

Couchbase Cloud Test Drive Manual

Introduction

The objective of this Test Drive is to familiarize you with the basics of Couchbase Server and Couchbase Sync Gateway. You will go through four exercises using this test drive:

1. Couchbase basics

- You will log into the Couchbase Console (the web-based UI that comes with Couchbase Server)
- Next, you will load the "travel-sample" bucket, a predefined data set that's great for experimentation.
- Finally, you will log into the Sync Gateway console briefly.

2. Key/value document storage

- You will use the Couchbase Console to add documents to a bucket.
- Next, you will locate documents using the key
- After that, you will learn how to edit documents.
- Finally, you will see how to delete a document.

3. Querying with N1QL

- You will use the Couchbase Console to view indexes on the "travel-sample" bucket.
- Next, you will execute SELECT queries in the Query Workbench.
- Finally, you will see how to use data modification queries like INSERT, UPDATE, and DELETE.

4. Full text search (FTS)

- You will use the Couchbase Console to create a full text search index
- After creating the index, you will execute the FTS using search terms

About Couchbase

Couchbase's mission is to be the data platform that revolutionizes digital innovation. To make this possible, Couchbase created the world's first Engagement Database. Built on the most powerful NoSQL technology, the open source Couchbase Data Platform includes Couchbase Server and Couchbase Mobile. The platform provides unmatched agility and manageability – as well as unparalleled performance at any scale – to deliver ever-richer and ever more personalized customer experiences.

Couchbase customers include industry leaders like AOL, Amadeus, AT&T, Cisco, Comcast, Concur, Disney, Dixons Carphone, eBay, General Electric, Marriott, Neiman Marcus, Ryanair, Rakuten/Viber, Tesco, Verizon, Wells Fargo, as well as hundreds of other household names.

Prerequisites

The only prerequisite is a web browser and an Azure account.

In this test drive, a 3 node cluster of Couchbase Server and 1 instance of Couchbase Sync Gateway will be at your disposal.

Note that Sync Gateway is only briefly touched on in this test drive. For more information on using Sync Gateway and the entire Couchbase Mobile solution, please contact partners@couchbase.com or sales@couchbase.com to plan a proof-of-concept.

Guide to using this test drive

The first lab needs to be done first. After that, you can elect to work on any of the remaining 3 labs in any order.

Code snippets and fields in JSON document will appear as a monospaced font, like this: `SELECT t.*
FROM `travel-sample` t`

What you will learn from this Test Drive

You will learn the basics of using Couchbase Server in this test drive.

After you complete these labs, we invite you to [visit the Azure Marketplace](#) to spin up your own Couchbase Server cluster.

Couchbase Server offers a full enterprise document database solution. A performant key/value database engine with a memory-first architecture is at the core of Couchbase Server. The Couchbase Console provides a built-in UI that makes it easy to perform ops and dev tasks out of the box. Couchbase Server also has robust querying options to match your use cases, including [N1QL \(which is SQL for JSON\)](#) and a language aware full text search engine built on the [Bleve](#) engine. After completing these labs, you will understand how to use all these components.

In addition, you will also be taking a very shallow look at Sync Gateway, which is part of the [Couchbase Mobile platform](#).

Test Drive Support

If you have any questions or problems with this test drive, please contact partners@couchbase.com or sales@couchbase.com.

Labs

Great. Now, we're ready to get started! Here are the labs we'll work through in order:

- 1 - Logging into Couchbase
- 2 - Key Value Document Storage
- 3 - Querying with N1QL
- 4 - Full Text Search

Lab 1 - Logging into Couchbase

This is the first lab for the Couchbase Azure Test Drive. You need to complete this lab before starting any of the other labs.

Objective

This lab is designed to get you started with Couchbase Server. You will start the Azure Test Drive. Then, you will login to the Couchbase Console. In the console, you will load up a bucket of sample data. Finally, you will view the Sync Gateway console.

This lab will not be covering any clients or SDK usage. You will not do anything with the Sync Gateway console in later labs (yet).

Steps

Start the test drive

The test drive should start launching once you sign up or sign in.

You may be prompted for additional information, and you will be asked to agree to the terms of service. Accept to proceed.

Once done, Azure will start deploying the test drive. This should only take a few minutes.

What Azure is doing:

- * Deploying 3 "nodes" of Couchbase Server on 3 Azure VMs
- * Setting up each node
- * Creating a cluster, and adding all 3 nodes to the cluster
- * Creating administrator credentials
- * Deploying 1 node of Sync Gateway on an Azure VM.

When you use the standard deployment for Couchbase in Azure Marketplace, it will do all these operations for you as well. However, you will have more control over the deployment. You can configure details like how many nodes, what type of VMs to use, the credentials, and more.

When the test drive is done deploying, you'll see "Access information" displayed. It also is sent to you via email

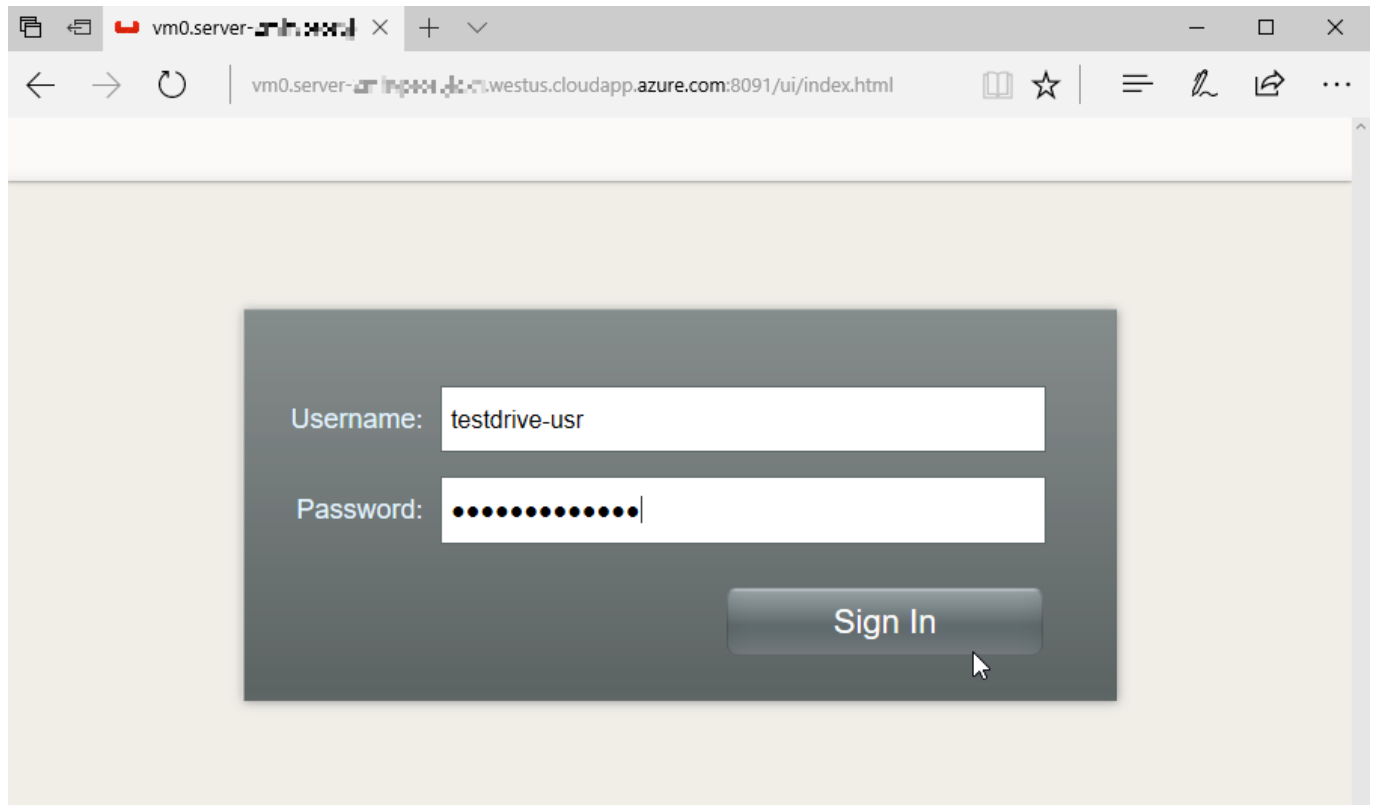
This includes:

- Server Admin URL (where you'll go in your browser to login to the Couchbase Server Console)
- Admin Credentials (how you'll login to the Console)
- Sync Gateway Admin URL (where you'll go in your browser to view the Sync Gateway console)

You may want to bookmark these URLs and make a note of the access information, since you will be using it throughout the remaining labs.

Login to the Couchbase Console

Start by going to the Server Admin URL. You will be prompted with a login form. Enter the credentials you received in the last step. Click "Sign In".



Once you successfully login, you will see the Couchbase Enterprise Edition console. You will be looking at the "Overview" section initially. Feel free to navigate around to anything that interests you. Here are a few places of interest to get you started:

- **Server Nodes** - This will show you a list of all the nodes in the cluster. Each "node" is a separate VM with Couchbase Server installed that joins together into a single "cluster". In a production deployment, you can add nodes to a cluster within the UI, with a command line utility, or with a REST API.
- **Data Buckets** - In Couchbase Server, a "bucket" is the fundamental data container. Each bucket contains documents. When clicking "Data Buckets", you will see a list of all the buckets that live in the cluster. To start with, the Test Drive has created a single bucket called "sync_gateway". Later, you'll be creating another bucket.
- **Query** - When you want to run N1QL queries, you can do so from this tab that takes you to the Query Workbench. More on this in **Lab 3 - Querying with N1QL**.

Load "travel-sample" data

In the Couchbase Console, click the "Settings" tab.

Once in Settings, click the "Sample Buckets" button. Under "Available Samples", find "travel-sample" and check the box. Click "Create" to start loading this sample bucket.

Settings

[Cluster](#) [Update Notifications](#) [Auto-Failover](#) [Alerts](#) [Auto-Compaction](#) [Sample Buckets](#)

Sample Buckets

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 can be accessed without a password! They are only recommended for non-production environments.

Installed Samples

There are no samples available to install.

Available Samples

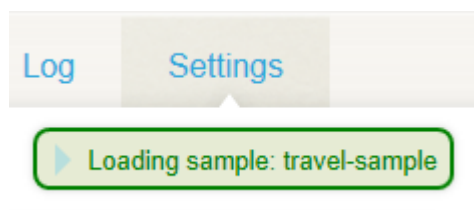
☐ beer-sample

☐ gamesim-sample

☒ travel-sample

Create

The "travel-sample" is a set of sample documents that will be loaded into a new bucket called "travel-sample". It contains five different kinds of documents: airlines, airports, routes, hotels, and landmarks. This sample data is meant to help you explore the features and functionality of Couchbase Server. You will use this bucket in the later test drive labs.



The bucket will take a minute or so to completely load. Click on "Data Buckets" to see a list of all the buckets in this cluster. The "sync_gateway" should still be there, and now "travel-sample" should be on the list too.

When complete, there should be 31,591 documents in the "travel-sample" bucket (look in the 'item count' column).

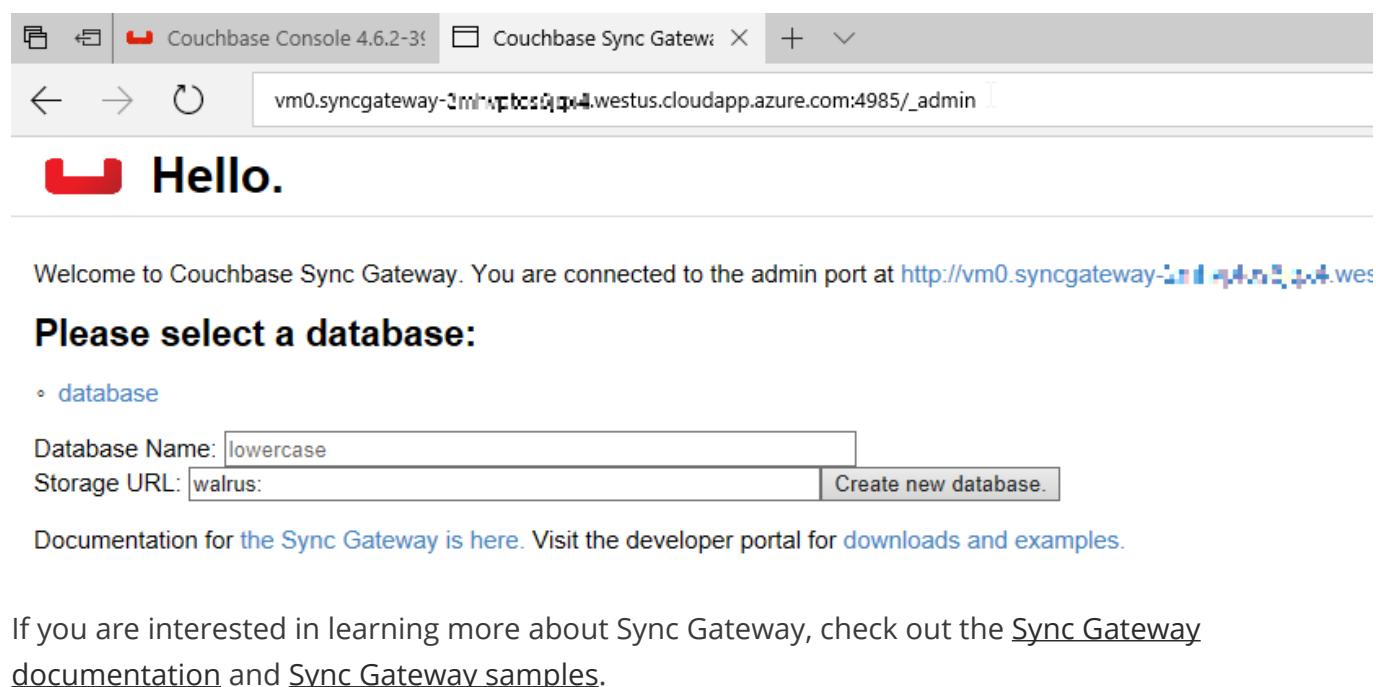
Data Buckets

Couchbase Buckets							Create New Data Bucket
Bucket Name	Data Nodes	Item Count	Ops/sec	Disk Fetches/sec	RAM/Quota Usage	Data/Disk Usage	
▶ sync_gateway	● 2	3	0	0	98.1MB / 200MB	16MB / 32.6MB	Documents Views
▶ travel-sample	● 2	31,591	0	0	136MB / 200MB	74.8MB / 122MB	Documents Views

Sync Gateway

Sync Gateway is separate from Couchbase Server, but it allows Couchbase Lite databases (on mobile devices) to sync with each other and with Couchbase Server automatically.

This lab does not include details on how to use Sync Gateway. You can visit the Sync Gateway Admin console via the URL that you saw in "Access information" earlier.



The screenshot shows a web browser window with two tabs: "Couchbase Console 4.6.2-3" and "Couchbase Sync Gateway". The address bar shows the URL "vm0.syncgateway-1m1p1cs0p4.westus.cloudapp.azure.com:4985/_admin". The page features the Couchbase logo and the text "Hello.". Below this, a welcome message states: "Welcome to Couchbase Sync Gateway. You are connected to the admin port at http://vm0.syncgateway-1m1p1cs0p4.westus.cloudapp.azure.com:4985/_admin". A section titled "Please select a database:" includes a radio button for "database". Below this, there are input fields for "Database Name:" (containing "lowercase") and "Storage URL:" (containing "walrus:"). A "Create new database." button is positioned to the right of the Storage URL field. At the bottom of this section, a link for "Documentation for the Sync Gateway is here." is provided, along with a reference to the "developer portal for downloads and examples."

If you are interested in learning more about Sync Gateway, check out the [Sync Gateway documentation](#) and [Sync Gateway samples](#).

Summary

You are now ready to begin labs 2, 3, and 4. The next lab is about Key Value Document Storage, but you can complete the labs in any order you choose.

For more information about Couchbase Server or Sync Gateway on Microsoft Azure, please contact partners@couchbase.com or sales@couchbase.com.

Lab 2 - Key/Value Document Storage

This is the second lab for the Couchbase Azure Test Drive. You need to have completed the first lab (Couchbase basics) before starting this lab.

Objective

This lab is designed to get you familiar with Couchbase Server. You will learn about buckets, keys and JSON documents work. You will be creating new documents, finding documents by key, making changes to documents, and deleting documents. At the end of this lab, you will understand how to do all those operations within the Couchbase Console.

This lab will not be covering any clients or SDK usage.

Steps

Introduction to documents

The fundamental unit of data in Couchbase Server (or any document database) is the document.

Each document is simply a JSON entity. It could be a JSON array, but it is typically a single JSON document. This is an example of a document that is in the travel-sample bucket that you loaded in the previous lab.

```
{
  "airline": "AF",
  "sourceairport": "TLV",
  "destinationairport": "MRS",
  "airlineid": "airline_137",
  "distance": 2881.617376098415,
  "equipment": "320",
  "stops": 0,
  "type": "route"
  "schedule": [
    {
      "day": 0,
      "flight": "AF198",
      "utc": "10:13:00"
    },
    {
      "day": 6,
      "flight": "AF496",
      "utc": "19:14:00"
    }, ...
  ], ...
}
```

This is a document that represents an airline route. It has a number of simple data fields: `airline`, `sourceairport`, `type`, and so on. But it also can store more complex, structured data. The `schedule` field is an array of JSON objects. In a relational database, this would typically be stored as multiple rows in separate table, but it is stored all in one piece in a *denormalized* form in a document database.

Each document is identified by a "key". In Couchbase Server, that key is not part of JSON object, but it is "metadata". A document key can be any value you want it to be. This document has a key of `route_10000`, for instance.

What is a bucket?

In Couchbase Server, documents are stored in named **buckets**. Within a bucket, each document must have a unique key. Beyond that, there is no restriction on the shape of the data from document to document. A bucket called **travel-sample**, for instance, could have several thousand airline route documents like the one above, but it could also have hundreds of documents that represent airlines, landmarks, airports, and hotels.

This flexibility is why document databases are said to be "schemaless".

Couchbase Server does not tell individual documents what fields they should or shouldn't have. This is why you often see fields like `type` in the above route example, to act as a sort of discriminator flag for organizational purposes. Although they can be helpful, discriminator fields like `type` are not required, "type" not a reserved word, and the field is not treated any differently in Couchbase than the other fields.

Create a new document

From the Couchbase Console, click "Data Buckets".

The "travel-sample" bucket should be listed (it was created in the first lab). Locate it on list of buckets and click the "Documents" button.

Couchbase Enterprise Edition

Documentation • Support • About • Sign Out

Overview Server Nodes **Data Buckets** Query Indexes XDCR Security Log Settings

Data Buckets

Couchbase Buckets Create New Data Bucket

Bucket Name	Data Nodes	Item Count	Ops/sec	Disk Fetches/sec	RAM/Quota Usage	Data/Disk Usage	
cbdownload	1	47,482	0	0	48.9MB / 256MB	51.2MB / 56.5MB	Documents Views
ccc	1	2,192	0	0	36MB / 512MB	36.6MB / 41.6MB	Documents Views
patients	1	607	0	0	46.1MB / 128MB	13.5MB / 22.1MB	Documents Views
<u>travel-sample</u>	1	31,591	0	0	75.7MB / 100MB	107MB / 133MB	Documents Views

You should now see a list of documents. The ID column shows the key for each document, and the Content column shows a partial excerpt of the JSON document. To create a new document, click the "Create Document" button.

travel-sample > Documents Current page: 1 5

Documents Filter Document ID Lookup ID Create Document

ID	Content	
airline_10	<pre>{"id":10,"type":"airline","name":"40-Mile Air","iata":"Q5","icao":"MLA","...</pre>	Edit Document Delete
airline_10123	<pre>{"id":10123,"type":"airline","name":"Texas Wings","iata":"TQ","icao":"TXW...</pre>	Edit Document Delete
airline_10226	<pre>{"id":10226,"type":"airline","name":"Atifly","iata":"A1","icao":"A1F","ca...</pre>	Edit Document Delete
airline_10642	<pre>{"id":10642,"type":"airline","name":"Jc royal.britannica","iata":null,"ic...</pre>	Edit Document Delete
airline_10748	<pre>{"id":10748,"type":"airline","name":"Locair","iata":"ZQ","icao":"LOC","ca...</pre>	Edit Document Delete

After clicking the "Create Document" button, you will be prompted for a document ID. It must be

unique to that bucket. If I typed in "airline_10" for instance, I would get an error message. Enter a creative ID like "lab2document". A document with that key and some default JSON will be created, and you will be taken to a page where you can make changes to the JSON.



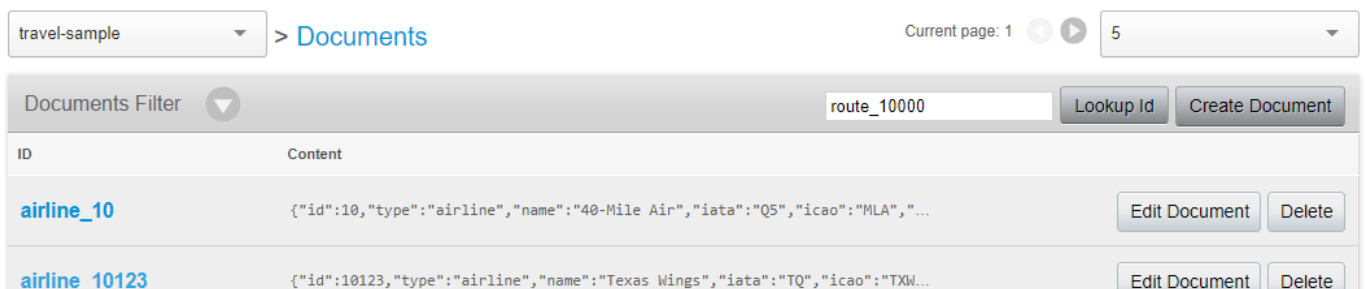
Enter the following JSON and then click the "Save" button.

```
{
  "name": "Matthew",
  "shoeSize": 13
}
```

If, at any point, the JSON you are entering isn't valid, an error message will appear on the screen. Once you're done, click the "Documents" link to go back to the list of documents. Feel free to experiment and create additional document if you'd like.

Finding a document by key

This bucket has a large amount of documents. You can go right to a specific document by using its key. Enter the key **route_10000** and click the "Lookup Id" button.



You should be taken to the document that was shown at the very beginning of this lab. It's an airline route from Ben Gurion Airport in Tel Aviv (TLV) to Marseille Provence Airport in France (MRS). We don't know the name of the airline from looking at just this document, but note that the `airlineid` has a value of **airline_137**. To find out the name of the airline, look up that document by key.

Make changes to a document

Lookup the document you created earlier (by key **lab2document**). Let's make a change to this document. Let's add something a little more interesting than a flat field:

```
{
  "name": "Matthew",
  "shoeSize": 13,
  "socialMedia": [
    { "twitter": "mgroves" },
    { "github": "mgroves" },
    { "linkedin": "https://www.linkedin.com/in/mgroves/" }
  ]
}
```

Click the "Save" button to make these changes. Notice that I evolved the structure of this data without updating any schema or affecting the other documents in this database. It's possible that some documents have `socialMedia` and some don't. That flexibility/tradeoff makes Couchbase suitable to agile development and to situations where you need to ingest data from multiple sources that have changing data structures.

Delete a document

Finally, you can delete a document. Just click the "Delete" button from the document list page, or from the individual document editing page.

Note that there are no referential constraints to take into account. If I deleted the **airline_137** document, for instance, the **route_10000** document would still have an `airlineid`, and it would still have a value of **airline_137**.

Summary

In this lab, you have learned all the basics of interacting with documents and buckets of documents. You have create a document, edited a document, looked up documents by key, and deleted a document. With this core set of operations and a denormalized data model, you can accomplish some very powerful things.

However, there are situations where key lookups aren't the best approach to finding documents. Please continue with lab 3 to learn how to query documents with N1QL and lab 4 to learn how find documents with a full text search.

Lab 3 - Querying with N1QL

This is the third lab for the Couchbase Azure Test Drive. You need to have completed at least the first lab (Couchbase basics) before starting this lab. The second lab (Key/Value Document Storage) is optional, but recommended.

Objective

This lab is designed to get you familiar with the N1QL (Non 1st-Normal-Form Query Language). N1QL is a flavor of SQL that is designed to work with JSON data. If you've used any flavor of SQL in

the past, you should be right at home. You will learn about Query Workbench, writing SELECT queries, basic JOINS, and data manipulate queries. At the end of this lab, you will have a basic understanding of N1QL.

N1QL is a very rich language, so this lab will only be scratching the surface. You are invited to check out the [extensive N1QL documentation](#) or the [interactive N1QL tutorial](#), once you have a handle for the basics.

This lab will not be covering any clients or SDK usage.

Steps

Query Workbench

From the Couchbase Console, click "Query". You will taken to an interactive page that allows you to enter a N1QL query, execute it, and see the results. Try a very simple query like `SELECT 1;`. You can type it in, end with a semicolon, and press "enter", or you can just click the "Execute" button. In the "Result" window, you'll see the full response to your request. It includes fields like `success` and `metrics`, but focus on the `results` field. It's an array of JSON documents. In this case, it's only one document with one field (assigned a name of `$1`, since you didn't specify a name), and a value of 1.

The screenshot displays the Couchbase Enterprise Edition Query Workbench interface. At the top, the Couchbase logo and "Enterprise Edition" text are visible, along with navigation links for Documentation, Support, About, and Sign Out. Below this is a navigation bar with tabs for Overview, Server Nodes, Data Buckets, Query (selected), Indexes, XDCR, Security, Log, and Settings. The main interface is divided into several sections. On the left, there's a "Bucket Analysis" section with a tree view showing "Fully Queryable Buckets" (cbdownload, ccc, patients, travel-sample) and "Queryable on Indexed Fields" and "Non-Indexed Buckets". The central area is the "Query" editor, which contains the text "1 SELECT 1;". Above the editor is an "Execute" button, and below it are navigation arrows and a "Clear History" button. To the right of the editor is a "Results" section. It shows the query status as "Status: success", with "Elapsed: 1.00ms", "Execution: 0", "Result Count: 1", and "Result Size: 31". Below this, the JSON response is displayed in a collapsible tree view. The response is a single document with fields: "requestID", "clientContextID", "signature" (containing "\$1": "number"), "results" (an array containing one document with "\$1": 1), "status": "success", and "metrics" (containing elapsedTime, executionTime, resultCount, and resultSize).

```
1 SELECT 1;
```

Execute

← 4/4 → Clear History Save Query

Bucket Analysis

- Fully Queryable Buckets
 - cbdownload
 - ccc
 - patients
 - travel-sample
- Queryable on Indexed Fields
- Non-Indexed Buckets

JSON Table Tree Results Save JSON

Status: success Elapsed: 1.00ms Execution: 0 Result Count: 1 Result Size: 31

```
1 {
2   "requestID": "4d90005c-4e50-4553-8ae0-7c6c886ee10c",
3   "clientContextID": "5c3aba7a-5b18-4dc4-bea2-996a48718b3e",
4   "signature": {
5     "$1": "number"
6   },
7   "results": [
8     {
9       "$1": 1
10    }
11  ],
12  "status": "success",
13  "metrics": {
14    "elapsedTime": "1.0005ms",
15    "executionTime": "0",
16    "resultCount": 1,
17    "resultSize": 31
18  }
19 }
```

Before doing anything more advanced, you must first understand the basics of indexing. In relational databases, you often don't need to create indexes on small tables, because a table scan is often fast enough for small amounts of data. So, indexes aren't required.

In a huge collection of documents, you must create an index on the fields you want to query,

otherwise the queries will not execute.

In Couchbase Server, You can optionally create an index on a bucket that is roughly equivalent to a table scan: `CREATE PRIMARY INDEX def_primary on bucketname`. Fortunately, this index has already been created on the "travel-sample" bucket automatically, along with several other indexes. Click the "Indexes" tab in the Couchbase Console to see what indexes exist.

▶ travel-sample	127.0.0.1:8091	def_airportname	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_city	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_faa	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_icao	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_name_type	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_primary	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_route_src_dst_day	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_schedule_utc	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_sourceairport	Standard Global Secondary Indexes	Ready	100 %
▶ travel-sample	127.0.0.1:8091	def_type	Standard Global Secondary Indexes	Ready	100 %

An index called `def_primary` already exists on travel-sample.

There are several other indexes on this bucket as well. For instance, if you example the `def_type` index, you'll see that the `type` field is being indexed.

▼ travel-sample	127.0.0.1:8091	def_type	Standard Global Secondary Indexes	Ready	100 %
Definition: <code>CREATE INDEX 'def_type' ON 'travel-sample'('type') WITH { "defer_build"=true }</code>					

Consider this query:

```
SELECT t.*
FROM `travel-sample` t
WHERE t.type = 'airline';
```

What happens when you execute that query depends on the indexes you have (or don't have). If I have the `def_type` index, then this query will make use of that index to quickly fetch any document with a type value of "airline".

If all you have is the `def_primary` index, then the above query will need to scan every document in

the bucket, which is significantly slower.

If you don't have any indexes at all, the above query will fail to execute with an error.

Try running the above query while experimenting with adding/removing indexes. Here are some commands you will need:

- Drop the primary index: `DROP INDEX `travel-sample`.`def_primary`;`
- Drop the type index: `DROP INDEX `travel-sample`.`def_type`;`
- Create the primary index: `CREATE PRIMARY INDEX `def_primary` ON `travel-sample``
- Create the type index: `CREATE INDEX `def_type` ON `travel-sample`(`type`)`

When you're done, make sure both `def_type` and `def_primary` indexes exist in their initial state.

JOIN

One feature that is somewhat unique to Couchbase Server is the ability to `JOIN` documents together in a query. Currently, a value can be joined to a document by its key.

Back to the `travel-sample`, here is a document with key `route_10000`.

```
{
  "sourceairport": "TLV",
  "destinationairport": "MRS",
  "airlineid": "airline_137",
  ...
}
```

Notice that it has a field, `airlineid`, which contains a value `airline_137`. The document with that key looks like:

```
{
  "type": "airline",
  "name": "Air France",
  ...
}
```

There is another document with that key, so we could join these two documents together:

```
SELECT r.sourceairport, r.destinationairport, r.airlineid, a.name
FROM `travel-sample` r
JOIN `travel-sample` a ON KEYS r.airlineid
```

Notice that this query is performing a `JOIN` from the "travel-sample" bucket to itself. It is selecting every document from "travel-sample" and aliasing the fields from there as "r". It's joining to any document where the key matches the value of `airlineid`. `JOIN` is an `INNER JOIN` by default, so this

query is implicitly limiting the set of documents returned in "r" by using `airlineid` (which is unique to route documents).

The above query will actually take quite a while to run (because it is using the primary index), and it will return 17000+ results. To narrow it down further, let's limit it to just `route_10000`.

```
SELECT r.sourceairport, r.destinationairport, r.airlineid, a.name
FROM `travel-sample` r
JOIN `travel-sample` a ON KEYS r.airlineid
WHERE META(r).id = 'route_10000';
```

More on `META` later, but for now, running that query should return one result.

```
"results": [
  {
    "airlineid": "airline_137",
    "destinationairport": "MRS",
    "name": "Air France",
    "sourceairport": "TLV"
  }
]
```

The `airlineid`, `destinationairport`, and `sourceairport` fields all come from the route document. But the `name` field comes from the airline document.

INSERT

With N1QL, you also get the ability to run `INSERT`, `UPDATE`, and `DELETE` queries (and more, like `MERGE` and `UPSERT`).

Let's go through all 3 to see how they work and how they differ from SQL that you've written before.

An `INSERT` will always be inserting into `KEY` and `VALUE` fields. The `VALUE` field can be a JSON literal, or the result of a `SELECT` query. Here's an example of an `INSERT` with a JSON literal:

```
INSERT INTO `travel-sample` (KEY, VALUE)
VALUES ("lab3document", { "name": "Matthew", "twitter": "@mgroves" } );
```

This will create a document with the key "lab3document". Now try finding that document by its key (refer back to [lab 2](#) if you need to). You can also use `RETURNING` in the query to return back the record that you just inserted:

```
INSERT INTO `travel-sample` t (KEY, VALUE)
VALUES (UUID(), { "name": "Matthew", "twitter": "@mgroves" } )
RETURNING t.*, META(t).id;
```

The `RETURNING` acts like a `SELECT`. In the above query, I've given the "travel-sample" bucket an alias of "t". In the `RETURNING`, I've asked for `t.*` (the entire document) and I used `META` to get its key. Since I used the `UUID` function to generate a unique key, the `RETURNING` comes in very handy in telling me the value of the key that was generated. Here are the results of the above `INSERT` query:

```
"results": [
  {
    "id": "4ecff4f1-dda0-4c9e-bbe1-3c0d95734253",
    "name": "Matthew",
    "twitter": "@mgroves"
  }
]
```

Try this for yourself in Query Workbench.

UPDATE

`UPDATE` in N1QL is used to make changes to an existing document. Assuming you still have the "lab3document" from above, here's an example of `UPDATE` in action:

```
UPDATE `travel-sample`
USE KEYS "lab3document"
SET name = "Matthew Groves", shoeSize = 13;
```

With this `UPDATE`, I'm changing the value of the existing `name` field to be my first and last name. Also note that I'm setting the value of `shoeSize`, which previously didn't exist in the document.

`RETURNING` can also be used with an `UPDATE`.

DELETE

Finally, let's get rid of this "lab3document" with a `DELETE` query.

```
DELETE FROM `travel-sample`
USE KEYS "lab3document";
```

This will delete the document with the matching key. If you use a `RETURNING` with a `DELETE`, it will return the document that you deleted.

META and other JSON tidbits (TODO)

`META` has been mentioned a few times already, and `UUID` was also mentioned. These are [N1QL functions](#) (of which there are many). `UUID` simply generates a universally unique identifier (similar to a GUID). `META` is a function that, applied to a document, returns metadata about that document.

The document key (`id`) is the main reason you'd use `META`, but there is other information in there that might be of some use.

Before wrapping up, there is one more basic thing about N1QL that should be made explicit. Earlier, a "route" document was used as an example. This document contains an array of objects as a field. N1QL needs a way to address both array elements and fields within an object hierarchy. Intuitively, you'll use the JavaScript array `[]` syntax and dotted `.` syntax. Here's an example of a query that gets the first item from the `schedule` array, and the `day` value within that item.

```
SELECT t.schedule[0].day
FROM `travel-sample` t
WHERE META(t).id = "route_10000";
```

Because there are arrays and hierarchies present in JSON data, the N1QL language has added a lot of statements, functions, and operators to query different shapes of data. Because of this, N1QL is considered a "superset" of SQL.

Summary

In this lab, you have learned the basics of querying documents with N1QL. You have learned how to use the Query Workbench, `SELECT` documents, modify documents with N1QL, and some unique features of N1QL to deal with JSON. With N1QL, you can query denormalized JSON documents while being productive very quickly with a familiar SQL syntax.

There is a lot more to learn about N1QL, even for SQL experts. Check out the [N1QL documentation for a full language reference](#), and if you have tricky N1QL questions, the [Couchbase N1QL forums](#) can help you find the answers.

N1QL is a great tool for finding documents that meet strict criteria. It is not necessarily the best tool for a generalized search. For that, please check out lab 4 to learn how find documents with a full text search.

Important note: If you can use a key/value operation, you should: it will be faster than running a query. However, key/value operations are sometimes not feasible for certain data access patterns, and that's where N1QL can help.

Lab 4 - Full Text Search

This is the fourth lab for the Couchbase Azure Test Drive. You need to have completed at least the first lab (Couchbase basics) before starting this lab. The second lab (Key/Value Document Storage) and the third lab (N1QL) are optional, but recommended.

Objective

This lab is designed to introduce you to the Full Text Search (FTS) capabilities built into Couchbase. With many databases, FTS capabilities are limited or non-existent, and are often farmed out to external tools like Elasticsearch, Solr, etc. Elasticsearch is a powerful tool and Couchbase does have an [Elasticsearch plug-in](#) available. However, if you don't need the full power of Elasticsearch, Couchbase's own FTS (built on the [Bleve](#) engine) is a viable way of delivering features without any extra integration work.

Couchbase's FTS contains many options for searching and indexing. This lab will only be scratching the surface. You are invited to check out the [extensive FTS documentation](#) once you have a handle for the basics.

This lab will not be covering any clients or SDK usage.

Steps

Search feature

In the "travel-sample" bucket, there are a number of "landmark" documents. These landmark documents contain information about places, including address, city, geolocation, name, url, and a description (content). Here's an example document (key `landmark_11772`):

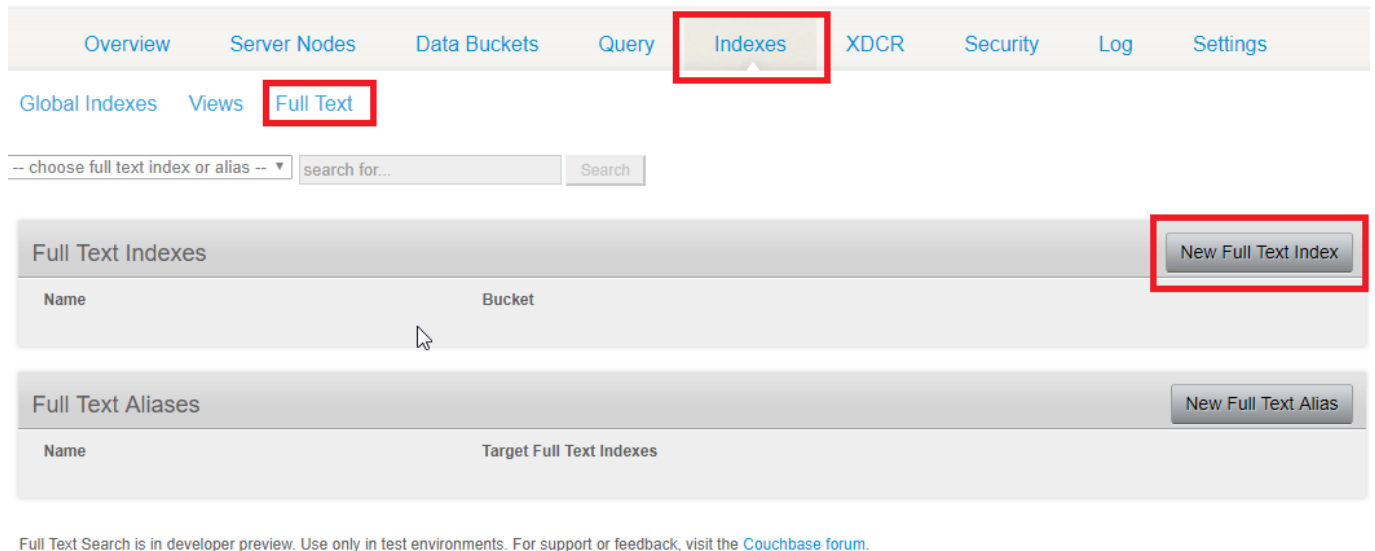
```
{
  {
    "name": "The Griddle Café",
    "address": "7916 W Sunset Blvd",
    "url": "http://www.thegriddlecafe.com",
    "content": "The Griddle Café is the best breakfast experience in LA. It features
pages of every type of pancake you can imagine, which also happen to be twice as
large as any pancake you've ever had, and still manage to be fluffy-thick and light
on the tummy. Coffee is fresh, in a French press, and the menu features more than
just breakfast. Short story: Food is awesome, service is great, but its always
crowded. Don't worry though, they serve fast and you will feel the wait is worth
it.",
    "geo": {
      "lat": 34.097866,
      "lon": -118.36221,
      "accuracy": "ROOFTOP"
    },
    "type": "landmark",
    "state": "California",
    ...
  }
}
```

Now put yourself in place of a user trying to find interesting places to go. If that user searches "breakfast", should this document be in the results? What about "bed and breakfast"? Maybe it should appear, but lower down the list in relevancy. What if your user types "brakfast" instead? These type of "language aware" queries that return results with relevancy scores are difficult to do with the more exacting nature of N1QL([3queryingwithn1ql.md](#)).

Building an index

Let's build a index to help our users find interesting results. But let's keep it very simple for this example.

From the Couchbase Console, go to "Indexes" then click "Full Text".



Click the "New Full Text Index" button to create a new FTS index. I'll name the index "travel-sample-idx". Next, I'll choose the "travel-sample" bucket. After that, I'll expand the "Type Mappings" section. There's already a "default" type mapping. If I saved this index now, this index would search on every field in every document (even airline documents, and even the geolocation fields and address fields in landmark documents, which will not be helpful to a user searching for "breakfast").

So, let's narrow it down to just the content field.

Click "Add Type Mapping". Enter "landmark" as the type name. Notice that the "Type Identifier" is set to "JSON type field" with a value of "type". This is the default behavior. But as I mentioned in an earlier lab, there's nothing magical, special, or reserved about the "type" name. You can specify "_type" or "doctype" instead. Let's leave it as "type", since that's what "travel-sample" uses.

Check the box "only index specified field" and click "ok" to save.

New Full Text Index

Index Name:

Bucket:

Type Identifier: ☒ JSON type field:
☐ Doc ID up to separator:
☐ Doc ID with regex:

Type Mappings

Add Type Mapping

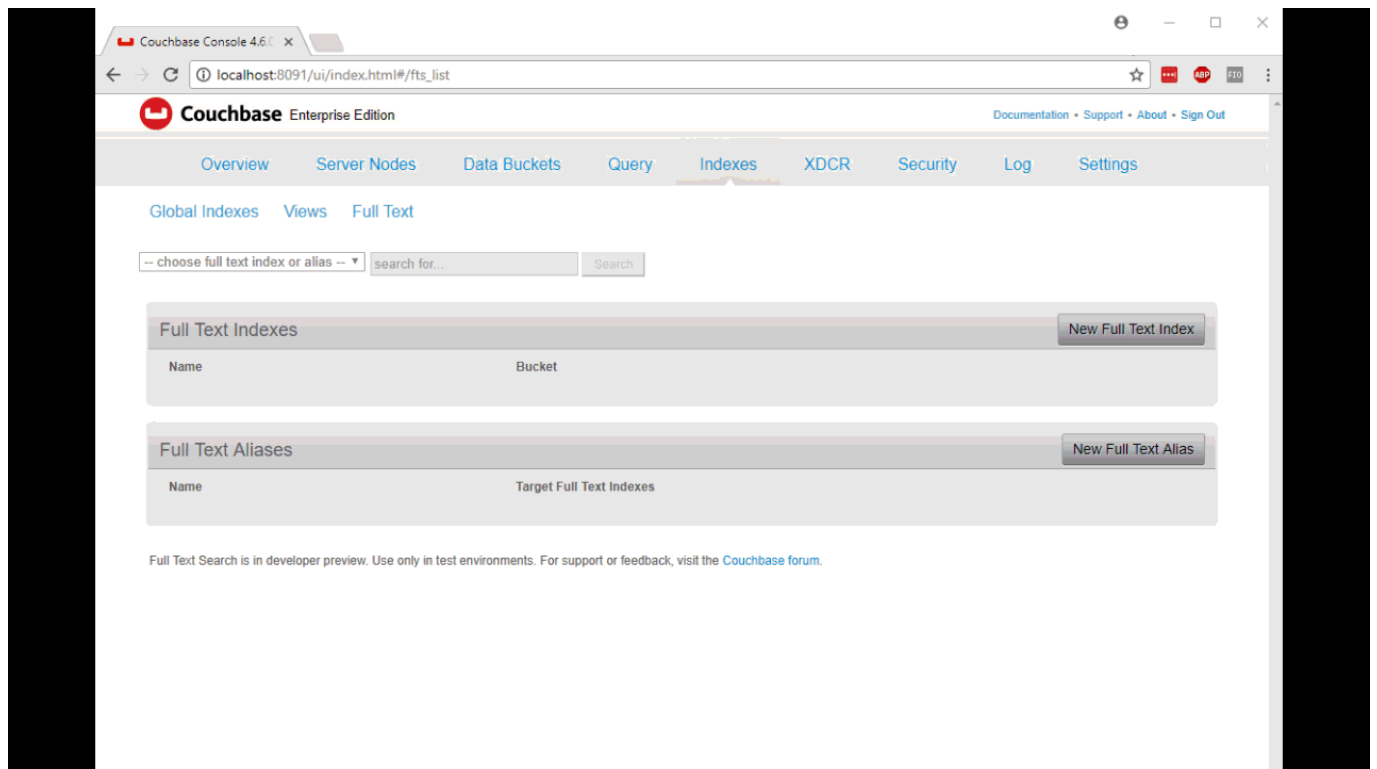
#	<input type="text" value="landmark"/>	<input type="text" value="inherit"/>	<input checked="" type="checkbox"/> enabled	<input checked="" type="checkbox"/> only index specified fields	<input type="button" value="ok"/> <input type="button" value="cancel"/>
#	default				

▶ Analyzers
▶ Custom Filters
▶ Advanced

☐ Show advanced settings

Next, we need to add a "child field" to this index. Right now, we only want to index the `content` field of landmark documents. Hover over the "landmark" record's and the "+" button. Click "insert child field". Enter "content" and the field and "content" as the "searchable as". Also check the "store" option (more on that later). Click "ok" to save.

After that, hover over the "default" record, click "edit" and uncheck the "enabled" option. We aren't going to use the default index, so this will save time when creating the initial index.



Finally, click the "Create Index" button. This will kick off the initial indexing process. You can keep clicking the "> Refresh" button to see the progress. Since this index is only on "landmark" documents, and only one field within those documents, the indexing should be done quickly.

Further changes/additions to documents will be automatically indexed on an incremental basis (or you can manually update if you choose).

Executing a search

Now that we have a search index, let's try out some searches. Click "Full Text" to back to the list of FTS indexes. In the dropdown box at the top, select "travel-sample-idx". Enter a search term like "breakfast" and click "Search".

[Global Indexes](#) [Views](#) [Full Text](#)

travel-sample-idx ▼

breakfast

Search

When searching for "breakfast", you should get 81 results. The most relevant results will appear at the top (just like a web search engine). You can check "Show Scoring" to see the actual score calculations if you'd like.

[Global Indexes](#) [Views](#) [Full Text](#)

Full Text Search: travel-sample-idx

breakfast

Search

☐ Advanced

[full text query syntax help](#)

81 results (2ms server-side)

landmark_7752

☐ Show Scoring

content
Breakfast and Lunch menus

landmark_2807

content
fantastic deli for breakfast and lunch.

landmark_37333

content
Pleasant place for a coffee break or breakfast.

landmark_33230

content
A great spot for breakfast or lunch.

Try search for "breakfast" and then "bed and breakfast". The former returns 81 results, the latter returns 85 results. Also notice that document **landmark_2807** appears in the results of both: 2nd in the former, 4th in the latter.

You can use a variety of operators within the search. You can use - to specify that a term must *not* be in the result, or + to specify that a term *must* be in the result. You can also use the ~ to specify a "fuzziness".

Try these searches and see what happens:

- +bed -breakfast
- -bed +breakfast
- brakfast ~2

- "bed and breakfast"

Finally, notice that the search results show a snippet of text with the relevant search terms highlighted (in yellow). When creating this index, you checked the "store" option. If you didn't check that option, you wouldn't get a the snippet and highlight: you would just get the document key and a score.

Summary

In this lab, you have learned the basics of using Couchbase Server's built in Full Text Search (FTS). You have learned how to create an index, limit the scope of the index, and execute a search.

For a complete reference to all the search options, check out the [bleve documentation](#) and the [FTS reference](#).

##Key Takeaways/Summary

Couchbase basics: Getting a cluster up and running in Azure. The Couchbase Console makes it easy to view your cluster and perform both dev and ops tasks.

Key/value document storage: Documents are keys and JSON values. You can add them quickly, find them by key, make changes by key (as long as it's valid JSON), and delete them by key.

Querying with N1QL: Key/value operations will always be the fastest, but querying multiple documents is easy with N1QL (which is just like SQL).

Full Text Search: A language-aware search for text is another way to locate document(s). You can leverage the many full text search options to help your users find the information they want.

Conclusions and Next Steps

If you're ready, please [visit the Azure Marketplace](#) to spin up your own Couchbase Server cluster. If you'd like to do more research, please check out the [Couchbase Developer portal](#). If you'd like to try running Couchbase on your own machine, you can [download Couchbase Server Enterprise or Couchbase Server Community](#) for free. It can run on Windows, Mac, and a variety of popular Linux distros.

Contact Information

If you have any questions about Couchbase Server or Couchbase Mobile, please contact partners@couchbase.com or sales@couchbase.com.