



# Open source data ingestion for RAGs with dlt

Akela Drissner

# About Me

- Head of Solutions Engineering at dltHub (a data engineering company)
- Previously: Rasa, a conversational AI company
- Background in Machine Learning with a focus on NLP
- LinkedIn: [linkedin.com/oakela](https://www.linkedin.com/oakela)



# Agenda of this workshop

1. Intro on data ingestion
2. Challenges of data ingestion
3. An intro to dlt and how it solves these challenges
4. Challenges of vector DBs
5. Intro to LanceDB
6. Practical implementation of a scalable RAG data pipeline


# What is data ingestion?

The process of extracting data from a producer, transporting it to a convenient environment, and preparing it for usage by normalising it, sometimes cleaning, and adding metadata.

Sometimes the format in which it appears is structured, and with an explicit schema (e.g. Parquet or a db table)

Most of the time, the format is weakly typed and without explicit schema (e.g. csv and json), in which case some normalization and cleaning is required

In many data science teams, data magically ✨ appears - because the engineer loads it.

 What is a schema? The schema specifies the expected format and structure of data within a document or data store, defining the allowed keys, their data types, and any constraints or relationships.

# What is a RAG and a vector DB?

Retrieval augmented generation (RAG):

- A framework to retrieve contextually relevant information and integrate it into an LLM's query
- [Learn more](#)

Vector DB:

- A database that lets you store, index and query embeddings of your data
- [Learn more](#)

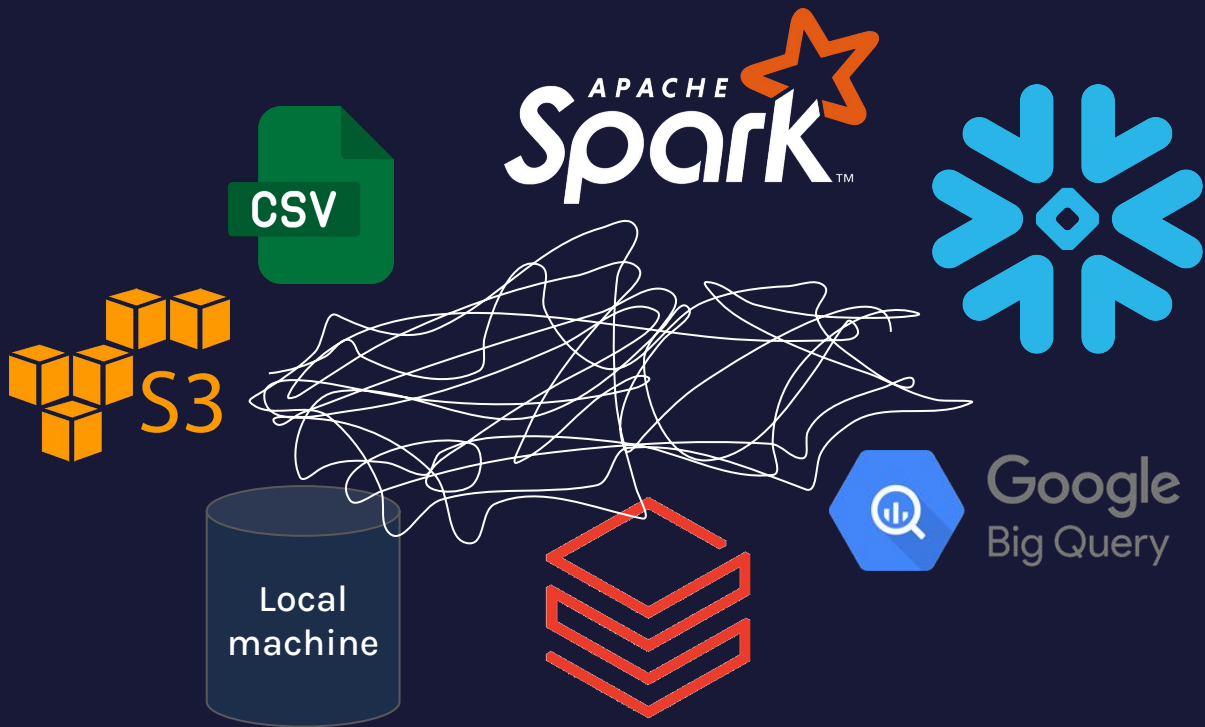
# Why is this course even needed? Why not just use some python scripts?

- Less work, no more breaking python scripts
- Make sure your data is of good quality (data contracts)
- Keep your data up to date without having to reload the entire dataset
- Make your data pipelines production ready!



# Challenges with data ingestion

# Moving data with scripts





# Getting data from APIs

```
marrage_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]
# can't really cast this to an array as it would be useless
# the number of surveys collected for each survey id is different, so that
# so we leave it as a list

# doing the same for the rest:
work_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]['work']
education_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]['education']
money_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]['money']
health_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]['health']
age_related = [[list(i.values()) for i in df.loc[df['survey_id'] == j]]['age']

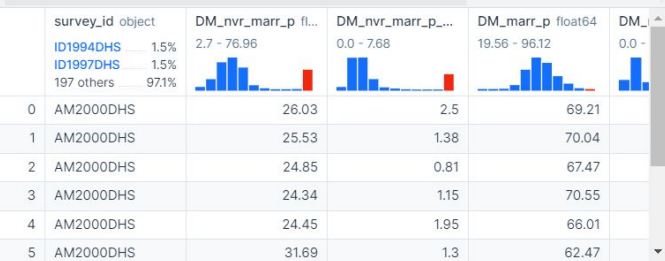
# check if all are correctly sorted for each survey id:
len(marrage_related) == len(work_related) == len(education_related) == len(money_related) == len(health_related) == len(age_related)
```

```
# to sort these into tables:
sids = []
marr_val = [[] for i in df['marriage_related'][0][0].keys()]
# for other variables
work_val = [[] for i in df['work_related'][0][0].keys()]
# ...

for i, el in enumerate(df['survey_id']):
    sids.extend([el for i in marrage_related[i]])
    for j, lst in enumerate(marr_val):
        lst.extend(np.array(marrage_related[i])[j]) # casting to np array
    # the others can be listed below (work_val, ..., age_val)
    # ^ which means this loop will be 5x the amount of code as above :)

marrage_related_df = {'survey_id': sids}
for i, el in enumerate(df['marriage_related'][0][0].keys()):
    marrage_related_df[el] = marr_val[i]

pd.DataFrame(marrage_related_df)
```



# Data & schema versioning

- Data is always changing - how do we keep it up to date?
- You need to be able to identify bad data
- You need to be able to version and roll back your data loads



# Scaling data ingestion

- Loading a few documents is one thing, what happens when you load thousands?
- Hardware limits: limited memory and disk space can cause your machines to crash
- Network limits: sometimes networks fail
- API limits: rate limiting

# Additional consideration for RAGs

- Pre-processing:
  - Extracting the data from PDFs, jsons etc
  - Making sure the resulting text data is clean
- Chunking: making sure the text is in manageable segments

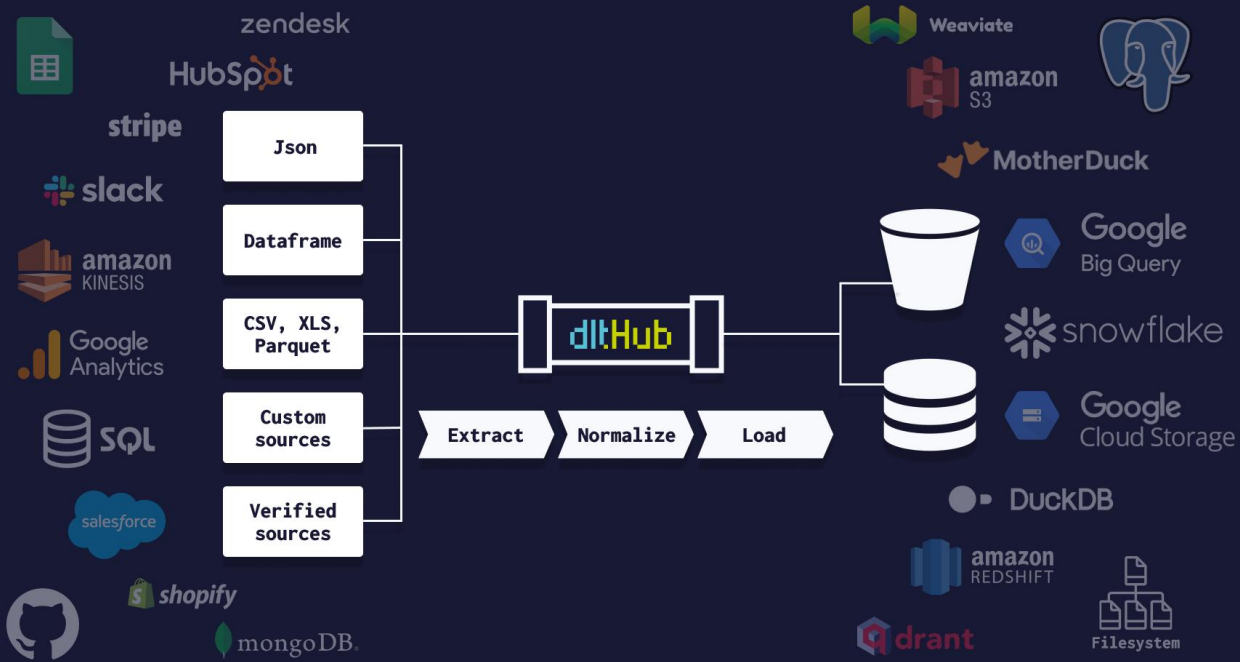
We won't cover cleaning text data or chunking in detail - please refer to [section 4](#) of the LLM Zoomcamp



dlt (data load tool)

# Introducing dlt

dlt is a Python library that automates data loading with features like **schema creation**, **normalization**, and **integration adaptability**.



# Introducing dlt



Easy install and set up.

```
>> pip install dlt
```

Easy to use, learning curve is shallow, declarative interface.

```
import dlt

pipeline = dlt.pipeline(
    pipeline_name='my_pipeline',
    destination='bigquery',
    dataset_name='my_data',
)

pipeline.run(data, table_name='users')
```

It's **Pythonic**, you don't have to learn new frameworks or programming languages.

# Integrations

## Verified sources:

30+ existing well-tested sources, such as Postgres CDC, SQL databases, REST API connector, Google Sheets, Zendesk, Stripe, Notion, Hubspot, GitHub and others.

## Destinations:

16 destinations, such as DuckDB, Postgres, Delta tables, BigQuery, Snowflake, and others.

**Reverse ETL** – build your own destination

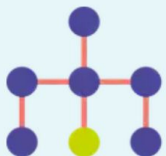
## Integrations:

**dbt**-runner, deploy helpers, Streamlit build-in app, etc.

Be it a Google Colab notebook, AWS Lambda function, an Airflow DAG, or your local laptop – **dlt** can be dropped in anywhere.



verified sources



destinations



helpers & integrations





# Normalizing nested data

dlt normalizes nested data

```
data = [  
  {  
    'id': 1,  
    'name': 'Alice',  
    'job': {  
      'company': "ScaleVector",  
      'title': "Data Scientist",  
    },  
    'children': [  
      {  
        'id': 1,  
        'name': 'Eve'  
      },  
      {  
        'id': 2,  
        'name': 'Wendy'  
      }  
    ]  
  }  
]
```

database varchar	schema varchar	name varchar	column_names varchar[]	column_types varchar[]	temporary boolean
from_json	mydata_20240123031..	_dlt_loads	[load_id, schema_n...	[VARCHAR, VARCHAR, BIG...	false
from_json	mydata_20240123031..	_dlt_pipeline_state	[version, engine_v...	[BIGINT, BIGINT, VARCH...	false
from_json	mydata_20240123031..	_dlt_version	[version, engine_v...	[BIGINT, BIGINT, TIMES...	false
from_json	mydata_20240123031..	json_data	[id, name, job_co...	[BIGINT, VARCHAR, VARC...	false
from_json	mydata_20240123031..	json_data__children	[id, name, _dlt_pa...	[BIGINT, VARCHAR, VARC...	false

id	name	job__company	job__title	_dlt_load_id	_dlt_id
0	1 Alice	ScaleVector	Data Scientist	1706022685.087654	gIv6L4sM9yRgeg

id	name	_dlt_parent_id	_dlt_list_idx	_dlt_id
0	1 Eve	gIv6L4sM9yRgeg		0 Raz2EXsPT9RZtA
1	2 Wendy	gIv6L4sM9yRgeg		1 +3N2PQM2Fnm2kA

# Schema evolution

`dlt` contains all **DE best practices**. Anyone who ever worked with Python can create the pipeline on a Senior level:

- Schema evolution

With `dlt` schema evolution is handled automatically. When modifications occur in the source data's schema, `dlt` detects these changes and updates the schema accordingly.

# Data contracts

`dlt` contains all **DE best practices**. Anyone who ever worked with Python can create the pipeline on a Senior level:

- Schema evolution
- Data contracts

You can use [data contracts](#) modes to tell `dlt` how to apply **contract** for a particular entity:

- **evolve**: No constraints on schema changes.
- **freeze**: Raise an exception if data is encountered that does not fit the existing schema.
- **discard\_row**: Discard any extracted row if it does not adhere to the existing schema.
- **discard\_value**: Discard data in an extracted row that does not adhere to the existing schema.

# Incremental loading

**dlt** contains all **DE best practices**. Anyone who ever worked with Python can create the pipeline on a Senior level:

- Schema evolution
- Data contracts
- Incremental loading

Incremental loading is a crucial concept in data pipelines that involves loading only new or changed data instead of reloading the entire dataset.

```
@dlt.resource(primary_key="id")
def repo_issues(
    access_token,
    repository,
    created_at = dlt.sources.incremental("created_at", initial_value="1970-01-01T00:00:00Z")
):
    # get issues since "created_at" stored in state on previous run (or initial_value on first run)
    for page in _get_issues_page(access_token, repository, since=created_at.start_value):
        yield page
        # last_value is updated after every page
        print(created_at.last_value)
```

# Performance management

**dlt** contains all **DE best practices**. Anyone who ever worked with Python can create the pipeline on a Senior level:

- Schema evolution
- Data contracts
- Incremental loading
- Performance management

**dlt** provides several mechanisms and configuration options to **manage performance** and **scale up** pipelines:

1. Parallel execution: extraction, normalization, and load processes in parallel.
2. Thread pools and async execution: sources and resources that are run in parallel.
3. Memory buffers, intermediary file sizes, and compression options.
4. Scalability through iterators and chunking.

# How does it address our challenges?

- **Messy python scripts:** with dlt you write minimal python code, it handles most complexities automatically
- **Extracting the data:** dlt automatically unnests json, types it and figures out the schema
- **Data versioning:** Each load has an id, is versioned and could be rolled back
- **Data quality:** You can define “data contracts” that reject data that isn’t of the right type
- **Scaling:**
  - Incremental loading - only load new or changed data
  - Performance management



# Challenges with vector DBs

# Maintaining data and embeddings

- Most vector DBs only store embeddings and their metadata
- Extra infrastructure and maintenance costs!

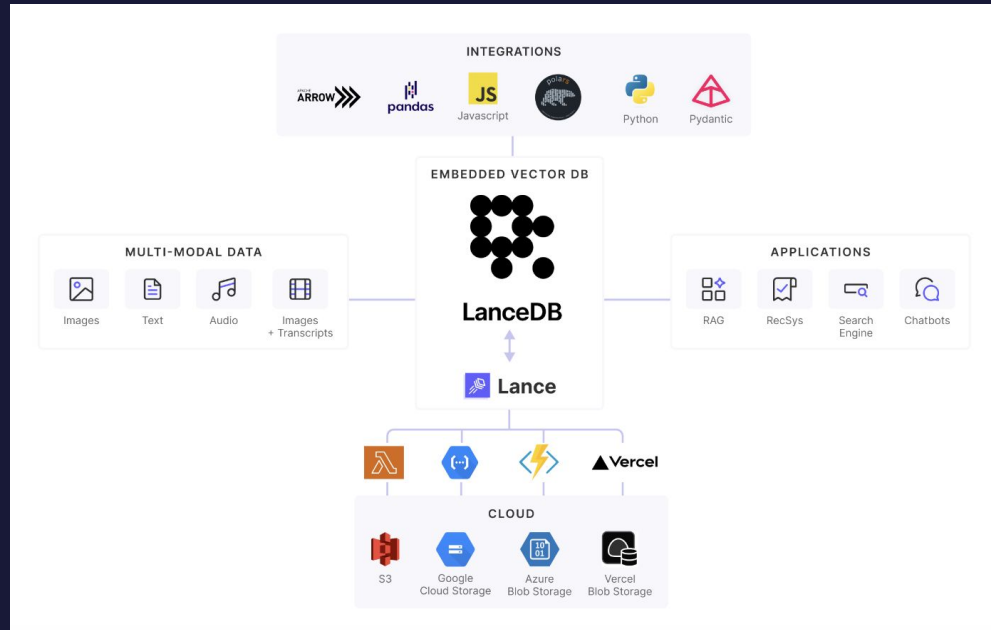




# Introducing LanceDB

# LanceDB: a scalable open source vector DB

- Stores your data (incl text and images), the embeddings and metadata
- Highly scalable + fast





# Link to colab

[https://colab.research.google.com/drive/1nNOybHdWQiwUUuJFZu\\_\\_xvJxL\\_ADU3xl?usp=sharing](https://colab.research.google.com/drive/1nNOybHdWQiwUUuJFZu__xvJxL_ADU3xl?usp=sharing)