Università degli Studi di Catania
Corso di Laurea Magistrale in Informatica

Riccardo Raciti
Rosario Cannavo'
Mario Benissimo

Progetto Multimedia

Relazione

Anno Accademico 2022–2023

Indice

Introduzione

L'obiettivo di questo progetto è quello di creare uno strumento di analisi e visualizzazione di semplice utilizzo basato su una pipeline realtime di image processing robusta e resiliente in grado di applicare degli algoritmi di Machine learning e Computer vision a delle rilevazioni fotografiche effettuate su Marte quotidianamente dai Rover Curiosity, Perseverance e Spirit per la ricerca di informazioni di interesse. L'endpoint implementato è un'interfaccia web messa a disposizione degli utenti che potranno visualizzare le analisi effettuate in modo rapido, avendo la possibilità di confrontare più risultati contemporaneamente.

1 Architettura Generale

Il sistema è progettato per elaborare le immagini in maniera scalabile e in tempo reale. L'architettura basata su microservizi permette di isolare e distribuire facilmente le diverse funzionalità del sistema. L'intera infrastruttura è basata su container Docker e orchestrata tramite Docker Compose. La totalità della comunicazione avviene tramite REST api e attraverso una rete privata che connette tutti i servizi. Grazie all'adozione di questo pattern architettturale moderno viene semplificata la gestione e la distribuzione del sistema, inoltre, vista la natura versatile ed estendibile del progetto, tale scelta permette di rimodulare l'intero sistema senza dover rimodulare totalmente l'infrastruttura esistente.

1.1 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Essa raccoglie il software in unità standardizzate chiamate **Docker container** che ricalcano esattamente l'essenza dei container Linux. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito. Docker permette di distribuire il codice più rapidamente, standardizzare il funzionamento delle applicazioni, trasferire il codice in modo ottimizzato e risparmiare denaro migliorando l'utilizzo delle risorse. Con Docker è possibile ottenere un singolo oggetto che può essere eseguito in modo affidabile in qualsiasi posizione. La sua sintassi è semplice e permette di tenere le risorse sotto controllo. Il fatto che sia già diffuso significa che offre un ecosistema di strumenti e applicazioni pronte all'uso.

1.1.1 Immagini Docker

Un'immagine Docker è un template di sola lettura che viene fornito con le istruzioni per il deploy dei container. In Docker, tutto ruota fondamentalmente intorno alle **immagini**. Un'immagine consiste in una collezione di file (o layer) che mettono insieme tutte le necessità, come le dipendenze, il codice sorgente e le librerie, necessarie per impostare un ambiente container completamente funzionale.Le immagini possono essere memorizzate in locale, o su un registro remoto come **Docker hub**[**?**], in modo da essere sempre fruibili e utilizzabili per la creazione di container su qualsiasi ambiente.

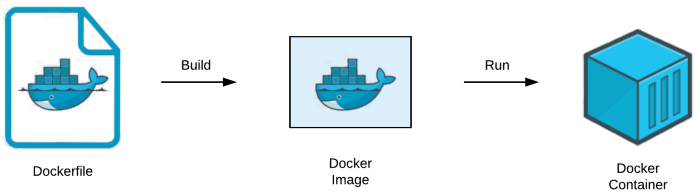

Figura 1: Pipeline di creazione di un'immagine docker

1.2 Dockerfile

I Dockerfile sono file utilizzati per creare in modo programmatico immagini Docker. Essi consentono di creare in modo rapido e riproducibile un'immagine Docker e quindi sono utili per la containerizzazione di applicativi. I Dockerfile sono formati da istruzioni per la creazione di un'immagine Docker, dove ogni istruzione è scritta su una riga e viene data nella forma:

$$< INSTRUCTION >< argument(s) >$$

Durante la creazione di un Dockerfile, il **client Docker** invierà un "build context" al **daemon Docker**, il processo che permette di eseguire i container, dove il contesto di costruzione include tutti i file e le cartelle nella stessa directory del file Docker. Grazie ad essi la containerizzazione di applicazioni diventa diretta e facile da utilizzare, rendendo Docker uno strumento ancora più performante.

1.2.1 Docker Compose

Un cluster di container è un gruppo di server o nodi (nodes) o host che lavorano insieme per eseguire, gestire e distribuire i container delle applicazioni. Un cluster di container consente di distribuire e gestire i container in modo efficace, migliorando la flessibilità e l'affidabilità dell'infrastruttura di gestione dei container. L'orchestrazione di container è un processo di gestione e coordinamento di un cluster di container. L'obiettivo dell'orchestrazione è di automatizzare il deployment, la scalabilità e la gestione di un'applicazione containerizzata, garantendo un'infrastruttura altamente disponibile e scalabile. Una tipologia di orchestrazione lightweight è Docker Composeche permette di gestire piccoli insiemi di container in modo semplice e unificato. **Docker Compose** è progettato per orchestrare e gestire un insieme di container su un singolo host. Con Docker Compose è possibile definire un file di composizione che descrive l'infrastruttura dell'applicazione e specifica i servizi, le reti e i volumi necessari per far funzionare l'applicazione. Inoltre, Docker Compose consente di avviare e fermare facilmente tutti i container dell'applicazione in modo coordinato. All'interno del progetto Docker Compose risulta essere essenziale per la fase di deploy in quanto permette di lanciare tutte le instanze contemporaneamente, gestire i guasti ed inoltre permette di gestire le variabili di ambiente che contengono le informazioni necesssarie per la comunicazione fra container. Senza l'uso di Docker Compose, non sarebbe possibile gestire la scalabilità e la resilienza ai guasti che permette al servizio di non andare interamente offline nonostante il possibile down di qualsiasi componente dell'architettura.

1.3 Nasa API

La NASA raccoglie molti dati, oltre 15 Terabyte al giorno. E per mandato della Casa Bianca questi dati sono gratuiti per il pubblico, in un formato utile per l'utente, ovvero le API. Trovare il modo migliore per distribuire, utilizzare e riutilizzare i dati della NASA è un problema. Le API della NASA forniscono una soluzione abbassando la barriera d'ingresso per le persone al di fuori della NASA per manipolare e accedere facilmente alle informazioni pubbliche. Le API utilizzate all'interno del progetto ricadono sotto il nome di **"Mars Rover Photos**. Questa API è stata progettata per raccogliere i dati delle immagini raccolte dai rover Curiosity, Opportunity e Spirit della NASA su Marte e renderli più facilmente disponibili ad altri sviluppatori, educatori e civili. Ogni rover ha il proprio set di foto memorizzate nel database, che possono essere interrogate separatamente. Esistono diverse possibilità di interrogazione dell'API. Le foto sono organizzate in base al sol (rotazione marziana o giorno) in cui sono state scattate, contando dalla data di atterraggio del rover. Una foto scattata durante il millesimo sol marziano di Curiosity, ad esempio, avrà l'attributo sol 1000. Se invece si preferisce cercare in base alla data terrestre in cui è stata scattata una foto, si può fare anche questo. Oltre alla ricerca per data, i risultati possono essere filtrati anche in base alla fotocamera con cui è stata scattata la foto e le risposte saranno limitate a 25 foto per chiamata. Le interrogazioni che dovrebbero restituire più di 25 foto saranno suddivise in diverse pagine, alle quali si può accedere aggiungendo un parametro "pagina" alla query. Le foto recuperate dalla pipeline e utilizzate per l'analisi sono quelle relative al giorno odierno e vengono aggiornate quotidianamente.

1.4 Logstash

Logstash è un motore di raccolta dati open source con capacità di pipelining in tempo reale. Logstash può unificare dinamicamente i dati provenienti da fonti diverse e normalizzarli in destinazioni a scelta. Pulisce e democratizza tutti i dati per diversi casi d'uso di analisi e visualizzazione avanzate a valle. Sebbene Logstash abbia originariamente guidato l'innovazione nella raccolta dei log, le sue capacità si estendono ben oltre questo caso d'uso. Qualsiasi tipo di evento può essere arricchito e trasformato con un'ampia gamma di input, filtri e plugin di output, con molti codec nativi che semplificano ulteriormente il processo di ingestione. Logstash accelera gli approfondimenti sfruttando un volume e una varietà maggiori di dati.

1.4.1 Plugin HTTP

Un plugin di input consente a una fonte specifica di eventi di essere letta da Logstash. Con questo input si possono ricevere eventi singoli o multilinea su http. Le applicazioni possono inviare una richiesta HTTP all'endpoint avviato da questo input e Logstash la convertirà in un evento per la successiva elaborazione. Gli utenti possono passare testo semplice, JSON o qualsiasi dato formattato e utilizzare un codec corrispondente con questo input. Per il Content-Type application/json viene utilizzato il codec json, mentre per tutti gli altri formati di dati viene utilizzato il codec plain. Nella pipeline implementata, Logstash risulta di fondamentale importanza. Il suo compito è quello di effettuare data ingestion quotidianamente così da fornire al resto dell'architettura sempre le immagini aggiornate da processare. Logstash è direttamente collegato ad un'endpoint specifico messo a disposizione dalla NASA e riesce grazie ad un plugin output a scrivere in real time le informazioni ritrovate sotto forma di documento json all'interno di una specifica

coda RabbitMQ. Il json ottenuto da logstash contiene i metadati relativi all'immagine e l'url dell'immagine stessa che verrà successivamente estratta in fase di pre-processing.

1.5 RabbitMQ

RabbitMQ è un message-oriented middleware (detto anche broker di messaggistica) che implementa il protocollo Advanced Message Queuing Protocol (AMQP). Il server RabbitMQ è scritto in Erlang e si basa sul framework Open Telecom Platform (OTP) per la gestione del clustering e del failover. Un broker di messaggi è un modello architetturale per la convalida, la trasformazione e il routing dei messaggi. Il broker media la comunicazione tra le applicazioni, riducendo al minimo la consapevolezza reciproca, ovvero quella che le applicazioni dovrebbero avere l'una rispetto all'altra al fine di poter scambiare messaggi, implementando in modo efficace il disaccoppiamento. Lo scopo principale di un broker è di prendere i messaggi in arrivo dalle applicazioni ed eseguire alcune azioni su di essi. Ad esempio, un broker di messaggi può essere utilizzato per gestire una coda di carico di lavoro o una coda di messaggi per più ricevitori, fornendo memoria affidabile, con una consegna di messaggi garantita. Nel tipico scenario dello scambio di messaggi, RabbitMQ introduce, oltre la presenza del Publisher, del Consumer e della Queue, un nuovo elemento: l'Exchange. Attraverso questa modifica avviene l'implementazione del protocollo AMQP. Con RabbitMQ il Publisher non invia più il messaggio direttamente alla coda, ma passa per l'Exchange, il quale crea la comunicazione con la coda attraverso un legame detto binding. Differenza tra RabbitMQ e JMS: nel RabbitMQ viene introdotto l'Exchange, per mediare lo scambio di messaggi tra il Publisher e una coda, associata ad uno specifico consumatore. RabbitMQ offre la possibilità di

utilizzare diversi tipi di Exchange al fine di soddisfare i differenti bisogni. Tuttavia si possono delineare tre principali categorie:

- Fanout;
- Direct;
- Topic;

Ognuna di queste categorie definisce un diverso comportamento rispetto a come viene indirizzato il messaggio dall'Exchange alle code. Utilizzando l'Exchange viene a determinarsi un sistema definito Pub/Sub. Infatti nella realtà non è presente un solo consumatore, bensì migliaia. Per questo motivo il messaggio pubblicato dal Publisher viene inviato a tutte le code sottoscritte ad uno specifico Exchange relativo al Publisher mittente. Per la realizzazione del progetto si è scelto di implementare un Exchange di tipo Direct. Il messaggio viene inoltrato alle code la cui chiave di associazione corrisponde esattamente alla chiave di Routing (etichetta) del messaggio. La scelta è stata effettuata in previsione di possibili flussi multipli all'interno del framework che prevedono però data source differenti, ad esempio, immagini di natura diversa da quelle presenti attualmente. Inoltre, nello scambio diretto è possibile associare più code con la stessa chiave di associazione, questo permette di avere più worker, ognuno agganciato ad una coda personale.

1.6 Pre-Processing

Una volta inseriti nella coda, i messaggi vengono letti in real time da uno script python che si occupa di recuperare ed effettuare il download effettivo delle immagini a partire dagli url presenti nei json ritornati dalle API messe a disposizione dalla NASA. In particolare questo script fa uso della libreria python *requests* per scaricare le immagini e della libreria *pillow* per

effettuare il download delle immagini in formato PNG. Dopo essere state scaricate, le immagini vengono all'occorrenza convertite in scala di grigi, per permettere agli algoritmi una computazione più semplice e organizzate in dei file che verranno memorizzati all'interno di una specifica collezione MongoDB chiamata **base_image**. Insieme alle immagini vengono memorizzati anche i dati satellite inerenti alle immagini come il nome della telecamera che ha scattato la foto e il nome del rover. Una volta terminata la fase di pre-processing ed aver effettuato le scritture sul database in modo corretto, lo script si occuperà di richiamare tramite una chiamata API predisposta il microservizio che si occuperà della fase di processamento. Grazie a questo sistema è stato possibile evitare l'utilizzo di timer multipli implementati attraverso *CronJob* che avrebberò causato problemi di sincronizzazione e di conseguenza downtime del servizio. Data la natura dell'architettura è possibile all'occorrenza aumentare il numero di container che si occupano della fase di preprocessing: basterà agganciare un consumer alla coda di messaggi e la gestione del traffico sarà autogestita. Lo stesso effetto si avrà in fase di scrittura sul database.

1.7 Processing

La parte centrale dell'architettura è rappresentata da un server scritto in *Flask*, un framework Python per la creazione di server backend. Il server espone una sola root, quella chiamata dal servizio di microprocessing per avviare la computazione. Il servizio di processing implementa gli algoritmi presenti all'interno dell'applicativo. Ogni algoritmo è implementato all'interno di una classe dedicata ed è eseguito all'interno di un thread specifico in modo da sfruttare la parallelizzazione e rendere più efficiente la computazione. Ogni algoritmo utilizza come input i dati presenti all'interno della collezione

base_image, quindi, le immagini inserite in fase di preprocessing. Dopo aver effettuato la computazione, ogni algoritmo memorizza i risultati su una collezione MongoDB dedicata. Dividendo la parte di scrittura sul database è stato possibile evitare busy waiting, infatti, gli algoritmi implementati hanno tempi di esecuzione differenti mentre il servizio risulta essere sempre disponibile.

1.8 Implementazione degli algoritmi

Di seguito una breve descrizione delle librerie e delle metodologie utilizzate per l'implementazione degli algoritmi. Una trattazione teorica degli algoritmi è presente nei capitoli successivi.

1.8.1 KMeans

Per l'implementazione dell'algoritmo KMeans è stato utilizzato il modulo KMeans della libreria *SkLearn*. Le feature utilizzate per l'esecuzione del clustering sono state estratte attraverso il modello **VGG16** messo a disposizione da Keras. VGG16 è un modello di rete neurale convoluzionale utilizzato per il riconoscimento delle immagini che utilizza solamente 16 layer. Inoltre, prima della computazione le feature sono state sottoposte a **PCA** per ottimizzare i risultati. Per effettuare la PCA è stato utilizzato il modulo *decomposition* messo a disposizione da SkLearn.

1.8.2 Histogram of Oriented Gradients

Una seconda implementazione per la ricerca di cluster è stata effettuta attraverso le feature estratte attraverso la procedura *Hog*. Il feature extractor è stato implementato in modo iterativo attraverso le funzioni messe a disposizione dalla libreria *numpy* mentre il clustering effettivo, una volta estratte

le feature è stato implementato attraverso il metodo *linkage* di *SciPy* che fa uso di matrici di distanza, ottenute computando sulle feature la distanza di *Jensen-Shannon*.

1.8.3 FAST

Per implementare l'algoritmo Features from Accelerated Segment Test, è stata utilizzata la libreria *OpenCv*. Nonostante OpenCv metta a disposizione già un metodo nativo per la rilevazione dei corner si è optato, per raffinare maggiormente il risultato finale, per una soluzione che combina dei metodi di libreria appartenenti a OpenCv stesso.
I metodi in questione sono sono:

- *cornerHarris*, per ogni pixel (x,y) calcola la matrice di covarianza $M^{(x,y)}$ in un'area 2x2;
- *dilate*, dilata l'immagine di origine utilizzando l'elemento di strutturazione specificato che determina la forma di una regione di pixel in cui viene selezionato il massimo.

Una volta ottenute queste informazioni, viene effetuata una sogliatura per trovare il valore ottimale. Questa soluzione è stata scelta perchè ogni immagine può avere un valore ottimo differente che le librerie native non avrebbero rilevato fornendo dei risultati parziali.

1.8.4 Segmentazione

Per effettuare la segmentazione è stato utilizzato l'algoritmo messo a disposizione da *OpenCv*. per completezza sono state memorizzate anche le immagini intermedie generate durante la computazione. Nello specifico sono

state memorizzate le immagini fornite dalle funzioni:

- *threshold*
- *morphologyEx*, fornisce l'immagine threshold priva di rumore;
- *dilate*, data l'immagine priva di rumore fornisce il background dell'immagine stessa;
- *distanceTransform*, data l'immagine priva di rumore fornisce il foreground dell'immagine stessa;
- *connectedComponents*, utilizzando questa funzione sull'immagine di foreground forinisce la maschera delle strutture presenti nell'immagine, tramite i contorni degli oggetti.
- *watershed*, data l'immagine originale e l'immagine dei contorni genera l'immagine segmentata, in modo da ottenere:
  - l'immagine originale con i bordi degli oggetti evidenziati.
  - l'immagine originale con ogni regione delimitata da bordi differenti colorata o evidenziata da colori differenti.

1.8.5 Slic

Per l'implementazione dell'algoritmo Simple Linear Iterative Clustering sono stati utilizzati i metodi messi a disposizione dalla libreria *SkImage*. Inoltre per valutare la bontà di questo algoritmo, sono stati implementati anche gli algoritmi *Felzenszwalbs*, *Quickshift* e *Watershed*. In questo modo l'utente finale avrà la possibilità di valutare personalmente la qualità dei risultati. Anche gli algoritmi aggiuntivi sono stati implementati attraverso i metodi nativi messi a disposizione dalla libreria *SkImage*.

1.9 MongoDB

MongoDB è un database NoSQL open source. In quanto database non relazionale, è in grado di elaborare dati strutturati, semi-strutturati e non strutturati. Utilizza un modello di dati non relazionale e orientato ai documenti e un linguaggio di query non strutturato. MongoDB è altamente flessibile e consente di combinare e memorizzare più tipi di dati. Inoltre, memorizza e gestisce quantità di dati maggiori rispetto ai database relazionali tradizionali. MongoDB utilizza un formato di storage dei documenti definito BSON, che è una forma binaria di JSON (JavaScript Object Notation) in grado di ospitare più tipi di dati. MongoDB memorizza gli oggetti di dati in raccolte e documenti, invece che nelle tabelle e righe utilizzate nei database relazionali tradizionali. Le raccolte comprendono insiemi di documenti, che sono l'equivalente delle tabelle di un database relazionale. I documenti sono costituiti da coppie chiave-valore, che sono l'unità di base dei dati in MongoDB.

La struttura di un documento può essere modificata semplicemente aggiungendo nuovi campi o eliminando quelli esistenti. I documenti possono definire una chiave primaria come identificatore univoco e i valori possono includere vari tipi di dati, compresi altri documenti, array e array di documenti.

1.9.1 Memorizzazione delle immagini

MongoDB memorizza i documenti in collection. Le collection sono analoghe alle tabelle dei database relazionali. Per la memorizzazione delle immagini all'interno del database la scelta è stata quella di dedicare una collection ad ogni categoria di elaborazione in modo da separare i dati, replicandoli eventualmente, ed evitare che potessero corrompersi. Per creare una connessione tra le immagini originali e quelle analizzate, ogni documento presenta

un campo che denota l'**ObjectID** (identificativo univoco di un documento all'interno del database) del documento originale, in questo modo non è necessario replicare le immagini in ogni collection, evitando inutile ridondanza. Inerentemente al formato utilizzato per le immagini, data l'eterogenerità del sistema e dei linguaggi utilizzati per manipolarle, la scelta è stata quella di utilizzare il formato **Base64**. A seguito di questa scelta si sono però presentati dei problemi di dimensionalità in quanto essendo molte delle immagini rappresentate da stringhe di notevoli dimensioni, non è stato possibile effettuare lo storing in modo tradizionale. Per ovviare a questo problema, la soluzione scelta, che rappresenta l'attuale stato dell'arte per problematiche di questo tipo, è ricaduta sull'utilizzo di GridFS.

1.9.2 GridFS

Invece di memorizzare un file in un singolo documento, GridFS divide il file in parti, o chunk, e memorizza ogni chunk come documento separato. Per impostazione predefinita, GridFS utilizza una dimensione predefinita dei chunk di 255 kB; in altre parole, GridFS divide un file in chunk di 255 kB con l'eccezione dell'ultimo chunk. L'ultimo chunk è grande solo quanto necessario. Allo stesso modo, i file che non superano la dimensione del chunk hanno solo un chunk finale, utilizzando solo lo spazio necessario più alcuni metadati aggiuntivi. GridFS utilizza due raccolte per memorizzare i file. Una collezione memorizza i chunk dei file, mentre l'altra memorizza i metadati dei file. La sezione Collection GridFS descrive in dettaglio ogni raccolta. Quando si interroga GridFS per un file, il driver riassembla i pezzi come necessario. È possibile eseguire query di intervallo sui file archiviati in GridFS. È anche possibile accedere alle informazioni di sezioni arbitrarie di file, ad esempio per "saltare" alla metà di un file video o audio. GridFS è utile non solo per

memorizzare file che superano i 16 MB, ma anche per memorizzare qualsiasi file a cui si desidera accedere senza dover caricare l'intero file in memoria. Grazie a GridFS è stato possibile memorizzare la stringa in chunks e quindi aggirare i problemi che si erano presentati, inoltre, l'adozione di questa tecnica ha permesso di rendere la ricerca di grandi quantità di immagini più efficiente grazie ad una parallelizzazione effettuata dalle funzioni di libreria.

1.10 SpringBoot

Per la realizzazione del backend che si occupa della gestione del sito web, la scelta è ricaduta su uno degli strumenti attualmente più utilizzati a livello enterprise, il framework Java **SpringBoot**. Spring Boot è un framework open source basato su Java utilizzato per creare microservizi.Essi è usato per costruire applicazioni Spring stand-alone e pronte per la produzione. Viene utilizzato per costruire applicazioni Spring stand-alone e pronte per la produzione. Spring Boot fornisce una buona piattaforma agli sviluppatori Java per sviluppare un'applicazione Spring stand-alone e pronta per la produzione, che può essere semplicemente eseguita. È possibile iniziare con configurazioni minime, senza la necessità di un'intera configurazione di Spring. L'interezza del Backend è composta da diverse *root* che si occupano di effettaure query al database, gestire la logica di business dei dati e ritornarli al frontend per presentarli. La parte più comune consiste nella corretta conversione delle immagini dal formato Base64 e nell'organizzazione dei dati in base ai metadati.

1.11 Prometheus

Prometheus è un sistema di monitoraggio open-source che utilizza un'architettura decentralizzata per raccogliere, archiviare e analizzare metriche

provenienti da diverse fonti. L'architettura di Prometheus è costituita da quattro componenti principali: server di raccolta dati, endpoint di esportazione, database a serie temporali e strumenti di interrogazione e visualizzazione. Il server di raccolta dati di Prometheus è il componente principale del sistema. Raccoglie le metriche da diverse fonti tramite endpoint HTTP esposti dalle applicazioni o da client specifici chiamati "exporter". Le metriche raccolte vengono quindi archiviate in un database a serie temporali. L'endpoint di esportazione è un'interfaccia HTTP utilizzata dalle applicazioni per esporre le metriche a Prometheus. Le metriche sono rappresentate come coppie chiave-valore e sono raggruppate in "famiglie di metriche" in base al loro nome e alle loro etichette. Il database a serie temporali di Prometheus è costituito da una serie di file di dati in cui le metriche raccolte vengono archiviate. Il database consente di interrogare e analizzare i dati nel tempo, consentendo di monitorare l'evoluzione delle metriche nel tempo e di individuare eventuali anomalie. Infine, gli strumenti di interrogazione e visualizzazione di Prometheus permettono di interrogare i dati del database e di visualizzarli sotto forma di grafici e tabelle. In particolare, il linguaggio di query **PromQL** di Prometheus offre una sintassi flessibile per selezionare e aggregare le metriche archiviate. L'architettura decentralizzata di Prometheus consente di distribuire il server di raccolta dati in modo da gestire carichi di lavoro di grandi dimensioni. Inoltre, grazie alla sua flessibilità e alla sua vasta gamma di strumenti di monitoraggio e allerta, questo tool è diventato uno strumento di monitoraggio molto popolare nella comunità di sviluppatori e negli ambienti cloud-native. All'interno del progetto Prometheus è stato utilizzato per effettuare il monitoring delle root Springboot in modo da avere sempre contezza delle risorse utilizzate. Per collegare Prometheus a Spring Boot, è stata utilizzata la libreria Spring Boot Actuator, che

fornisce un endpoint per le metriche in formato Prometheus. Per ogni root sono state predisposte delle query che permettono di monitorare il numero di richieste e il tempo necessario a soddisfarle, in questo modo è possibile scalare opportunamente il singolo servizio se necessario.

1.12 Front-end

La parte finale della pipeline consiste di una dashboard web che permette di visualizzare le immagini del giorno e i risultati ottenuti durante la fase di processing. In particolare ogni sezione della pagina fa riferimento ad un differente algoritmo e presenta una breve descrizione di quest'ultimo. Per l'implementazione del front-end le principali tecnologie utilizzate sono state il framework **Bootstrap** per il rendering grafico e l'implementazione della responsiveness della dashboard e **Jquery** per la parte logica inerente alla comunicazione con il backend. In combinazione con html è stato utilizzato il framework di rendering **Thymeleaf** messo a disposizione da Springboot. Attraverso Thymeleaf è stato possibile implementare in modo nativo le gallery di immagini e il passaggio di contenuti multimediali tra front-end e back-end.

2 Clustering

Una parte del progetto è stata incentrata sulla produzione di cluster in modo da poter classificare le immagini e poterle distinguere e categorizzare per similarità. I primi algoritmi implementati sono stati $KMeans$, PCA e clustering tramite HOG.

2.1 KMeans

Con il termine $K - means$ si indica una classe di algoritmi di clustering partizionali basati su assegnamenti di punti. Essi permettono di suddividere un insieme di oggetti in k gruppi sulla base delle loro feature. Lavorano su spazi euclidei e assumono la conoscenza a-priori del numero di cluster k, che costituisce un iperparametro per l'algoritmo. Sono tuttavia presenti alcune tecniche per dedurre il miglior valore di k attraverso una serie di esperimenti.

2.2 PCA

PCA è l'acronimo di $AnalisiComponentiPrincipali$, è una tecnica finalizzata a ridurre la dimensionalità di un insieme di dati con finalità esplorative, di visualizzazione dei dati o feature extraction, per un eventuale uso in analisi successive. L'analisi delle componenti principali (PCA) è una tecnica finalizzata a derivare, a partire da un set di variabili numeriche correlate, un insieme più ridotto di variabili ortogonali "artificiali". L'insieme ridotto di proiezioni ortogonali lineari (noto come "componenti principali" o "principal components", "PC") è ottenuto combinando linearmente in maniera appropriata le variabili originarie.

2.3 HOG

$Histogram of Oriented Gradients$ (HOG), è un descrittore di feature che viene spesso utilizzato per estrarre feature dai dati dell'immagine. In generale, è una rappresentazione semplificata dell'immagine che contiene solo le informazioni più importanti sull'immagine. Il descrittore di feature HOG conta le occorrenze dell'orientamento del gradiente nelle porzioni localizzate di un'immagine. È ampiamente utilizzato nelle attività di visione artificiale per il rilevamento di oggetti. Il descrittore HOG si concentra sulla struttura o sulla forma di un oggetto, è importante notare che è diverso dall'estrazione delle caratteristiche dei corner che possiamo estrarre per le immagini perché nel caso delle caratteristiche HOG, vengono estratti sia i corner che la direzione. L'immagine completa è suddivisa in regioni più piccole (porzioni localizzate) e per ogni regione vengono calcolati i gradienti e l'orientamento. Infine l'HOG genera un istogramma per ciascuna di queste regioni separatamente. Gli istogrammi vengono creati utilizzando i gradienti e gli orientamenti dei valori dei pixel, da cui il nome Histogram of Oriented Gradients.

3 Corner Detection

3.1 FAST

Si è scelto di implementare un algortimo di corner detection vista l'utilità all'interno della computer vision. Un corner può essere definito come l'intersezione di due angoli o come un punto per il quale ci sono due direzioni di angoli dominanti e diverse in un area locale del punto.
Un punto di interesse è un punto in un'immagine che ha una posizione ben definita e può essere rilevato in modo robusto. Ciò significa che un punto di interesse può essere un angolo ma può anche essere, ad esempio, un punto isolato di intensità locale massima o minima, terminazioni di linea o un punto su una curva in cui la curvatura è localmente massima.
In pratica, la maggior parte dei cosiddetti metodi di rilevamento degli angoli rileva i punti di interesse in generale e in effetti, il termine "angolo" e "punto di interesse" sono usati in modo più o meno intercambiabile in letteratura. Di conseguenza, se si devono rilevare solo gli angoli, è necessario fare un'analisi locale dei punti di interesse rilevati per determinare quali di questi sono angoli reali. L'algoritmo implementato è FAST (Features from accelerated segment test). Il vantaggio di FAST è la sua efficienza computazionale. Riferendosi al suo nome, è davvero più veloce di molti altri noti metodi di estrazione delle caratteristiche, come ad esempio DoG.
Il rilevatore di corner FAST utilizza un cerchio di 16 pixel (un cerchio Bresenham di raggio 3) per classificare se un punto candidato p è effettivamente un angolo. Ogni pixel nel cerchio è etichettato con un numero intero da 1 a 16 in senso orario. Se un insieme di N pixel contigui del cerchio è composto da punti di un'intensità maggiore di una certa soglia, il pixel candidato p verrà classificato come angolo.

4 Segmentazione

Un altro importante strumento dell'image computing è la *segmentazione*.
La segmentazione è un processo di partizionamento di un'immagine in regioni disgiunte e omogenee.
Sia R l'intera regione spaziale occupata dall'immagine. Il processo di segmentazione può essere visto come il partizionamento di R in n sottoregioni, R_1, R_2 ..., R_n tali che:

1. $\bigcup_{n=1}^{k} R_i$ = R.
2. R_i è un insieme connesso, i = 1, 2, . . . , n.
3. $R_i \cap R_j \neq \emptyset$ per tutti i calori di i e j, con i e j diversi tra loro
4. Q(R_i) = $True$ per i = 1, 2, . . . , n.
5. Q($R_i \cap R_j$) = $False$ per ogni coppia di regioni adiacenti R_i intersezione R_j

Ogni pixel deve appartenere ad una regione. I punti appartenenti ad una regione devono essere connessi.
Le regioni devono essere disgiunte e i pixel appartenenti ad una regione devono soddisfare un certo predicato Q.
Due regioni adiacenti devono essere diverse nel senso del predicato Q.
Ad esempio il predicato Q potrebbe essere il seguente:
Q(R_i) = TRUE se l'intensità media dei pixel di R_i è inferiore a m e la loro deviazione standard è minore di σ (con m e σ parametri costanti).
Per poter suddividere le immagini in gruppi di pixel adiacenti simili sono stati utilizzati degli algortimi specifici.

4.1 SLIC

Questo filtro crea dei superpixel basati sull'algoritmo di clustering k-means. I superpixel sono piccoli cluster di pixels che condividono proprietà similari. I superpixel semplificano le immagini con un gran numero di pixels rendendole più facili da trattare in molti ambiti (computer vision, pattern recognition e intelligenza artificiale). Il colore del superpixel è la media dei colori dei pixel nella regione corrispondente.

4.2 Felzenszwalb

Effettua la segmentazione di un'immagine basandosi sulla teoria dei grafi. Sia G = (V, E) un grafo non diretto con vertici $v_i \in$ V, l'insieme di elementi da segmentare e angoli $(v_i, v_j) \in$ E corrispondente a coppie di vertici vicini. Ogni angolo $(v_i, v_j) \in$ E ha un peso corrispondente w((v_i, v_j)), che è una misura non negativa della dissomiglianza tra gli elementi vicini v_i e v_j. Nel caso della segmentazione dell'immagine, gli elementi in V sono pixel e il peso di un arco è una misura della dissimilarità tra i due pixel collegati da quel-l'arco (ad esempio, la differenza di intensità, colore, movimento, posizione o qualche altro attributo locale).

4.3 Quickshift

Quick shift è un algoritmo di ricerca rapida che effettua la segmentazione dell'immagine facendo inferenza sui colori, simile al mean shift. L'algoritmo segmenta un'immagine RGB (o qualsiasi immagine con più di un canale) identificando gruppi di pixel nelle dimensioni spaziali e cromatiche congiunte. I segmenti sono locali (superpixel) e possono essere utilizzati come base per ulteriori elaborazioni. Data un'immagine, l'algoritmo calcola una foresta di pixel i cui rami sono etichettati con un valore di distanza. Questo specifica

una segmentazione gerarchica dell'immagine, con segmenti corrispondenti a sottoalberi. I superpixel utili possono essere identificati tagliando i rami la cui etichetta di distanza è superiore a una determinata soglia (la soglia può essere fissata a mano o determinata mediante convalida incrociata).

4.4 Watershed

E' una tecnica di morfologia matematica, basata sul principio che ogni immagine in scala di grigio può essere considerata come una superficie topografica (una superficie tridimensionale definita sul piano dell'immagine ed avente l'intensità luminosa come terza dimensione). In tale interpretazione si identificano tre tipi di punti:

1. Punti appartenenti ad una regione di minimo;
2. Punti in cui una goccia d'acqua scorrerebbe verso uno di tali minimi (bacino di raccolta, catchment basin o watershed);
3. Punti in cui l'acqua potrebbe scendere verso più di un minimo con uguale probabilità (linee di separazione, linee di watershed o watershed lines).

L'obiettivo è quello di individuare le linee di watershed sull'immagine. L'idea è semplice: supponiamo che ogni regione di minimo sia perforata e che l'intera topografia sia riempita facendo fluire l'acqua dal basso a velocità costante. Quando la salita dell'acqua sta per unire due distinti bacini adiacenti si costruisce una diga per evitare la fusione. L'insieme finale di tali dighe costituisce le linee di watershed.

Conclusioni

Gli obiettivi prefissati, quali, la realizzazione di tutti le componenti dell'architettura progettata e l'implementazione degli algoritmi proposti, sono stati raggiunti, dando vita ad una pipeline real time per l'analisi delle rilevazioni fotografiche effettuate su marte dai rover presenti sul pianeta. Durante la fase di sviluppo sono state riscontrate delle difficoltà inerenti alla realizzazione della comunicazione tra microservizi rispettando la sincronia delle operazioni e la memorizzazione di file di grandi dimensioni. Tali problematiche sono state superate rispettivamente adottando un modello di comunicazione tra servizi basato su REST API ed effettuando lo sharding dei dati sfruttando la tecnologia GridFS messa a disposizione da MongoDB. Per via della genericità dell'architettura, possibili sviluppi futuri prevedono la possibilità di scegliere più fonti da analizzare, rendendo di fatto la pipeline adatta all'analisi di qualsiasi tipo di immagine. In conclusione, è possibile affermare che grazie al pattern di sviluppo adottato il prodotto finito risulta essere portabile e facilmente scalabile, rendendolo idoneo ad una distribuzione soggetta a traffico elevato. In questo modo anche gli utenti meno esperti avranno la possibilità di effettuare analisi su delle immagini in modo semplice e intuitivo utilizzando la dashboard messa a disposizione.