Immagine che contiene Elementi grafici, schermata, grafica, testo

Il contenuto generato dall'IA potrebbe non essere corretto.

Pietro Coloretti – Leonardo Gennaioli – Iacopo Sbalchiero

Scalable and Reliable Services – Ingegneria Informatica M

Università degli studi di Bologna - A.A. 2024-2025

Sommario

[Abstract 3](#_Toc203171852)

[Requisiti Funzionali 3](#_Toc203171853)

[Requisiti Non Funzionali 4](#_Toc203171854)

[Tecnologie Utilizzate 5](#_Toc203171855)

[Architettura del Sistema 6](#_Toc203171856)

[Orchestratore 6](#_Toc203171857)

[Formatter 8](#_Toc203171858)

[Anonymizer 10](#_Toc203171859)

# Abstract

AnonimaData ha avuto l'obiettivo di progettare e implementare un servizio scalabile e affidabile per l'anonimizzazione di dataset, garantendo la protezione della privacy dei dati sensibili. La piattaforma permette agli utenti di caricare dataset in formato CSV o JSON e di applicare algoritmi di anonimizzazione all'avanguardia come k-anonymity, l-diversity, t-closeness e differential privacy, configurandone dinamicamente i parametri.

AnonimaData è stato concepito per essere generalizzabile a dataset tabulari arbitrari, supportando schemi, tipi di dati e configurazioni di colonne diverse. Il servizio offre un'interfaccia utente web intuitiva che facilita la gestione del processo, incluse funzionalità di anteprima del dataset anonimizzato. I dati anonimizzati vengono archiviati sia in un database per un accesso strutturato sia come file CSV nello storage a oggetti, garantendo disponibilità e facilità di download.

L'intera infrastruttura del sistema è stata distribuita su Google Cloud Platform (GCP), con la gestione e il provisioning delle risorse interamente automatizzati tramite Terraform, assicurando scalabilità, affidabilità e riproducibilità. Infine, il servizio integra un sistema di autenticazione utente (Google OAuth 2.0) per garantire un accesso sicuro e controllato. AnonimaData rappresenta una soluzione completa per la gestione della privacy dei dati, fornendo uno strumento robusto e flessibile per la conformità normativa e la protezione delle informazioni personali.

# Requisiti Funzionali

Le funzionalità principali che il sistema deve fornire agli utenti includono:

* **Caricamento Dataset:** Gli utenti devono poter caricare dataset in formato **CSV** e **JSON**.
* **Selezione Algoritmo di Anonimizzazione:** Il sistema deve presentare una lista di algoritmi di anonimizzazione disponibili, tra cui:
  + k-anonimato
  + l-diversità
  + t-vicinanza
  + Privacy Differenziale
* **Configurazione Parametri Algoritmo:** Per ogni algoritmo selezionato, gli utenti devono poter configurare dinamicamente i parametri specifici (es. valore di *k* per k-anonimato, *epsilon* per la Privacy Differenziale).
* **Anteprima Dataset Anonimizzato:** Il sistema deve permettere agli utenti di visualizzare un'anteprima del dataset anonimizzato prima del salvataggio definitivo.
* **Archiviazione Dataset Anonimizzato:** I dataset anonimizzati devono essere archiviati in due modalità distinte:
  + **In un database:** Per consentire interrogazioni e gestione strutturata.
  + **Come file CSV:** Archiviati in uno storage a oggetti (es. Google Cloud Storage) per download e riutilizzo.
* **Interfaccia Utente Web:** Deve essere disponibile un'interfaccia web intuitiva e user-friendly per gestire l'intero processo di anonimizzazione.
* **Autenticazione Utente:** Il sistema deve supportare l'autenticazione degli utenti tramite un provider esterno **(Google OAuth 2.0)**.
* **Generalizzabilità Dataset:** La piattaforma deve essere in grado di elaborare dataset tabulari arbitrari, supportando diversi schemi, tipi di dati e configurazioni di colonne senza richiedere modifiche al codice.

# Requisiti Non Funzionali

Il sistema deve rispettare alcuni vincoli principali, oltre che avere determinate qualità:

* **Scalabilità:** Il servizio deve essere in grado di gestire un aumento del numero di utenti e di dataset di grandi dimensioni senza degrado significativo delle prestazioni. Questo implica l'utilizzo di servizi cloud scalabili e architetture distribuite.
* **Affidabilità:** Il sistema deve essere resiliente ai guasti e garantire la disponibilità continua del servizio. I dati devono essere archiviati in modo ridondante e devono essere previsti meccanismi di recupero in caso di errore.
* **Performance:** Il tempo di elaborazione per l'anonimizzazione dei dataset deve essere ragionevole, anche per dataset di grandi dimensioni. L'interfaccia utente deve essere reattiva.
* Sicurezza:
  + **Protezione dei Dati:** Tutti i dati, sia quelli originali che quelli anonimizzati, devono essere protetti da accessi non autorizzati.
  + **Crittografia:** I dati sensibili devono essere crittografati sia a riposo che in transito.
  + **Conformità Privacy:** Il sistema deve aderire ai principi di protezione della privacy, garantendo che i dati anonimizzati non possano essere re-identificati.
* **Usabilità:** L'interfaccia utente web deve essere intuitiva, facile da navigare e offrire una buona esperienza d'uso agli utenti.
* **Deployabilità:** L'intero sistema deve essere completamente deployabile su Google Cloud Platform (GCP) e la gestione dell'infrastruttura deve essere automatizzata tramite **Terraform**.
* **Robustezza:** Il sistema deve gestire errori e input non validi, fornendo feedback chiari all'utente.
* **Portabilità (Dati):** I dataset anonimizzati scaricabili (CSV) devono essere in un formato standard e facilmente importabile in altre applicazioni.

# Tecnologie Utilizzate

**Google Cloud Platform (GCP)**: L'intero sistema è progettato per essere distribuito su GCP, fornendo l'infrastruttura sottostante necessaria per la scalabilità, l'affidabilità e la gestione dei servizi. GCP offre un'ampia gamma di servizi, inclusi calcolo (es. Cloud Run per i microservizi), storage (Cloud Storage per i file CSV e un database per i metadati/dati anonimizzati), e messaggistica (Pub/Sub) che sono fondamentali per l'interconnessione dei componenti del backend.

**Terraform**: Per la gestione e il provisioning dell'infrastruttura su GCP, viene utilizzato Terraform. Questo strumento di Infrastructure as Code (IaC) permette di definire, versionare e deployare l'intera infrastruttura cloud in modo dichiarativo e automatizzato, garantendo coerenza, riproducibilità e facilità di gestione dell'ambiente.

**Docker**: Ogni servizio backend è containerizzato utilizzando Docker. Questo permette di isolare l'ambiente di esecuzione di ciascun servizio, garantendo che le dipendenze siano gestite in modo coerente e che i servizi possano essere deployati in qualsiasi ambiente compatibile con Docker.

**Python (e Flask per API REST)**: Il linguaggio di programmazione principale per lo sviluppo dei servizi backend è Python. In particolare, il framework Flask viene utilizzato per costruire le API REST che espongono le funzionalità dei servizi (come l'upload di file, la richiesta di anonimizzazione e il recupero dello stato). Python, con le sue librerie ricche per la manipolazione dei dati (es. Pandas per i dataset), è ideale per le operazioni di analisi e anonimizzazione dei dati.

**React (per Frontend)**: React è una scelta popolare per la costruzione di Single Page Applications (SPA) dinamiche e interattive, ideale per la gestione del processo di upload, configurazione degli algoritmi e visualizzazione dei risultati. Comunica con il backend tramite le API REST esposte da Flask.

**Firebase**: Firebase è utilizzato per l'autenticazione degli utenti, in particolare tramite **Firebase Authentication (con Google OAuth 2.0)**. L’utilizzo di Firebase permette di delegare la gestione dell'identità e dell'autorizzazione a un servizio robusto e scalabile.

# Architettura del Sistema

L'architettura di AnonimaData è modulare e distribuita, basata su microservizi che comunicano principalmente tramite Google Pub/Sub. Questa impostazione garantisce scalabilità, resilienza e manutenibilità. I componenti chiave includono il Frontend, l'Orchestratore, il Formatter e l'Anonymizer. Uno schema riassuntivo è visibile in Figura 1.

Immagine che contiene testo, schermata, diagramma, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

Figura – Schema dell’architettura di AnonimaData

## Orchestratore

L'Orchestratore è il cuore del backend di AnonimaData e si pone come punto di ingresso principale per le interazioni degli utenti e la gestione del flusso di lavoro complessivo. Le sue responsabilità principali sono le seguenti:

* **Interfaccia con il Frontend (API REST):** L'Orchestratore espone una serie di endpoint API REST che consentono al frontend di interagire con il sistema. Questi includono funzionalità per:
  + **Upload e avvio analisi (*/upload\_and\_analyze*):** Riceve i file caricati dagli utenti, li codifica in Base64 e pubblica un messaggio sul topic *DATA\_UPLOAD\_REQUESTS* di Pub/Sub per avviare il processo di analisi da parte del Formatter.
  + **Recupero dello stato delle operazioni (*/get\_status/<job\_id>*):** Permette al frontend di interrogare lo stato di un lavoro specifico (analisi o anonimizzazione) tramite un *job\_id*.
  + **Richiesta di anonimizzazione (*/request\_anonymization*):** Riceve le configurazioni e i parametri scelti dall'utente per l'anonimizzazione (metodo, parametri, selezioni colonne), recupera i dati preprocessati e i metadati associati al *job\_id*, e pubblica un messaggio sul topic *ANONYMIZATION\_REQUESTS* di Pub/Sub, destinato all'Anonymizer.
  + **Recupero elenco file e statistiche *(/get\_files*):** Fornisce al frontend una lista dei file caricati dall'utente, inclusi il loro stato, le dimensioni, le informazioni sull'anonimizzazione e un URL per il download del file completo.
  + **Download file anonimizzati (*/download/<job\_id>*):** Gestisce la richiesta di download di un dataset anonimizzato completo, recuperandolo dalla memoria (in questo caso, da un dizionario in-memory *job\_status\_map*) e inviandolo come file CSV.
  + **Rimozione dei file anonimizzati (*/delete/<job\_id>*):** Permette all’utente di eliminare tutti i dati relativi ad un dataset, che esso sia stato anonimizzato o che sia ancora in fase di anonimizzazione.
* **Gestione dello Stato dei Job:** L'Orchestratore comunica con un database PostgreSQL e salva uno stato dettagliato di ogni operazione (job). Questo include lo stato corrente (es. *uploaded, analyzed, anonymization\_requested, completed, error*), l'avanzamento, i dettagli, i timestamp, i dati preprocessati, i metadati, i dati anonimizzati e i campioni anonimizzati.
* **Verifica Permessi Utente (Firebase Authentication):** Ogni richiesta API che implica l'accesso a risorse utente è protetta da un decoratore *@firebase\_auth\_required*. Questo decoratore estrae il token di autenticazione Firebase dall'header Authorization, lo verifica tramite *firebase\_admin.auth.verify\_id\_token()*, e se valido, imposta l'ID utente (*request.user\_id*) sulla richiesta. Questo garantisce che solo gli utenti autenticati e autorizzati possano accedere ai propri dati e processi. L'Orchestratore verifica che l'utente che effettua la richiesta sia lo stesso che ha avviato il job.
* **Comunicazione Inter-Servizio (Pub/Sub):** L'Orchestratore funge da hub di comunicazione Pub/Sub. Pubblica messaggi sui topic di Pub/Sub (es. *DATA\_UPLOAD\_REQUESTS* e *ANONYMIZATION\_REQUESTS*) per delegare le operazioni di analisi e anonimizzazione ad altri servizi. Riceve anche notifiche da Pub/Sub su endpoint dedicati (es. */receive\_analysis\_results, /receive\_anonymization\_results, /receive\_error\_notifications*) quando il Formatter o l'Anonymizer completano le loro operazioni o riscontrano errori. Questo approccio asincrono e basato su eventi disaccoppia i servizi, migliorando la scalabilità e la robustezza.
* **Gestione Errori:** L'Orchestratore è configurato per ricevere notifiche di errore dal Pub/Sub (tramite il topic *ERROR\_NOTIFICATIONS* e l'endpoint */receive\_error\_notifications*). In caso di errore in qualsiasi fase (analisi o anonimizzazione), lo stato del job viene aggiornato a *'error'*, fornendo dettagli sullo stage e il messaggio di errore.
* **Preparazione Dati per l'Anonimizzazione:** Quando riceve una richiesta di anonimizzazione, l'Orchestratore recupera i DataFrame di Pandas contenenti i dati preprocessati e i metadati precedentemente salvati all’interno del database. Questi DataFrame vengono convertiti in stringhe CSV/JSON e codificati in Base64 prima di essere inviati come payload nel messaggio Pub/Sub all'Anonymizer. Questo assicura che l'Anonymizer riceva i dati necessari in un formato pronto per l'elaborazione.
* **Persistenza Dati (PostgreSQL e Google Cloud Storage):** L'Orchestratore è responsabile della gestione dello stato persistente di tutti i job di anonimizzazione e dei dati correlati. Nel nostro ambiente di produzione, la persistenza dei dati è gestita tramite database relazionale (PostgreSQL) e Storage ad Oggetti (Google Cloud Storage).

## Formatter

Il Formatter è un componente cruciale dell'architettura che si attiva dopo il caricamento iniziale di un dataset. Il suo compito principale è duplice: **standardizzazione del formato e analisi approfondita del dataset**.

* **Ingresso Dati (Pub/Sub):** Il Formatter non riceve le richieste direttamente dal frontend, ma si sottoscrive a un topic Pub/Sub (presumibilmente *DATA\_UPLOAD\_REQUESTS* come indicato in *google\_pubsub\_manager.py*). Quando l'Orchestratore riceve un nuovo upload di dataset, pubblica un messaggio su questo topic contenente l'ID del job, il nome del file e il contenuto del file codificato in Base64. Il Formatter, ricevendo questo messaggio tramite un endpoint push, decodifica il payload e inizia l'elaborazione.
* Standardizzazione del Dataset:
  + Il servizio è progettato per gestire dati in vari formati iniziali.
  + La funzione *read\_dataset\_for\_web* è responsabile di leggere il contenuto del file (che l'Orchestratore ha inviato codificato in Base64) e di trasformarlo in un DataFrame di Pandas. Questo passaggio è fondamentale per standardizzare il formato dei dati, indipendentemente dal formato di input originale, in una struttura uniforme e facilmente manipolabile per le successive fasi.
  + Il dataset viene quindi convertito in un **CSV ben formattato**. Questa conversione interna garantisce che tutti i dati, una volta analizzati, siano in un formato coerente, facilitando il passaggio al servizio Anonymizer.
* Analisi delle Colonne e Rilevamento dei Tipi di Dato:
  + Dopo aver standardizzato il dataset, il Formatter esegue un'analisi approfondita di ogni colonna. La funzione *structure\_dataset* è il fulcro di questa operazione.
  + L'obiettivo è identificare automaticamente il tipo di dato presente in ciascuna colonna. Tali tipi di dato possono essere:
    - **Numerico:** Numeri interi, decimali, float.
    - **Testuale:** Stringhe generiche.
    - **Data/Ora:** Date, timestamp, orari.
    - **Categorico:** Campi con un numero limitato di valori discreti (es. sesso, stato civile).
    - **Identificativi Sensibili (Quasi-Identificatori):** Campi che, se combinati, potrebbero portare alla re-identificazione (es. età, CAP, etnia).
    - **Attributi Sensitivi:** Campi che contengono informazioni private (es. salario, diagnosi medica).
    - **Email, Numeri Telefonici, Indirizzi, Codici Fiscali, ecc.:** Riconoscimento di pattern specifici che indicano dati personali altamente sensibili.
* Salvataggio dei Dati Preprocessati e Metadati:
  + Una volta completata l'analisi, il DataFrame preprocessato e i metadati generati (informazioni sulle colonne, tipi di dati, ecc.) vengono salvati.
  + Tali informazioni verranno quindi comunicate all’Orchestratore che procederà al salvataggio in memoria persistente, in particolare all’interno del **Google Cloud Storage** viene salvato il dataset formattato, mentre i metadati pertinenti vengono salvati in **PostgreSQL**, associati all'ID del job.
* **Notifica all'Orchestratore (Pub/Sub):** Dopo aver completato l'elaborazione e aver salvato i dati preprocessati e i metadati, il Formatter pubblica un messaggio su un topic Pub/Sub (*ANALYSIS\_RESULTS*). Questo messaggio contiene l'ID del job, lo stato di successo, e le informazioni sui metadati generati. L'Orchestratore si sottoscrive a questo topic per ricevere la notifica e aggiornare lo stato del job, rendendo le informazioni disponibili al frontend.
* **Gestione Errori:** Se si verifica un errore durante l'elaborazione (es. file corrotto, formato non supportato), il Formatter cattura l'eccezione e pubblica un messaggio sul topic *ERROR\_NOTIFICATIONS*, consentendo all'Orchestratore di aggiornare lo stato del job a "errore" e notificare l'utente.

## Anonymizer

L'Anonymizer è il servizio backend dedicato all'applicazione degli algoritmi di protezione della privacy sui dataset. Riceve le richieste dall'Orchestratore, utilizzando i dati preprocessati e i metadati, genera il dataset anonimizzato finale.

* Ingresso Dati e Richiesta di Anonimizzazione (Pub/Sub):
  + Similmente al Formatter, l'Anonymizer non riceve richieste dirette dal frontend. Si sottoscrive al topic Pub/Sub *ANONYMIZATION\_REQUESTS*.
  + Quando l'Orchestratore riceve una richiesta di anonimizzazione dall'utente (incluso l'algoritmo scelto e i suoi parametri), costruisce un messaggio Pub/Sub contenente l'ID del job, il dataset preprocessato (come stringa CSV/JSON codificata in Base64), i metadati associati (anche questi codificati), il metodo di anonimizzazione selezionato e i relativi parametri. Questo messaggio viene poi pubblicato sul topic *ANONYMIZATION\_REQUESTS*.
  + L'Anonymizer, tramite un endpoint push, riceve il messaggio, decodifica il payload JSON e ne estrae il *job\_id*, il *filename*, il *file\_content\_base64* (che rappresenta il dataset preprocessato), il *metadata\_content\_base64*, il *method* e i *params*.
* Processo di Anonimizzazione (Core Logic in anonymizer.py):
  + Il servizio *anonymization\_service.py* invoca il modulo *anonymizer.py* per l'effettiva logica di anonimizzazione, tramite la funzione *process\_anonymization*.
  + anonymizer.py è il cuore algoritmico del sistema. Le sue responsabilità includono:
    - **Lettura Dati e Metadati:** Decodifica il contenuto del dataset preprocessato (CSV) e dei metadati, trasformandoli in DataFrame Pandas utilizzabili.
    - **Implementazione degli Algoritmi:** Contiene le implementazioni robuste dei diversi algoritmi di anonimizzazione richiesti:
      * **k-anonimato:** Garantisce che ogni record nel dataset anonimizzato sia indistinguibile da almeno k-1 altri record rispetto a un set di attributi quasi-identificatori.
      * **l-diversità:** Estende il k-anonimato, garantendo che all'interno di ogni gruppo di k record, ci siano almeno l valori distinti per gli attributi sensibili, mitigando gli attacchi di omogeneità.
      * **t-vicinanza:** Ulteriore miglioramento rispetto a l-diversità, assicura che la distribuzione dei valori degli attributi sensibili all'interno di ogni gruppo di k record sia vicina alla distribuzione globale, contrastando gli attacchi di skewness.
      * **Privacy Differenziale:** Aggiunge rumore calibrato ai dati per garantire che la presenza o assenza di un singolo individuo nel dataset non influenzi significativamente l'output di una query, fornendo una garanzia di privacy molto forte.
    - **Utilizzo dei Metadati:** Il modulo *anonymizer.py* fa un uso estensivo dei metadati generati dal Formatter, in particolare i tipi di dato e le indicazioni su quali colonne sono quasi-identificatori o attributi sensibili. La classe *Anonymizer* nel file *anonymizer.py* è inizializzata con il DataFrame e i metadati, e può anche considerare selezioni dell’utente (is\_quasi\_identifier, should\_anonymize), che dovrebbero essere presenti nei metadati estesi. Questo permette una configurazione dinamica e intelligente dell'anonimizzazione.
    - **Generalizzazione e Soppressione:** Gli algoritmi applicano tecniche come la generalizzazione (sostituzione di valori specifici con intervalli o categorie più ampie, es. "20-30 anni" invece di "25 anni") e la soppressione (rimozione completa di valori sensibili o interi record).
    - **Gestione di Diversi Tipi di Dato:** Le implementazioni degli algoritmi sono in grado di trattare correttamente diversi tipi di dato (numerici, categorici, testuali) applicando le tecniche di anonimizzazione appropriate per ciascuno.
  + Salvataggio del Dataset Anonimizzato: Una volta completato il processo di anonimizzazione, il dataset risultante (un nuovo DataFrame Pandas) viene convertito in una stringa CSV per essere inviata all’Orchestratore.
* Notifica all'Orchestratore (Pub/Sub):
  + Al termine dell'anonimizzazione (successo o fallimento), l'Anonymizer pubblica un messaggio sul topic *ANONYMIZATION\_RESULTS* (o *ERROR\_NOTIFICATIONS* in caso di fallimento).
  + Questo messaggio include il *job\_id*, lo stato (es. 'completed'), i dettagli del risultato (es. percorso GCS del file anonimizzato completo, un campione di dati anonimizzati per l'anteprima frontend, le statistiche sull'anonimizzazione), il metodo e i parametri utilizzati, e il timestamp di completamento. L'Orchestratore consumerà questo messaggio per aggiornare lo stato interno del job e notificare il frontend.
* **Gestione Errori:** Come gli altri servizi, l'Anonymizer implementa una robusta gestione degli errori. Qualsiasi eccezione durante il processo di anonimizzazione (es. parametri non validi, problemi di elaborazione dati) viene catturata, e un messaggio di errore viene pubblicato sul topic *ERROR\_NOTIFICATIONS*, indirizzando l'errore all'Orchestratore per la notifica all'utente.

# Flusso di anonimizzazione

Il processo di anonimizzazione dei dati tramite la piattaforma AnonimaData si articola nei seguenti passaggi, corrispondenti allo schema semplificato in Figura 2. Alcune componenti tecniche, come l'autenticazione tramite Firebase, i controlli sui token e le operazioni sui bucket e database, sono omesse nel disegno per chiarezza visiva.

1. Autenticazione utente: l’utente accede alla webapp (in esecuzione su Cloud Run) e viene reindirizzato a Google per l’autenticazione OAuth tramite Firebase.
2. Emissione del token: Firebase rilascia un ID token, che il frontend utilizza per firmare tutte le richieste successive.
3. Upload file: l’utente carica un file da anonimizzare attraverso l’interfaccia web. Il frontend allega il token all’invio.
4. Verifica del token e registrazione del job: il backend dell’Orchestratore (anch’esso su Cloud Run) riceve la richiesta, verifica l’autenticazione presso Firebase, assegna un job\_id univoco, lo associa all’UUID dell’utente e registra tutto nel database.
5. Pubblicazione richiesta di analisi: l’Orchestratore invia una richiesta su Pub/Sub, includendo il file (in base64) e il job\_id.
6. Intervento del Formatter: un servizio formatter (Cloud Run) sottoscrive il canale Pub/Sub, riceve il messaggio, decodifica il file, lo analizza (es. identificazione colonne, tipi di dato, struttura tabellare) e lo uniforma.
7. Risposta del Formatter: al termine, pubblica su un secondo canale Pub/Sub i file prodotti (dataset uniformato e descrizione delle colonne) e il job\_id.
8. Polling stato da parte del frontend: mentre il processo è in corso, il frontend interroga periodicamente l’Orchestratore passando il job\_id, per conoscere lo stato corrente del job.
9. Memorizzazione intermedia: quando l’Orchestratore riceve la risposta del Formatter, salva temporaneamente il file uniformato nel bucket, aggiorna le colonne nel database e imposta lo stato del job come "analizzato".
10. Configurazione dell’anonimizzazione: il frontend riceve lo stato aggiornato e visualizza i dettagli sulle colonne all’utente, che seleziona quali dati anonimizzare e con quali algoritmi e parametri.
11. Pubblicazione richiesta di anonimizzazione: il frontend invia la configurazione all’Orchestratore, che la pubblica (insieme al file formattato) su un terzo canale Pub/Sub.
12. Esecuzione degli Anonymizer: uno o più servizi Anonymizer (Cloud Run) ricevono la richiesta e applicano le trasformazioni richieste alle colonne selezionate (es. k-anonimato, mascheramento, hashing).
13. Polling stato anonimizzazione: anche durante questa fase, il frontend continua a interrogare periodicamente l’Orchestratore per aggiornamenti sul job.
14. Pubblicazione del risultato finale: al termine, l’Anonymizer pubblica il file anonimizzato e il relativo job\_id su un quarto canale Pub/Sub.
15. Aggiornamento finale dell’Orchestratore: l’Orchestratore riceve il file anonimizzato, lo salva nel bucket, elimina il file formattato, aggiorna lo stato nel database e genera un’anteprima dei dati anonimizzati.
16. Restituzione dell’esito: alla successiva interrogazione del frontend, l’Orchestratore restituisce lo stato "completato" e l’anteprima.
17. Download finale: l’utente può visualizzare il risultato ed eventualmente scaricare il file anonimizzato. Il dataset rimane disponibile sulla piattaforma per future consultazioni o download.

Lo schema evidenzia l’architettura basata su microservizi Cloud Run orchestrati da Pub/Sub e supportati da Firebase, database e bucket GCP, assicurando scalabilità, tracciabilità dei job e separazione delle fasi (upload, analisi, anonimizzazione).

Immagine che contiene schermata, testo, design

Il contenuto generato dall'IA potrebbe non essere corretto.

Figura – Flusso di anonimizzazione di un dataset