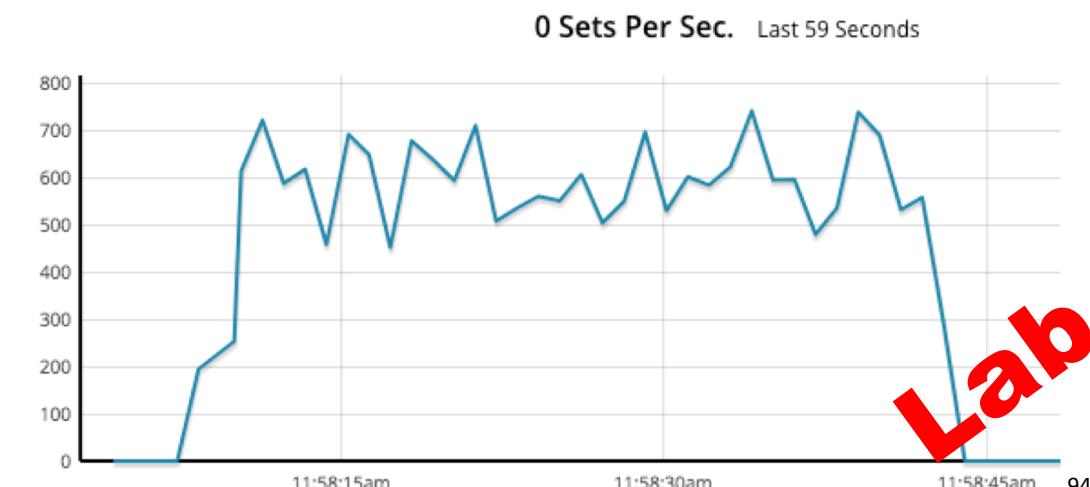
- Print the time elapsed to STDOUT
- Rebuild the JAR and run the application
- Test both methods creating 20000 messages
- Compare results sync vs. async: check both time and operations per second in the console

```
# 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 12: 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 12: 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



# 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/5.5/fts/fts-query-types.html>



# Lab 13: 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"
- Set Index Type to Scorch
- Click "Create Index"

CB5 Docker > 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**:

JSON type field: type

Doc ID up to separator: delimiter

Doc ID with regex: regular expression

**+ Add Type Mapping**

**Type Mappings**

# hotel | only index specified fields

field	description	ok
type	text	cancel
searchable as	description	delete
analyzer	inherit	

index    store    include in \_all field    include term vectors

# default | disabled | dynamic

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

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

**Create Index**      **Cancel**

**Lab**



# Lab 13: FTS – Create index

- Search for “farm”

Full Text Search > Search Results

farm   show advanced query settings

full text query syntax help

Results for sidx\_hotel\_desc

Show Scoring 7 results (<1ms 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

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

**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 14: 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 14: FTS – Java example

- Rebuild the JAR and run the application
- Test using the search command 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 SPRING BOOT





# Travel-sample demo on couchbaselabs

- This application uses Couchbase Server 5.0 together with Spring Boot, Angular2 and Bootstrap.
- Documentation: <https://github.com/couchbaselabs/try-cb-java/tree/5.0-updates>
- From command line execute:

```
git clone -b 5.0-updates https://github.com/couchbaselabs/try-cb-java.git
```

- To run the app execute:

```
cd try-cb-java/
```

```
mvn spring-boot:run -Dstorage.host=127.0.0.1 -Dstorage.bucket=travel-sample -Dstorage.password=password -Dstorage.username=travel
```

- Access application: <http://localhost:8080>
- Details: <https://blog.couchbase.com/traveling-with-couchbase-using-the-java-sdk/>



# Travel-sample demo on couchbaselabs

The screenshot shows a web browser window titled "TryCb" displaying the CBTravel application. The URL is "localhost:8080/home". The top navigation bar includes links for HipChat, ToRead, Couchbase, VECI, Varios, and GitTalent. A user "Manuel" is logged in, with the email "manuel.hurtado@couchbase.com" and a "LOGOUT" link. The shopping cart is shown as "0 items (\$ 0)".

The main content area has a yellow header bar with "(Show Narration)" and the CBTravel logo. Below this is a "Find a Flight" form with fields for "AIRPORT OR CITY" (From: John F Kennedy Intl, To: Los Angeles Intl) and "TRAVEL DATES" (Leave: 01/12/2018, Return: 03/12/2018). A "Find Flights" button is present. The next section, "Available Outbound Leg Flights", lists flights from "John F Kennedy Intl -> Los Angeles Intl".

Airline	Flight	Departure	From	To	Aircraft	Price	Reservation
American Airlines	AA882	19:34:00	JFK	LAX	32B 762	561\$	<button>Choose</button>
American Airlines	AA380	11:49:00	JFK	LAX	32B 762	522\$	<button>Choose</button>
American Airlines	AA657	05:47:00	JFK	LAX	32B 762	1242\$	<button>Choose</button>
Delta Air Lines	DL320	22:16:00	JFK	LAX	76W 752	1803\$	<button>Choose</button>
Delta Air Lines	DL415	01:34:00	JFK	LAX	76W 752	934\$	<button>Choose</button>
Delta Air Lines	DL904	19:11:00	JFK	LAX	76W 752	966\$	<button>Choose</button>
Delta Air Lines	DL310	02:30:00	JFK	LAX	76W 752	66\$	<button>Choose</button>

# THANK YOU

