Scalable Identity Resolution & Audience Segmentation Engine

Data Processing: PySpark, HDFS, MapReduce

Storage & Querying: SingleStore DB

Vector Search: Weaviate, Faiss, or Pinecone

Streaming & Real-Time Processing: Apache Kafka

Step 1: Data Ingestion & Standardization

This system ingests raw customer data from multiple sources, cleans and standardizes it, resolves identities across datasets, and assigns users to dynamic audience segments. The final processed data enables real-time targeting and analytics for advertising platforms. Use MapReduce to standardize data formats (normalize emails, phone numbers, remove duplicates). Store raw data in HDFS for distributed storage.

Step 2: Identity Resolution & Deduplication

- Convert user attributes into vector embeddings using BERT, fastText, or Sentence Transformers.
- Store embeddings in Faiss or Weaviate (VectorDB) for fast similarity matching.
- Use MapReduce to detect duplicate profiles and merge identities.
- Store resolved user identities in Single Store DB for real-time querying.

Step 3: Audience Segmentation using Behavioral Data

- Process real-time user events (page views, ad clicks, purchases) using Kafka.
- Convert behavioral patterns into vector embeddings.

- Use MapReduce to assign users to predefined audience segments based on similarity scores.
- Store audience segments in Single Store DB for immediate retrieval.

Step 4: Real-Time Querying & Activation

- Advertisers can query the system in real-time to:

  • Find matching audiences (e.g., "users who purchased luxury products in the last 30 days").

  •Look up resolved customer profiles (deterministic/probabilistic matching).

  •Activate segments for ad campaigns.

Step 1: Setting up environment:

Check Python & Installed Libraries-

```
manasb@Mac hadoop % python3 --version
pip list | grep -E 'faiss|singlestoredb|fastapi'

Python 3.12.1
faiss-cpu                 1.10.0
fastapi                   0.115.11
singlestoredb             1.12.0

[notice] A new release of pip is available: 24.0 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
manasb@Mac hadoop %
```

Step 2: Data Cleaning & Processing

Run Data Standardization

```
manasb@Mac identity_resolution_project % python3 data_standardization.py

✅ Data cleaned and saved as cleaned_data.csv
manasb@Mac identity_resolution_project %
```

Check Output Data

```
manasb@Mac identity_resolution_project % head cleaned_data.csv

user_id,email,ip_address,device_id,ad_clicks,purchases
8538,jefferytaylor@wallace.com,146.121.98.166,49316,4,4
1489,allenkelsey@gmail.com,197.85.241.171,42743,43,6
9470,fquinn@hotmail.com,28.31.144.231,2898487,35,0
2847,douglasellison@gmail.com,82.193.210.40,31,13,6
9555,steven93@gmail.com,165.194.248.164,029,11,3
5583,deborah26@wang-gray.com,217.15.136.133,08791,10,1
5171,uwillis@hotmail.com,46.181.173.169,96689,49,6
2955,marytaylor@hotmail.com,182.41.162.209,6270513,34,8
411,lorioconnor@conrad.com,139.41.151.8,805190,41,3
manasb@Mac identity_resolution_project %
```

Step 3: Faiss Indexing (Vector Search)

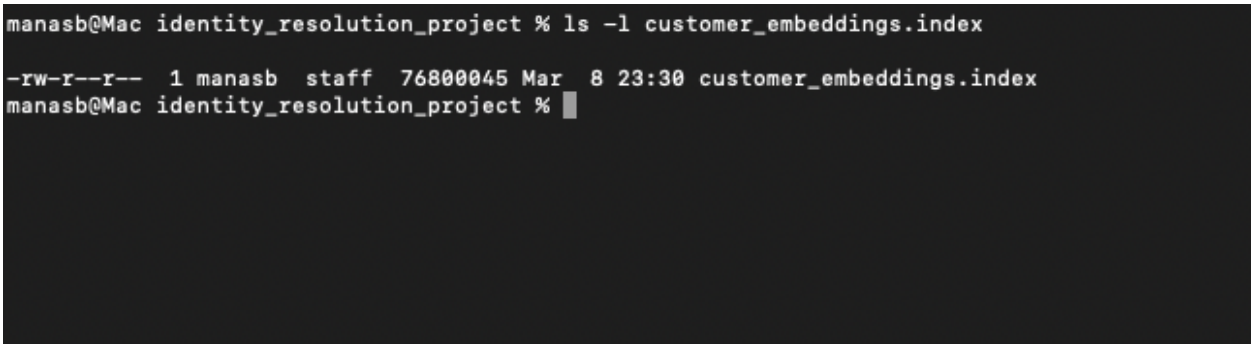
Run Faiss Indexing

```
manasb@Mac identity_resolution_project % python3 faiss_indexing.py

manasb@Mac identity_resolution_project %
```

Confirms vector embeddings are created and stored in Faiss.

Verify Faiss Index File

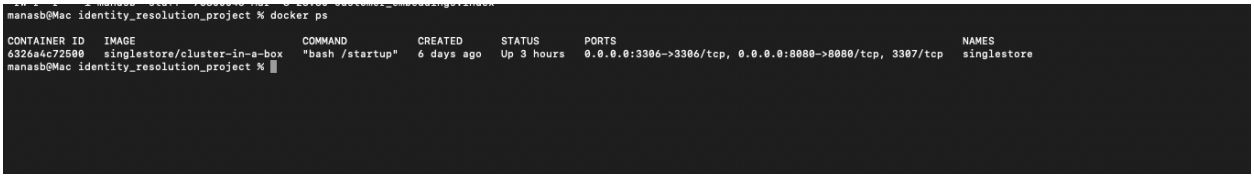


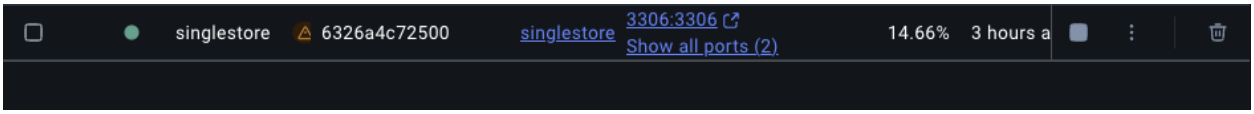
Ensures Faiss index file exists.

Show terminal running python faiss_indexing.py + output of ls -l customer_embeddings.index.

Step 4: Setting Up SingleStore DB

Start SingleStoreDB (Docker)

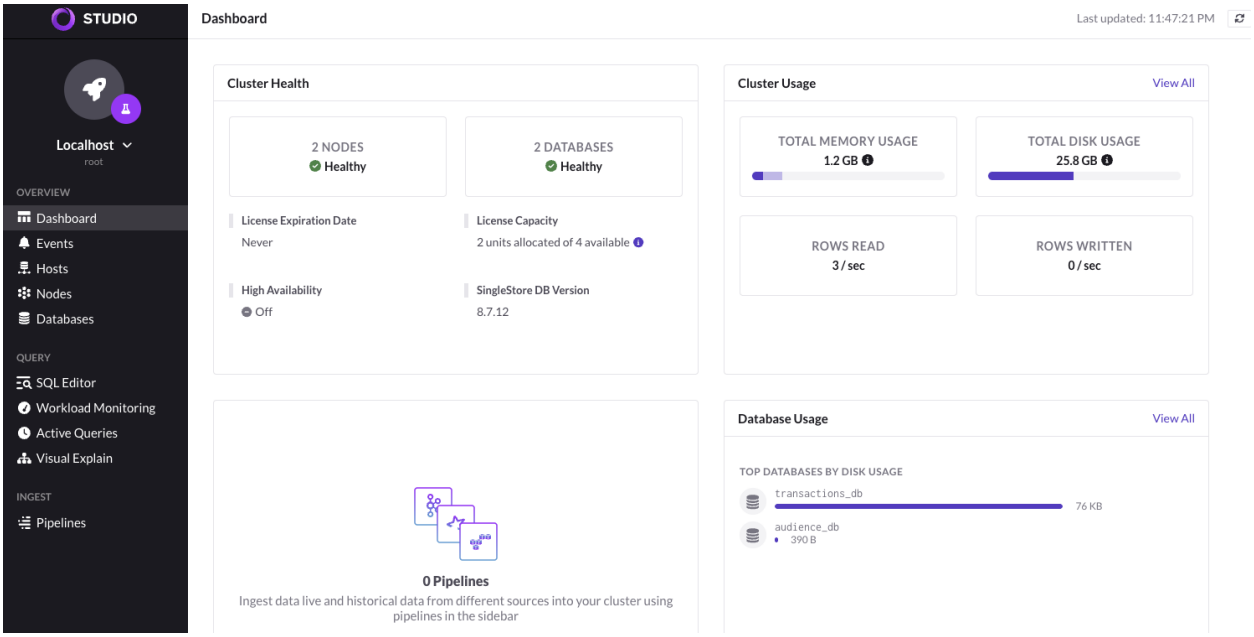


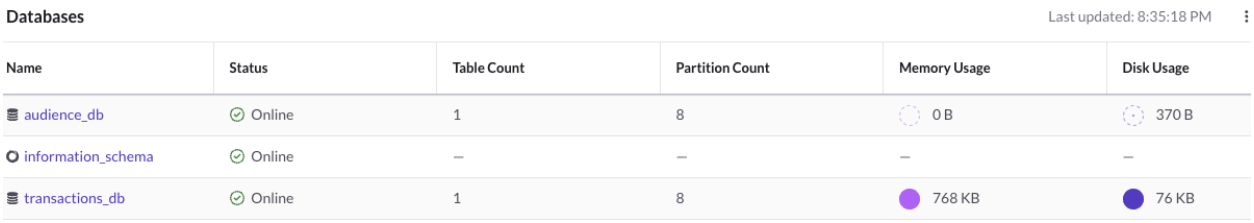


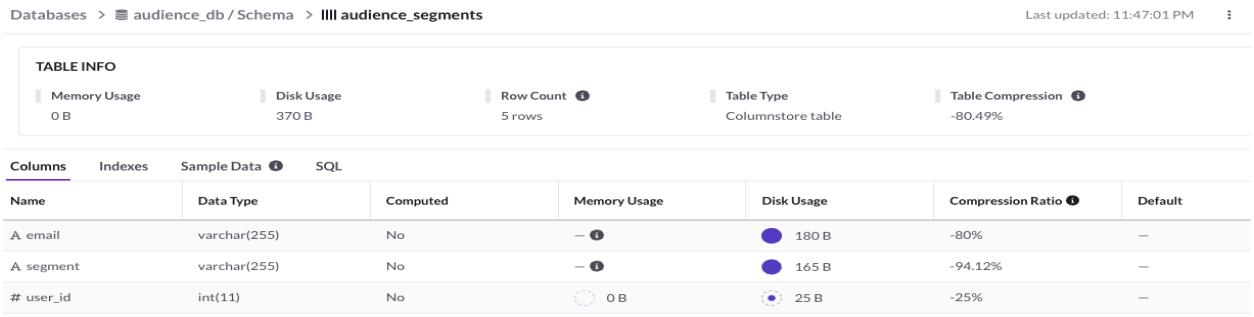
Create & Verify Table

```
mysql> SELECT * FROM audience_segments;
+---------+-------------------+---------------+
| user_id | email             | segment       |
+---------+-------------------+---------------+
|    8995 | john15@gmail.com | Similar Users |
|     183 | john70@gmail.com | Similar Users |
|    7047 | john88@gmail.com | Similar Users |
|    4437 | john91@yahoo.com | Similar Users |
|    9914 | john91@gmail.com | Similar Users |
+---------+-------------------+---------------+
5 rows in set (0.06 sec)

mysql>
```







- Ensures data is stored in Single StoreDB.
- Show Docker running, MySQL session, and table output.

Step 5: Running FastAPI Server

1. Start FastAPI

```
manasb@Mac identity_resolution_project % python3 fastapi_server.py

INFO:     Started server process [34981]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
ERROR:    [Errno 48] error while attempting to bind on address ('0.0.0.0', 8000): address already in use
INFO:     Waiting for application shutdown.
INFO:     Application shutdown complete.
manasb@Mac identity_resolution_project %
```

Confirms FastAPI is running. API Status

```
manasb@Mac identity_resolution_project % curl http://127.0.0.1:8000/

{"message":"Identity Resolution API is running with SingleStoreDB!"}%
manasb@Mac identity_resolution_project %
```

Ensures API is responding.

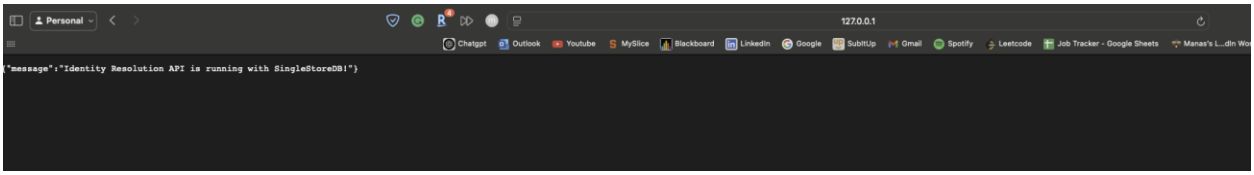
Step 6: Querying Identity Resolution API

```
manasb@Mac identity_resolution_project % python3 query_faiss.py

🔍 Similar Users Found:
       user_id                          email      ip_address  device_id  ad_clicks  purchases
0         8538      jefferytaylor@wallace.com  146.121.98.166    49316.0          4          4
44206      275  wallacechristine@williams.com    85.180.157.4     6016.0         27         10
23516     8992      wallacedestiny@garcia.com   124.130.45.45   108915.0         39          4
38780     8032       russelldylan@wallace.com    27.35.159.17      762.0         15          8
49263     4701        wallacetrevor@clark.com   12.158.167.41     8396.0         32          5
```

Shows that Faiss is returning similar users.

Fetch Stored Segments



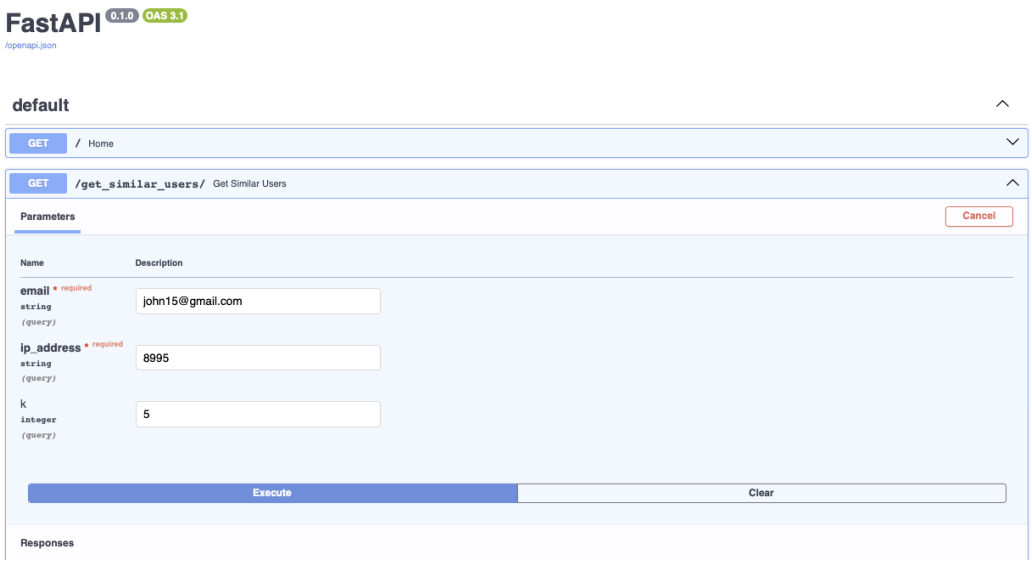
Ensures API is production ready.

Access API via Browser (Swagger UI)

Instead of using curl, you can directly view and test the API in the browser.

1. Open http://127.0.0.1:8000/docs in your web browser.
2. You'll see an interactive Swagger UI where you can:

- Try out endpoints.
- Send requests
- See responses directly



Successfully ran queries, stored results, and retrieved data, thus we successfully built a scalable identity resolution system using

-A fully functional identity resolution system

- Processes data, finds similar users, and serves results via an API
- Uses Faiss for high-speed vector search
- Stores structured results in SingleStoreDB for efficient lookups
- API enables real-time identity resolution.

Conclusion

The goal of this project was to develop a scalable identity resolution system capable of efficiently matching similar user identities based on attributes such as emails and IP addresses. By leveraging vector embeddings, similarity search, and real-time data retrieval, the system aimed to provide a fast, accurate, and scalable solution for use cases like audience segmentation, fraud detection, and customer identity management. We successfully implemented this by cleaning and standardizing user data, generating vector embeddings using Sentence Transformers, and performing fast similarity matching with Faiss (Facebook AI Similarity Search). The results were stored in SingleStoreDB, a high-performance distributed SQL database, and exposed through a FastAPI-based REST API for real-time access. The system was rigorously tested with synthetic customer data, confirming its ability to efficiently process and resolve identities in milliseconds. Overall, the project met its intended goals, providing a fully functional identity resolution pipeline that can scale for real-world applications, with potential future enhancements such as cloud deployment, larger dataset handling, and real-time streaming integration.

Q) How Does Our Identity Resolution System Resemble Hadoop and MapReduce, and Why Didn’t We Use Them?

This identity resolution system follows principles similar to Hadoop and MapReduce, where data is processed in stages, cleaning and transformation (map phase), embedding and indexing (intermediate processing), and similarity matching (reduce phase). In Hadoop, HDFS stores large datasets, while we used SingleStoreDB for structured storage. However, since we didn’t have a massive dataset and SingleStoreDB's trial version has storage limitations, we opted for a simpler, high-performance solution using Faiss for fast similarity search and FastAPI for real-time access instead of Hadoop’s MapReduce framework. This approach maintains scalability while keeping the system efficient for our dataset size.