**Couchbase RAG with Eventing**

Jason Cao, Solution Engineer

Version 1, **June 2024**

This demo tries to showcase Couchbase’s capability as a general-purpose data platform including vector search. We’ll be building a chatbot application that allows you to upload any raw JSON data into Couchbase and run RAG, as well as the many scenarios common in building an enterprise chatbot application.

At a high level, this is what the demo consists of:

1. Eventing is leveraged to automate many of the AI data pipeline process including:
   1. Data cleansing (using Javascript function)
   2. Sensitive data masking (using LLM)
   3. Object labeling (using LLM)
   4. Re-arrange objects to destination collection (Eventing)
   5. Automatic Embedding
2. Run chatbot with the RAG-ready data
3. Couchbase as a multi-model, SQL friendly db (and Capella IQ if you’re running on Capella)

Naturally, this demo can be run on both Capella and Servers. Run either section below according to your preferences. Here are my thoughts:

|  |  |  |
| --- | --- | --- |
|  | Capella | Server |
| Setup | Eventing and FTS index creation is manual. | Creation of Bucket/scope/collection, Eventing functions, and FTS index is automated. |
| Demo Wow-factor | Capella IQ is included. | No Capella IQ. |
| Customer share-ability | Only if customer has active Capella subscription | Can be shared to any customers and prospect if they agreed to use Server only for testing workloads such as this demo |

# SETUP

## LLM

GPT-4o is used in this demo. You need to have an OPENAI\_API\_KEY from OpenAI.

## AWS VM

* In your AWS account, create a EC2 instance and run the following startup script:

#!/bin/bash

sudo yum update -y

sudo yum install git -y

sudo yum install python3 -y

sudo yum install python3-pip -y

git clone https://github.com/sillyjason/chatbot-with-couchbase

cd chatbot-with-couchbase

python3 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

cat > .env <<- EOF

#EE Environment Variables

EE\_HOSTNAME=ec2-13-213-39-81.ap-southeast-1.compute.amazonaws.com

EVENTING\_HOSTNAME=ec2-13-213-39-81.ap-southeast-1.compute.amazonaws.com

SEARCH\_HOSTNAME=ec2-13-213-39-81.ap-southeast-1.compute.amazonaws.com

#Capella Environment Variables

CB\_HOSTNAME=cb.tp6sl4zqsvdh02la.cloud.couchbase.com

CAPELLA\_API\_KEY\_TOKEN=QWdpV2tldmxQa05Rc0NMVTY0dWtzRUYzaXpKZ09VdDM6U0VuNGVUQTVZbjlMRGghVTBjSTVwU29FQlpOU2xTeGs0MmdySFc3bnhVd2hJZWxCa3RkMTBOSkNoMHJoZyNnbA==

ORG\_ID=42730eb3-53ab-451a-b5eb-8eeb9a92084c

PROJECT\_ID=4af36ee8-eb11-45ee-bcde-845bc07f1ba3

CLUSTER\_ID=2697449c-4727-46a4-8e74-9cfc80e0972c

#Chatbot Endpoint

CHATBOT\_APP\_END\_POINT=ec2-18-139-115-48.ap-southeast-1.compute.amazonaws.com

#CB User Credential

CB\_USERNAME=admin

CB\_PASSWORD=C0uchbase123!

#LLM Keys

OPENAI\_API\_KEY=sk-proj-iWYWRTPUAHnTskOu7F0LT3BlbkFJCCaAk52SsB7XIvFLKPr6

EOF

* Make sure to update Security Group setting to allow ALL inbound/outbound traffics
* After the instance is created, note down its Public IPv4 DNS. It should look like this: ec2-18-139-115-48.ap-southeast-1.compute.amazonaws.com

## Capella Cluster Setup

\*If you intend to run the demo with Server, you can skip section 1.3 entirely.

### Cluster Creation

On your Capella account, spin up a **7.6** cluster with these services: Data, Index, Query, Search, Eventing.

Basic machines are fine. Service Group setup is below:

A screenshot of a computer

Description automatically generated

### Whitelist IP and User Credential Creation

Whitelist all IP.

Create db access credentials. Note down the credential for later use.

### Create Bucket Structures

Create the following bucket/scope/collection structure:

|  |  |  |
| --- | --- | --- |
| Bucket | Scope | Collection |
| meta   * 1000MB ram | \_default | \_default |
| main   * rest of the RAM * Enable flushing | raw | raw |
| formatted |
| data | products |
| policies |
| chats | bot |
| human |

### Create Eventing Functions

Download the 3 Eventing files below from this [Github repo](https://github.com/sillyjason/chatbot-with-couchbase) at this directory: templates/assets/eventing:

* embedding.json
* metadata\_labelling.json
* reformatting.json

In your Capella cluster, upload these 3 files to create the functions.

At step 2 of creation, find the URL bindings, replace the AWS hostname with the DNS record of the AWS VM instance we created in step 1.2. Now the binding should look like this: <http://ec2-13-215-49-139.ap-southeast-1.compute.amazonaws.com:5000>.

No need to update the Javascript function.

A screenshot of a computer

Description automatically generated

\*Repeat the DNS replacement action with ALL 3 functions. Then, deploy all functions.

### Create FTS Index

* Go to Search tab. Create FTS index.
* Then go to “Advanced Mode” where there is the “Import from File” option

A screenshot of a computer

Description automatically generated

* Download the JSON file from the repo directory /assets/fts-index.json, then upload here. You might need to manually select bucket and scope like below. (make sure you’re on version 7.6 otherwise it won’t work)

A screenshot of a computer

Description automatically generated

## Server Setup

\*If you intend to run the demo on Capella, you can skip section 1.4 entirely.

* On AWS account, spin up a 3-node CB cluster with at least these services enabled: Data, Query, Index, Search, Eventing. Make sure the Server version is **EE 7.6**
* Deploy all services on all 3 nodes. For instance size, I used t3.2xlarge and it worked fine. Now your cluster should look like this:

A close-up of a number

Description automatically generated

* The creation of bucket/scope/collections, Eventing functions, and FTS indexs can all be automated with REST calls with Server. So we’re done here!

## Get, Set, Run!

### SSH

* SSH into the VM instance you created in 1.2
* Run “sudo -i” to switch to root user; run “cd /” to go to root folder
* You should have already downloaded the repo and installed dependencies from the startup script in 1.2. Now “cd” into the app folder and activate the python virtual environment “venv”. For details see screen below for commands

A screenshot of a computer program

Description automatically generated

### .ENV Setup

At SSH session, run “nano .env”

The environment variables needed are listed here. Here’s how-to:

-------------------- .env setup begin --------------------

**\#EE Environment Variables** (only needed when you’re running with EE)

EE\_HOSTNAME=The EE node with **Data** service enabled. In our setup where all services are deployed on all machines, any node will do

EVENTING\_HOSTNAME= The EE node with **Eventing** service enabled. In our setup where all services are deployed on all machines, any node will do

SEARCH\_HOSTNAME= The EE node with **Search** service enabled. In our setup where all services are deployed on all machines, any node will do

**#Capella Environment Variables** (only needed when you’re running on Capela)

CB\_HOSTNAME=Capella connection string. It should look like this: cb.xxxxx.cloud.couchbase.com (remove the leading “couchbases://”)

CAPELLA\_API\_KEY\_TOKEN=[optional] your Capella Token. Used for automatic scope creation

ORG\_ID=[optional] Your Capella ORG\_ID. Used for automatic scope creation

PROJECT\_ID=[optional] Your Capella PROJECT\_ID. Used for automatic scope creation

CLUSTER\_ID=[optional] Your Capella CLUSTER\_ID. Used for automatic scope creation

**#Chatbot Endpoint** (required)

CHATBOT\_APP\_END\_POINT=the hostname of VM you setup in 1.2

**#CB User Credential** (required)

CB\_USERNAME=username you used for Capella or EE

CB\_PASSWORD= password you used for Capella or EE

**#LLM Keys** (required)

OPENAI\_API\_KEY=your openai api key

-------------------- .env setup end --------------------

### Additional Setup

* Bucket/scope/collection can be automated by running “python3 setupbuckets.py” for both EE and Capella but more convoluted for Capella
  + since for Capella you’ll need your ORG\_ID, PROJECT\_ID, CLUSTER\_ID and CAPELLA\_API\_KEY\_TOKEN. In our case manual creation is easier which we already did earlier. So you can skip this one
  + For Server, this script does it for you. You’ll see messages on success creations
* Regardless of Capella or EE, run “python3 setupothers.py”.
  + On EE this will create the Eventing functions and FTS index for you automatically.
  + For both Capella and EE, this script will also update the endpoint address for accessing the chatbot app.
  + For Capella, look *for “Endpoint updated successfully in index.html”* in the logs and you’re fine. Ignore the rest of the stuff; for EE, you should see both this message and success message of eventing/FTS index creation.
* All good. Run “python3 app.py --capella” (for Capella) or “python3 app.py” (for EE). If you’see this, we’re good to go!

A screen shot of a computer

Description automatically generated

# DEMO

## Demo AI Data Pipeline

We’ll use a series of Eventing functions to simplify the AI data pipeline, which needless to say, is a big challenge for any company looking to build a GenAI application, considering the enormity and complexity of their unstructured data source. Here is what happens with CB

A screenshot of a diagram

Description automatically generated

### Import Data

Download the “raw-data.json” file under directory templates/assets/. At “Import” tab under Data Tools, import the file into main.raw.raw collection.

A screenshot of a computer

Description automatically generated

It’s worth pointing out the steps this raw data must go through before it’s RAG ready:

* Data format inconsistency. For example, “last\_upddate” field
* Empty spaces. For example, “content” field
* Data of different nature (2 product JSON and 1 email regarding internal policy JSON)
* Sensitive data. For example, ID or email in “content” field

After import, go to main.raw.raw collection to check data successfully imported;

Then, go to main.raw.formatted collection to see data after first processing

* You’ll notice the format inconsistency, empty space, and sensitive data issues are taken care of (\*note that in this demo, the sensitive data masking is done by prompting OPENAI with a API call. In production this defeats the purpose of data protection and instead, the model should be deployed locally hence local inferencing, a totally viable approach, just not demoed in this version…yet)
* Also note, every doc is added a “type” field which is an enumeration of [“internal\_policies”, “insurance\_product”]

Go to scope “data”. The 3 raw JSON objects are now pigeonholed into corresponding collections, automatically. Also note embedding is added.

Now let’s do some chatbot’ing

## Chatbot

Note the hostname of VM deployed in 1.2. Access this link in your browser and voila. {your\_hostname}:5000

A screenshot of a chatbot

Description automatically generated

Take a look at the raw data, and feel free to ask any questions.

Also note there are elements of user engagements. Feel free to give ratings to the comments.

At this moment we’ve demoed how Couchbase acts as both vector store and AI data pipeline orchestration platform. But we can demo more.

## Maintenance, Query, etc.

\*This is when we showcase Couchbase’s power as a general-purpose data store. Feel free to continue your talk track with any convincing SQL scenarios, and here is my version:

### NoSQL Flexibility

What if in quick iteration, customer need to update their schema? In our example, collection main.chats.human has the field “user\_id”, but not main.chats.bot. If it’s decided that the field be added to the ‘bot’ collection too, just to go Query tab and run the following:

SET b.user\_id = (SELECT RAW d.user\_id FROM human AS d WHERE meta().id = b.user\_msg\_id)[0]

WHERE b.user\_msg\_id IS NOT NULL;

\*The query might fail with “index not created” message. In this case, follow Index Advisor to create index and re-run the query

Easy. This is good time to explain the superiority of NoSQL when fast iteration is required.

### Capella IQ + SQL JOIN

If customer choose to have a separate vector database, they’ll end up storing vectors, transactions, and SQL docs in separate datastore. Imagine running query that needs joining these disconnected tables.

With Couchbase, easy.

What makes it even easier, if Capella IQ.

Go to Query, turn on IQ, and enter this prompt:

Show the most returned products in "product\_ids" array field and their corresponding frequencies. Also add product name to the results. the query should be joined by collection "main"."data"."products" where the document id matches the record. fetch the "product\_name" field

Let the system do the magic for you.

\*We all know, any LLM can be fickle. It’s completely normal if Capella IQ tells you “he doesn’t know”. Don’t panic, he usually works at second nudge.

At this moment, hopefully we’ve delivered a strong point that Couchbase serves much more than a vector DB.

# Tailor the demo with your own raw-data for any customer/industry:

This demo was built to facilitate this possibility but extra steps are needed.

Reach out to @jason.cao on Slack and I’ll be happy to walk you through!