Architecture and Administration Basics

Java Labs

1 Installation & Configuration SDK

Documentation & Examples

Open the documentation for libcouchbase!

- https://developer.couchbase.com/documentation/server/current/sdk/java/sample-app-backend.html
- https://developer.couchbase.com/documentation/server/current/sdk/java/start-using-sdk.html
- http://docs.couchbase.com/sdk-api/couchbase-java-client-2.7.2/

- Check the Sample App backend https://docs.couchbase.com/java-sdk/2.7/sample-application.html

Including the SDK

```
// Gradle
dependencies {
    ...
    compile group: 'com.couchbase.client', name: 'java-client', version:'2.5.4'
    ...
}
```

```
// Maven
<dependencies>
    <dependency>
        <groupId>com.couchbase.client</groupId>
        <artifactId>java-client</artifactId>
        <version>2.7.3</version>
    </dependency>
</dependencies>
```

TIP. Check last version available here:

https://developer.couchbase.com/documentation/server/current/sdk/java/start-using-sdk.html

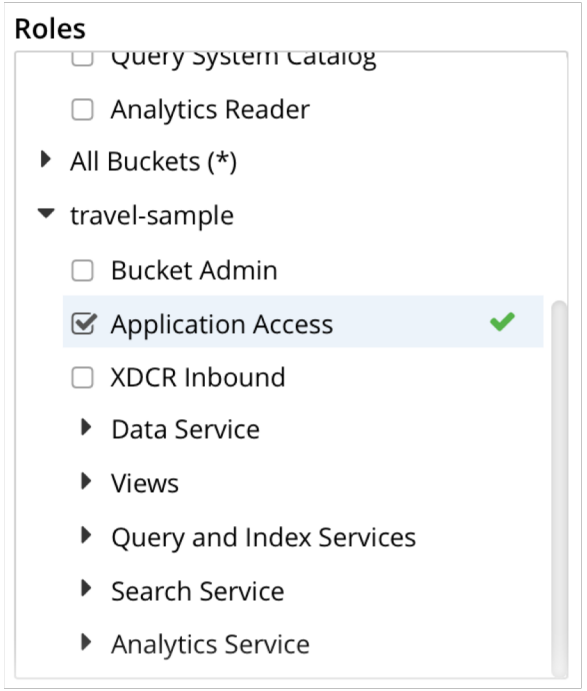
Lab 1: Preparing for the Lab

- Clone the source repository in your home folder.

```
git clone https://github.com/vgasiunas/cb-workshop-2d.git
```

- Make sure Couchbase Server is running on your Machine and you have travel-sample bucket

- Create a user for the application access
  - Go the Security Tab in Couchbase
  - Create a new User.
  - Username = "travel"
  - Password = "couchbase"
  - Role: Application Access on 'travel-sample'



Lab

Lab 1: Load the Project into NetBeans (optional)

Perform the following steps:

- Start a Server X on your Oracle Virtual Box from a Terminal.

```
startxfce4
```

- Start NetBeans IDE from the Desktop.
- Open the Existing projects “Lab”, “LabSolutions” (solutions) in NetBeans IDE
- Open the “LabSolution” project => This is the solution.
- Open the “Lab” project => This is where you start.

Lab

Lab 1: Build & Run

Update dependency on the libcouchbase:

- Open **`java/Lab/pom.xml`**
- Change Couchbase Java client dependency to the latest

Build a simple executable jar:

```
cd cb-workshop-2d/java/Lab
mvn clean compile assembly:single
```

Run main class:

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

Lab

2

Java SDK
Connecting to Couchbase

Connection Basics

```
// Java
import com.couchbase.client.java.CouchbaseCluster;
import com.couchbase.client.java.CouchbaseBucket;

CouchbaseEnvironment env = DefaultCouchbaseEnvironment.builder()
    .connectTimeout(15000)
    // more custom configuration
    .build();

CouchbaseCluster cluster = CouchbaseCluster.create(env,
                                "<host1>", ..., "<hostN>");
cluster.authenticate("<username>", "<password>");
CouchbaseBucket bucket = cluster.openBucket("<bucket-name>");
```

- Reuse cluster and bucket objects, e.g. as singletons
- Make sure to provide more than one hostIP for high availability

Lab 2: Couchbase Connection

- Use source file: MainLab.java
- Implement method initConnection()
- Read config values from System properties:
  - cbworkshop.clusteraddress
  - cbworkshop.user
  - cbworkshop.password
  - cbworkshop.bucket
- Connect to the bucket with the given credentials
- Run application
- Check output:

```
Nov 20, 2017 1:44:25 PM com.couchbase.client.core.CouchbaseCore <init>
INFO: CouchbaseEnvironment: {sslEnabled=false, sslKeystoreFile='null',
sslTruststoreFile='null', sslKeystorePassword=false, sslTruststorePassword=false,
sslKeystore=null, sslTruststore=null, bootstrapHttpEnabled=true,
bootstrapCarrierEnabled=true, bootstrapHttpDirectPort=8091,
bootstrapHttpSslPort=18091, . . .}

Nov 20, 2017 1:44:33 PM
com.couchbase.client.core.config.DefaultConfigurationProvider$8 call
INFO: Opened bucket travel-sample

Nov 20, 2017 1:44:35 PM com.couchbase.client.core.node.CouchbaseNode signalConnected
INFO: Connected to Node 127.0.0.1/localhost
```

3

Java SDK
Key-Value Operations

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

Lab 3: Create Object

- Implement method: `create(String[] words)`
- Compose a JSON document like this:
  - Use the command line parameters from `words`:
    - document key
    - from
    - to
  - Compose key with prefix "`msg::`" + provided key
  - Set timestamp to `System.currentTimeMillis()`
  - Set type to "`msg`"
- Use insert
  - Try several times. See results in console
  - Try same key (Error should appear!)
- Try upsert instead of insert

Key:
msg::some_text

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

Lab

Lab 4: Read Object

- Implement method: `read(String[] words)`
- Use the command line parameter:
  - Document key
- Read the document
- Write the json string to `System.out`
- 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"}
```

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

Lab

Lab 5: Update Object

- Implement method: `update(String[] words)`
- Use the command line parameters:
  - Document key (prefix with "airline_" in code)
- Read the document
- Modify attribute "name": set the same value converted `toUpperCase`
- Use `replace` to modify

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

Lab

Lab 6: Delete Object

- Implement method: delete(String[] words)
- Use the command line parameter:
  - Document key (prefix with "msg::" in code)
- Delete document
- Tip: use create, then delete same key
- Try to read it to test if it is actually deleted

```
# create 1001 luis ana
# read msg::1001
{"from":"luis","to":"ana","type":"msg","timestamp":1511192232720}
# delete 1001
# read msg::1001
java.lang.NullPointerException
#       at com.cbworkshop.MainLab.read(MainLab.java:95)
        at com.cbworkshop.MainLab.process(MainLab.java:60)
        at com.cbworkshop.MainLab.main(MainLab.java:30)
```

Lab

4

Java SDK
Subdocument API

The Goal: Working with Parts of a Document

- Get parts of a JSON Document
- Update individual JSON attributes 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
SubdocOptionsBuilder builder = new SubdocOptionsBuilder();
builder.createParents(true);
bucket().mutateIn(”nraboy")
        .upsert(”profile.firstname", "Nicolas", builder)
        .execute();
```

Chain Subdocument Operations

```
DocumentFragment<Mutation> result = bucket
    .mutateIn(”nraboy")
    .replace(”profile.firstname", ”Nic")
    .insert(”profile.gender", ”Male", builder)
    .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 7: SubDocument API example

- Implement method: `subdoc(String[] words)`
- Use the command line parameter:
  - Document key (prefix with "msg::" from code)
- Using SubDocument API:
  - Change the actual value of the "`from`" attribute to "`administrator`"
  - Add a new attribute: "`reviewed`", with value `System.currentTimeMillis()`

```
# create 1005 juan santiago
# read msg::1005
{"from":"juan","to":"santiago","type":"msg","timestamp":1511196994278}
# subdoc 1005
# read msg::1005
{"reviewed":1511197006619,"from":"Administrator","to":"santiago","type":"msg","timestamp":1511196994278}
```

Lab

5

Java SDK
Executing N1QL

Query Raw String vs. DSL

Raw string query

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

Using the query builder

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

Iterate over the query result

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

Query with Parameters

Sample code

```
String sourceairport = ...;
String destinationairport = ...;

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";

JsonObject params = JsonObject.create()
    .put("src", sourceairport)
    .put("dst", destinationairport);

N1qlQuery query = N1qlQuery.parameterized(queryStr, params,
                                          N1qlParams.build().adhoc(false));
```

Query Consistency

- **not_bounded** (fastest)
  - Returns data that is currently indexed and accessible by the index or the view.
- **at_plus**
  - 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**
  - Requires all mutations, up to the moment of the query request, to be processed before the query execution can start.

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

Lab 8: Simple Query

- Implement method: query (String[] words)
- Execute the query: "`SELECT * FROM `travel-sample` LIMIT 10`"
- Print the results to STDOUT
- Use both implementations, raw and query builder

```
# 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 9: Query with parameters

- Implement method: queryAirports(String[] words)
- Use the command line parameters:
  - sourceairport
  - destinationairport
- Write a query to find airlines (airline names) flying from sourceairport to destinationairport. Use JOIN
- Use a parametrized query
- **TIP**: Highest traffic airport codes: 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

6

Java SDK
Asynchronous programming

Why asynchronous programming?

- Synchronous programming is straightforward, e.g. simple loop to create multiple documents

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

- But difficult to achieve high throughput
  - E.g. if insert takes 1ms, maximum throughput is 1000 op/s
- Multithreading can increase throughput, but creates a lot of overhead
- Asynchronous programming provides an efficient way to achieve high throughput

Observable Pattern

- Observable = a stream of data
- RxJava operators to manipulate the stream

	Single	Multiple
Sync ("pull")	T	Iterable<T>
Async ("push)	Future<T>	Observable<T>

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 process the stream events with the Couchbase Java SDK and merge the results asynchronously.

- **last()** to wait until the last event of the stream is received

- **toList()** to transform the events into a single list of results. Useful for reading data

- **toBlocking().single()** transforms the stream into a synchronous call returning the result

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

- If you wrap the code in a helper method, you can provide very nice encapsulated batching semantics

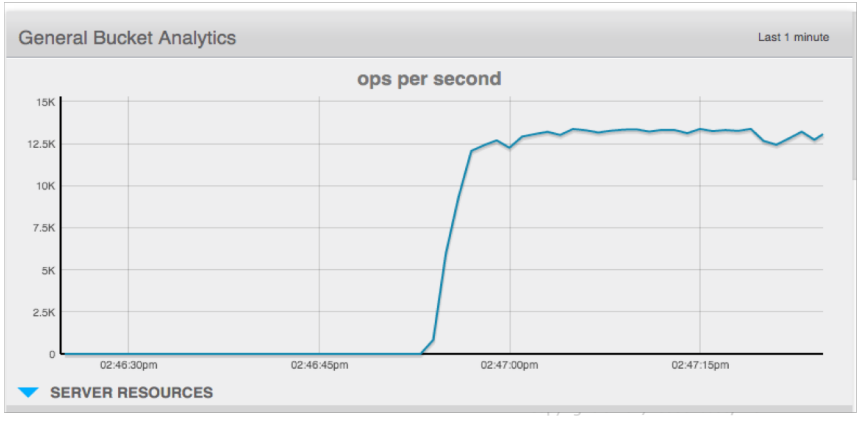
```
public List<JsonDocument> bulkGet(final Collection<String> ids) {
    return Observable
        .from(ids)
        .flatMap(new Func1<String, Observable<JsonDocument>>() {
            @Override
            public Observable<JsonDocument> call(String id) {
                return bucket.async().get(id);
            }
        })
        .toList()
        .toBlocking()
        .single();
}
```

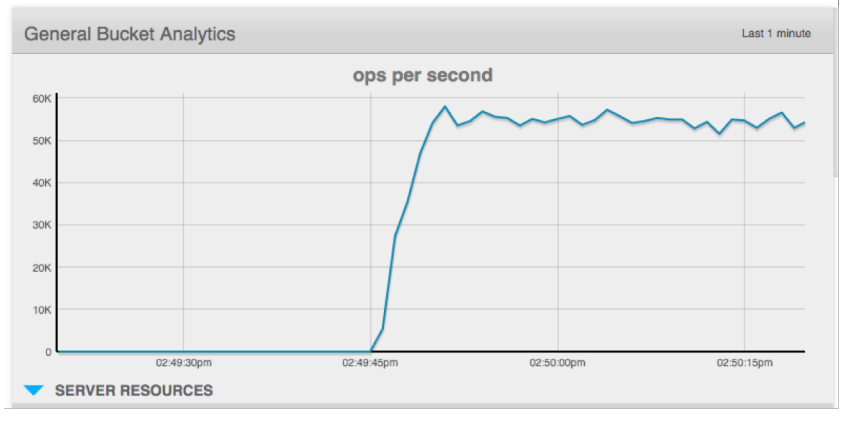
Batching with RxJava

Here are two code samples, both synchronous, 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();
}
```





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 before 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();
```

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

Sync version:

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

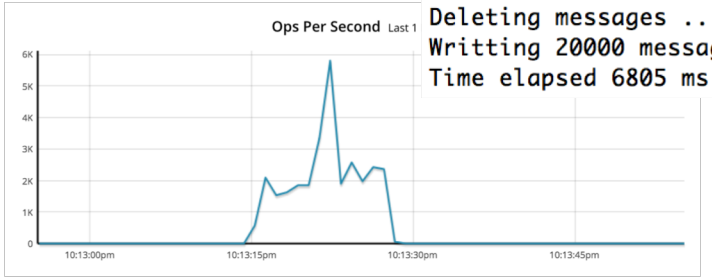
Async version:

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

Lab 10: Bulk Write Performance

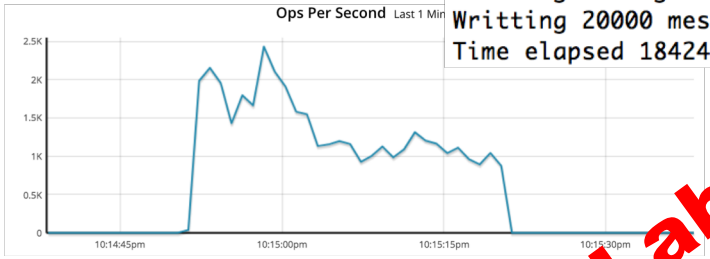
- Implement method: bulkWrite (String[] words) : Async version
- Implement method: bulkWriteSync (String[] words) : Sync version
- Read parameters from command line:
  - size: Number of messages to insert. Keys will be from msg::1 to msg::[size]
- Delete all messages in the bucket: DELETE FROM `travel-sample` WHERE type="msg"
- Create an array of JsonDocuments of messages
- Insert the messages into the bucket (in async / sync way)
- Print the time elapsed to STDOUT
- Compare results sync vs. async. Check both time and operations per second in the console

Async




```
# bulkwrite 20000
Deleting messages ...
Writting 20000 messages
Time elapsed 6805 ms
```

Sync




```
# bulkwritesync 20000
Deleting messages ...
Writting 20000 messages
Time elapsed 18424 ms
```

Lab

Query Async mode

```
bucket.async()
    .query(select("*").from(i("`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()
     );
```

- Unlike the synchronous method, does not block the calling thread
- The query results are processed asynchronously as they arrive by the `subscribe` handlers

Lab 11: Simple Query – Async version

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

```
# 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

Thank you

Couchbase