Fabric in un caso reale di analisi sinistri con open AI

Marco Pozzan

Sponsor

altitudo

beanTech
IT moves your business

OVERNET.
upgrade your digital skills

[stesi]
Powered by Innovation

Chi sono

Consulente e formatore in ambito business intelligence

Nel 2023 Fondatore e CTO start-up Regolo Farm per Business Intelligence e Power Platform

Nel 2020 Fondatore e CTO start-up Cargo BI per insurance data analytics per le compagnie assicurative

Dal 2017 mi occupo di architetture big Data e in generale di tutta la proposizione data platform di Microsoft.

Docente all'Università di Pordenone per i corsi IFTS di analisi Big Data

Community Lead di 1nn0va (www.innovazionefvg.net)

MCP,MCSA,MCSE dal 2017 MCT e dal 2014 MVP per SQL Server e relatore in diverse conferenze sul tema.

- Marco.pozzan@regolofarm.com
- @marcopozzan.it
- www.marcopozzan.it

MVP
Microsoft
Most Valuable Professional

Microsoft CERTIFIED
AZURE FUNDAMENTALS
Trainer

Microsoft CERTIFIED
Professional

Microsoft CERTIFIED
AZURE DATA ENGINEER
ASSOCIATE
Solutions Associate
SQL Server 2012/2014

Agenda

- Che cosa è Fabric?
- Introduzione alla Demo
  - Che cosa è LLM?
  - Che cosa sono i RAG?
  - Cosa è LangChain?
  - Collegamento a Fabric da Notebook

What is Microsoft Fabric?

A unified SaaS platform for all your analytics needs

Real-time Analytics

Ingest & query real-time data within seconds

Store realtime data in OneLake

BI

Reports & Dashboards With Power BI

Data Science

Run notebooks within seconds without managing clusters

Train models and experiment between different models versions

Data Warehouse

Powered by Delta Parquet

Accessible through SQL and Spark

Decoupled storage & processing

Organized around workspaces

OneLake

Unified lakehouse With an open format

Data Integration

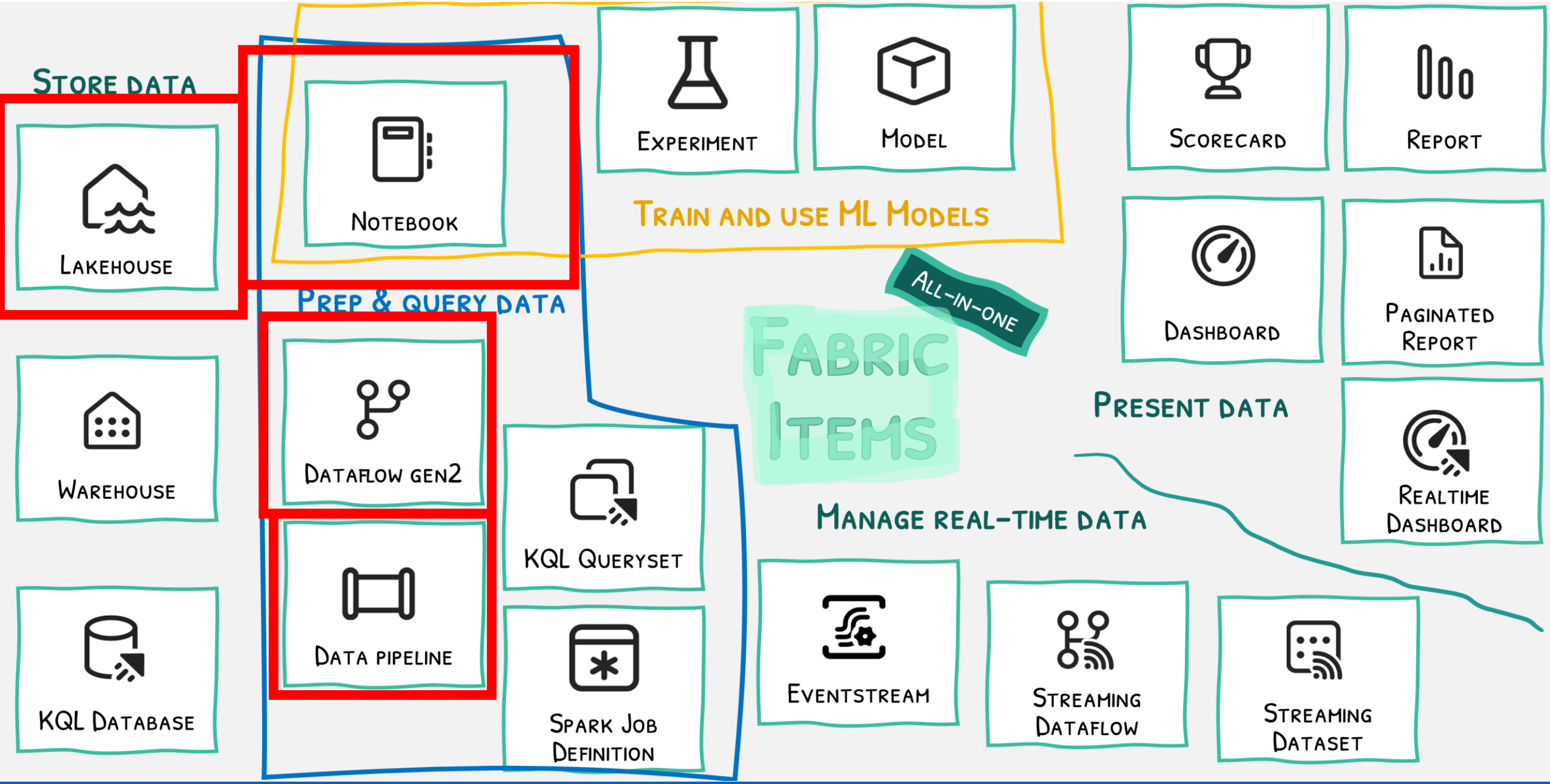
Use no-code or notebooks for ETL or ELT Jobs

Schedule & run jobs without managing (spark) clusters

Unified security

Across all types of data stores & engines

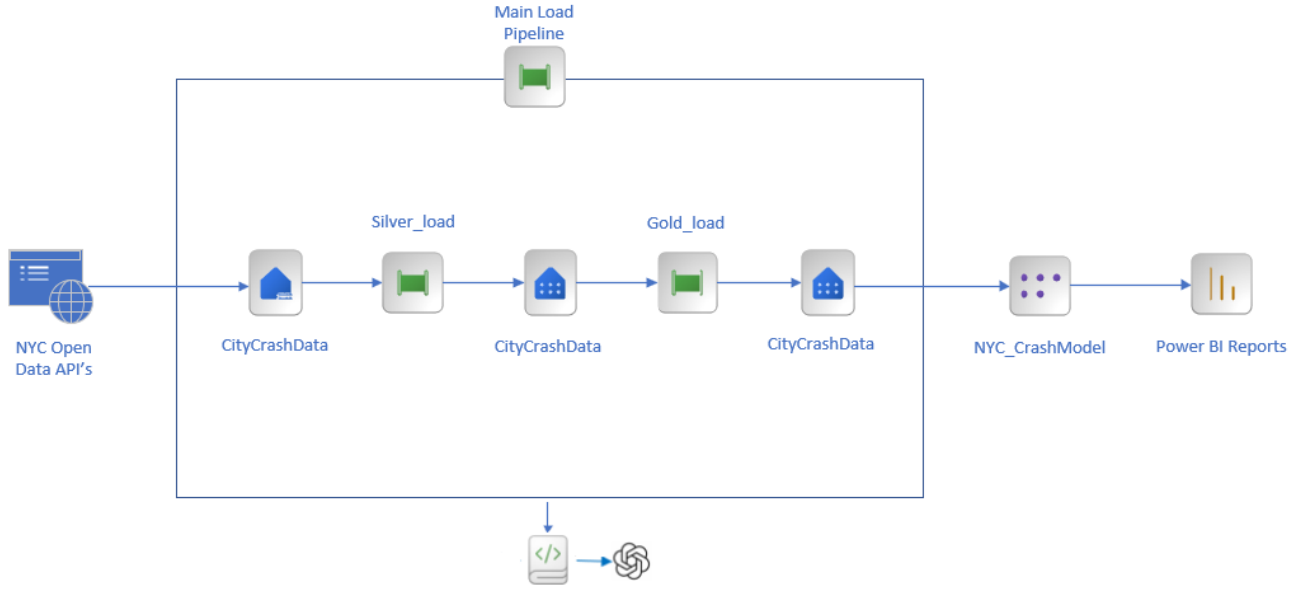
Shortcuts Easy access to data stored in other clouds



Demo

A New York City, negli Stati Uniti, è richiesto un rapporto di polizia standard quando si verifica una collisione di veicoli in cui qualcuno rimane ferito, muore o se ci sono almeno $ 1.000 di danni. Questi dati vengono raccolti e pubblicati online come fonte di dati aperta su NYC Open Data e vengono aggiornati quotidianamente.

Lo scenario è quello di fornire report Power BI aggiornati quotidianamente per fornire analisi descrittive e approfondimenti su posizione, fattori contribuenti e tipi di veicoli per collisioni di veicoli utilizzando Fabric. Questa soluzione fornisce anche le basi per altre persone come Data Analyst e Data Scientist per analisi con Open AI.



Che cosa è LLM?

- I Large Language Model (LLM), come ChatGPT di OpenAI, hanno rivoluzionato il modo in cui interagiamo e utilizziamo l'intelligenza artificiale.
- Questi modelli sono abili nel generare testo simile a quello umano e possono essere sfruttati per un'ampia gamma di applicazioni. Tuttavia, per massimizzare la loro utilità in contesti specifici, come ambienti aziendali o domini specializzati, la personalizzazione è essenziale.

Che cosa è la Retrieval Augmented Generation (RAG)?

- Di base, gli LLM sono formati su ampi set di dati che comprendono un'ampia gamma di conoscenze generali. Sebbene ciò li renda versatili, significa anche che potrebbero non avere la precisione necessaria per attività specializzate.

- Un metodo efficace per migliorare la pertinenza e l'accuratezza degli output LLM è tramite Retrieval Augmented Generation (RAG). RAG integra le risposte del modello con informazioni recuperate dinamicamente, assicurando che l'output sia contestualmente rilevante e aggiornato. Ciò è particolarmente prezioso quando si ha a che fare con dati aziendali specifici o query dei clienti, in cui le risposte generiche risultano carenti.

Che cosa è LangChain

- LangChain è un framework per lo sviluppo di applicazioni basate su modelli linguistici di grandi dimensioni (LLM).

- LangChain semplifica ogni fase del ciclo di vita dell'applicazione LLM:

- **Sviluppo:** crea le tue applicazioni utilizzando i blocchi di costruzione open source, i componenti e le integrazioni di terze parti di LangChain. Utilizza LangGraph per creare agenti con stato con streaming di prima classe e supporto umano nel ciclo.
- **Produzione**: utilizza LangSmith per ispezionare, monitorare e valutare le tue catene, in modo da poterle ottimizzare e distribuire in modo continuo con sicurezza.
- **Distribuzione**: trasforma le tue applicazioni LangGraph in API e assistenti pronti per la produzione con LangGraph Cloud.

Come si configura LangChain

- Codice che inizializza l'istanza chat_model di LangChain che fornisce un'interfaccia per richiamare un provider LLM tramite chat API.
- Il motivo per cui selezioniamo il **modello chat** è che il modello **gpt-35-turbo** è ottimizzato per la chat

```
1   import os
2   from langchain.chat_models import AzureChatOpenAI
3
4   llm = AzureChatOpenAI(model=os.getenv("OPENAI_CHAT_MODEL"),
5                         deployment_name=os.getenv("OPENAI_CHAT_MODEL"),
6                         temperature=0)
```

- La temperatura è un parametro che controlla la "creatività" o la casualità del testo generato. Una temperatura più bassa rende l'output più "focalizzato" o deterministico. Poiché qui abbiamo a che fare con un database, è importante che LLM generi una risposta relativa ai fatti.

Configurazione del prompt per LLM

- **Prompt** è l'input che inviamo all'LLM per generare un output. Prompt può anche essere progettato per contenere istruzioni, contesto e impostare il tono e formattare i dati di output.

```
from langchain.prompts.chat import ChatPromptTemplate

final_prompt = ChatPromptTemplate.from_messages(
    [
        ("system",
         """
          You are a helpful AI assistant expert in querying SQL Database to find answers to user's question about Collision.
         """
         ),
        ("user", "{question}\n ai: "),
    ]
)
```

- Utilizzare Prompt Template è un buon modo per strutturare queste proprietà, incluso l'input dell'utente finale da fornire all'LLM.
- Noi utilizziamo il modulo **ChatPromptTemplate** di LangChain, che si basa su **ChatML** (Chat Markup Language).

Configurazione dell'agent

- Il componente ***Agent*** di LangChain è un wrapper attorno a LLM, che decide i migliori passaggi o azioni da intraprendere per risolvere un problema.
- L'Agent ha in genere accesso a un set di funzioni chiamate ***Tools*** (o Toolkit) e può decidere quale Tool utilizzare in base all'input dell'utente.
- Ogni agent può eseguire varie attività NLP, come parsing, calcoli, traduzione ecc.

Configurazione dell’agent

- Un ***Agent Executor*** è un'interfaccia eseguibile dell'Agent e del suo set di Tools.
- L'agent executor è responsabile della chiamata dell'Agent, dell'ottenimento dell'azione e dell'input dell'azione, della chiamata dello strumento a cui l'azione fa riferimento con l'input corrispondente, dell'ottenimento dell'output dello strumento e quindi del passaggio di tutte queste informazioni all'Agent per ottenere l'azione successiva che dovrebbe intraprendere. Di solito è un processo iterativo finché l'Agent non raggiunge la Final Answer o l'output.

Come ci si connette al lakehouse di Fabric

- Per collegarsi al lakehouse di fabric usiamo la libreria sqlalchemy passando come credenziali ClientID e Secret del Service Principal

```python
from sqlalchemy import create_engine
import sqlalchemy as sa

driver = '{ODBC Driver 18 for SQL Server}'
odbc_str =  sa.URL.create(
    "mssql+pyodbc",
    username="feca4d00-63fe-47b6-95d1-4fe2ca30dd48",
    password="tsd8Q~GoCYMTvUGYejamAUtgBxku.6ovPZAeVbQy",
    host="ehauyipajrjepfj3id6yqalcnm-rxx6zuqhqlterk4ko7sdyhvdsm.datawarehouse.fabric.microsoft.com",
    database="CityCrashData",
    query={"driver": "ODBC Driver 18 for SQL Server","Authentication":"ActiveDirectoryServicePrincipal"}
)
print(odbc_str)
db_engine = create_engine(odbc_str)
connection = db_engine.raw_connection()
```

Configurazione dell’agent

- L’agent è progettato per interagire con il database SQL.E’ dotato di toolkit per connettersi al database SQL e leggere sia i metadati che dati

```
from langchain.agents import AgentType, create_sql_agent
from langchain.sql_database import SQLDatabase
from langchain.agents.agent_toolkits.sql.toolkit import SQLDatabaseToolkit

db = SQLDatabase(db_engine, view_support=True)
# test the connection
print(db.dialect)
print(db.get_usable_table_names())

sql_toolkit = SQLDatabaseToolkit(db=db, llm=llm)
sql_toolkit.get_tools()

sqldb_agent = create_sql_agent(
    llm=llm,
    toolkit=sql_toolkit,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
    verbose=True
)
```

- Abbiamo utilizzato ZERO_SHOT_REACT_DESCRIPTION come valore del parametro agent_type, che indica all'agente di non utilizzare memoria.

Come funziona l’agent?

- L’agent è progettato per mandare domande e ricevere risposte.

```
1   sqldb_agent.run(final_prompt.format(
2           question="what is total of persone morte"
3     ))
4     #  question="what is the number of persone morte in the year 2023?"
```

Grazie!