LIVE

TEXPRESSO

Maffeis Riccardo 1085706 Zanotti Matteo 1085443

Job Principale TedX

Abbiamo integrato e adattato il dataset watch_next, che contiene i suggerimenti relativi ai talk TEDx, all'interno del nostro Data Warehouse.

Nel job PySpark abbiamo integrato i video correlati raggruppando i suggerimenti per ogni talk tramite una aggregazione (collect_list). Successivamente, abbiamo unito questi dati al dataset principale TEDx, arricchendolo con una colonna contenente tutti i talk suggeriti per ciascun video, che ci permetterà mediante i tag di collegare le news.

```
#READ RELATED_VIDEOS DATASET
related_videos_dataset_path = "s3://tedxpresso-data-mp/related_videos.csv"
related_videos_dataset = spark.read.option("header","true").csv(related_videos_dataset_path)

#CREATE THE AGGREGATE MODEL, ADD RELATED_VIDEOS TO TEDX_DATASET
related_videos_dataset_agg = related_videos_dataset.groupBy(col("id").alias("id_ref")).agg(collect_list("related_id").alias("related_ids"))
related_videos_dataset_agg.printSchema()
tedx_dataset_agg1 = tedx_dataset_agg.join(related_videos_dataset_agg, tedx_dataset_agg.id == related_videos_dataset_agg.id_ref, "left") \
    .drop("id_ref")

tedx_dataset_agg1.printSchema()
```

Jobs Secondari

NewsData

X

Reddit

NewsAPI

Job NewsAPI

L'integrazione di **NewsAPI** è stata realizzata per consentire l'estrazione di notizie in tempo reale da fonti giornalistiche autorevoli. Attraverso l'utilizzo di una **API key gratuita**, è stato possibile accedere ai dati in formato **JSON**, successivamente elaborati tramite un job PySpark. I contenuti ottenuti sono stati salvati sia su **Amazon S3** per la conservazione e la scalabilità, sia all'interno del **database**, dove sono stati unificati con il modello dati esistente.

```python
# Funzione per chiamata API
def fetch_news(page):
    url = 'https://newsapi.org/v2/everything'
    params = {
        'q': query,
        'language': 'it',
        'pageSize': 100,
        'page': page,
        'apiKey': api_key
    }
    response = requests.get(url, params=params)
    if response.status_code != 200:
        raise Exception(f"Errore API ({response.status_code}): {response.text}")
    return response.json().get('articles', [])
```

Job NewsData

Un'ulteriore fonte di notizie è stata integrata utilizzando una **seconda API specializzata**, caratterizzata dalla possibilità di effettuare un numero maggiore di richieste giornaliere, anche in presenza di una singola chiave di accesso gratuita. Questo servizio si è rivelato particolarmente utile per garantire una copertura informativa più ampia e costante nel tempo. Anche in questo caso, i dati ricevuti in formato **JSON** sono stati gestiti all'interno di un job PySpark e successivamente **salvati su Amazon S3** e **inseriti nel database**.

```
# === 3. Chiamata all'API NewsData.io ===
base_url = "https://newsdata.io/api/1/news"
params = {
    "apikey": api_key,
    "country": "it",
    "language": "it",
}

response = requests.get(f"{base_url}?{urlencode(params)}")
if response.status_code != 200:
    raise Exception(f"❌ NewsData.io API error {response.status_code}: {response.text}")
```

Job RSS Reddit

L'integrazione dei dati provenienti da Reddit è stata effettuata utilizzando esclusivamente il feed RSS ufficiale, in quanto rappresenta una soluzione gratuita, stabile e facilmente gestibile.
Questo approccio non richiede l'utilizzo di una chiave di accesso (API key), semplificando così la fase di autenticazione. Il processo di salvataggio avviene in modo analogo con memorizzazione in formato JSON su Amazon S3 e nel database.
Tuttavia, l'elaborazione presenta una peculiarità: i contenuti RSS vengono gestiti tramite il formato Atom, e la richiesta HTTP deve essere accompagnata da uno User-Agent compatibile (es. Mozilla).

```python
# === FUNZIONE PARSING RSS ATOM ===
def fetch_reddit_rss_data(url):
    try:
        response = requests.get(url, headers={
            'User-Agent': 'Mozilla/5.0 (compatible; AWS Glue Job)'
        }, timeout=10)

        if response.status_code != 200:
            print(f"HTTP Error {response.status_code}")
            return []

        root = ET.fromstring(response.content)
        ns = {'atom': 'http://www.w3.org/2005/Atom'}
        entries = root.findall("atom:entry", ns)
        print(f"Trovati {len(entries)} elementi nel feed.")

        items = []
        for entry in entries:
            title = entry.findtext("atom:title", default="", namespaces=ns)

            link_el = entry.find("atom:link[@rel='alternate']", ns)
            if link_el is None:
                link_el = entry.find("atom:link", ns)
            link = link_el.attrib['href'] if link_el is not None else None

            published = entry.findtext("atom:published", default="", namespaces=ns)
            summary = entry.findtext("atom:content", default="", namespaces=ns)

            try:
                published_dt = datetime.strptime(published, "%Y-%m-%dT%H:%M:%S%z")
                published_str = published_dt.isoformat()
            except:
                published_str = None

            items.append({
                "title": title,
                "link": link,
                "published": published_str,
                "summary": summary
            })

        return items
```

Job X

L'integrazione della piattaforma **X è** stata effettuata attraverso l'utilizzo delle API ufficiali, le quali richiedono il possesso di una **chiave di accesso (API key)** ottenuta tramite la registrazione a un **account developer**. Il processo di estrazione dei dati è risultato tecnicamente semplice grazie alla struttura ben documentata delle API, che permette di accedere a contenuti come tweet, hashtag, menzioni e profili. Tuttavia, **le limitazioni imposte dal piano gratuito**, in particolare in termini di **numero massimo di richieste giornaliere**, si sono rivelate troppo restrittive per un utilizzo continuativo e su larga scala, per questa ragione non verrà utilizzata nella raccolta generale dei dati.

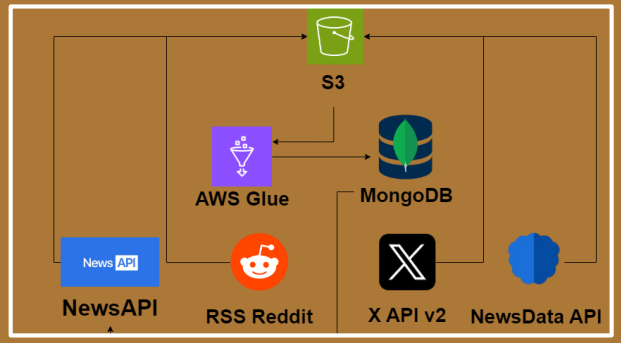
```
# --- 2) Headers e params per 1 singola richiesta di 2 tweet matching la query ---
headers = {
    'Authorization': f'Bearer {args["api_key"]}',
    'Accept': 'application/json'
}
params = {
    'query': args['query'],        # stringa di ricerca, es. "data",
    'tweet.fields': 'id,text',
    'max_results': 2
}

# --- 3) Chiamata all'API X ---
response = requests.get(args['api_endpoint'], headers=headers, params=params, timeout=30)
response.raise_for_status()
records = response.json().get('data', [])
```

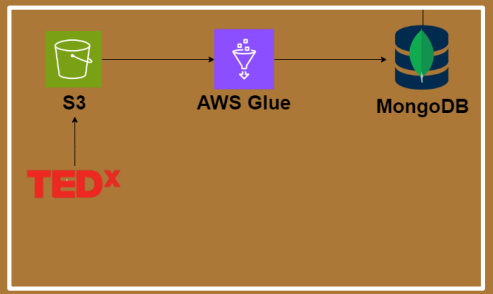
Sviluppi

Il progetto si è articolato in due componenti principali:
da un lato, la raccolta dei dati provenienti da fonti API esterne, e dall'altro, la fase di integrazione, collegamento e filtraggio dei contenuti TEDx con le informazioni ottenute.

Per garantire un aggiornamento continuo e un rispetto dei limiti imposti dai piani gratuiti delle API, è stato implementato un meccanismo di scheduling automatico tramite AWS EventBridge.

Questo sistema consente di eseguire le chiamate API in modo distribuito e programmato, prelevando i dati ogni due ore, alternando le fonti per evitare il superamento delle soglie di utilizzo.





CRITICITÀ

Limiti richieste API	Necessaria schedulazione per gestione dati	API key necessarie
Salvataggio di grandi quantità di dati	Limiti di aggiornamento basati sul tempo	Affidabilità API

NEWS

Integrazioni future?

01 Aggiunta di API a pagamento

02 Aggiunta di AI per la traduzione

03 Integrazione con piattaforme video

04 Aggiornamento a richiesta dell'utente

LIVE

TEXPRESSO