Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

Automazione della pipeline per il calcolo delle Community Health Metrics in Wikipedia

Automazione con Apache Airflow e osservabilità tramite Grafana

Supervisore
Montresor Alberto
Cristian Consonni

Laureando
Denina Andrea

Anno accademico 2024/2025

Ringraziamenti

...thanks to...

Indice

Sommario

L'elaborato descrive l'attività svolta durante il tirocinio curriculare, finalizzata alla progettazione e implementazione di una pipeline dati automatizzata per il calcolo delle Community Health Metrics di Wikipedia. Queste metriche costituiscono indicatori fondamentali per valutare lo stato di salute e la vitalità delle comunità online, fornendo uno strumento utile sia per i ricercatori che per i responsabili delle piattaforme collaborative. Il lavoro prende avvio da uno script Python monolitico, privo di modularità, di strumenti di schedulazione e di funzionalità di monitoraggio. Tale soluzione risultava poco scalabile e difficilmente manutenibile, soprattutto in un contesto in cui i dati da elaborare, i MediaWiki History Dumps, sono caratterizzati da dimensioni elevate e aggiornamenti periodici.

La principale motivazione del progetto è stata quindi quella di adottare un approccio più moderno e strutturato, capace di garantire affidabilità, osservabilità e riproducibilità delle elaborazioni. Per raggiungere questo obiettivo è stato scelto di utilizzare Apache Airflow come strumento di orchestrazione dei workflow, che ha consentito di suddividere il processo in task modulari e indipendenti e di programmarne l'esecuzione periodica. Tutta l'infrastruttura è stata containerizzata con Docker e orchestrata tramite Docker Compose, in modo da garantire portabilità, isolamento e semplicità di deploy. Per quanto riguarda l'osservabilità, è stato predisposto uno stack di monitoraggio basato su StatsD, Prometheus e Grafana, che permette di raccogliere e visualizzare metriche relative alle performance e allo stato della pipeline, rendendo possibile l'individuazione di anomalie in tempo reale.

Il risultato finale è una pipeline completamente automatizzata, in grado di elaborare i dump storici di Wikipedia in maniera affidabile e scalabile. Oltre al miglioramento dell'efficienza, il sistema offre ora un controllo puntuale delle esecuzioni, con dashboard dedicate che consentono di analizzare sia l'andamento delle task sia l'utilizzo delle risorse. Il contributo personale del lavoro si è concentrato sul refactoring del codice esistente, sulla definizione e implementazione dei DAG di Airflow, sulla configurazione dell'infrastruttura containerizzata e sull'integrazione del sistema di monitoraggio. Il mio contributo si è concluso con il deploy dell'applicazione su un server di Wikimedia, dove è ora attiva in produzione. In questo modo il progetto Community Health Metrics può contare su un'infrastruttura moderna e affidabile, capace di calcolare automaticamente i dati necessari alle visualizzazioni.

1 Introduzione

La presente tesi è il risultato del tirocinio curriculare che ho svolto negli scorsi mesi, nell'ambito del progetto Community Health Metrics di Wikipedia. Tale iniziativa ha l'obiettivo di misurare lo stato di salute delle comunità di Wikipedia attraverso 6 metriche, fornendo strumenti utili per analizzare la partecipazione degli editor e supportarne lo sviluppo nel tempo.

Il punto di partenza del mio lavoro era costituito da uno script monolitico che, pur permettendo di calcolare le metriche desiderate, presentava diversi limiti in termini di modularità, scalabilità e assenza di strumenti di monitoraggio.

L'attività di tirocinio si è quindi focalizzata sul refactoring di questo sistema, con l'obiettivo di realizzare una pipeline dati moderna e automatizzata, basata su Apache Airflow, e integrata con un sistema di monitoraggio fondato su Prometheus e Grafana. Il lavoro si è infine concluso con il deploy in produzione della pipeline su un server Wikimedia, dove è attualmente utilizzata per il calcolo mensile delle metriche.

1.1 Contesto

Il progetto Community Health Metrics propone sei insiemi di indicatori, detti Vital Signs, per valutare crescita e rinnovamento delle comunità di Wikipedia. Tre riguardano l'intera popolazione di active editors (chi crea contenuti): Retention, Stability e Balance; tre riguardano funzioni comunitarie specifiche: Special functions, Administrators e Global participation.

adesso presenterò in dettaglio come vengono misurati questi indicatori e quali sono le metriche associate:

- **Retention**: Misura la capacità di trattenere i nuovi editori nel tempo. L'indicatore base è la retention rate: quota di nuovi editori che effettuano almeno un'altra modifica 60 giorni dopo la prima. È un segnale precoce della qualità dell'onboarding e dell'inclusione dei newcomer.

- **Stability**: Cattura la persistenza degli editori attivi. Si conta il numero di mesi consecutivi di attività per ogni editor e si analizza la distribuzione in sei fasce: 1 mese, 2 mesi di fila, 3–6, 7–12, 13–24, più di 24 mesi. Una comunità "stabile" combina sia nuove presenze sia contributori di lungo periodo.

- **Balance**: Valuta l'equilibrio tra "generazioni" di editor molto attivi. L'indicatore considera numero e percentuale di very active editors per anno e per generazione (definita come il lustrum dell'anno del primo edit). Very active editor = utente registrato (non bot) con più di 100 edit al mese nel mainspace. L'obiettivo è che più generazioni contribuiscano alla "spina dorsale" produttiva, senza dipendere solo dai veterani.

- **Special functions**: Questo indicatore prende in considerazione gli editor che svolgono compiti speciali all'interno di Wikipedia, ossia attività che vanno oltre la semplice scrittura di contenuti. In particolare, vengono distinte due categorie: da un lato coloro che si occupano della manutenzione tecnica, contribuendo nei namespace relativi a template e infrastruttura del software; dall'altro chi partecipa alle attività di coordinamento, interagendo negli spazi dedicati all'organizzazione e alla governance della comunità.

- **Administrators**: Gli amministratori svolgono un ruolo centrale nel mantenimento dell'ordine e della qualità all'interno delle comunità di Wikipedia, poiché hanno la responsabilità di supervisionare i contenuti, applicare le regole e garantire il corretto funzionamento del progetto. Per valutare la capacità di questo gruppo di supportare la comunità, l'indicatore osserva sia l'evoluzione storica degli utenti che hanno ricevuto i permessi amministrativi, sia il numero di

amministratori effettivamente attivi nel tempo. Un altro aspetto rilevante riguarda il rapporto tra amministratori ed editor attivi: questo valore fornisce un'indicazione della sostenibilità del gruppo di gestione rispetto alla dimensione complessiva della comunità. In questo modo è possibile comprendere se il numero di amministratori è sufficiente a garantire trasparenza ed efficienza, evitando situazioni di sovraccarico o, al contrario, un eccesso di concentrazione di potere decisionale.

- **Global participation**: Questa metrica misura il grado di apertura e di interconnessione di una comunità linguistica di Wikipedia con il resto del movimento Wikimedia. Da un lato considera il livello di partecipazione degli utenti al progetto Meta-Wiki, osservando in particolare quanti editor attivi hanno come lingua primaria quella della comunità analizzata. Dall'altro analizza la composizione degli editor in base alla loro lingua primaria, distinguendo tra chi contribuisce principalmente nella Wikipedia in questione e chi invece proviene da altre edizioni linguistiche. In questo modo è possibile valutare la capacità della comunità di attrarre contributori multilingue e di favorire interazioni cross-wiki, elementi importanti per garantire apertura, collaborazione e scambio con il resto dell'ecosistema Wikimedia.

Le metriche descritte vengono calcolate a partire dai MediaWiki History Dumps, un insieme di file che documentano in maniera completa la cronologia delle modifiche di Wikipedia e che vengono aggiornati mensilmente. Questi dataset rappresentano la base informativa su cui si fonda il progetto Community Health Metrics, in quanto permettono di ricostruire l'evoluzione delle comunità e di derivarne gli indicatori presentati. Nel capitolo successivo verranno presentati nel dettaglio la loro struttura e i dataset utilizzati nella fase di sviluppo della pipeline.

1.2 Dataset

Il dataset dei MediaWiki History Dumps contiene la cronologia completa degli eventi relativi a revisioni, utenti e pagine delle wiki Wikimedia a partire dal 2001. I dati sono organizzati in uno schema denormalizzato, in cui tutte le informazioni sono raccolte in un unico formato, includendo campi precomputati utili alle analisi, come il numero di edit per utente e per pagina o le operazioni di revert. Questo rende i dumps una fonte strutturata e coerente per lo studio dell'evoluzione delle comunità. Il dataset dei MediaWiki History Dumps viene aggiornato con cadenza mensile, generalmente entro la fine della prima settimana del mese. Ogni rilascio contiene l'intera cronologia a partire dal 2001 fino al mese corrente: questo approccio è necessario poiché eventi come rinomini di utenti, revert o spostamenti di pagine possono modificare retroattivamente lo stato delle tabelle, rendendo poco affidabili eventuali aggiornamenti incrementali. I dumps sono organizzati secondo un sistema di partizionamento che tiene conto della dimensione delle diverse edizioni linguistiche. Per le wiki più grandi, come enwiki, wikidatawiki e commonswiki, i file sono suddivisi su base mensile; per quelle di dimensioni intermedie la suddivisione è annuale; mentre per le comunità più piccole l'intero storico è raccolto in un singolo file. In questo modo si evita la generazione di file eccessivamente voluminosi, mantenendo ogni dump entro la dimensione di circa 2 GB. I dumps sono distribuiti in formato TSV, compresso con Bzip2 per ridurre le dimensioni. La struttura delle directory segue una convenzione che comprende la versione del dump (nel formato YYYY-MM, con la disponibilità limitata agli ultimi due mesi), il nome della wiki e l'intervallo temporale della partizione, ad esempio: /2019-12/ptwiki/2019-12.ptwiki.2018.tsv.bz2.

Nel contesto del progetto ci interessano esclusivamente i dumps relativi alle comunità linguistiche di Wikipedia, poiché costituiscono la base per il calcolo delle metriche presentate. Nella fase di sviluppo ho scelto di lavorare su un sottoinsieme di questi dati, selezionando alcune edizioni linguistiche dei dialetti italiani: piemontese, lombardo, ligure, veneto, sardo, siciliano e napoletano. La scelta è stata guidata dal fatto che tali dataset, oltre ad avere dimensioni ridotte, sono distribuiti in un unico file all-time, caratteristica che li rende particolarmente adatti per le fasi di test e sviluppo.

Per chiarezza, nella tabella seguente riporterò le edizioni considerate, con il nome della lingua, il codice corrispondente e la dimensione del file dump in megabyte:

Lingua	Codice	Dimensione (MB)
Piemontese	pmswiki	64.6
Lombardo	lmowiki	91.6
Ligure	lijwiki	18.9
Veneto	vecwiki	78.9
Napoletano	napwiki	45.3
Siciliano	scnwiki	57.4
Sardo	scwiki	14.6

Tabella 1.1: Dump utilizzati nella fase di sviluppo (edizione maggio 2025)

1.3 Problema affrontato

La versione iniziale dell'architettura, disponibile su GitHub, era costituita da un unico script Python, denominato `vital_signs.py`. Questo programma aveva il compito di estrarre i dati dai *MediaWiki History Dumps* e di popolare due database SQLite. Le operazioni venivano eseguite in maniera sequenziale: in una prima fase lo script generava il database `vital_signs_editors.db`, contenente le informazioni di base sugli editor. Successivamente, a partire da questi dati, calcolava gli indicatori definiti come *Vital Signs* e li inseriva nel database `vital_signs_web.db`.

Quest'ultimo database rappresentava l'output finale del processo ed era utilizzato da un'applicazione sviluppata con il framework Dash, che permetteva di visualizzare graficamente gli indicatori attraverso una serie di dashboard. Questa applicazione è stata sviluppata da un altro tirocinante ed era stata deployata su un server wikimedia dedicato, dove era accessibile pubblicamente.

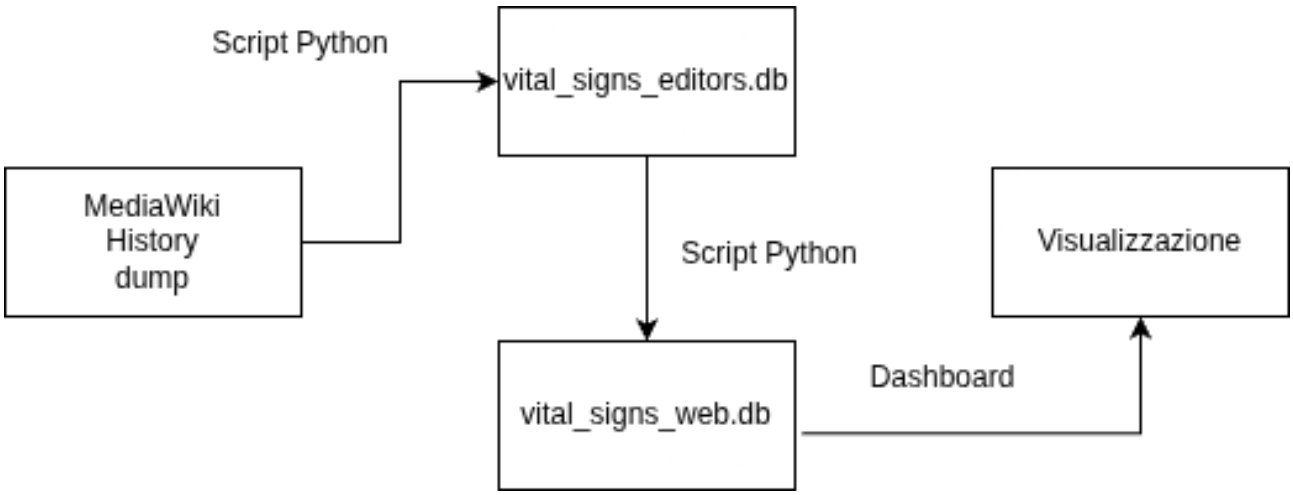

Figura 1.1: Rappresentazione logica della vecchia architettura

L'architettura presentava diversi limiti:

- **Monoliticità**: Tutte le operazioni erano contenute in un unico script, rendendo difficile la manutenzione e l'estensione del codice. Non c'era modularità, né possibilità di riutilizzare le singole parti del processo.

- **Assenza di schedulazione**: Lo script doveva essere eseguito manualmente, senza alcun meccanismo di schedulazione. Questo rendeva impossibile automatizzare l'aggiornamento dei dati, che doveva essere fatto manualmente ogni mese.

- **Mancanza di monitoraggio**: Non c'erano strumenti per monitorare lo stato di esecuzione dello script, né per raccogliere metriche sulle performance. In caso di errori, non era possibile risalire facilmente alla causa.

- **Prestazioni limitate**: L'approccio sequenziale e la mancanza di parallelizzazione rendevano il processo lento, soprattutto con dataset di grandi dimensioni. Non c'era modo di scalare l'elaborazione per gestire volumi crescenti di dati.

- **Difficoltà di deploy**: trattandosi di uno script, il deploy risulta più complicato rispetto ad un approccio basato su container. Non c'era isolamento tra le dipendenze e l'ambiente di esecuzione, rendendo difficile la gestione delle versioni e delle librerie.

1.3.1 Obiettivi del tirocinio

L'obiettivo principale del tirocinio è stato il refactoring dello script esistente e la progettazione di una data pipeline completamente automatizzata utilizzando Apache Airflow. In particolare, il lavoro ha previsto:

- la strutturazione del workflow in task modulari e indipendenti, per garantire chiarezza e riusabilità;

- l'introduzione di meccanismi di schedulazione, così da consentire esecuzioni periodiche e affidabili senza intervento manuale;

- l'integrazione di strumenti di monitoraggio per tracciare lo stato della pipeline e raccogliere metriche di performance;

- lo sviluppo dell'intera infrastruttura in un ambiente Docker, con tutti i componenti (Airflow, database e monitoring stack) containerizzati e orchestrati tramite `docker-compose`, al fine di semplificare il deploy e garantire portabilità;

- la sostituzione del database `SQLite` con `PostgreSQL`, scelta resa necessaria dal fatto che le *task* in Airflow vengono eseguite in parallelo e quindi richiedono un sistema in grado di gestire operazioni concorrenti in scrittura;

- il deployment della nuova architettura su un server Wikimedia, dove la pipeline è ora attiva in produzione e calcola mensilmente le metriche richieste.

Nei capitoli successivi verranno presentate in dettaglio le tecnologie utilizzate (Capitolo 2), l'implementazione della nuova architettura (Capitolo **??**), e infine i risultati ottenuti e le conclusioni del lavoro (Capitolo **??**).

2 Tecnologie utilizzate

In questa sezione vengono presentate, con un adeguato livello di dettaglio, le tecnologie utilizzate per la realizzazione della pipeline dati sviluppata durante il tirocinio. Lo stack tecnologico adottato è stato progettato per garantire modularità, scalabilità e osservabilità del sistema, caratteristiche fondamentali in un contesto di automazione e monitoraggio di processi dati periodici. Come si può notare, tutte le tecnologie utilizzate seguono la filosofia open source. Con il termine open source si intende software il cui codice sorgente è pubblico e liberamente accessibile: questo permette a chiunque di utilizzarlo, modificarlo e distribuirlo. Grazie a questa apertura, la collaborazione tra sviluppatori è facilitata e diventa più semplice correggere errori o aggiungere nuove funzionalità.

Le tecnologie possono essere suddivise in tre principali categorie funzionali:

- **Containerizzazione**: Docker e Docker Compose sono stati utilizzati per creare un ambiente di esecuzione isolato, facilmente replicabile e portabile. Tutti i componenti dell'infrastruttura vengono eseguiti come container indipendenti ma coordinati, permettendo una gestione semplificata dell'intero sistema.

- **Orchestrazione**: Apache Airflow è stato scelto come strumento per l'automazione e la schedulazione dei flussi di lavoro. La sua interfaccia intuitiva e il paradigma configuration as code lo rendono adatto a definire e monitorare pipeline complesse.

- **Monitoraggio e raccolta delle metriche**: il sistema di osservabilità è stato implementato utilizzando StatsD Exporter, Prometheus e Grafana. Le metriche prodotte da Airflow vengono raccolte, elaborate e visualizzate in tempo reale tramite dashboard interattive, al fine di garantire il corretto funzionamento del sistema e facilitare l'identificazione di anomalie o colli di bottiglia.

Nei paragrafi seguenti, ogni tecnologia verrà descritta singolarmente.

2.1 Docker

Docker è una piattaforma open-source che consente di creare, distribuire ed eseguire applicazioni all'interno di pacchetti detti container. È importante notare che quando si parla di Docker, di solito ci si riferisce a Docker Engine, il software che ospita i container. Docker utilizza la virtualizzazione a livello di sistema operativo (containerizzazione), in particolare utilizza alcune feature del Kernel Linux per garantire l'isolamento e il controllo dei pacchetti. Alcuni esempi di strumenti del Kernel che vengono utilizzati sono i cgroup e i namespace. I cgroups (control groups) sono una funzionalità del kernel linux che permette di limitare, monitorare e assegnare le risorse del sistema ai diversi processi in esecuzione. In questo modo, ogni container può essere configurato per utilizzare solo una certa quantità di risorse, evitando che un singolo container possa consumare tutte le risorse dell'host a discapito degli altri. I namespace, invece, sono un meccanismo che isola la visione che ha un'applicazione delle risorse di sistema. Grazie ai namespace, ogni container vede solo i processi, le interfacce di rete, i punti di mount e le variabili d'ambiente che gli sono stati assegnati, come se fosse l'unico sistema attivo sulla macchina. Questo garantisce un elevato livello di isolamento rispetto agli altri container e al sistema operativo. Questo approccio, basato sulla containerizzazione 2.1, rende i container molto più leggeri (nell'ordine di megabyte, contro i gigabyte di molte VM)[IBM docker] e più rapidi da avviare rispetto alle macchine virtuali tradizionali. L'efficienza ottenuta consente di eseguire molti più container sullo stesso hardware rispetto alle VM e di rendere lo sviluppo e il deployment delle applicazioni più agili e flessibili.

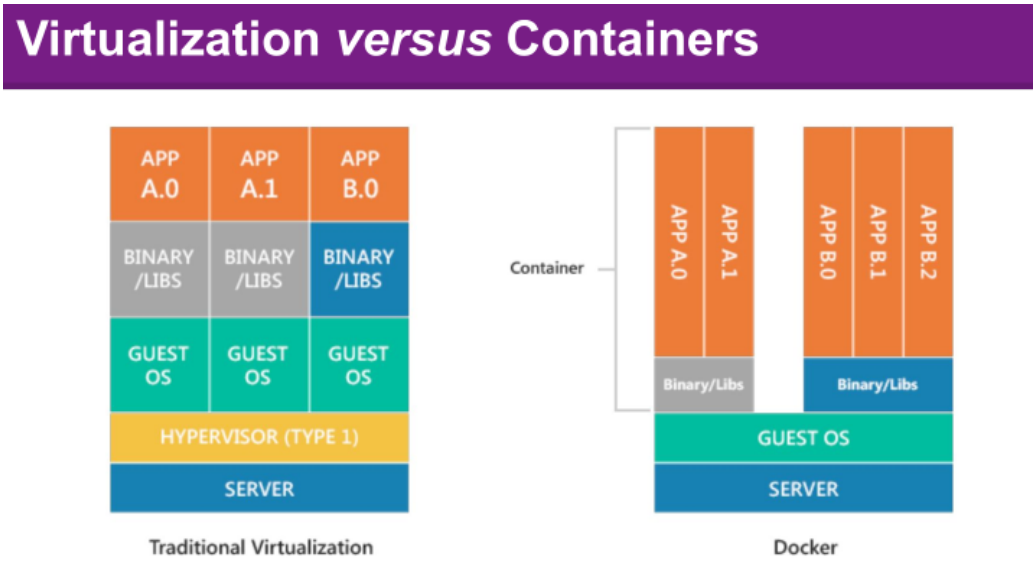

Figura 2.1: Confronto tra le macchine virtuali tradizionali e i container di Docker

2.1.1 Principali componenti

L'ecosistema Docker è composto da tre elementi principali:

- **Software**: Il cuore del sistema è il Docker daemon (dockerd), un processo che gestisce i container e le altre risorse Docker. Gli utenti interagiscono con il daemon tramite il Docker Client, ovvero un'interfaccia a riga di comando (docker) che comunica attraverso le API di Docker Engine.
- **Oggetti**: Gli oggetti Docker rappresentano le entità fondamentali per il funzionamento delle applicazioni containerizzate. I principali sono:
  - **Immagini**: template di sola lettura da cui vengono creati i container.
  - **Container**: ambienti isolati in cui vengono eseguite le applicazioni.
  - **Servizi**: permettono ai container di essere scalati su più Docker daemon creando così uno swarm (un insieme di di daemon cooperante tramite l'API di Docker).

  Gli oggetti di Docker verranno trattati con maggiore dettaglio nelle seguenti sottosezioni.
- **Registry**: I registry sono archivi centralizzati dove vengono conservate le immagini Docker. Possono essere pubblici o privati; il più noto è Docker Hub, il registry predefinito. I registry permettono di caricare e scaricare immagini.

2.1.2 Immagini e Dockerfile

Le applicazioni containerizzate in Docker vengono distribuite sotto forma di immagini Docker, ovvero template immutabili da cui si generano i container. Ogni immagine racchiude il file system e tutte le istruzioni necessarie per avviare l'applicazione all'interno di un container. La creazione di un'immagine è automatizzata tramite un file di configurazione testuale chiamato Dockerfile, in cui vengono specificati in modo dichiarativo tutti i passaggi necessari: dalla scelta dell'immagine di base (ad esempio una distribuzione Linux minimale), alla copia dei file dell'applicazione, fino all'installazione delle dipendenze e alla configurazione dell'ambiente (cartella di lavoro, utenti, variabili d'ambiente, ecc.). Questo approccio garantisce la riproducibilità e la coerenza dell'ambiente software, eliminando i tipici problemi di compatibilità tra sistemi diversi. Le immagini Docker possono inoltre essere versionate e facilmente condivise tramite i registry, favorendo la collaborazione e la distribuzione delle applicazioni.

2.1.3 Container

Un container Docker è un'istanza attiva di un'immagine: rappresenta un ambiente isolato, leggero e portatile che esegue un'applicazione con tutte le sue dipendenze, come definito nell'immagine da cui è stato creato. Tuttavia, nella pratica, molte applicazioni moderne richiedono l'interazione tra più servizi distinti, ciascuno eseguito in un proprio container. Per gestire e orchestrare in modo semplice insiemi di container che compongono un'applicazione complessa si utilizza Docker Compose, uno strumento che permette di definire e avviare configurazioni multi-container tramite un unico file YAML. Inoltre, poiché i container sono per loro natura effimeri e i dati scritti al loro interno vengono persi alla loro rimozione, Docker mette a disposizione i volumi: uno strumento pensato per garantire la persistenza dei dati e la condivisione delle informazioni tra container diversi o tra host e container.

2.1.4 Docker Compose

Docker Compose è uno strumento che permette di definire, configurare e gestire applicazioni composte da più container. Tramite un file in formato YAML (docker-compose.yml), è possibile descrivere tutti i servizi che compongono l'applicazione, specificando per ciascuno l'immagine da usare, le variabili d'ambiente, le reti, le dipendenze e i volumi associati. Questo approccio consente di avviare, arrestare o aggiornare l'intero stack applicativo con un singolo comando, semplificando la gestione di architetture complesse e migliorando la riproducibilità degli ambienti sia in fase di sviluppo che di produzione. Compose è particolarmente utile quando si desidera far lavorare insieme più servizi (ad esempio un database, un'applicazione backend e un frontend), ciascuno eseguito nel proprio container ma coordinati tra loro.

2.1.5 Volumi

I volumi in Docker rappresentano una soluzione per la persistenza dei dati nei container. Di default, i dati creati all'interno di un container sono temporanei e vengono eliminati quando il container viene rimosso. I volumi permettono di salvare dati in uno spazio dedicato e indipendente dal ciclo di vita dei

singoli container, garantendo che le informazioni rimangano disponibili anche dopo la terminazione o la ricreazione dei container. Inoltre, i volumi possono essere utilizzati per condividere dati tra più container o per facilitare l'integrazione con il sistema host. Questa caratteristica è fondamentale per applicazioni che richiedono la conservazione dei dati, come database, sistemi di caching o storage di file.

Esistono due tipi di volumi:

- **Bind Mount**: Consente di collegare una cartella o un file presente sul filesystem dell'host direttamente all'interno di uno o più container. Questo tipo di volume risulta particolarmente utile quando è necessario accedere a file specifici già esistenti o quando si dispone di spazio limitato sul disco gestito da Docker, poiché i dati vengono memorizzati direttamente nel percorso scelto sull'host.

- **Named Volume**: Si tratta di volumi gestiti direttamente da Docker, che vengono creati e organizzati in modo automatico dal Docker Engine all'interno di uno spazio dedicato dell'host. Sono ideali per la persistenza dei dati tra diverse esecuzioni dei container, e permettono una gestione più semplice e indipendente dal percorso fisico sul filesystem.

In generale, i bind mount sono consigliati quando si vuole controllare esattamente dove vengono salvati i dati o quando si ha poco spazio disponibile nell'area gestita da Docker, mentre i named volume offrono maggiore astrazione e portabilità nell'uso dei dati tra container diversi.

2.2 Apache Airflow

Airflow è una piattaforma open-source progettata per la gestione automatizzata dei flussi di lavoro. Consente di definire, pianificare e monitorare in modo programmatico i workflow, tramite una pratica interfaccia utente integrata. Creato originariamente da Airbnb, Airflow è sviluppato in Python ed è progettato secondo il paradigma del Configuration as Code.

La Configuration-as-Code (CaC) è un approccio che prevede la gestione delle configurazioni tramite codice, invece che attraverso modifiche manuali o strumenti proprietari. In pratica, le configurazioni vengono trattate come vero e proprio codice applicativo. Questo metodo offre diversi vantaggi: garantisce coerenza e ripetibilità nelle configurazioni, riduce il rischio di errori umani e semplifica il ripristino in caso di problemi. Inoltre, essendo gestite come codice, le configurazioni possono essere versionate tramite sistemi di controllo versione (come Git), permettendo di tracciare facilmente le modifiche e tornare a versioni precedenti se necessario.

Airflow sfrutta Python per definire i flussi, a differenza di altre piattaforme simili, che tipicamente utilizzano linguaggi markup come XML. Questo approccio permette agli sviluppatori di importare librerie o script esistenti nel loro workflow.

Ogni workflow viene rappresentato come un DAG (Directed Acyclic Graph), ovvero un grafo orientato e aciclico, dove i nodi rappresentano i task (le singole operazioni da eseguire) e le frecce rappresentano le dipendenze e l'ordine di esecuzione tra i task.

2.2.1 Concetti principali

Per comprendere il funzionamento di Apache Airflow, è fondamentale introdurre alcuni concetti chiave che costituiscono la base della piattaforma e ne determinano la logica operativa.

Tra questi i principali sono:

- **DAG**: è la struttura fondamentale che descrive un workflow, ossia una sequenza di operazioni da eseguire secondo una precisa logica di dipendenze. Un DAG definisce quali task devono essere eseguiti e in quale ordine, rappresentando graficamente e logicamente il flusso di lavoro in cui ogni nodo è un task e ogni freccia una dipendenza. Nel codice, un DAG è definito come un oggetto Python in cui si specificano parametri come:

  - ID e descrizione del DAG;
  - schedule_interval (frequenza di esecuzione, ad esempio giornaliera o mensile);

- start_date e altri parametri opzionali come ad esempio retries (quante volte può essere rieseguita una task);
- i task che lo compongono e le relative dipendenze (ad esempio, usando task1 ¿¿ task2).

La definizione di un DAG in Airflow consente di:

- Modellare pipeline di qualsiasi complessità,
- Visualizzare la struttura e lo stato delle esecuzioni tramite l'interfaccia grafica,
- Gestire in modo chiaro retry, fallback, e dipendenze tra task.

In seguito verrà mostrato un esempio di DAG:

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

with DAG(
    dag_id="esempio_dag",
    start_date=datetime(2025, 1, 1),
    schedule_interval="@daily",
    catchup=False
) as dag:
    task1 = PythonOperator(
        task_id="primo_task",
        python_callable=lambda: print("Hello Airflow!")
    )
    task2 = PythonOperator(
        task_id="secondo_task",
        python_callable=lambda: print("Secondo task!")
    )
    task1 >> task2
```

- **Task**: rappresenta l'unità base di esecuzione: ogni task corrisponde a una singola operazione o processo all'interno di un workflow. Le task vengono organizzate all'interno dei DAG e collegate tra loro tramite dipendenze che ne determinano l'ordine di esecuzione. Esistono due principali tipi di task, nella mia implementazione ho utilizzato solo gli Operator ma li citerò per fornire un maggiore contesto:
  - **Operator**: sono template di task predefiniti, pensati per svolgere operazioni comuni (ad esempio eseguire codice Python, eseguire comandi Bash, inviare email, ecc.). Sono il tipo di task più utilizzato. I principali sono PythonOperator, BashOperator, EmailOperator.
  - **Sensor**: sono una sottoclasse di operator. Sono progettati per attendere il verificarsi di un evento esterno.

  In Airflow, i concetti di Task e Operator sono in parte intercambiabili, ma è comunque utile pensarli come elementi distinti: in pratica, gli Operator e i Sensor rappresentano dei modelli (template), e ogni volta che ne istanzi uno all'interno di un DAG, stai effettivamente creando una Task. Ogni esecuzione di una task (detta Task Instance) attraversa diversi stati durante il proprio ciclo di vita:
  - **none**: la task non è ancora stata messa in coda perché le dipendenze non sono soddisfatte
  - **scheduled**: la task è stata programmata per l'esecuzione, avendo tutte le dipendenze soddisfatte
  - **queued**: la task è in attesa di essere eseguita da un worker

- **running**: la task è in esecuzione
- **success**: la task è terminata con successo
- **failed**: la task è fallita a causa di un errore
- **skipped**: la task è stata saltata (ad esempio per una branch condizionale)
- **upstream_failed**: una task a monte è fallita, impedendo l'esecuzione
- **up_for_retry**: la task è fallita, ma sono previsti tentativi di retry
- (altri stati specifici come restarting, up_for_reschedule, deferred, removed, etc.)

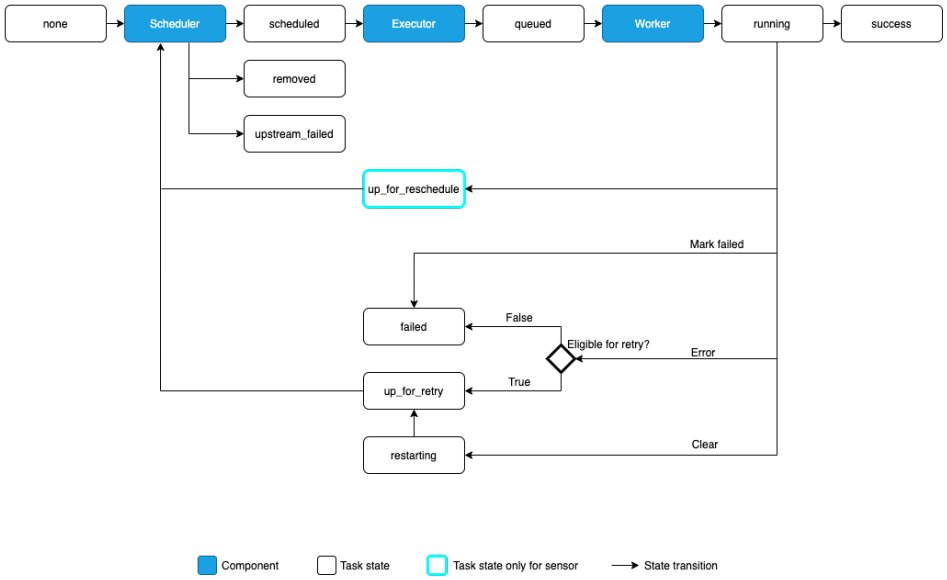

Figura 2.2: Diagramma di tutti i possibili stati di una task

Come si può vedere in figura 2.2, l'obiettivo ideale è che una task attraversi lo stato da none a success, passando per scheduled, queued e running. Per impostazione predefinita, una task viene eseguita solo se tutte le sue upstream (le task che la precedono nel flusso logico) sono completate con successo, ma Airflow offre anche strumenti avanzati di controllo del flusso (branching, trigger rules, ecc.).

2.2.2 Architettura

L'architettura di Airflow è formata da diversi componenti. Un'installazione minima di Apache Airflow è formata in questo modo:

- **Scheduler**: Il cuore del sistema, si occupa di pianificare l'esecuzione dei workflow (DAG) secondo la programmazione definita, individuando i task che devono essere eseguiti e inviandoli all'esecutore (executor). L'executor, che è una proprietà configurabile dello scheduler, gestisce l'esecuzione dei task e può lavorare in modalità locale o distribuita, a seconda delle esigenze e delle risorse disponibili.

- **Webserver**: Fornisce un'interfaccia web intuitiva che permette di visualizzare, monitorare e gestire i DAG e i task, analizzare i log, forzare l'esecuzione manuale dei workflow e diagnosticare eventuali problemi.

- **Cartella /dags**: Una directory locale (o su storage condiviso) in cui risiedono i file Python che definiscono i DAG. Lo scheduler esegue periodicamente la scansione di questa cartella per individuare nuovi workflow o aggiornamenti ai workflow esistenti.

- **Meta Database**: Un database relazionale (tipicamente PostgreSQL o MySQL) utilizzato da tutti i componenti di Airflow per tracciare lo stato delle esecuzioni dei DAG e dei task, memorizzare configurazioni, cronologia delle esecuzioni, log degli errori e altri metadati fondamentali.

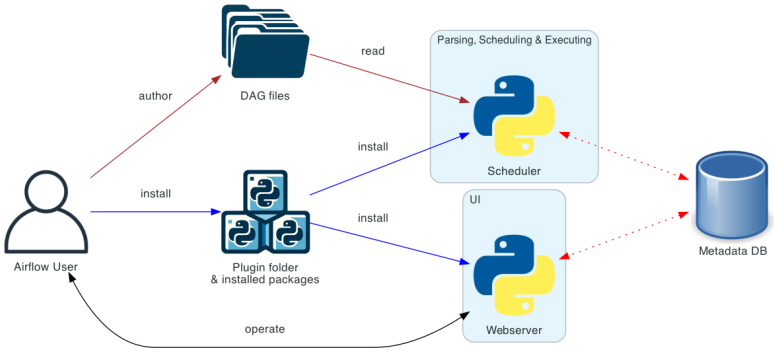

Figura 2.3: Fig 2.3 Diagramma del deployment minimo di Airflow

In figura 2.3 viene mostrata la configurazione più semplice di Airflow, in cui tutti i componenti principali (scheduler, webserver e database) vengono eseguiti sulla stessa macchina. In questo scenario si utilizza solitamente il LocalExecutor: scheduler e worker condividono lo stesso processo Python e i file dei DAG vengono letti direttamente dal filesystem locale. Anche l'interfaccia web (webserver) viene eseguita sulla stessa macchina. È importante far notare la differenza tra executor e worker: L'executor in Airflow determina la modalità e la logica con cui i task vengono messi in esecuzione, gestendo l'assegnazione dei task ai worker. Il worker è invece il processo che riceve i task dall'executor e li esegue.

2.2.3 Perché utilizzare Airflow?

Apache Airflow offre numerosi vantaggi nella gestione dei workflow. In primo luogo, la flessibilità della piattaforma consente di definire pipeline complesse come codice Python, integrandosi con praticamente qualsiasi tecnologia. Inoltre, l'automazione tramite la schedulazione integrata permette di eseguire i workflow a intervalli prestabiliti, e la gestione delle dipendenze garantisce che i task vengano eseguiti nell'ordine corretto secondo le relazioni definite. Airflow è altamente scalabile, potendo operare da un singolo processo locale fino a cluster distribuiti capaci di gestire carichi di lavoro ingenti. Offre anche un'ottima osservabilità: tramite un'interfaccia web si possono visualizzare chiaramente lo stato dei DAG, i log dei task e altri indicatori, facilitando il monitoraggio e il troubleshooting dei processi. Infine, sono previsti meccanismi di retry automatico e robusta gestione degli errori, così che eventuali fallimenti vengano intercettati e i task possono essere ritentati automaticamente per garantire la continuità delle pipeline.

2.3 Monitoraggio e metriche

Monitorare significa raccogliere, analizzare e visualizzare metriche e log per osservare il comportamento di un sistema nel tempo. Nell'ambito dell'informatica e dell'ingegneria del software, il monitoring si concentra principalmente sul tracciamento di metriche predefinite e indicatori chiave di performance (KPI) con l'obiettivo di rilevare deviazioni dal comportamento atteso. Tipicamente vengono utilizzati strumenti di monitoring per raccogliere dati come utilizzo di CPU, memoria, tempo di risposta e tassi

di errore. Questi dati vengono analizzati in tempo reale e, al superamento di determinate soglie, il sistema può inviare degli alert per consentire una risposta tempestiva a possibili problemi.

Nel mio progetto, il monitoring della pipeline dati è stato realizzato tramite l'integrazione di strumenti open source come StatsD, Prometheus e Grafana. Airflow esporta automaticamente metriche relative allo stato delle task, alle risorse utilizzate (CPU/RAM) e alla durata delle esecuzioni, che vengono poi raccolte e visualizzate in dashboard interattive. Questo consente di identificare rapidamente task fallite o anomalie di performance e intervenire in modo reattivo, garantendo affidabilità ed efficienza nella gestione dei workflow automatizzati.

2.3.1 StatsD

STATSD È UN DEMONE

Bibliografia