**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 ri-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'Orchestrator, il Formatter e l'Anonymizer.

**Orchestrator**

L'Orchestrator 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'Orchestrator 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\_analysis\_status/<job\_id>, /get\_anonymization\_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>/full*):** 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.
* **Gestione dello Stato dei Job (In-Memory):** L'Orchestrator mantiene uno stato dettagliato di ogni operazione (job) in un dizionario in-memory (*job\_status\_map*). 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'Orchestrator verifica che l'utente che effettua la richiesta sia lo stesso che ha avviato il job.
* **Comunicazione Inter-Servizio (Pub/Sub):** L'Orchestrator 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'Orchestrator 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'Orchestrator recupera i DataFrame di Pandas contenenti i dati preprocessati e i metadati precedentemente salvati in memoria (nella *job\_status\_map*). 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'Orchestrator è responsabile della gestione dello stato persistente di tutti i job di anonimizzazione e dei dati correlati. Nel nostro ambiente di produzione, la persistenza effettiva 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'Orchestrator 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'Orchestrator 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.
  + Come discusso in precedenza, nel nostro ambiente di produzione questi dati vengono caricati su **Google Cloud Storage** (per il DataFrame) mentre i metadati pertinenti vengono salvati in **PostgreSQL/Cloud SQL**, associati all'ID del job.
* **Notifica all'Orchestrator (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'Orchestrator 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'Orchestrator 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'Orchestrator e, 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'Orchestrator 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.
  + Questa stringa CSV (e potenzialmente un campione per l'anteprima) viene poi caricata su **Google Cloud Storage** in un bucket dedicato ai dataset anonimizzati.
* **Notifica all'Orchestrator (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'Orchestrator 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'Orchestrator per la notifica all'utente.