Progetto Sistemi Distribuiti e Cloud Computing
MLOps and AWS

Martina Giacobbe, mat. 263662

June 2025

Contents

Introduzione

Il presente progetto nasce con l'obiettivo di sviluppare una pipeline automatizzata per il ciclo di vita completo di un modello di Machine Learning, sfruttando i servizi cloud di Amazon Web Services (AWS) in un'ottica completamente serverless e in linea con i paradigmi Function-as-a-Service (FaaS). Tale approccio consente una gestione semplificata dell'infrastruttura, favorendo scalabilità, modularità e manutenibilità.

La pipeline è progettata per ricevere un dataset, processarlo automaticamente, addestrare un modello predittivo e rendere disponibile un endpoint HTTP per l'inferenza in tempo reale. Tutto ciò è orchestrato tramite trigger automatici su Amazon S3 e Lambda Functions, con container Docker su Amazon ECR per garantire ambienti di esecuzione personalizzati.

1 Analisi dei Requisiti

1.1 Requisiti Funzionali

I requisiti funzionali specificano le capacità che il sistema deve offrire per rispondere alle necessità dell'utente:

- **Trigger automatico su S3:** Il caricamento di un file CSV nel bucket S3 nella directory `raw/` genera un evento che innesca automaticamente l'esecuzione della funzione Lambda di preprocessing. Questo garantisce un'elaborazione immediata dei dati in ingresso senza necessità di intervento manuale.
- **Preprocessing del dataset:** La funzione Lambda dedicata esegue la pulizia dei dati tramite:
  - Rimozione di righe con valori mancanti;
  - Encoding delle variabili categoriche tramite LabelEncoder;
  - Normalizzazione delle feature numeriche attraverso MinMaxScaler.

  I dati trasformati vengono poi salvati nella directory `preprocessed/` dello stesso bucket S3.
- **Addestramento del modello:** Una seconda funzione Lambda legge i dati preprocessati, addestra un modello (RandomForestClassifier di scikit-learn) e ne salva la serializzazione in formato `.pkl` nella directory `model/`.
- **Servizio di inferenza:** Una terza Lambda, accessibile da endpoint HTTP tramite Amazon API Gateway, accetta dati in formato JSON, applica il medesimo preprocessing e restituisce la predizione.
- **Architettura completamente serverless:** Tutti i componenti sono distribuiti tramite Lambda Functions, S3, ECR e API Gateway. Non sono necessari server persistenti.

1.2 Requisiti Non Funzionali

- **Scalabilità:** La soluzione serverless permette di scalare automaticamente in base alla quantità di dati e alle richieste di inferenza.
- **Disponibilità:** L'utilizzo di servizi gestiti garantisce alta disponibilità e tolleranza ai guasti.
- **Prestazioni:** Il preprocessing e l'inferenza sono ottimizzati in termini di tempo grazie all'uso di container specializzati.
- **Sicurezza:** Le policy IAM limitano l'accesso ai soli componenti autorizzati.
- **Modularità e manutenibilità:** Ogni funzione è contenuta in un container indipendente con le sue dipendenze.
- **Portabilità:** L'intera infrastruttura è facilmente replicabile in ambienti locali.
- **Tracciabilità:** Tutti i dati e i modelli sono salvati su S3 con struttura coerente e versionabile.

2 Architettura e Componenti

2.1 Amazon S3

Amazon S3 (Simple Storage Service) è un servizio web di memorizzazione offerto da Amazon Web Services. È un servizio di archiviazione di oggetti scalabile, senza limiti di quantità e con una durevolezza del dato garantita al 99%. I dati vengono archiviati in contenitori chiamati bucket, i quali possono ospitare qualsiasi tipo di dati, da file di testo a immagini e video. Il bucket `ssdc-bucket` è strutturato come segue:

- `raw/` - input manuale del dataset iniziale;
- `preprocessed/` - output del preprocessing automatico;
- `model/` - modelli addestrati in formato serializzato.

2.2 Amazon ECR

Amazon Elastic Container Registry (Amazon ECR) è un servizio di registro di immagini Docker completamente gestito da AWS. Questo servizio è sicuro, scalabile e affidabile, permettendo di archiviare e gestire facilmente le immagini dei container per le applicazioni. Amazon ECR supporta anche gli standard Open Container Initiative (OCI) e Docker Registry HTTP API V2, rendendo possibile l'uso dei comandi di Docker CLI per interagire con il registro. Inoltre, il servizio supporta i repository privati con autorizzazioni basate su risorse utilizzando AWS Identity and Access Management (IAM), garantendo che solo gli utenti specificati o le istanze Amazon EC2 possano accedere ai repository e alle immagini.

Ogni funzione Lambda ha bisogno di dipendenze Python non sempre disponibili nel runtime standard (come `scikit-learn`, `pandas`, `joblib`). Per questo motivo si è scelto di utilizzare Docker containers caricati su Amazon ECR. Ogni cartella Lambda contiene:

- `requirements.txt`;
- `Dockerfile` per l'ambiente;
- `Codice Lambda` in Python;
- `deploy.sh`, un file ad hoc che effettua l'accesso all'account AWS e al servizio ECR per mezzo di variabili d'ambiente, carica l'immagine Docker generata su un apposito spazio in ECR ed infine crea la funzione lambda su AWS Lambda; effettua cioè push su ECR e deploy della funzione su AWS Lambda.

2.3 AWS Lambda

AWS Lambda è un servizio di elaborazione serverless che consente di eseguire codice senza dover effettuare il provisioning o gestire server. Con AWS Lambda, tutto ciò che è necessario fare è fornire il proprio codice in uno dei runtime di linguaggio supportati da Lambda. AWS Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutta l'amministrazione delle risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning e la scalabilità automatica della capacità e la registrazione. Quando il codice non è in esecuzione, non viene addebitato alcun costo.

Tre funzioni Lambda principali:

1. **Preprocessing:** attivata da trigger S3;
2. **Model Training:** invocata dal preprocessing via `boto3`;
3. **Inference:** invocata da client via HTTP POST.

Iniziando dalla funzione preprocessing.py, essa sarà la prima funzione ad essere chiamata nella pipeline a seguito del caricamento del dataset nel bucket. Di fatto, il caricamento del dataset deve fungere da evento trigger per l'attivazione della funzione. Per permettere ciò, è stato necessario specificare manualmente le condizioni del trigger da interfaccia grafica. Per garantire una maggiore flessibilità e chiarezza di utilizzo, si è scelto di sviluppare manualmente il percorso della pipeline inserendo nella funzione lambda una porzione di codice che, invocando la libreria boto3, effettua automaticamente una chiamata alla lambda successiva.

La funzione lambda preprocessing, dunque, una volta caricato il dataset pulito sul bucket, procede a chiamare la funzione di model training per l'addestramento.

La funzione lambda model training avrà quindi il compito di addestrare il modello e caricarlo nell'apposita sottocartella del bucket s3.

Infine, l'ultima lambda è la funzione inference la quale non viene invocata come parte della pipeline ma direttamente da una richiesta POST http lato client. Di fatto, per questa funzione è stato necessario definire un access point pubblico tramite l'URL della funzione stessa. Al fine di fare ciò, sono stati modificati i permessi di accesso alla funzione e sono stati rilassati i vincoli CORS per la comunicazione tra host.

2.4 IAM

AWS Identity and Access Management (IAM) è un servizio web che aiuta a controllare in modo sicuro l'accesso alle risorse AWS. Con IAM, è possibile gestire le autorizzazioni che controllano le risorse AWS a cui gli utenti possono accedere. IAM offre autenticazione e autorizzazione per i servizi AWS, valutando se una richiesta AWS è consentita o negata.

Per permettere comunicazioni sicure, è stato definito un ruolo IAM `uploadDB` che consente l'accesso a S3 solo alle Lambda autorizzate. Le policy sono definite in modo restrittivo per massimizzare la sicurezza.

2.5 Interfaccia Utente Streamlit

Per facilitare l'utilizzo del servizio di inferenza anche da parte di utenti non tecnici, è stata sviluppata un'interfaccia utente semplice ed efficace utilizzando **Streamlit**, un framework Python open-source pensato per la creazione rapida di dashboard e web app interattive per progetti di data science e machine learning.

L'applicazione Streamlit consente di compilare manualmente tutti i campi richiesti dal modello di classificazione, corrispondenti alle feature presenti nel dataset Adult Income (es. età, istruzione, ore settimanali di lavoro, stato civile, ecc.). Una volta inseriti i dati, l'interfaccia genera automaticamente una `POST request` verso l'endpoint pubblico associato alla funzione Lambda di inferenza, sfruttando la libreria `requests` di Python.

Figure 1: Screenshot dell'interfaccia Streamlit per la compilazione e l'invio della richiesta

Funzionalità principali

L'app presenta le seguenti funzionalità:

- Interfaccia grafica intuitiva basata su moduli di input (text box, dropdown, slider);
- Validazione automatica dei dati immessi;
- Invio automatizzato della richiesta al servizio di inferenza;
- Visualizzazione del risultato in tempo reale (es. classe predetta: >50K o 50K).

Vantaggi dell'approccio Streamlit

- Nessuna necessità di scrivere codice lato utente;
- Interazione fluida anche da browser web senza installazioni aggiuntive;
- Facilità di estensione per aggiungere visualizzazioni o parametri avanzati;
- Possibilità di deployment come servizio online (es. tramite Streamlit Cloud o container Docker su EC2).

L'aggiunta di questa componente migliora notevolmente l'usabilità complessiva del progetto, permettendo anche a stakeholder o analisti di interagire con il sistema predittivo senza accedere direttamente ai servizi AWS.

3 Flusso di Esecuzione della Pipeline

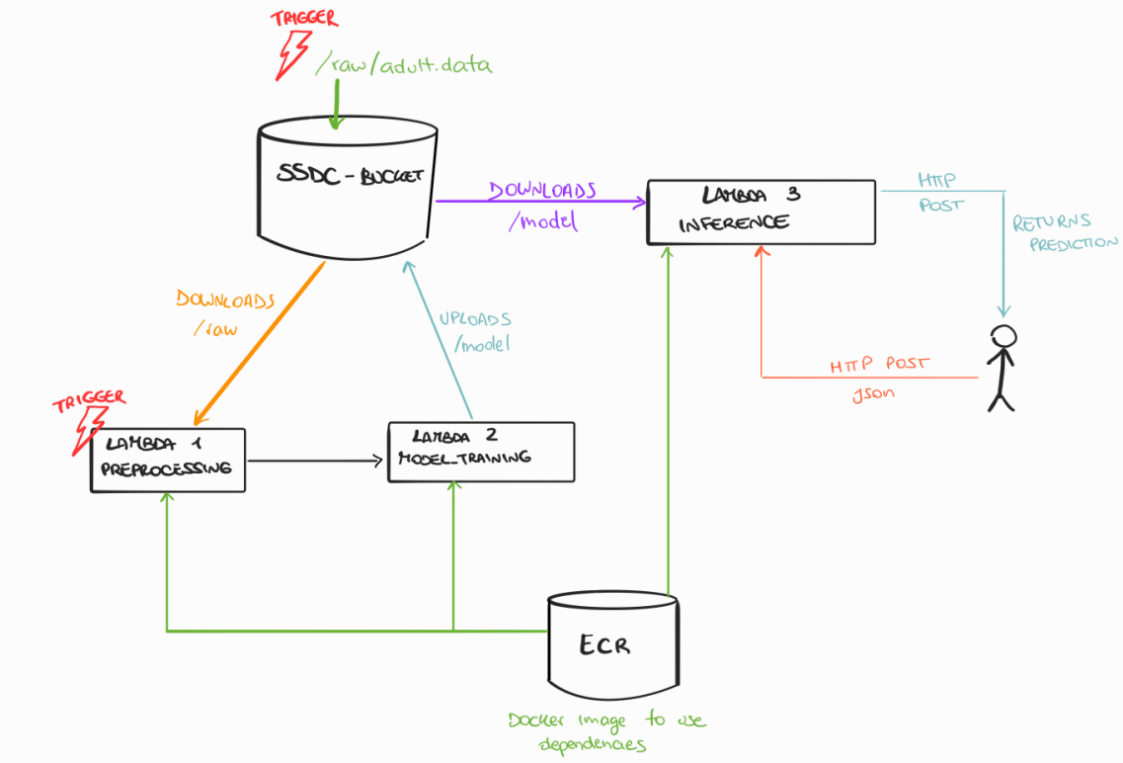

Figure 2: Flusso di Esecuzione del Sistema

1. Un utente carica il file CSV nella directory `raw/` del bucket.
2. Questo attiva la funzione `preprocessing.py` che effettua la pulizia dei dati.
3. I dati preprocessati vengono caricati in `preprocessed/`.
4. La funzione `model_training.py` viene quindi invocata via `boto3`.
5. Dopo l'addestramento, il modello viene salvato in `model/`.
6. Il client invia una richiesta HTTP POST a un endpoint pubblico.
7. La funzione `inference.py` esegue la predizione e restituisce il risultato.

4 Verifica del Rispetto dei Requisiti

Requisito	Stato	Verifica
Trigger su S3	Verificato	Upload su `raw/` attiva `preprocessing.py`
Preprocessing automatico	Verificato	Pulizia, encoding e normalizzazione implementate
Addestramento modello	Verificato	Addestramento tramite RandomForestClassifier e salvataggio `.pkl`
Inferenza via HTTP	Verificato	Funzione esposta pubblicamente tramite API Gateway
Sicurezza IAM	Verificato	Ruolo `uploadDB` assegnato correttamente
Persistenza e tracciabilità	Verificato	Tutti i file versionati su S3 con path distinti

5 Sviluppi Futuri

Il progetto attuale implementa una pipeline serverless per il machine learning efficace, modulare e scalabile. Tuttavia, vi sono numerose direzioni di sviluppo che potrebbero arricchirne le funzionalità, migliorarne l'affidabilità, o estenderne l'applicabilità a scenari più complessi e reali. Di seguito vengono elencati e descritti alcuni possibili sviluppi futuri, sia in ambito tecnico che architetturale.

5.1 Integrazione con strumenti di CI/CD

Attualmente, il deployment delle funzioni Lambda e dei container ECR avviene tramite script Bash (deploy.sh). Una naturale evoluzione sarebbe l'integrazione con una pipeline CI/CD basata su strumenti come AWS CodePipeline, GitHub Actions o GitLab CI. Questo permetterebbe:

- Deploy automatico al push del codice;
- Validazione automatica dei container;
- Controllo delle versioni tramite tagging automatico su ECR;
- Test end-to-end prima del deployment in ambiente di produzione.

5.2 Monitoraggio e logging avanzato

Un altro sviluppo importante riguarda il monitoraggio dell'intera pipeline. Potrebbero essere integrati strumenti come Amazon CloudWatch, AWS X-Ray o Datadog per ottenere:

- Tracciamento dei tempi di esecuzione e delle metriche delle funzioni Lambda;
- Rilevamento automatico di anomalie o malfunzionamenti;
- Logging centralizzato per l'analisi delle performance e dei costi.

5.3 Gestione avanzata dei modelli

Attualmente viene gestita una singola versione del modello. In futuro, si potrebbe integrare un sistema di versioning dei modelli tramite Amazon SageMaker Model Registry, così da:

- Tenere traccia delle performance delle diverse versioni del modello;
- Effettuare rollback in caso di regressioni;
- Automatizzare test di A/B testing tra modelli.

5.4 Validazione e gestione dei dati

La pipeline potrebbe essere arricchita con meccanismi di:

- Validazione dello schema del dataset (es. via AWS Glue Data Catalog o Great Expectations);
- Controllo di qualità del dato (outlier, valori nulli non previsti, codifiche errate);
- Tracciabilità del lineage dei dati.

5.5 Supporto a diversi modelli e automazione dell'ottimizzazione

Attualmente il sistema addestra un unico modello. Una naturale evoluzione sarebbe:

- Permettere la selezione dinamica tra più modelli (Random Forest, XGBoost, ecc.);
- Automatizzare l'hyperparameter tuning (via AWS SageMaker o librerie come Optuna);
- Integrare AutoML per dataset generici.

5.6 Front-end e interfaccia utente

Ad oggi l'interfaccia utente è limitata all'invocazione della funzione `inference` tramite web server locale. Potrebbe essere utile:

- Fornire un'interfaccia utente per la configurazione dei parametri di training o la selezione del modello;
- Offrire una dashboard di analytics per l'utente finale.
- Effettuare il deploy su un access point pubblico tramite dominio.

5.7 Generalizzazione della pipeline

Infine, si potrebbe rendere la pipeline general-purpose, parametrizzandola per supportare dataset e problemi di machine learning differenti. In quest'ottica, la pipeline potrebbe diventare un framework riutilizzabile in diversi contesti aziendali o accademici.

5.8 Compliance e auditability

In scenari produttivi, specialmente in ambito finanziario o sanitario, sarà essenziale includere:

- Meccanismi di audit e compliance (es. GDPR);
- Logging di tutte le inferenze con timestamp e identificativo utente;
- Versioning trasparente di dati, codice e modelli per garantire la riproducibilità.

In sintesi, la pipeline attuale rappresenta una solida base, ma può crescere in molte direzioni fino a costituire un sistema completamente maturo per la messa in produzione di soluzioni ML moderne.

6 Conclusioni

Il progetto ha dimostrato con successo come sia possibile implementare una pipeline completa di Machine Learning attraverso un'architettura interamente serverless, sfruttando al massimo i servizi offerti da Amazon Web Services (AWS). A partire dall'upload di un dataset grezzo in un bucket S3, fino all'inferenza su nuovi dati tramite un'API HTTP, l'intero flusso è stato automatizzato mediante funzioni Lambda orchestrate opportunamente e supportate da container Docker customizzati tramite Elastic Container Registry (ECR).

Tra i principali risultati raggiunti, possiamo evidenziare:

- La modularità della pipeline, che consente di isolare le diverse fasi (preprocessing, training, inferenza), facilitando debugging, manutenzione e riuso del codice;
- La portabilità delle funzioni Lambda grazie all'utilizzo di container Docker, che risolvono il problema della compatibilità tra ambienti e garantiscono il corretto caricamento delle dipendenze;
- L'efficace utilizzo di S3 come storage centralizzato, che assicura persistenza e tracciabilità dei dati e dei modelli;
- L'interfaccia utente realizzata in Streamlit, che rende il sistema accessibile anche a utenti non tecnici e favorisce l'adozione reale della soluzione.

In termini di requisiti funzionali, il sistema soddisfa pienamente le specifiche iniziali: ogni fase della pipeline è attivata da eventi precisi, il dataset viene preprocessato correttamente, il modello viene addestrato e reso disponibile per l'inferenza, e infine le richieste POST possono essere effettuate facilmente dall'esterno. Anche i requisiti non funzionali sono stati tenuti in considerazione, in particolare per quanto riguarda scalabilità, sicurezza, modularità e portabilità.

Il progetto rappresenta un esempio concreto di come le tecnologie FaaS e serverless possano semplificare l'implementazione e la gestione di soluzioni ML complesse, eliminando la necessità di provisioning manuale e rendendo il sistema altamente scalabile e resiliente per scenari di produzione reali.

In sintesi, il sistema progettato rappresenta una solida base per lo sviluppo di pipeline ML moderne, robuste e facilmente scalabili, offrendo un equilibrio efficace tra semplicità architetturale, efficienza operativa e potenziale evolutivo.