MlOps, AWS

Umberto Domenico Ciccia 263844

May 31, 2025

Contents

1 MLOps **3**
- 1.1 Cosa si intende per MlOps? . . . 3
- 1.2 Perché MlOps . . . 3
- 1.3 Quali sono i principi del MLOps? . . . 4
  - 1.3.1 Controllo delle versioni . . . 4
  - 1.3.2 Automazione . . . 5

2 Tecnologie utilizzate **6**
- 2.1 AWS . . . 6
- 2.2 Terraform . . . 7
- 2.3 Kubernetes . . . 8
- 2.4 Helm . . . 8
- 2.5 Bash . . . 9
- 2.6 GitHub Actions . . . 9

3 MLOps su AWS con Glue e SageMaker **10**
- 3.1 Introduzione . . . 10
- 3.2 Obiettivi . . . 12
- 3.3 Dataset . . . 13
- 3.4 Architettura . . . 13
  - 3.4.1 Panoramica Generale . . . 13
  - 3.4.2 ETL con AWS Glue . . . 13
  - 3.4.3 Orchestrazione con EventBridge e Lambda . . . 14
  - 3.4.4 Pipeline MLOps su SageMaker . . . 14
  - 3.4.5 API di Servizio e Frontend . . . 14
  - 3.4.6 CI/CD Pipeline per il Frontend . . . 15
  - 3.4.7 Deploy Kubernetes con Ingress . . . 15
- 3.5 Deployment con Terraform . . . 15

3.5.1 CI/CD GitOps per l'Infrastruttura (IaC) 16
3.5.2 Continuous Delivery su Kubernetes 16
3.5.3 Kubernetes Helm chart 16
3.6 Vantaggi dell'Architettura 17
3.7 Considerazioni Tecniche 17
3.8 Interfaccia Utente con Streamlit 18
3.9 Problematiche Affrontate 18

4 Conclusioni **19**
4.1 Possibili Sviluppi Futuri 19
4.2 Riferimenti 19

Chapter 1

MLOps

1.1 Cosa si intende per MlOps?

Le MlOps sono un insieme di pratiche che automatizzano e semplificano i workflow e le implementation di modelli di machine learning (ML). Il machine learning e l'intelligenza artificiale (IA) sono funzionalità fondamentali che possono essere implementate per risolvere problemi complessi del mondo reale e offrire valore ai clienti. MlOps è una pratica che unifica lo sviluppo (Dev) di applicazioni di ML con l'implementazione e le operazioni (Ops). Le aziende possono utilizzare le pratiche MLOps per automatizzare e standardizzare i processi durante il ciclo di vita dei modelli di ML. Questi processi includono lo sviluppo del modello, il test, l'integrazione, il rilascio e la gestione dell'infrastruttura.

1.2 Perché MlOps

Ad alto livello, per intraprendere il ciclo di vita del machine learning, l'aziende deve in genere iniziare con la preparazione dei dati. Vengono recuperati dati di diversi tipi da varie origini e vengono eseguite attività come aggregazione, pulizia dei duplicati e attivita di preprocessing.

Successivamente, i dati vengono utilizzati per addestrare e convalidare il modello di ML. È quindi possibile implementare il modello addestrato e convalidato come servizio di previsione al quale altre applicazioni possono accedere tramite API.

L'analisi esplorativa dei dati spesso richiede di sperimentare modelli dif-

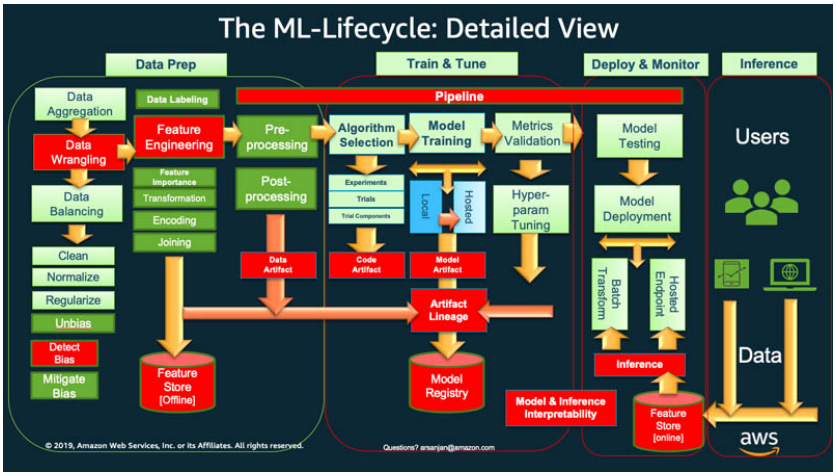


Figure 1.1: MlOps

ferenti, fino a quando la versione migliore del modello non è pronta per l'implementazione. Porta a implementazioni delle versioni del modello e a rilasci di versioni dei dati frequenti. Il monitoraggio degli esperimenti e la gestione della pipeline di addestramento del ML sono fasi essenziali prima che le applicazioni possano integrare o utilizzare il modello nel loro codice.

Il MLOps è fondamentale per gestire in modo sistematico e simultaneo il rilascio di nuovi modelli di ML con modifiche al codice dell'applicazione e ai dati. Un'implementazione MLOps ottimale tratta le risorse di ML in modo simile ad altre risorse software dell'ambiente di integrazione e distribuzione continua (CI/CD). I modelli di ML vengono implementati insieme alle applicazioni e ai servizi che utilizzano e a quelli che se ne avvalgono come parte di un processo di rilascio unificato.

1.3 Quali sono i principi del MLOps?

1.3.1 Controllo delle versioni

Questo processo prevede il monitoraggio delle modifiche nelle risorse di machine learning in modo da poter riprodurre i risultati e ripristinare le versioni precedenti, se necessario. Ogni specifica di modello o codice di addestramento di ML passa attraverso una fase di revisione del codice. Ciascuna è dotata di versioni per rendere riproducibile e verificabile l'addestramento dei modelli

di ML.

La riproducibilità in un flusso di lavoro di ML è importante in ogni fase, dall'elaborazione dei dati all'implementazione del modello di ML. Significa che ogni fase dovrebbe produrre risultati identici a fronte dello stesso input.

1.3.2 Automazione

È possibile automatizzare varie fasi della pipeline di machine learning per garantire ripetibilità, coerenza e scalabilità. Ciò include le fasi dall'importazione dei dati, la pre-elaborazione, l'addestramento dei modelli e la convalida fino all'implementazione.

L'automazione è più efficiente con l'infrastruttura come codice (IaC). È possibile utilizzare strumenti per definire e gestire l'infrastruttura. Questo contribuisce a garantire che sia riproducibile e possa essere implementato in modo coerente in vari ambienti.

Chapter 2

Tecnologie utilizzate

In questo capitolo vengono descritte le principali tecnologie e i servizi impiegati per realizzare la pipeline MLOps end-to-end su AWS, il deploy del frontend su Kubernetes e l'automazione tramite CI/CD e GitOps.

2.1 AWS

Amazon Web Services (AWS) ha costituito il fulcro dell'architettura cloud del progetto. In particolare, sono stati utilizzati i seguenti servizi:

- **Amazon S3**: storage di oggetti per la memorizzazione del dataset grezzo (bucket "raw-data") e dei dati preprocessati (bucket "processed-data"). Ogni upload in S3 innesca eventi per la pipeline.
- **AWS Glue**: servizio serverless di ETL utilizzato per la fase di preprocessing dei dati. Un workflow Glue, avviato da una funzione Lambda, esegue lo script `pre_processing.py` per pulire, normalizzare e salvare i dati in un bucket S3 dedicato.
- **Amazon EventBridge**: bus di eventi che rileva automaticamente l'arrivo di nuovi file nei bucket S3. Viene configurato per innescare funzioni Lambda sia in ingresso (upload raw) sia in uscita (upload processed), garantendo l'orchestrazione della pipeline.
- **AWS Lambda**: funzioni serverless che coordinano l'avvio dei workflow Glue. Queste funzioni, scritte in Pytho.

- **Amazon SageMaker**: piattaforma ML fully managed utilizzata per creare e orchestrare pipeline di training e deployment del modello. La SageMaker Pipeline include:
  - *TrainingPreprocessingStep*: ulteriore trasformazione dei dati tramite lo script `training_preprocessing.py`.
  - *TrainingStep*: addestramento di un modello di regressione lineare basato su XGBoost.
  - *ModelRegisterStep*: registrazione del modello nel Model Registry di SageMaker.
  - *ModelDeployStep*: deployment dell'endpoint in real-time per inferenze REST.

2.2 Terraform

Terraform è stato impiegato per la definizione e il provisioning dell'infrastruttura cloud in modalità Infrastructure as Code (IaC). Il repository `iac/` include moduli Terraform modulari e riutilizzabili per ciascun servizio AWS:

- `terraform-aws-s3`: definisce i bucket S3 per dati raw e processed, assicura versioning e politiche di lifecycle.
- `terraform-aws-glue`: crea database, crawler e workflow Glue, configurando i job di ETL con risorse e ruoli IAM minimi necessari.
- `terraform-aws-eventbridge`: stabilisce le regole EventBridge che monitorano gli upload nei bucket S3 e instradano gli eventi alle Lambda Functions.
- `terraform-aws-lambda`: deploya le funzioni Lambda, caricando il codice Python packeggiato e assegnando permessi IAM granulati.
- `terraform-aws-sagemaker`: definisce le componenti della SageMaker Pipeline, includendo pipeline definition, pipeline execution role, model registry e endpoint di deploy.

Per ogni modulo sono incluse variabili di input configurabili (region, environment, instance_type, ecc.) e output per concatenare le risorse. La pipeline

GitOps utilizza le GitHub Actions per eseguire `terraform init`, `terraform plan` e `terraform apply` in modo automatizzato ad ogni push sul ramo principale del repository `iac/`.

2.3 Kubernetes

Kubernetes è stato scelto per la gestione del container del frontend Streamlit, con i seguenti componenti principali:

- **Deployment**: definisce il numero di repliche del pod che esegue l'immagine Docker del frontend, risolvendo automaticamente eventuali crash tramite rolling update.
- **Service**: espone il Deployment internamente nel cluster su porta TCP (es. 8501 per Streamlit).
- **ConfigMap & Secret**: forniscono parametri di configurazione e segreti (es. endpoint SageMaker, chiavi API) al container senza rideployare l'immagine.

Kubernetes garantisce scalabilità orizzontale, resilienza e facilità di rollout dei nuovi container.

2.4 Helm

Helm è il package manager per Kubernetes utilizzato per semplificare il deploy delle risorse del frontend. È stato sviluppato un **Helm chart custom** che include:

- `Chart.yaml`: metadati (nome, versione, descrizione).
- `values.yaml`: valori di default configurabili (nome immagine, tag, port, replica count, risorse CPU/RAM, configurazioni Ingress).
- `templates/deployment.yaml`: template parametrizzato per il Deployment del container Streamlit.
- `templates/service.yaml`: template per il Service di tipo LoadBalancer.

- `templates/_helpers.tpl`: definizioni di helper functions per la generazione di nomi delle risorse e altre utilità.

Gli utenti finali possono eseguire il deploy con un singolo comando. In questo modo, il chart gestisce automaticamente l'aggiornamento dei pod, la creazione dei service e dell'ingress, rendendo il deployment ripetibile e configurabile tramite GitOps.

2.5 Bash

La Bash è stata impiegata per creare script di automazione locale.

2.6 GitHub Actions

Le GitHub Actions sono state integrate per realizzare pipeline CI/CD e GitOps sia per il codice del frontend sia per l'infrastruttura Terraform. Le azioni principali sono:

- **Frontend CI/CD Workflow** (`.github/workflows/frontend.yml`):
  1. `on: push` sul ramo `main` nella cartella `frontend/`.
  2. Check-out del codice e setup di Docker Buildx.
  3. Build e push dell'immagine Docker (tag `${GITHUB_SHA}`).
- **IaC GitOps Workflow** (`.github/workflows/iac.yml`):
  1. `on: push` sul ramo `main` nella cartella `iac/`.
  2. Setup di Terraform CLI e AWS CLI.
  3. `terraform init` per scaricare provider e moduli.
  4. `terraform plan` con output in file `plan.out`.
  5. Condizionale: se `plan` non contiene cambiamenti, il job termina; altrimenti, prosegue con `terraform apply -auto-approve`.

Questi workflow garantiscono l'automazione completa del ciclo di vita dell'applicazione e dell'infrastruttura, riducendo al minimo gli interventi manuali e mantenendo versioning e tracciabilità in Git.

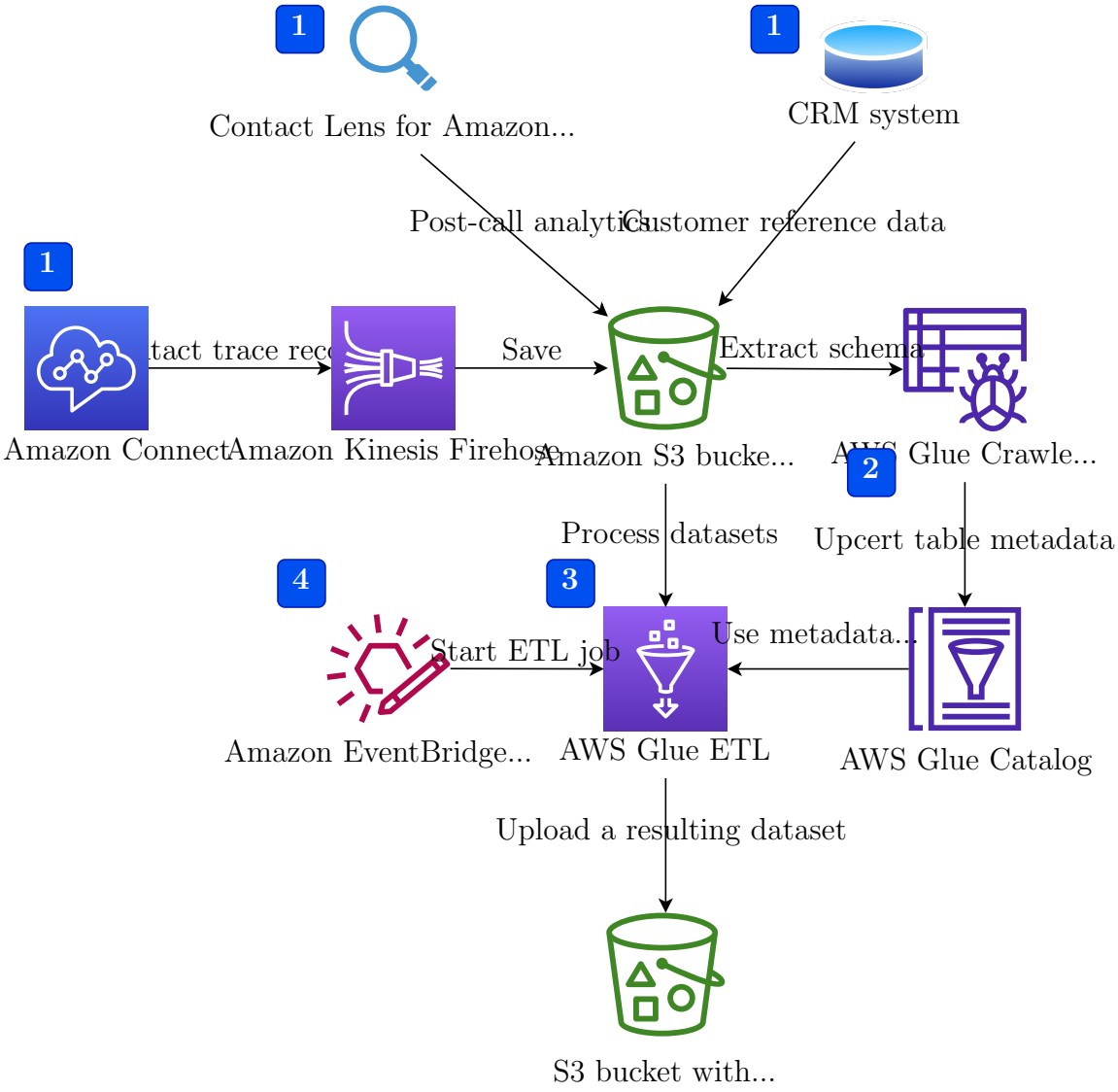
Chapter 3

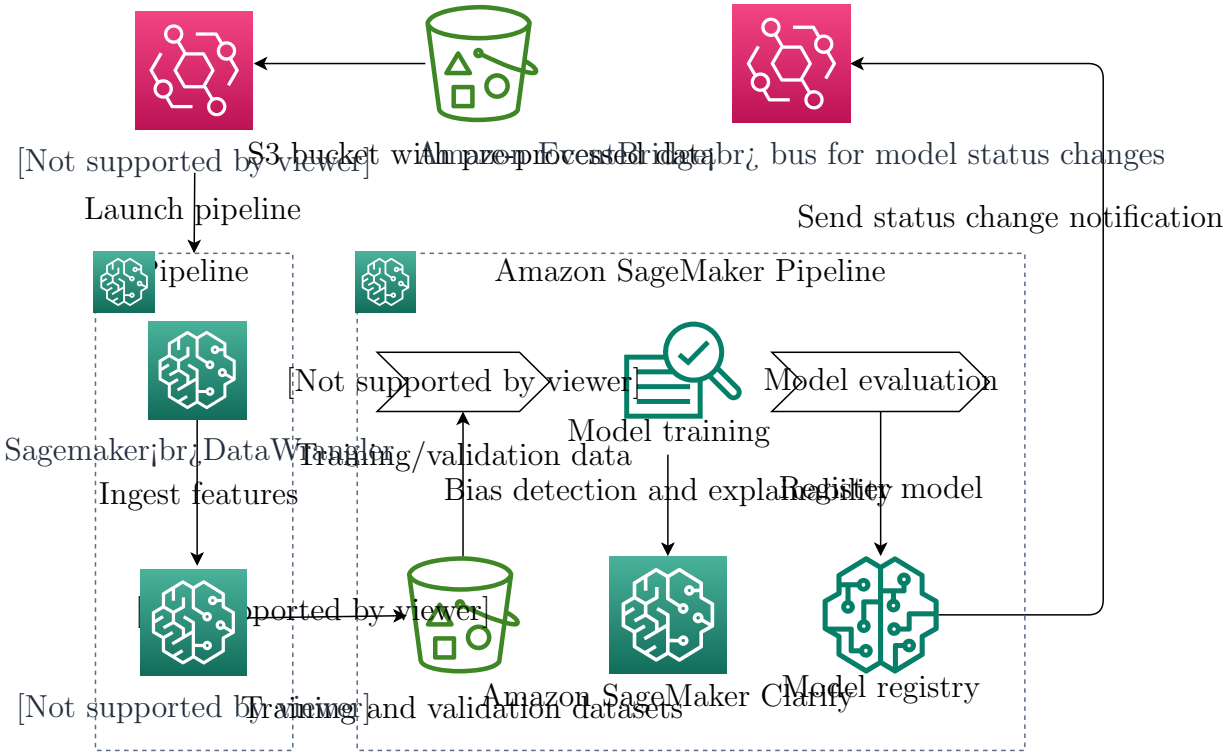
MLOps su AWS con Glue e SageMaker

3.1 Introduzione

Nel contesto dell'informatica moderna, l'integrazione tra machine learning e sistemi cloud distribuiti ha assunto un ruolo centrale. Questo progetto ha come obiettivo la realizzazione di una pipeline MLOps (Machine Learning Operations) completa su AWS, capace di gestire il ciclo di vita di un modello di regressione lineare basato sul dataset California Housing.

L'architettura proposta sfrutta diversi servizi AWS come Glue, Lambda, EventBridge e SageMaker per l'automazione dei processi di preprocessing, addestramento, deployment e monitoraggio. La pipeline è orchestrata tramite eventi e funzioni serverless, garantendo una soluzione scalabile, modulare e interamente Infrastructure as Code grazie a Terraform. Inoltre, è stato sviluppato un frontend in Streamlit per l'interazione utente deployato su un cluster k8s locale (locale a causa gli elevati costi di EKS).


1
Contact Lens for Amazon...
1
CRM system
Post-call analytics
Customer reference data
1
Amazon Connect
Amazon Kinesis Firehose
Save
Amazon S3 bucke...
Extract schema
AWS Glue Crawle...
2
Process datasets
Upcert table metadata
4
3
Start ETL job
Use metadata...
Amazon EventBridge...
AWS Glue ETL
AWS Glue Catalog
Upload a resulting dataset
S3 bucket with...



3.2 Obiettivi

- Automatizzare il processo di preprocessing tramite AWS Glue.
- Innescare automaticamente l'addestramento del modello tramite EventBridge e Lambda.
- Eseguire il training, la registrazione e il deployment del modello con SageMaker.
- Fornire un'interfaccia utente semplice per effettuare previsioni con Streamlit.
- Gestire l'intera infrastruttura con Terraform, in linea con le best practice DevOps.

3.3 Dataset

Il dataset "California Housing" contiene dati provenienti dal censimento del 1990, includendo caratteristiche come:

- Mediana del reddito
- Età media degli edifici
- Numero medio di stanze
- Popolazione per area
- Coordinate geografiche

Lo scopo è prevedere il valore medio delle abitazioni in un'area, rendendolo ideale per un modello di regressione lineare.

3.4 Architettura

3.4.1 Panoramica Generale

L'architettura è composta da quattro macro-componenti:

1. Pipeline ETL (Extract, Transform, Load)
2. Pipeline MLOps (Training e Deployment)
3. API di Serving del Modello
4. Frontend Streamlit deployato su un cluster k8s

3.4.2 ETL con AWS Glue

1. Input: Upload del file grezzo in un bucket S3.
2. Trigger: Evento su S3 rilevato da EventBridge.
3. Azione: La Lambda invoca un workflow Glue.
4. Elaborazione: Il job Glue applica il preprocessing (rimozione outlier, encoding, scaling).

5. Output: Scrittura dei dati preprocessati in formato parquet su un bucket S3 separato.

Script utilizzato: pre_processing.py

3.4.3 Orchestrazione con EventBridge e Lambda

Due regole di EventBridge coordinano il flusso di lavoro:

1. ETL Trigger: Avvio del Glue workflow alla ricezione di un file.
2. Training Trigger: Avvio della pipeline SageMaker dopo l'upload del file preprocessato.

Lambda agisce da ponte tra il rilevamento evento e l'esecuzione dei servizi AWS.

3.4.4 Pipeline MLOps su SageMaker

- Preprocessing Addizionale: training_preprocessing.py esegue un'ulteriore elaborazione dei dati.
- Training: Utilizzo del framework XGBoost per regressione lineare.
- Model Registry: Il modello viene salvato e versionato in SageMaker Model Registry.
- Deployment: Viene creato un endpoint per l'inferenza in tempo reale.

3.4.5 API di Servizio e Frontend

- API: Espone un endpoint REST per richieste di predizione.
- Frontend: Interfaccia utente in Streamlit, consente di caricare input e visualizzare output.

3.4.6 CI/CD Pipeline per il Frontend

Per garantire l'automazione nello sviluppo e distribuzione del frontend Streamlit, è stata implementata una pipeline CI/CD tramite GitHub Actions. Ogni push sul ramo principale del repository frontend/ avvia automaticamente il processo di:

- Build dell'immagine Docker del frontend.
- Push sul registry di Github, per risparmiare sui costi.

Questa strategia consente l'aggiornamento continuo dell'interfaccia utente, favorendo un ciclo di sviluppo rapido, controllato e privo di interventi manuali.

3.4.7 Deploy Kubernetes con Ingress

Il frontend è esposto pubblicamente tramite un Ingress Controller su un cluster k8s locale. La configurazione prevede:

- Un Deployment Kubernetes per il container Streamlit.
- Un Service di tipo LoadBalancer.

3.5 Deployment con Terraform

L'infrastruttura è descritta e distribuita tramite Terraform, rispettando il principio dell'Infrastructure as Code (IaC). I moduli principali includono:

- terraform-aws-s3: gestione bucket S3.
- terraform-aws-glue: configurazione di Glue jobs e workflows.
- terraform-aws-eventbridge: regole di evento.
- terraform-aws-lambda: deployment della funzione Lambda.
- terraform-aws-sagemaker: pipeline, endpoint e model registry.

3.5.1 CI/CD GitOps per l'Infrastruttura (IaC)

Il repository contenente i moduli Terraform (iac/) è integrato con una pipeline GitHub Actions che segue l'approccio GitOps. Ogni modifica al codice infrastrutturale (merge su main) comporta:

- Esecuzione automatica di terraform plan per validare le modifiche.
- Esecuzione di terraform apply in un ambiente di esecuzione controllato.
- Deploy automatico delle risorse aggiornate su AWS.

Questo approccio garantisce tracciabilità, versioning e ripetibilità delle modifiche all'infrastruttura, minimizzando il rischio di errori manuali.

3.5.2 Continuous Delivery su Kubernetes

Nel contesto del frontend containerizzato, la pipeline CI/CD si estende anche al deployment su Kubernetes. Dopo il push dell'immagine Docker su ECR, viene aggiornato automaticamente il Deployment Kubernetes tramite comandi kubectl apply, seguiti da rollout status, il tutto orchestrato tramite GitHub Actions o ArgoCD (in futuro).

Questa integrazione assicura che ogni modifica approvata venga deployata automaticamente in produzione, mantenendo una forte coerenza tra codice, infrastruttura e ambiente di esecuzione.

3.5.3 Kubernetes Helm chart

Per facilitare e standardizzare il processo di deploy delle risorse Kubernetes legate al frontend, è stato sviluppato un Helm chart personalizzato. L'obiettivo era rendere il deployment ripetibile, configurabile e portabile, evitando la duplicazione di file YAML e rendendo il sistema più gestibile in ambienti reali.

Il chart include i seguenti componenti:

- Deployment: definisce il pod che esegue il container Docker del frontend, configurabile tramite values.yaml (image, replica count, risorse, ecc.).
- Service: espone il pod all'interno del cluster.

- Secrets: mantiene le credenziali aws.

Questo approccio ha permesso di versionare e gestire il deployment del frontend su qualsiasi cluster Kubernetes (es. Amazon EKS), mantenendo consistenza tra ambienti di test, staging e produzione. Inoltre, la combinazione con GitHub Actions consente l'automatizzazione del deploy a ogni nuova release dell'immagine Docker.

3.6 Vantaggi dell'Architettura

- Scalabilità: i servizi serverless si adattano automaticamente al carico.
- Modularità: ogni componente può essere testato o aggiornato singolarmente.
- Automazione: l'intero ciclo di vita del modello è orchestrato senza intervento manuale.
- Ripetibilità: l'infrastruttura può essere ricreata identicamente in qualsiasi momento.
- Monitoraggio: log centralizzati in CloudWatch per ogni step.

3.7 Considerazioni Tecniche

- Gestione permessi IAM: ogni servizio ha policy minime per operare in sicurezza.
- Gestione degli errori: le Lambda registrano eventuali fallimenti per debug.
- Persistenza dei dati: S3 viene usato sia come input che come storage di output.
- Logging & Monitoring: integrazione con AWS CloudWatch e eventuale alerting.

3.8 Interfaccia Utente con Streamlit

Il frontend permette all'utente di:

- Inserire manualmente dati di input.
- Visualizzare in tempo reale le previsioni.
- Monitorare l'endpoint e visualizzare eventuali anomalie.

La scelta di Streamlit è giustificata dalla sua semplicità e rapidità di prototipazione.

3.9 Problematiche Affrontate

- Configurazione corretta dei permessi IAM: fondamentale per sicurezza ed esecuzione fluida.
- Formato del workflow Glue.
- Formato della pipeline di Sagemaker.

Chapter 4

Conclusioni

Il progetto dimostra l'efficacia dell'approccio MLOps in ambienti cloud, unendo automazione, scalabilità e best practice DevOps. L'integrazione dei servizi AWS permette una gestione completa del ciclo di vita di un modello ML, dall'ingestione dati al deployment, fino all'inferenza in tempo reale con un'interfaccia user-friendly.

4.1 Possibili Sviluppi Futuri

- Integrazione di un sistema di monitoraggio del modello (drift detection).
- Migrazione verso un'architettura multi-model.
- Inserimento di uno step di approvazione manuale nella pipeline.
- Ottimizzazione dei costi tramite modelli batch-based.

4.2 Riferimenti

- AWS Glue Documentation: https://docs.aws.amazon.com/glue/
- AWS SageMaker Documentation: https://docs.aws.amazon.com/sagemaker/
- Terraform AWS Provider: https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- Kubernetes: https://kubernetes.io/

- Helm: https://helm.sh/
- Github Actions: https://github.com/features/actions
- Repository: https://github.com/umbertocicciaa/aws-mlops