COUCHBASE CAPELLA Workshop

**Lab Handbook**

Lab 3: Connecting SDK

# Legend - Please Read!

We use the following formatting conventions in all lab modules:

* Yellow highlighted values (in the text or on the screenshots) must be replaced with the correct value in your environment, e.g., 192.168.61.101
* Orange highlighted values are used for additional emphasis in the context of a task
* Commands in **Courier New** font in bold are what you are expected to execute (see below):
  + The “**$**” prefix indicates the command to be executed in the terminal on your local computer (Mac and Linux).
  + The “**>**” prefix is only used to indicate Windows-specific commands to be executed on your local computer. We will continue to refer to the Windows *PowerShell*  as “terminal” in all the labs, and all universal examples of commands will assume the Mac and/or Linux prefix, i.e. “**$**”, and some commands may need to be modified before being run on Windows, e.g. change / to \, remove trailing & character to run commands in the background.
  + The text without any prefix is the output of the command.

|  |
| --- |

* **Red font** indicates important instructions, please pay very close attention to those.
* Items in **bold** or *italic* font are names or UI elements to pay attention to and/or click on.

Unless stated otherwise, all commands, parameters, settings, logins, passwords, etc. are case-sensitive!

The output of the commands in your terminal will likely differ from the output shown in the labs. Generally, this should not be an issue, as long as the output is similar, and there are no obvious error messages. Please ask for help if you get stuck.

Remember: copying and pasting the code and/or commands may produce errors due to auto-formatting in text editors, e.g. regular dash characters get replaced with extended ASCII dashes, regular quotation marks with fancy quotes; some special characters may be invisible. Such characters will not let your code run correctly. If this happens to you, try typing the commands instead of copying and pasting them.

# Lab Overview

The goal of this lab is to use the Python SDK to connect to a Couchbase Capella cluster and perform some core functionality with the SDK.

The lab assumes the participant has Python already installed. Participants are encouraged to use a version >= 3.5. See the Appendix for information on installing Python.

# Lab Setup

## Step 1: Create a working directory

Create a working directory, the directory name is up to the lab participant. The working directory will be the directory where the Python code will reside.

| **$ mkdir ~/cbc\_workshop** |
| --- |

## Step 2: Setup the Web UI

Move into the working directory created in [Step 1](#_tyjcwt).

| **$ cd ~/cbc\_workshop/** |
| --- |

## Step 3: Verify Python is installed

| **$ python -V**  Python 3.8.5 |
| --- |

\*\*NOTE\*\*: It is highly recommended to use virtual environments when working with Python. It is the easiest way to keep your python project organized (it will save you valuable time in the long run). The Appendix provides details on using virtual environments.

## Step 4: Install Couchbase

| **$ pip install couchbase**  Collecting couchbase  Downloading couchbase-3.0.9.tar.gz (1.5 MB)  |████████████████████████████████| 1.5 MB 4.1 MB/s  Installing build dependencies ... done  Getting requirements to build wheel ... done  Installing backend dependencies ... done  Preparing wheel metadata ... done  Requirement already satisfied: mypy-extensions in ~/.virtualenvs/cbc\_workshop/lib/python3.8/site-packages (from couchbase) (0.4.3)  Requirement already satisfied: six in ~/.virtualenvs/cbc\_workshop/lib/python3.8/site-packages (from couchbase) (1.15.0)  Requirement already satisfied: attrs>=19.1.0; python\_version > "3.5" in ~/.virtualenvs/cbc\_workshop/lib/python3.8/site-packages (from couchbase) (20.3.0)  Requirement already satisfied: wrapt>=1.11.2 in ~/.virtualenvs/cbc\_workshop/lib/python3.8/site-packages (from couchbase) (1.12.1)  Requirement already satisfied: pyrsistent>=0.15.2 in ~/.virtualenvs/cbc\_workshop/lib/python3.8/site-packages (from couchbase) (0.17.3)  Building wheels for collected packages: couchbase  Building wheel for couchbase (PEP 517) ... done  Created wheel for couchbase: filename=couchbase-3.0.9-cp38-cp38-macosx\_10\_15\_x86\_64.whl size=1337870 sha256=27e3f7d492bbe631dbd1ba0f20ae2145f628a99f9577b220780176a0f997b4fb  Stored in directory: /private/var/folders/fj/\_yt793dj6d9f976tf6mz\_nt40000gr/T/pip-ephem-wheel-cache-iapa7cgg/wheels/84/18/5c/0d8d47d9811eb569ff3ef9af86447e761ca9052eb4aea10058  Successfully built couchbase  Installing collected packages: couchbase  Successfully installed couchbase-3.0.9 |
| --- |

### 

## Step 4: Create the Python module for the workshop

In Python, a module is a file that ends in “.py”. So, open a preferred text editor, create a new file and save it as *cbc\_workshop.py*. Use the following code to get started.

| 0:  1:  2:  3:  4:  5: | if \_\_name\_\_ == '\_\_main\_\_':  try:  print('Hello Couchbase Cloud Workshop!')  except Exception as ex:  import traceback  traceback.print\_exc() |
| --- | --- |

After saving *cbc\_workshop.py*, running the module should output the following:

| **$ python cbc\_workshop.py**  Hello Couchbase Cloud Workshop! |
| --- |

# 

# Lab Steps

## Step 1: Connecting to a Couchbase Capella cluster

### 1a: Determine IPv4 Address

Navigate to this [site](https://whatismyipaddress.com/) or use the following *curl* command. Note the IP address.

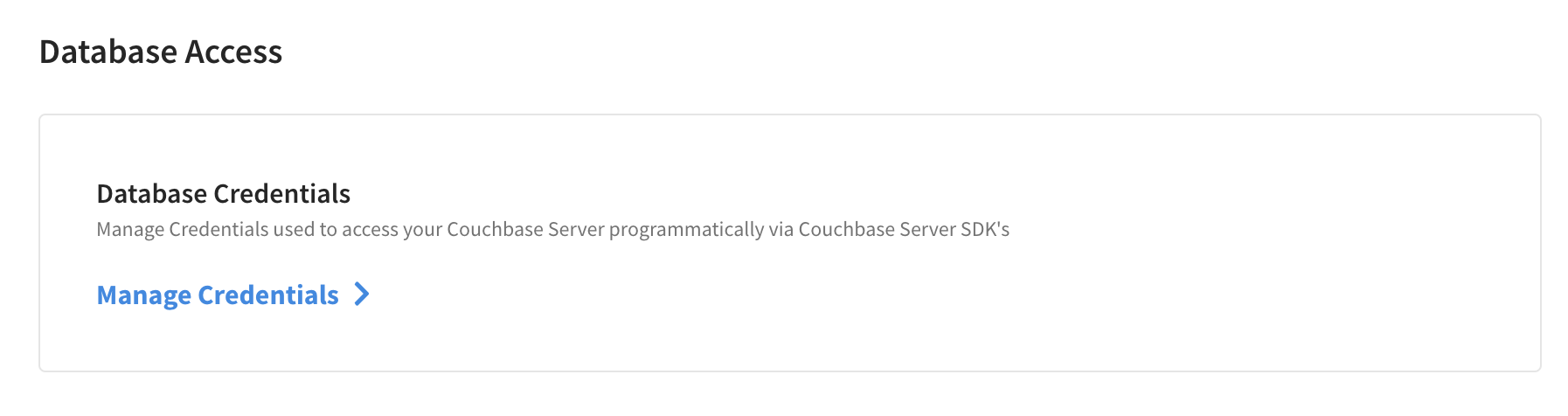
| **$ curl ifconfig.me**  109.93.42.42 |
| --- |

### 1b: Add IPv4 Address to Couchbase Capella

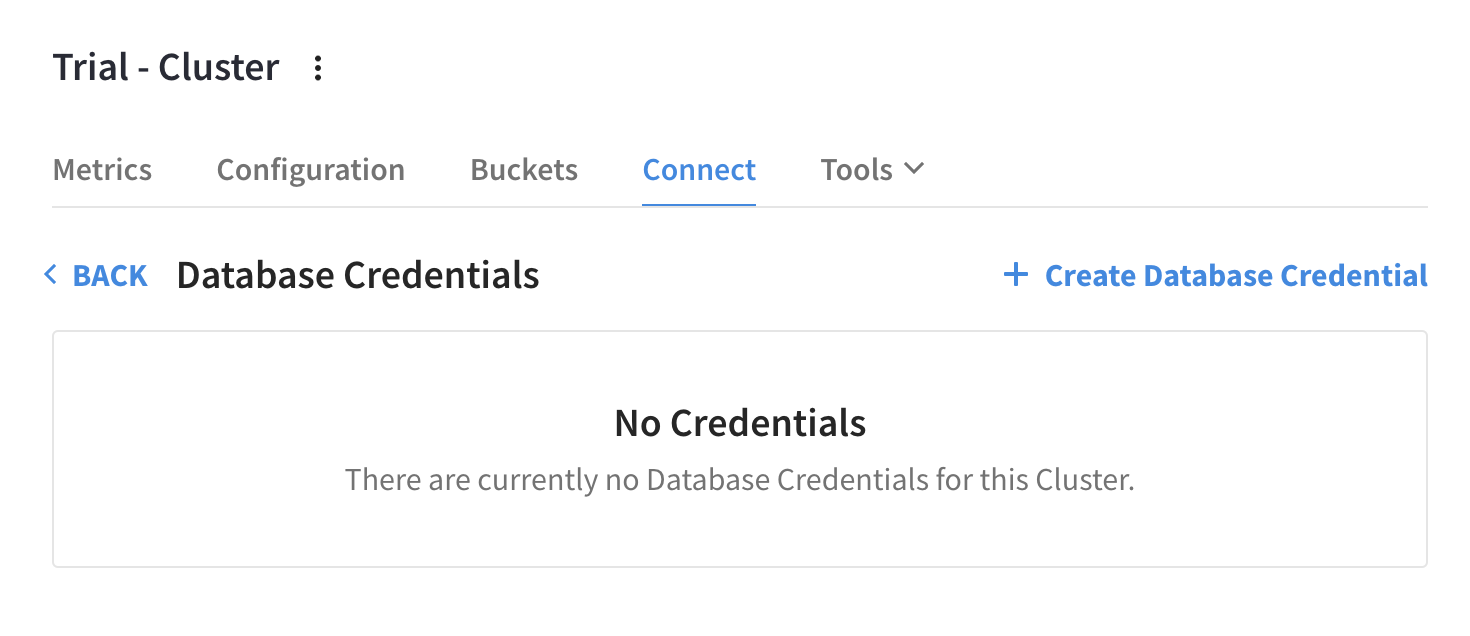
1. Go to the *Couchbase Capella Control Plane*
2. Select the cluster you want to connect with the SDK
3. Click on the *Connect* tab
4. Click the *Manage Allowed IPs* button
5. In the fly-out panel, paste your IP address from the previous step in the *IP Address or CIDR Block* field
6. Keep the *Save as Temporary* checkbox checked and set the number of hours
7. Click the *Add IP* button
8. Click the *Save* button

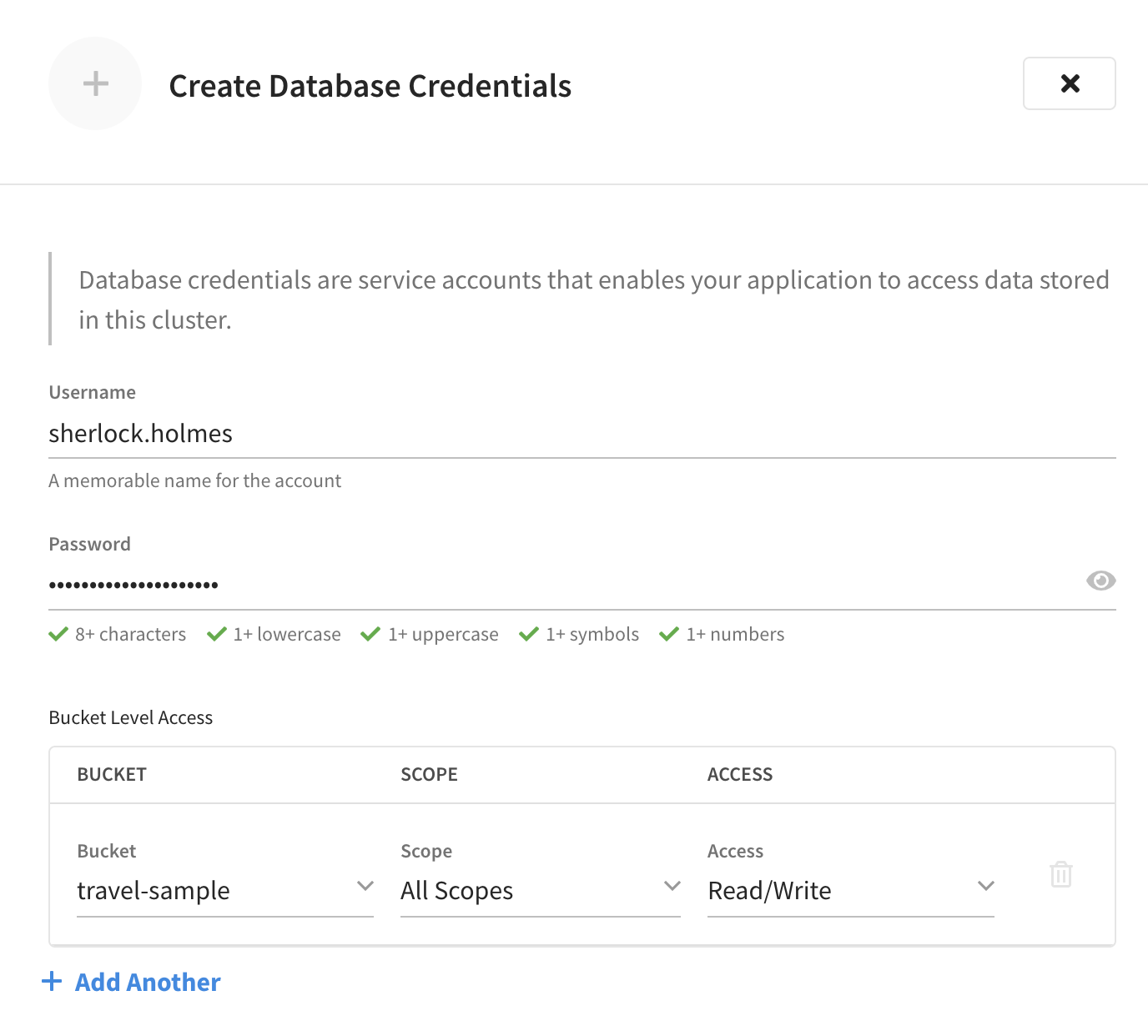
### 1c: Create database credentials:

In the Connect tab of the Couchbase Capella Control Plan, click on **Database Access -> Manage Credentials**

****

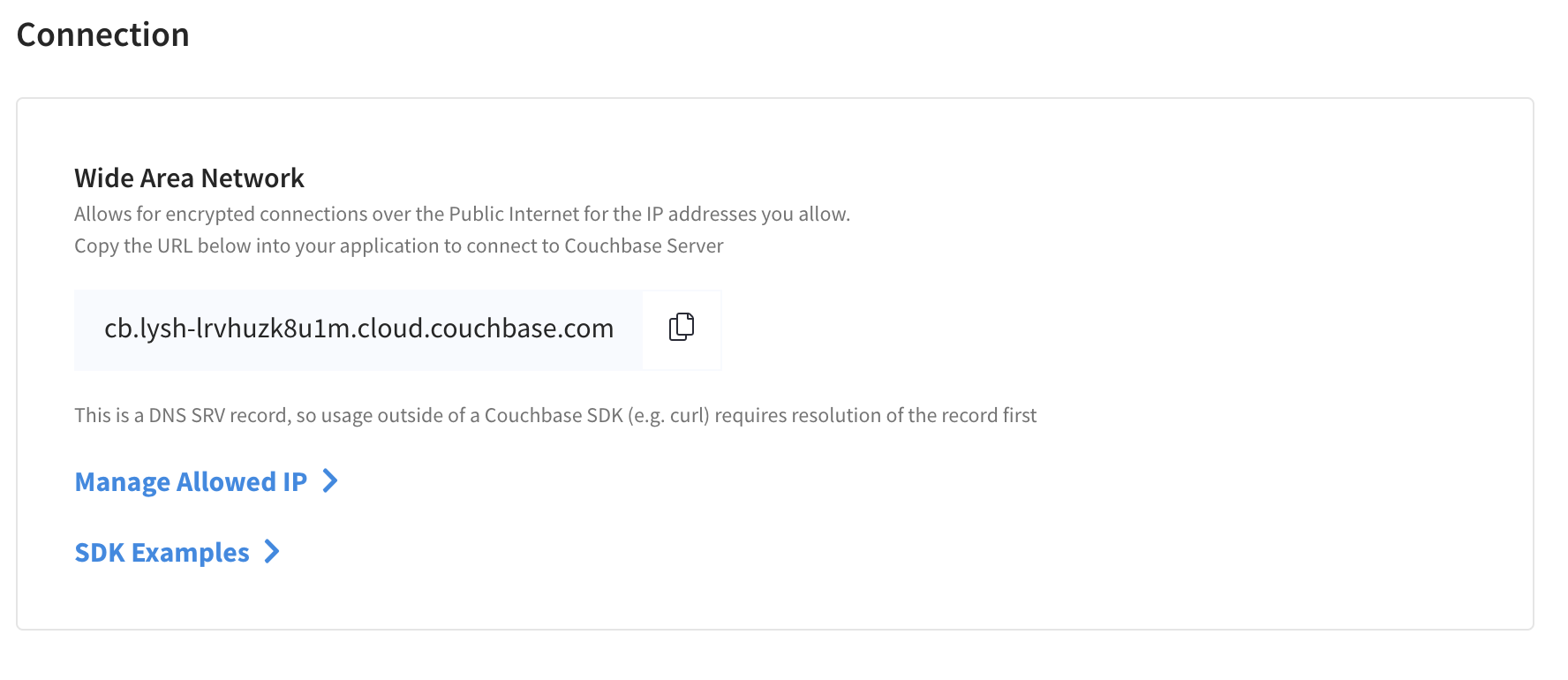
Create Database Credential for travel-sample bucket with read/write access

****

****

### 1d: Connection string

In the Connect tab of the Couchbase Capella Control Plane, the Wide Area Network field, copy the address. The SDK connection string should look like the following with the highlighted portion matching the participants specific address.

****

| couchbase://xxxxxx.dp.cloud.couchbase.com?ssl=no\_verify |
| --- |

### 1e: Update Python module

Add the connection logic to the *cbc\_workshop.py* module. The updated *cbc\_workshop.py* module can be seen below. See the SDK [documentation](https://docs.couchbase.com/python-sdk/current/howtos/managing-connections.html) for more details concerning connection management.

| 0:  1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22: | from couchbase.cluster import Cluster, ClusterOptions  from couchbase.auth import PasswordAuthenticator  if \_\_name\_\_ == '\_\_main\_\_':  try:  # Cluster specific information  cloud\_endpoint = \  'ef53717b-8698-4386-8af4-bcf1bdc87eca.dp.cloud.couchbase.com'  username = 'sherlock.holmes'  password = 'password'  bucket\_name = 'travel-sample'    # connect to cluster  conn\_str = 'couchbases://{}?ssl=no\_verify'.format(cloud\_endpoint)  cluster\_opts = ClusterOptions(PasswordAuthenticator(username,  password))  cluster = Cluster(conn\_str, cluster\_opts)  bucket = cluster.bucket(bucket\_name)  collection = bucket.default\_collection()    except Exception as ex:  import traceback  traceback.print\_exc() |
| --- | --- |

Notes about the code:

* Lines 5-9: Cluster information that is specific to a participant’s cluster:
  + Cloud endpoint
  + Username and password
  + Bucket name
* Line 11: Formatting the connection string. The secure couchbase**s** formatis used to connect to the cloud. The *ssl=no\_verify* is used in the connection string in order to skip SSL validation and this is **NOT** recommended for typical applications and only used for demonstration purposes.
* Line 13: Creating a cluster object. With the 3.x SDKs query, search and analytics is performed at the query level. Once Couchbase Server 7.0 is released performing various operations will be allowed at the scope level as well.
* Line 14: Creating a bucket object
* Line 15: Creating a collection object. The 3.x SDKs perform K/V operations at the collection level. Until Couchbase Server 7.0 is GA, the SDKs use a default collection object.
* Lines 17 - 18: try/except to show output if there is an error. The lab participant is free to add logic accordingly to test out various methods of handling errors.

## Step 2: CRUD (K/V Operations)

Add the K/V logic to the *cbc\_workshop.py* module. The updated *cbc\_workshop.py* module can be seen below. See the SDK documentation for more details concerning [Key Value operations](https://docs.couchbase.com/python-sdk/current/howtos/kv-operations.html) and [Sub-document operations](https://docs.couchbase.com/python-sdk/current/howtos/subdocument-operations.html).

| 0:  1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45:  46:  47:  48:  49:  50:  51:  52:  53:  54:  55:  56:  57:  58:  59:  60:  61:  62:  63:  64:  65:  66:  67:  68:  69:  70:  71:  72:  73:  74:  75:  76:  77: | from pprint import pprint  from couchbase.cluster import Cluster, ClusterOptions  from couchbase.auth import PasswordAuthenticator  import couchbase.subdocument as SD  if \_\_name\_\_ == '\_\_main\_\_':  try:  # Cluster specific information  cloud\_endpoint = \  'ef53717b-8698-4386-8af4-bcf1bdc87eca.dp.cloud.couchbase.com'  username = 'sherlock.holmes'  password = 'password'  bucket\_name = 'travel-sample'    # connect to cluster  conn\_str = 'couchbases://{}?ssl=no\_verify'.format(cloud\_endpoint)  cluster\_opts = ClusterOptions(PasswordAuthenticator(username,  password))  cluster = Cluster(conn\_str, cluster\_opts)  bucket = cluster.bucket(bucket\_name)  collection = bucket.default\_collection()  #KV operation data  test\_key = 'testDoc::0'  test\_doc = {  'type': 'testDoc',  'info': 'This is a test',  'address': {  'home': {  'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'state': 'WI',  'zipCode': '40243-4741',  'country': 'CA'  }  }  }  sd\_path = 'address.home.address1'  sd\_update = '8575 Lewis Springs Mountains Blvd'  #KV - insert  ins\_result = collection.insert(test\_key, test\_doc)  print('\nInserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - get  get\_result = collection.get(test\_key)  pprint(get\_result.content)  #KV - upsert  ins\_result = collection.upsert(test\_key, test\_doc)  print('\nUpserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - subdoc - mutate-in  mti\_result = collection.mutate\_in(test\_key,  [SD.upsert(sd\_path, sd\_update)])  print('Mutated sub-doc w/ key: {} and path: {}; CAS: {}\n'.format(test\_key,  sd\_path,  mti\_result.cas))  #KV - subdoc - lookup-in  lki\_result = collection.lookup\_in(test\_key,  [SD.get(sd\_path), SD.exists(sd\_path)])  print('Lookup in result, key: {}, path: {}, value: {}\n'.format(test\_key,  sd\_path,  lki\_result.content\_as[str](0)))  #KV - remove  rm\_result = collection.remove(test\_key)  print('Removed doc w/ key {}; CAS: {}\n'.format(test\_key,  rm\_result.cas))  except Exception as ex:  import traceback  traceback.print\_exc() |
| --- | --- |

Notes about the code:

* Line 0: importing the pretty print Python Module
* Line 4: importing the Couchbase sub-document module
* Lines 25-40: Test data
* Lines 43 - 45: K/V *insert* operation and CAS from result.
  + Required input: document key and document
* Lines 48 - 49: K/V *get* operation and document content from result.
  + Required input: document key
* Lines 52 - 54: K/V *upsert* operation
  + Required input: document key and document
* Lines 57 - 61: K/V sub-doc *mutate in* operation
  + Required input: document key, sub-document operation with path and value for operation
* Lines 64 - 68: K/V sub-document *lookup in* operation
  + Required input: document key, sub-document operation with path
* Lines 71 - 73: K/V *remove* operation
  + Required input: document key
* Lines 75 - 77: try/except to show output if there is an error. The lab participant is free to add logic accordingly to test out various methods of handling errors.

After saving *cbc\_workshop.py*, running the module should output the following:

| **$ python cbc\_workshop.py**    Inserted doc w/ key: testDoc::0; CAS: 1613764919809736704  {'address': {'home': {'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'country': 'CA',  'state': 'WI',  'zipCode': '40243-4741'}},  'info': 'This is a test',  'type': 'testDoc'}  Upserted doc w/ key: testDoc::0; CAS: 1613764919812227072  Mutated sub-doc w/ key: testDoc::0 and path: address.home.address1; CAS: 1613764919813210112  Lookup in result, key: testDoc::0, path: address.home.address1, value: 8575 Lewis Springs Mountains Blvd  Removed doc w/ key testDoc::0; CAS: 1613764919815241728 |
| --- |

## Step 3: N1QL Operations

### 

Add the N1QL logic to the *cbc\_workshop.py* module. The updated *cbc\_workshop.py* module can be seen below. See the SDK documentation for more details concerning [query operations](https://docs.couchbase.com/python-sdk/current/howtos/n1ql-queries-with-sdk.html).

| 0:  1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45:  46:  47:  48:  49:  50:  51:  52:  53:  54:  55:  56:  57:  58:  59:  60:  61:  62:  63:  64:  65:  66:  67:  68:  69:  70:  71:  72:  73:  74:  75:  76:  77:  78:  79:  80:  81:  82:  83:  84:  85:  86:  87:  88:  89:  90:  91:  92:  93:  94:  95:  96:  97:  98:  99:  100:  101:  102:  103:  104:  105:  106:  107:  108:  109:  110:  111:  112:  113:  114:  115:  116:  117:  118:  119:  120:  121:  122:  123:  124:  125:  126:  127:  128:  129:  130:  131:  132:  133:  134:  135: | from pprint import pprint  from couchbase.cluster import Cluster, ClusterOptions, QueryOptions  from couchbase.auth import PasswordAuthenticator  import couchbase.subdocument as SD  if \_\_name\_\_ == '\_\_main\_\_':  try:  # Cluster specific information  cloud\_endpoint = \  'ef53717b-8698-4386-8af4-bcf1bdc87eca.dp.cloud.couchbase.com'  username = 'sherlock.holmes'  password = 'password'  bucket\_name = 'travel-sample'    # connect to cluster  conn\_str = 'couchbases://{}?ssl=no\_verify'.format(cloud\_endpoint)  cluster\_opts = ClusterOptions(PasswordAuthenticator(username,  password))  cluster = Cluster(conn\_str, cluster\_opts)  bucket = cluster.bucket(bucket\_name)  collection = bucket.default\_collection()  #KV operation data  test\_key = 'testDoc::0'  test\_doc = {  'type': 'testDoc',  'info': 'This is a test',  'address': {  'home': {  'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'state': 'WI',  'zipCode': '40243-4741',  'country': 'CA'  }  }  }  sd\_path = 'address.home.address1'  sd\_update = '8575 Lewis Springs Mountains Blvd'  #KV - insert  ins\_result = collection.insert(test\_key, test\_doc)  print('\nInserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - get  get\_result = collection.get(test\_key)  pprint(get\_result.content)  #KV - upsert  ins\_result = collection.upsert(test\_key, test\_doc)  print('\nUpserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - subdoc - mutate-in  mti\_result = collection.mutate\_in(test\_key,  [SD.upsert(sd\_path, sd\_update)])  print('Mutated sub-doc w/ key: {} and path: {}; CAS: {}\n'.format(test\_key,  sd\_path,  mti\_result.cas))  #KV - subdoc - lookup-in  lki\_result = collection.lookup\_in(test\_key,  [SD.get(sd\_path), SD.exists(sd\_path)])  print('Lookup in result, key: {}, path: {}, value: {}\n'.format(test\_key,  sd\_path,  lki\_result.content\_as[str](0)))  #KV - remove  rm\_result = collection.remove(test\_key)  print('Removed doc w/ key {}; CAS: {}\n'.format(test\_key,  rm\_result.cas))    #basic query w/ GROUP BY  query\_str = '''  SELECT  t.type AS DocType,  COUNT(1) AS DocCount  FROM `{}` t  GROUP BY t.type  '''.format(bucket\_name)  q\_result = cluster.query(query\_str)  for r in q\_result.rows():  pprint(r)  print()    #sub-query example  query\_str = '''  SELECT  name,  country,  (SELECT raw avg(s.ratings.Overall)  FROM t.reviews as s)[0] AS overall\_avg\_rating  FROM `{}` AS t  WHERE  t.type = "hotel"  ORDER BY  overall\_avg\_rating DESC  LIMIT 10;  '''.format(bucket\_name)  q\_result = cluster.query(query\_str)  for r in q\_result.rows():  pprint(r)  print()  #positional query params  query\_str = '''  SELECT COUNT(1) AS DocCount  FROM `{}` t  WHERE t.type=$1  '''.format(bucket\_name)  query\_opts = QueryOptions(positional\_parameters=['hotel'])  q\_result = cluster.query(query\_str, query\_opts)  for r in q\_result.rows():  pprint(r)  print()  #named query params  query\_str = '''  SELECT COUNT(1) AS DocCount  FROM `{}` t  WHERE t.type=$doc\_type  '''.format(bucket\_name)  query\_opts = QueryOptions(named\_parameters={'doc\_type':'airline'})  q\_result = cluster.query(query\_str, query\_opts)  for r in q\_result.rows():  pprint(r)  print()  except Exception as ex:  import traceback  traceback.print\_exc() |
| --- | --- |

Notes about the code:

* Line 2: importing the Couchbase QueryOptions class
* Lines 76 - 82: Simple query with GROUP BY clause
* Line 83: Executing query operation on cluster
* Lines 84 - 85: Output query results row-by-row
* Lines 89 - 101: Query with sub-query
* Line 102: Executing query operation on cluster
* Lines 103 - 104: Output query results row-by-row
* Lines 108 - 112: Simple query using positional query parameters
  + Note the lack of quotes in the query string
* Line 113: Creating QueryOptions object to have name parameters
* Line 114: Executing query operation on cluster and providing position parameter query options
* Lines 115 - 116: Output query results row-by-row
* Lines 120 - 124: Simple query using named query parameters
  + Note the lack of quotes in the query string
* Line 125: Creating QueryOptions object to have name parameters
* Line 126: Executing query operation on cluster and providing named parameter query options
* Lines 127 - 128: Output query results row-by-row

After saving *cbc\_workshop.py*, running the module should output the following:

| **$ python cbc\_workshop.py**    Inserted doc w/ key: testDoc::0; CAS: 1613777143116070912  {'address': {'home': {'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'country': 'CA',  'state': 'WI',  'zipCode': '40243-4741'}},  'info': 'This is a test',  'type': 'testDoc'}  Upserted doc w/ key: testDoc::0; CAS: 1613777143118495744  Mutated sub-doc w/ key: testDoc::0 and path: address.home.address1; CAS: 1613777143119544320  Lookup in result, key: testDoc::0, path: address.home.address1, value: 8575 Lewis Springs Mountains Blvd  Removed doc w/ key testDoc::0; CAS: 1613777143121772544  {'DocCount': 24024, 'DocType': 'route'}  {'DocCount': 187, 'DocType': 'airline'}  {'DocCount': 1968, 'DocType': 'airport'}  {'DocCount': 917, 'DocType': 'hotel'}  {'DocCount': 4495, 'DocType': 'landmark'}  {'country': 'United Kingdom',  'name': 'Park Plaza County Hall',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'La Reserve Hotel Chelsea',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'Marriott London County Hall',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom', 'name': 'Abbey Hotel', 'overall\_avg\_rating': 5}  {'country': 'France', 'name': 'Avignon Hotel Monclar', 'overall\_avg\_rating': 5}  {'country': 'France', 'name': 'La Pradella', 'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'Culloden House Hotel',  'overall\_avg\_rating': 5}  {'country': 'France',  'name': 'Auberge-Camping Bagatelle',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': '8 Clarendon Crescent',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom', 'name': 'The Bulls Head', 'overall\_avg\_rating': 5}  {'DocCount': 917}  {'DocCount': 187} |
| --- |

## Step 4: FTS Operations

### 

Add the Full Text Search (FTS) logic to the *cbc\_workshop.py* module. The updated *cbc\_workshop.py* module can be seen below. See the SDK documentation for more details concerning [full text search operations](https://docs.couchbase.com/python-sdk/current/howtos/full-text-searching-with-sdk.html).

| 0:  1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29:  30:  31:  32:  33:  34:  35:  36:  37:  38:  39:  40:  41:  42:  43:  44:  45:  46:  47:  48:  49:  50:  51:  52:  53:  54:  55:  56:  57:  58:  59:  60:  61:  62:  63:  64:  65:  66:  67:  68:  69:  70:  71:  72:  73:  74:  75:  76:  77:  78:  79:  80:  81:  82:  83:  84:  85:  86:  87:  88:  89:  90:  91:  92:  93:  94:  95:  96:  97:  98:  99:  100:  101:  102:  103:  104:  105:  106:  107:  108:  109:  110:  111:  112:  113:  114:  115:  116:  117:  118:  119:  120:  121:  122:  123:  124:  125:  126:  127:  128:  129:  130:  131:  132:  133:  134:  135:  136:  137:  138:  139:  140:  141:  142:  143:  144:  145:  146:  147:  148:  149:  150: | from pprint import pprint  from couchbase.cluster import Cluster, ClusterOptions, QueryOptions  from couchbase.auth import PasswordAuthenticator  import couchbase.subdocument as SD  from couchbase.search import TermQuery, SearchOptions  if \_\_name\_\_ == '\_\_main\_\_':  try:  # Cluster specific information  cloud\_endpoint = \  'ef53717b-8698-4386-8af4-bcf1bdc87eca.dp.cloud.couchbase.com'  username = 'sherlock.holmes'  password = 'password'  bucket\_name = 'travel-sample'    # connect to cluster  conn\_str = 'couchbases://{}?ssl=no\_verify'.format(cloud\_endpoint)  cluster\_opts = ClusterOptions(PasswordAuthenticator(username,  password))  cluster = Cluster(conn\_str, cluster\_opts)  bucket = cluster.bucket(bucket\_name)  collection = bucket.default\_collection()  #KV operation data  test\_key = 'testDoc::0'  test\_doc = {  'type': 'testDoc',  'info': 'This is a test',  'address': {  'home': {  'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'state': 'WI',  'zipCode': '40243-4741',  'country': 'CA'  }  }  }  sd\_path = 'address.home.address1'  sd\_update = '8575 Lewis Springs Mountains Blvd'  #KV - insert  ins\_result = collection.insert(test\_key, test\_doc)  print('\nInserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - get  get\_result = collection.get(test\_key)  pprint(get\_result.content)  #KV - upsert  ins\_result = collection.upsert(test\_key, test\_doc)  print('\nUpserted doc w/ key: {}; CAS: {}\n'.format(test\_key,  ins\_result.cas))  #KV - subdoc - mutate-in  mti\_result = collection.mutate\_in(test\_key,  [SD.upsert(sd\_path, sd\_update)])  print('Mutated sub-doc w/ key: {} and path: {}; CAS: {}\n'.format(test\_key,  sd\_path,  mti\_result.cas))  #KV - subdoc - lookup-in  lki\_result = collection.lookup\_in(test\_key,  [SD.get(sd\_path), SD.exists(sd\_path)])  print('Lookup in result, key: {}, path: {}, value: {}\n'.format(test\_key,  sd\_path,  lki\_result.content\_as[str](0)))  #KV - remove  rm\_result = collection.remove(test\_key)  print('Removed doc w/ key {}; CAS: {}\n'.format(test\_key,  rm\_result.cas))    #basic query w/ GROUP BY  query\_str = '''  SELECT  t.type AS DocType,  COUNT(1) AS DocCount  FROM `{}` t  GROUP BY t.type  '''.format(bucket\_name)  q\_result = cluster.query(query\_str)  for r in q\_result.rows():  pprint(r)  print()    #sub-query example  query\_str = '''  SELECT  name,  country,  (SELECT raw avg(s.ratings.Overall)  FROM t.reviews as s)[0] AS overall\_avg\_rating  FROM `{}` AS t  WHERE  t.type = "hotel"  ORDER BY  overall\_avg\_rating DESC  LIMIT 10;  '''.format(bucket\_name)  q\_result = cluster.query(query\_str)  for r in q\_result.rows():  pprint(r)  print()  #positional query params  query\_str = '''  SELECT COUNT(1) AS DocCount  FROM `{}` t  WHERE t.type=$1  '''.format(bucket\_name)  query\_opts = QueryOptions(positional\_parameters=['hotel'])  q\_result = cluster.query(query\_str, query\_opts)  for r in q\_result.rows():  pprint(r)  print()  #named query params  query\_str = '''  SELECT COUNT(1) AS DocCount  FROM `{}` t  WHERE t.type=$doc\_type  '''.format(bucket\_name)  query\_opts = QueryOptions(named\_parameters={'doc\_type':'airline'})  q\_result = cluster.query(query\_str, query\_opts)  for r in q\_result.rows():  pprint(r)  print()  #FTS TermQuery data  search\_term = 'london'  fuzzy = 1  #FTS operations  result = cluster.search\_query('default',  TermQuery(term=search\_term, fuzziness=fuzzy),  SearchOptions(limit=100))  doc\_ids = list(map(lambda r: r.id, result))  docs\_found = result.metadata().metrics.total\_rows  print(  '\nFound {} docs that matched search for {} with fuzziness of {}\n'  .format(docs\_found, search\_term, fuzzy))  if len(doc\_ids) > 0:  sample\_doc = collection.get(doc\_ids[0])  pprint(sample\_doc.content)  except Exception as ex:  import traceback  traceback.print\_exc() |
| --- | --- |

Notes about the code:

* Line 5: importing the Couchbase FTS TermQuery and SearchOptions class
* Lines 136 - 138: Executing an FTS search on the cluster
  + Required input: name of the index to use (index must exist on Couchbase Server), type of FTS query. See SDK documentation for further details on types of FTS queries available
  + Optional input: SearchOptions, see SDK documentation for further details on available options
* Line 139: Gathering the document keys from the results
* Line 140: Gathering information about results found. This will not work until all results have been gathered and is one of the reasons for gather doc\_ids in line 139
* Line 144 - 146: If documents are found, output a single document after retrieving via the K/V *get* operation

After saving *cbc\_workshop.py*, running the module should output the following:

| **$ python cbc\_workshop.py**    Inserted doc w/ key: testDoc::0; CAS: 1613777143116070912  {'address': {'home': {'address1': '8575 Lewis Springs Mountains',  'city': 'West Mohammed',  'country': 'CA',  'state': 'WI',  'zipCode': '40243-4741'}},  'info': 'This is a test',  'type': 'testDoc'}  Upserted doc w/ key: testDoc::0; CAS: 1613777143118495744  Mutated sub-doc w/ key: testDoc::0 and path: address.home.address1; CAS: 1613777143119544320  Lookup in result, key: testDoc::0, path: address.home.address1, value: 8575 Lewis Springs Mountains Blvd  Removed doc w/ key testDoc::0; CAS: 1613777143121772544  {'DocCount': 24024, 'DocType': 'route'}  {'DocCount': 187, 'DocType': 'airline'}  {'DocCount': 1968, 'DocType': 'airport'}  {'DocCount': 917, 'DocType': 'hotel'}  {'DocCount': 4495, 'DocType': 'landmark'}  {'country': 'United Kingdom',  'name': 'Park Plaza County Hall',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'La Reserve Hotel Chelsea',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'Marriott London County Hall',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom', 'name': 'Abbey Hotel', 'overall\_avg\_rating': 5}  {'country': 'France', 'name': 'Avignon Hotel Monclar', 'overall\_avg\_rating': 5}  {'country': 'France', 'name': 'La Pradella', 'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': 'Culloden House Hotel',  'overall\_avg\_rating': 5}  {'country': 'France',  'name': 'Auberge-Camping Bagatelle',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom',  'name': '8 Clarendon Crescent',  'overall\_avg\_rating': 5}  {'country': 'United Kingdom', 'name': 'The Bulls Head', 'overall\_avg\_rating': 5}  {'DocCount': 917}  {'DocCount': 187}  Found 809 docs that matched search for london with fuzziness of 1  {'activity': 'see',  'address': None,  'alt': None,  'city': 'London',  'content': 'Remains of the wall that surrounded the City of London for almost '  'two thousand years. The parts around the Barbican are mostly '  'Tudor due to maintenance (Roman remains can be seen in and around '  'the Tower of London). Other local remains are the',  'country': 'United Kingdom',  'directions': 'Near the street called &quot;London Wall&quot;',  'email': None,  'geo': {'accuracy': 'RANGE\_INTERPOLATED', 'lat': 51.5177, 'lon': -0.0952},  'hours': None,  'id': 37138,  'image': 'https://en.wikivoyage.org/wiki/File:London Wall bastion, Barber '  "Surgeons' Hall Gardens, London EC2.jpg",  'image\_direct\_url': '',  'name': 'London Wall',  'phone': None,  'price': None,  'state': None,  'title': 'Wikimania 2014 London Guidebook',  'tollfree': None,  'type': 'landmark',  'url': None} |
| --- |

### 

### 