Java Labs

1 Installation & Configuration SDK

Documentation & Examples

Open the documentation for Java SDK!

- https://docs.couchbase.com/java-sdk/current/hello-world/start-using-sdk.html
- https://docs.couchbase.com/sdk-api/couchbase-java-client/

- Check the Sample Application
  https://docs.couchbase.com/java-sdk/current/hello-world/sample-application.html

3

Including the SDK

```
// Gradle
dependencies {
    ...
    implementation 'com.couchbase.client:java-client:3.2.5'
    ...
}
```

```
// Maven
<dependencies>
    <dependency>
        <groupId>com.couchbase.client</groupId>
        <artifactId>java-client</artifactId>
        <version>3.2.5</version>
    </dependency>
</dependencies>
```

TIP. Check last version available here:

https://docs.couchbase.com/java-sdk/current/hello-world/start-using-sdk.html

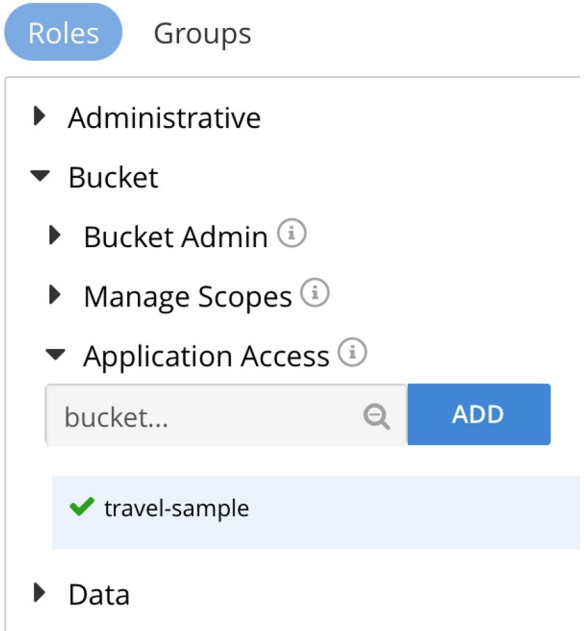
Lab 1: Preparing for the Lab

- Clone the source repository in your home folder.

  git clone https://github.com/couchbase-ps/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

5

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
```

Or run using **docker**

```
docker-compose up -d && docker attach java-lab
```

Lab

2

Java SDK
Connecting to Couchbase

Connection Basics

```
// Java
import com.couchbase.client.java.Cluster;
import com.couchbase.client.java.Collection;

ClusterEnvironment env = ClusterEnvironment.builder()
                .timeoutConfig(TimeoutConfig
                        .connectTimeout(Duration.ofSeconds(15)))
                   // more custom configuration
                .build();

Cluster cluster = Cluster.connect("<host1>, ..., <hostN>",
                    ClusterOptions.clusterOptions("<username>", "<password>")
                                  .environment(env));

Collection collection = cluster.bucket("<bucket-name>").defaultCollection();
```

- 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:

```
mars 04, 2022 11:18:59 AM com.couchbase.client.core.cnc.LoggingEventConsumer$JdkLogger info
INFOS: [com.couchbase.core][CoreCreatedEvent]
{"clientVersion":null,"clientGitHash":null,"coreVersion":null,"coreGitHash":null, ...}
{"coreId":"0xa564412c00000001","seedNodes":[{"address":"localhost"}]}

mars 04, 2022 11:18:59 AM com.couchbase.client.core.cnc.LoggingEventConsumer$JdkLogger info
INFOS: [com.couchbase.node][NodeConnectedEvent] Node connected
{"coreId":"0xa564412c00000001","managerPort":"8091","remote":"localhost"}

mars 04, 2022 11:19:00 AM com.couchbase.client.core.cnc.LoggingEventConsumer$JdkLogger info
INFOS: [com.couchbase.core][BucketOpenedEvent][222ms] Opened bucket "travel-sample"
{"coreId":"0xa564412c00000001"}
```

Lab

3 Java SDK Key-Value Operation

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);
collection.insert(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
collection.get("person-1").contentAsObject();
```

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 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
LookupInResult result = collection.lookupIn("nraboy",
                  List.of(LookupInSpec.get("profile")));
```

Update Part of a Document

```
// Java
 collection.mutateIn("nraboy", List.of(
                    MutateInSpec.upsert("profile.firstname", "Nicolas")));
```

Chain Subdocument Operations

```
Collection
    .mutateIn("nraboy",
        List.of(
                MutateInSpec.replace("profile.firstname", "Nic"),
                MutateInSpec.insert("profile.gender", "Male"),
                MutateInSpec.remove("data")),
         mutateInOptions()
    .durability(PersistTo.ACTIVE, ReplicateTo.NONE));
```

```
LookupInResult result = collection.lookupIn("nraboy", List.of(
                                        LookupInSpec.get("sub.value"),
                                        LookupInSpec.exists("fruits")));

String subValue = result.contentAs(0, String.class);
boolean fruitsExist = result.contentAs(1, 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 String

Raw string query

```
QueryResult queryResult = cluster.query(
                "SELECT * FROM `travel-sample` LIMIT 10");
```

Iterate over the query result

```
for (JsonObject row : queryResult.rowsAsObject()) {
        System.out.println(row.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);

QueryResult queryResult = cluster.query(queryStr, queryOptions()
        .parameters(params));
```

Query Consistency

- **not_bounded** (fastest)
  - Returns data that is currently indexed and accessible by the index or the view.

- **request_plus**
  - Requires all mutations, up to the moment of the query request, to be processed before the query execution can start.

```
// Java
QueryResult result = cluster.query(queryStr, queryOptions()
        .scanConsistency(QueryScanConsistency.REQUEST_PLUS));
```

Lab 8: Simple Query

- Implement method: query (String[] words)
- Execute the query: "`SELECT * FROM` `travel-sample` `LIMIT 10`"
- Print the results to STDOUT

```
# 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 Reactive programming

Why reactive 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
- Reactive programming provides an efficient way to achieve high throughput

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2021. All rights reserved.

Flux Pattern

- Flux = a stream of data
- Reactor operators to manipulate the stream

	Single	Multiple
Sync (Pull)	T	Iterable<T>
Reactive (Push)	Mono<T>	Flux<T>

33

Batching with Reactor

- Implicit batching is performed by utilizing a few operators:
  - Flux.just() or Flux.from() to generate a Flux 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
- collectList() to transform the events into a single list of results. Useful for reading data
- block() transforms the stream into a synchronous call returning the result

Batching with Reactor

- The following example creates a Flux 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 = Cluster.connect(…);
Collection collection = cluster.bucket("bucketName").defaultCollection();

Flux.just("key1", "key2", "key3", "key4")
                  .flatMap(key -> collection.reactive().get(key))
                  .map(GetResult::contentAsObject)
                  .collectList()
                  .block();
```

Batching with Reactor

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

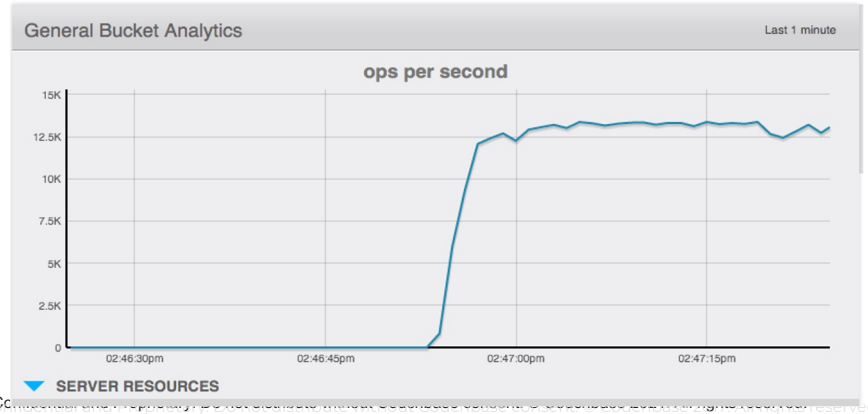
```
public List<JsonObject> bulkGet(final Collection<String> ids) {
    return Flux.just("key1", "key2", "key3", "key4")
                            .flatMap(key -> collection.reactive().get(key))
                            .map(GetResult::contentAsObject)
                            .collect(Collectors.toList())
                            .block();
}
```

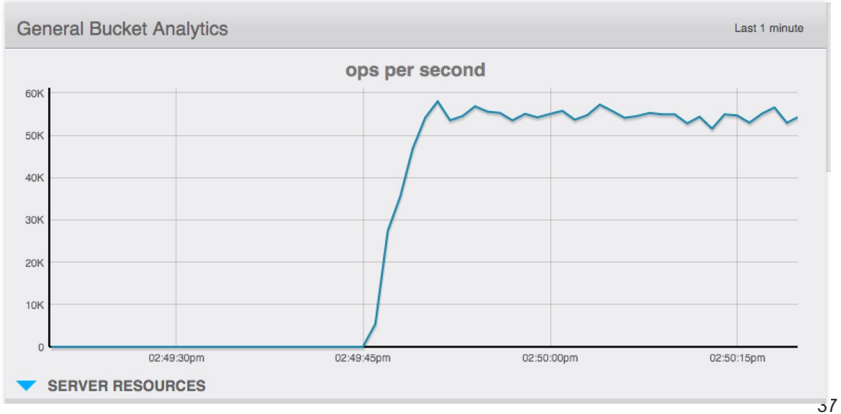
Batching with Reactor

Here are two code samples, both synchronous, that showcase serialized and batched

```
while (true) {
    List<JsonObject> loaded = new ArrayList<>();
    int docsToLoad = 10;
    IntStream.range(0, docsToLoad)
            .forEach(i -> {
                try {
                    loaded.add(
                        collection.get("doc-" + i)
                            .contentAsObject());
                } catch (DocumentNotFoundException e) {}
            });
}
```

```
while (true) {
    int docsToLoad = 10;
    Flux.range(0, docsToLoad)
                .flatMap(i -> collection.reactive()
                .get("doc-:" + i))
                .onErrorResume(
                          DocumentNotFoundException.class,
                           e -> Mono.empty())
                .toIterable();
}
```





Batching with Reactor - 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 **toIterable()** as shown before or just use **blockLast()** as shown here to wait until the last document is properly inserted.

```
// Generate a number of dummy JSON documents
int docsToCreate = 100;
Flux.range(0, docsToCreate)
        // Insert them in one batch, waiting until the last one is done.
        .flatMap(i -> collection.reactive()
                                .insert("msg::" + i, JsonObject.create()
                                .put("counter", i)
                                .put("name", "Foo Bar")))
        .blockLast();
```

Bulk Read Reactive

```
List<String> priceKeys = ...;
List<JsonObject> res = Flux.fromIterable(priceKeys)
        .flatMap(k -> collection.reactive().get(k))
        .map(GetResult::contentAsObject)
        .collectList()
        .block();
```

Bulk Write Reactive

Sync version:

```
Map<String, JsonObject> documents = ...;
documents.forEach((key, doc) ->
        collection.insert(key, doc));
```

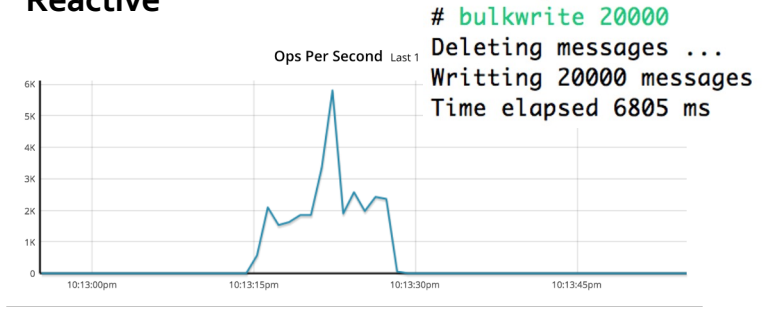
Reactive version:

```
Map<String, JsonObject> documents = ...;
Flux.fromStream(documents.entrySet().stream())
        .flatMap(e -> collection.reactive()
                .insert(e.getKey(), e.getValue()))
        .blockLast();
```

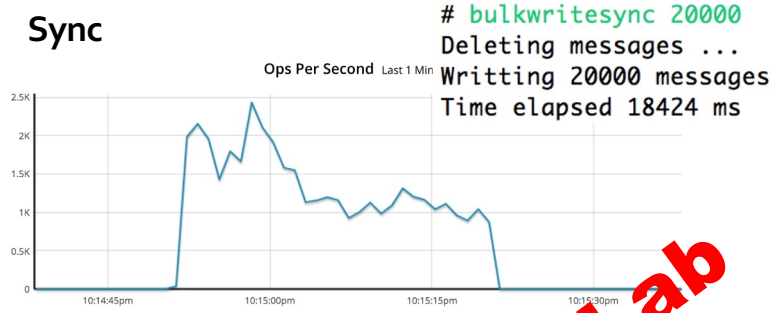
Lab 10: Bulk Write Performance

- Implement method: bulkWrite (String[] words) : Reactive 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 a list of JsonObject of messages
- Insert the messages into the collection (in reactive / sync way)
- Print the time elapsed to STDOUT
- Compare results sync vs. reactive. Check both time and operations per second in the console

Reactive



Sync



Lab

Query Reactive mode

```
cluster.reactive()
        .query("SELECT * FROM `travel-sample` LIMIT 5")
        .flatMapMany(ReactiveQueryResult::rowsAsObject)
        .subscribe(row -> System.out.println(row.toString()),
                e -> System.err.println("N1QL Error/Warning: " + e));
```

- 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 – Reactive version

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

Thank you