Universita' Politecnica delle Marche

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Tesi di laurea triennale

Porting di un'applicazione per l'analisi del planogram basata su computer visione con architettura container su piattaforma AWS

Porting of an application for vision based planogram integrity with container architecture on AWS platform

Candidato:
Denis Bernovschi
Matricola 1077134

Relatore:
Emanuele Frontoni

Correlatore:
Marina Paolanti

Anno Accademico 2019-2020

Ai miei genitori,
al mio fianco da sempre

Ringraziamenti

E' giunto il momento di fare i doverosi ringraziamenti. Comincio con il ringraziare la mia famiglia, per avere reso possibile questa magnifica avventura e per essere sempre al mio fianco, vi sarò per sempre grato. Un importante ringraziamento va poi a tutti quei professori, che lezione dopo lezione mi hanno trasmesso la loro passione, in particolare vorrei ringraziare il mio relatore Prof. Emanuele Frontoni e la mia correlatrice, la Prof. Marina Paolanti. Vorrei ringraziare inoltre il team Grottini, nelle persone di Marco Contigiani e Rocco Pietrini per avere reso questo progetto, nonostante le distanze e le difficoltà, una piacevole esperienza. Infine, ma non per l'ultimo vorrei ringraziare gli amici, gli amici di una vita, in particolare Vittoria, Anna e Giovi per l'affetto, gli amici incontrati lungo questo bellissimo percorso, o come li chiamo io, gli amici di merende, in particolare Angela, bea e Laura DS, senza le quali, i pomeriggi di studio non sarebbero stati così indimenticabili. Dulcis in fundo, vorrei ringraziare i miei fantastici coinquilini: Luca, Alex, Matteo che ad ogni difficoltà erano sempre pronti a spronarmi a non mollare mai e le "coinquiline" Laura F. e Sara per le fantastiche serate tra film, tisane e sconfitte a burraco. Perché se mai mi chiederanno cosa resterà di questi anni, afferrati e scivolati via, sono sicuro che risponderei, gli amici, gli amici del "tranquillo siamo qui noi".

Sommario

Tramite questo elaborato, vorrei porre in evidenza, in primis l'architettura generale di Amazon Web Services (AWS), ponendo maggior attenzione sui services essenziali. Tratteremo poi un'analisi sull'applicazione "An IoT Edge-Fog-Cloud Architecture for Vision Based Planogram Integrity". Successivamente, andremmo ad evidenziare il processo di realizzazione della piattaforma ospitante, integrando i vari service per la nostra applicazione, creando ad hoc una webpage per verificare il corretto funzionamento. Concludendo questo studio, tratteremo le problematiche emerse e le potenzialità di uno sviluppo applicativo cloud-based.

Indice

Elenco delle figure

Listings

1 Introduzione

Questa tesi nasce dall'incontro tra l'azienda Grottini Lab e l'Università Politecnica delle Marche. Grottini Lab è un'azienda marchigiana che opera a livello internazionale nel settore del Retail Intelligence e Communication Technology. Il suo obiettivo è lo studio dell'interazione del consumatore con i prodotti esposti nei punti vendita e l'analisi del loro comportamento in ambienti retail, il tutto finalizzato a valutazioni in merito alle vendite. Le tecnologie utilizzate dall'azienda per effettuare tali studi sono tecnologie di Computer Vision ed Intelligenza Artificiale.

Figura 1: Logo Grottini Lab

2 Materiali e Metodi

Questa sezione sarà interamente dedicata alla descrizione della strumentazione hardware e software necessaria allo studio oggetto di questa tesi. In particolare, porremo attenzione alle diverse peculiarità di AWS [1] e all'integrazione tra le diverse componenti a noi necessarie. Osserveremo poi la library OpenCV ponendo attenzione alle principali nozioni. Infine per quanto concerne la strumentazione necessaria allo sviluppo e all'implementazione del oggetto di studio di questa tesi, descriveremo le camere. Per quanto riguarda invece la parte software tratteremo oltre all'IDE necessario per sviluppo delle varie parti di codice e delle librerie, il Software Docker ponendo attenzione sul CLI (AWS CLI), e il Browser Chrome con le specifiche di sviluppo attivate al fine di verificare il corretto funzionamento del applicativo.

2.1 AWS

Amazon Web Services **(AWS)** offre servizi per infrastrutture IT sotto forma di Web Services, secondo un modello noto come cloud computing. Uno dei principali vantaggi del cloud computing è l'opportunità di eliminare gli investimenti iniziali in infrastrutture, sostituendoli con costi variabili contenuti, che variano in relazione alle esigenze dell'azienda. Grazie al cloud, non si deve più pianificare con molto anticipo l'acquisto di infrastrutture IT, serviranno bensì pochi minuti per raggiungere i propri obiettivi. Oggi Amazon Web Services offre una piattaforma infrastrutturale affidabile, scalabile e conveniente basata sul cloud, operativa i vari paesi del mondo, grazie ai vari data center sparsi sul territorio mondiale.

2.1.1 AWS - SDK per C++

Il kit SDK [2] è un insieme di librerie C++ completamente open source che semplifica l'integrazione di applicazioni C++ con servizi AWS quali S3, Kinesis e DynamoDB etc. .

2.1.2 Principali caratteristiche

- **Crittografia dei dati lato client per Amazon S3**: Questo intuitivo meccanismo di crittografia lato client consente di potenziare la protezione dei dati di applicazione archiviati in Amazon S3. Dato che crittografia e decrittografia vengono eseguite lato client, le chiavi di crittografia private non lasciano mai l'applicazione.

[1] AWS: Amazon Web Services
[2] SDK: Software Development Kit

- **Gestione trasferimenti di Amazon S3**: Tramite un'interfaccia API intuitiva, Amazon S3 TransferManager migliora throughput, prestazioni e affidabilità facendo ampio uso di caricamenti in più parti e multi-thread di Amazon S3.
- **Compatibilità con varie piattaforme**: Il kit è stato testato su diverse piattaforme: Windows, Linux, Mac, Android e iOS.
- **Supporto per CMake**: Gli utenti potranno usare CMake per compilare ed eseguire collegamenti al kit SDK senza perdersi nei dettagli sulle relative dipendenze.
- **Gestione della memoria personalizzata**: Collega un gestore di memoria per personalizzare il modo in cui le assegnazioni di memoria vengono allocate e ritirate.
- **Compatibilità aggiornata con tutti i servizi AWS**: Supporta tutti i servizi pubblici di AWS e viene aggiornato di frequente per supportare le modifiche più recenti delle API.

2.1.3 AWS - CLI

L'interfaccia a riga di comando di AWS, o AWS CLI (Command Line Interface), è uno strumento unificato che consente di gestire i servizi AWS

2.2 OpenCV

Poniamo particolare attenzione ad OpenCV (Open Source Computer Vision Library), una libreria multipiattaforma completamente open source, rilasciata sotto licenza BSD e utile per la realizzazione di software nell'ambito della computer vision e del machine learning. E' caratterizzata da innumerevoli interfacce in C, C++, Java e Python che permettono il supporto completo nelle diverse piattaforme Linux, Microsoft, OsX, Android e IOS. Tra gli algoritmi che compongono la libreria, quelli più interessanti ai fini dello sviluppo del progetto sono quelli che si occupano dell'individuazione degli oggetti. Il linguaggio di programmazione che è stato utilizzato con OpenCV è il C++.

Figura 2: Logo OpenCV

2.3 Camere

Le camere utilizzate finora sono delle Raspberry CAM (Fig.3) che provvedono all'acquisizione dell'immagine.

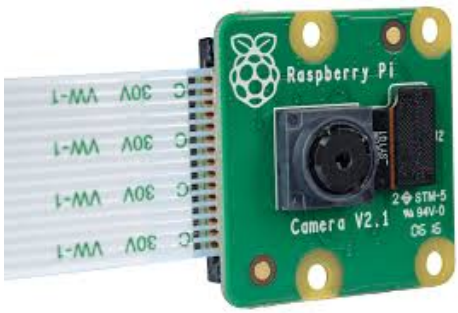


Figura 3: Rasperry Cam

Successivamente tramite il collegamento ad una scheda "Rasperry Py" come possiamo vedere in Fig. 4, l'immagine acquisita viene processata dall'applicazione che vedremo nella sezione 4. Infine restituisce il risultato memorizzandolo sulla memoria della Scheda "Rasperry Py".

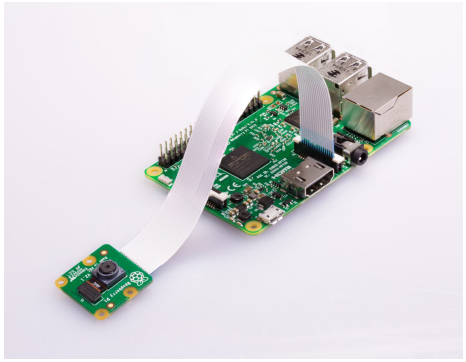


Figura 4: Rasperry Cam

Sviluppi futuri Al fine di un controllo più capillare, si procederà con lo sviluppo di un'applicazione per dispositivi mobile, multipiattaforma. Evitando così in primis l'installazione e la calibrazione delle camere appena viste. In secondo luogo, così facendo potremmo evitare anche i problemi di acquisizione immagini non soddisfacenti, come ad esempio immagini che ritraggono persone dinanzi lo scaffale, che impedirebbero la corretta visione del planogram.

2.4 IDE

Come ambiente di sviluppo integrato **(integrated development environment ovvero IDE)**, ho scelto XCode poiché mi permette uno sviluppo maggiormente efficiente in diversi linguaggi di programmazioni. In particolare grazie a diversi tool per HTML [3] , JS [4], ... è possibile avere un'anteprima diretta del possibile risultato live durante lo sviluppo. Mentre per modifiche brevi durante l'esecuzione e per lo sviluppo dell'immagine Docker, attraverso il dockerfile, ho utilizzato l'editor "nano" disponibile per macchine linux-based.

2.5 Chrome

Google Chrome è un web browser multi-piattaforma sviluppato da Google. Ho scelto Chrome poiché offre una enorme possibilità per gli sviluppatori, di controllo di eventuali errori sulle richieste effettuate attraverso i diversi di protocolli. Inoltre permette alcune funzionalità per lo sviluppo di web-page tramite alcune estensioni.

2.6 Docker

Docker è un progetto open-source che automatizza il deployment di applicazioni all'interno di contenitori software, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo di Linux. Una breve spiegazione della sua enorme diffusione è riassunta in questi punti:

- La community Docker Open Source si impegna a migliorare queste tecnologie a vantaggio di tutti.
- Docker Inc., sviluppa i progetti della community Docker rendendoli più sicuri, e condivide le migliorie apportate con la community più ampia. Inoltre, offre soluzioni migliorate di livello enterprise.

Docker, considera i container come macchine virtuali modulari estremamente leggere, offrendo la flessibilità di creare, distribuire, copiare e spostare i container da un ambiente all'altro, ottimizzando così le app per il cloud.

3 Amazon Web Services

3.1 S3 - Storage

Quando si parla di **AWS S3** , si parla innanzitutto di Cloud Storage, un Web Service che ci permette di salvare, accedere e/o fare un backup dei nostri da-

[3] HTML: l'HyperText Markup Language
[4] JS: JavaScript

ti attraverso internet. In particolare si basterà una browser, una connessione a internet e saremo in grado di operare con i nostri dati. L'utilizzo di S3 è fortemente spinto in logica Cloud Computing in quanto ci permette un sistema "ALL-IN" all'interno di AWS, fornendo così un meccanismo di sicurezza aggiuntivo e di una maggior velocità di esecuzione e/o elaborazione dei nostri dati.

AWS S3 al suo interno è provvisto di diverse possibilità di storage aggiuntive e diversi servizi aggiuntivi, che possiamo riassumere brevemente in questo elenco:

- **EBS** (Elastic Block Store) attached istance, ovvero la possibilità di utilizzare dischi a stato solido, noti come SSD [5].
- **EFS** (Elastic File System), ovvero diversi File System per diversi OS [6] garantendo così un meccanismo di multi-piattaforma
- **GLACIER** è un servizio di storage che garantisce sicurezza, durabilità e flessibilità per l'archiviazione e il backup dei dati
- **STORAGE GATEWAY** è un ulteriore servizio che di storage su cloud ibrido che ti offre l'accesso locale, praticamente illimitato, a uno storage su cloud; è comunemente utilizzato per trasferire dati da locale al cloud e viceversa. Fornendo la possibilità di salvare anche dati direttamente sul cloud, accessibili poi anche da locale.
- **SNOWBALL** è un servizio fondamentale per chi opera con grosse quantità di dati, in zone remote o senza una connessione ad internet. SNOWBALL, provvede l'attrezzatura e il recupero di quest'ultima al fine di salvare su cloud i dati. *Snowball edge* e *Snowball mobile* permettono di attuare il meccanismo appena citato.

3.1.1 Come funziona

S3 ci permette di creare diversi **Bucket** che possiamo immaginarci come dei cloud service diversi tra loro, che però di base offrono tutte il medesimo servizio, ovvero quello di salvare, leggere, e/o modificare gli **Object**. I **Object** non sono altro che i file con cui intendiamo operare, seguiti da chiavi e metadati che ci permetto di generare un ID univoco, ad ogni creazione di un oggetto. Per esempio possiamo vedere nel dettaglio come opera S3.

1. **Object** ovvero l'oggetto che intendiamo salvare: "Folder(key)/Pippo.jpg" la stringa così composta ci permette di generare il Version ID

[5] SSD: solid-state drive
[6] OS: Operating System

2. **Bucket** qui va riportato il nome del bucket che intendiamo utilizzare "NameBucket"
3. **LinkAdress** l'URL [7] ovvero l'indirizzo alla quale la nostra risorsa sarà accessibile sarà così formato:
   https://s3.amazonaws.com/NameBucket/Folder/Pippo.jpg.

Osservazione: il nome del bucket deve essere univoco, infine in fase di creazione del bucket è necessario fornire la regione ove intendiamo posizionare il nostro bucket.

Altre funzionalità Una volta creato il bucket, possiamo definire delle *Policy* per quanto riguarda la sicurezza, il versioning e lifecycle del nostro bucket e dei singoli file memorizzati al suo interno. In particolare è buona prassi settare anche il tipo di storage class durante la creazione del bucket, fornendo così un ulteriore efficienza.

Tipi di Storage Class Qui di seguito sono riportate le principali tipologie di S3 storage class: S3 Standard Class, Frequent Data Access, Infrequent Data Access, Glicer (High performances security), One Zone - IA (Infrequently accessed and stored in 1 single region), Standard reduced redundancy storage.

3.1.2 Policy & Funzionalità

- **Lifecycle Management**: Amazon S3 applica un insieme di regole che definiscono l'azione di un gruppo di oggetti
  1. Spostare i dati tra diverse classi di archiviazione
  2. Applica un insieme di regole a un gruppo di oggetti (ad esempio cancella automaticamente i dati dopo un certo periodo . . .)
- **Bucket Policy**: Regole di sicurezza (Access Deny/Allow)
- **Data Protection**
  1. Data Encryption (At rest o In Motion) JSON Script - Policy GEN (BULLER ARN)
  2. Versioning
- **Cross Region Replication**
- **Transfer Acceleration**: Consente di trasferire dati da bucket da regioni diverse con percorso ottimizzato: Edge Location $\iff$ Cloud Front

[7] URL: Uniform Resource Locator

3.2 ECS - Manage Container

AWS Elastic Container Service è un servizio offerto dal mondo AWS che ci permette la gestione dei container creati, come ad esempio, il loro avvio o il loro spegnimento. Ci permette inoltre di non operare solo un singolo container, ma con un *cluster*, ovvero gruppi di container. In sintesi una delle funzionalità cardine è la possibili di gestire i container creati tramite "Docker" che vedremo nella sezione 3.7

PRIMA ECS vs DOPO ECS Prima di ECS i problemi più comune erano : l'applicazione non funziona, server down o memoria piena. Con l'introduzione di ECS, abbiamo la possibilità di mantenere la disponibilità dell'applicazione e di permettere inoltre di garantire ad ogni utente di scalare container quando è necessario, senza alcun intervento da parte dell'utente.

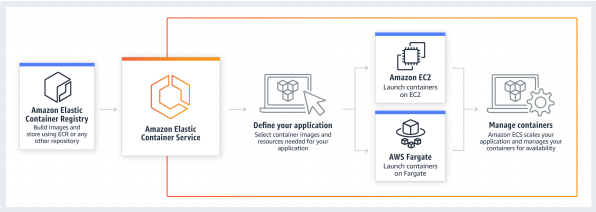


Figura 5: Elastic Container Service

3.2.1 Come funziona

Una volta creato un container, attraverso la procedura guidata ove va specificato le caratteristiche hardware della "macchina virtuale" a noi necessaria e impostata l'immagine docker, la creazione del container termina qui.

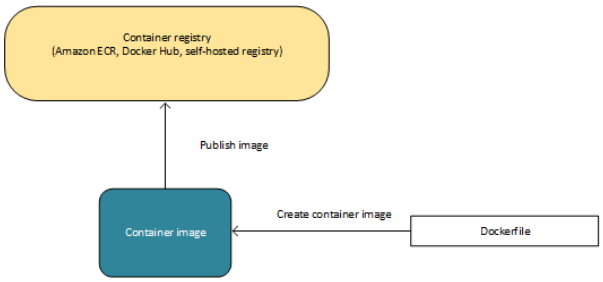


Figura 6: Immagine container

Tuttavia, il nostro container non è ancora in grado di svolgere alcun task, ecco quindi che ne va definito uno, che ci permetterà di operare con il nostro container.

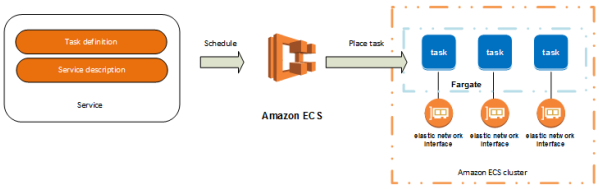


Figura 7: Panoramica sul servizio Fargate

Si possono eseguire i task su un'infrastruttura serverless gestita da AWS Fargate come nel nostro caso. Ma per ulteriori controlli dell'infrastruttura, è possibile anche eseguire attività e servizi su un cluster di Amazon EC2. Amazon ECS consente di avviare e arrestare le applicazioni basate su container con una semplice API, rendendo così il tutto molto user friendly. Tra le varie potenziali ci sono: il recupero dello stato del cluster da un servizio centralizzato e, per utenti esistenti di Amazon EC2, accedere a molte caratteristiche familiari di EC2. Come visto per gli altri servizi AWS è possibile pianificare il posizionamento dei container nel cluster in base a esigenze di risorse, policy di isolamento e requisiti di disponibilità. Con Amazon ECS, non si devono gestire i propri sistemi di gestione dei cluster e della configurazione o sul dimensionamento dell'infrastruttura di gestione. Amazon ECS può essere utilizzato per creare un'esperienza di implementazione e compilazione coerente, gestire e di scalare i carichi di lavoro batch e di Extract-Transform-Load (ETL) e creare applicazioni sofisticate su un modello di micro servizi.

"ECS, allows migrates application to the cloud without changing code"

Vantaggi dell'utilizzo di AWS ECS I vantaggi principali dell'utilizzo di AWS ECS sono : Sicurezza migliorata tramite policy consenti & nega, efficienza dei costi, facilmente estensibile scalabile e una compatibilità migliorata.

3.2.2 Funzionamento di ECS

Nell'immagine sottostante è visibile un schema molto basilare del funzionamento di ECS (Fig.8)

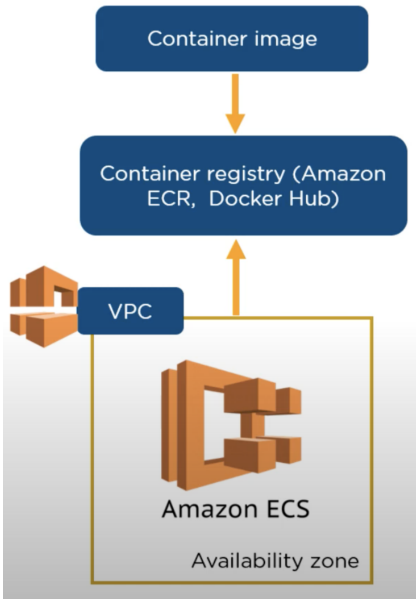


Figura 8: Schema ECS

Nel dettaglio ...

- Container Image Modello di istruzioni che viene utilizzato per creare contenitori (Quale sistema operativo, Software aggiuntivo ...)
- Container Registry Si tratta di un servizio che viene utilizzato per ospitare e distribuire immagini Docker tra gli utenti
- VPC Servizio di fornire un isolamento del task; consentendo di lanciare risorse AWS, come AWS ECS istanze in una rete virtuale, che si è specificato precedentemente.

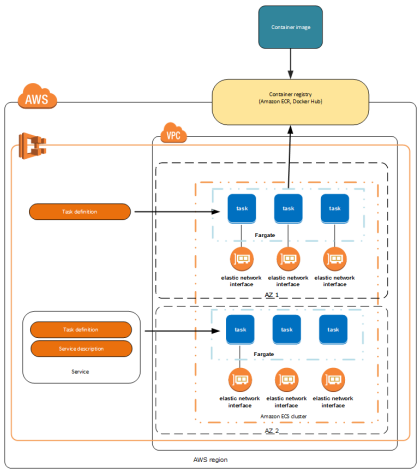


Figura 9: Panoramica completa di ECS

Task

- I tasks vengono avviati utilizzando Fargate
- Fargate è un motore di elaborazione in ECS che permette ad un utente di lanciare container senza dover monitorare i cluster
- I Task hanno 2 componenti : ECS container instance (Una parte di AWS EC2 istanza che gestisce AWS ECS container agent) e Container Agent (Responsabile della comunicazione tra AWS ECS e l'istanza. Inoltre, fornisce lo stato dei contenitori in esecuzione)

Elastic Network Interface Elastic network interface una rete di interfaccia virtuale che può essere collegata a un'istanza (ad esempio, VPC)

Amazon ECS Cluster Un cluster è un insieme di istanze container ECS, gestisce il processo di pianificazione, monitoraggio e scalabilità delle richieste.

3.3 Lambda

AWS Lambda è un servizio disponibile tramite AWS console, che consente di eseguire codice senza dover effettuare il provisioning né gestire server. Con Lambda, puoi eseguire codice per qualsiasi tipo di applicazione o servizio di back-end, senza bisogno di amministrazione. Una volta caricato il codice, Lambda si prende carico delle azioni necessarie per eseguirlo e ricalibrarne le risorse con la massima disponibilità. Puoi configurare il codice in modo che venga attivato automaticamente da altri servizi AWS oppure richiamato direttamente da qualsiasi applicazione Web o mobile.

Vantaggi

- Nessun server da gestire AWS Lambda esegue automaticamente il codice senza dover effettuare il provisioning né gestire server. Devi solo scrivere il codice e caricarlo in Lambda. Dimensionamento continuo
- Dimensionamento continuo AWS Lambda ridimensiona automaticamente le risorse dell'applicazione eseguendo il codice in risposta a ogni trigger. Il codice viene eseguito in parallelo ed elabora ciascun trigger separatamente, ricalibrando le risorse in base al carico di lavoro.
- Prestazione costante Con AWS Lambda puoi ottimizzare la durata di esecuzione del codice scegliendo la dimensione di memoria adatta alla tua funzione. Puoi anche abilitare Provisioned Concurrency per mantenere inizializzate le funzionalità, pronte a reagire in meno di cento millisecondi.

3.3.1 Come funziona

Una breve rappresentazione grafica di come funziona Lambda

Figura 10: Come funziona ?

3.4 AWS Cognito

Amazon Cognito fornisce autenticazione, autorizzazione e gestione degli utenti per le applicazioni Web e mobili. Gli utenti possono accedere direttamente con un nome utente e una password, oppure tramite terze parti. I due componenti principali di Amazon Cognito sono i **pool di utenti** e i **pool di identità**.

I pool di utenti sono directory utente che forniscono opzioni di registrazione e di accesso agli utenti delle tue app.

I pool di identità consentono di concedere agli utenti l'accesso ad altri servizi AWS.

"È possibile usare i pool di identità e i pool di utenti separatamente o insieme."

3.4.1 Come funziona

Attraverso il grafico sottostante è possibile visionare l'workflow per autenticare l'utente e quindi concedere a quest'ultimo l'accesso a un altro servizio AWS. Nella prima fase l'utente dell'applicazione accede mediante un pool di utenti e riceve i token del pool di utenti dopo aver completato l'autenticazione. Quindi, l'applicazione scambia i token del pool di utenti con le credenziali AWS attraverso un pool di identità. L'utente dell'applicazione può quindi utilizzare le credenziali AWS per accedere ad altri servizi AWS come Amazon S3 o DynamoDB.

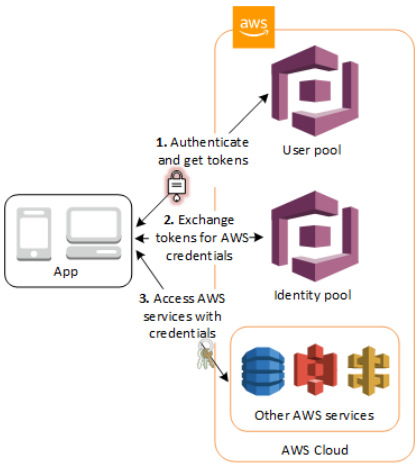


Figura 11: Come funziona ?

3.5 IAM

AWS IAM [8] consente di gestire in sicurezza l'accesso ai servizi e alle risorse AWS. Grazie ad IAM, è possibile creare e gestire utenti e gruppi AWS e utilizzare autorizzazioni per consentire o negare l'accesso alle risorse AWS. IAM è una funzionalità maggiormente utilizzata dell'account AWS principale, noto anche come Root IAM.

Casi d'uso

- Controllo di accesso granulare alle risorse AWS IAM consente ai tuoi utenti di controllare l'accesso alle API del servizio AWS e a risorse specifiche. IAM consente inoltre di aggiungere condizioni specifiche, per esempio l'ora del giorno, per controllare l'utilizzo di AWS da parte degli utenti, l'indirizzo IP di origine, l'eventuale uso del protocollo SSL oppure di un dispositivo con autenticazione a più fattori.

[8] IAM: Identity and Access Management

- Multi-Factor Authentication per utenti con elevati privilegi Proteggi il tuo ambiente AWS con l'autenticazione a più fattori o MFA. MFA chiede agli utenti di provare il possesso fisico di un token MFA hardware o di un dispositivo mobile compatibile con MFA fornendo un codice MFA valido.
- Analisi degli accessi IAM ti aiuta ad analizzare gli accessi nel tuo ambiente AWS. È inoltre possibile identificare e perfezionare facilmente le policy per consentire l'accesso solo ai servizi utilizzati. Questo ti aiuta ad aderire meglio al principio del privilegio minimo.
- Integrazione con directory aziendali Puoi usare IAM per concedere ai tuoi dipendenti e alle tue applicazioni accesso federato alla Console di gestione AWS e alle API del servizio AWS, impiegando i tuoi sistemi di identità già esistenti, come Microsoft Active Directory.

3.6 Cloud Watch

Amazon CloudWatch è un servizio di monitoraggio e osservabilità. CloudWatch fornisce dati e analisi concrete per monitorare le applicazioni, rispondere ai cambiamenti di prestazioni a livello di sistema, ottimizzare l'utilizzo delle risorse e ottenere una visualizzazione unificata dello stato di integrità operativa. CloudWatch raccoglie dati di monitoraggio e operativi sotto forma di log, parametri ed eventi, fornendo una visualizzazione unificata delle risorse AWS, sulle applicazioni e i servizi eseguiti in AWS e su server locali. Si può utilizzare CloudWatch per rilevare comportamento anomalo nei tuoi ambienti, impostare allarmi, visualizzare log e parametri uno di fianco all'altro, intraprendere azioni automatiche, risolvere problemi e scoprire informazioni per garantire che le applicazioni vengano eseguite senza intoppi.

Vantaggi I vantaggi principali di utilizzo del Cloud Watch sono: Osservabilità in un'unica piattaforma su più applicazioni e sull'infrastruttura, Il metodo più facile per raccogliere parametri in AWS e in locale, Migliora le prestazioni operative e l'ottimizzazione delle risorse, Assicurati visibilità operativa e informazioni utili, Ricavare analisi concrete dai log.

3.7 Docker

Docker è un progetto open source che automatizza l'implementazione di applicazioni software all'interno di container fornendo un ulteriore livello di astrazione e automazione di OS-LEVEL VIRTUALIZATION SU LINUX. In particolare Docker è uno strumento per automatizzare l'implementazione di un'applicazio-

ne come contenitori leggeri in modo che l'applicazione possa funzionare in modo efficiente in diversi ambienti.

Figura 12: Docker

Docker Container È un leggero pacchetto di software che consiste di tutte le dipendenze (Code, Framework, Libraries, ...) necessarie per eseguire l'applicazione.

I vantaggi dell'utilizzo di Docker I vantaggi principali sono: servizi ad alta scalabilità ed efficienza, uptime Shoortboot-up (Kernel Light), volumi di dati riutilizzabili, applicazioni isolate. Inoltre l'utilizzo di Docker ci permette, di pacchettizzare un'applicazione con tutte le sue dipendenze in un'unità standardizzata per lo sviluppo software.

3.7.1 Container

Sfruttando la meccanica di basso livello del sistema operativo host, container fornire la maggior parte dell'isolamento delle macchine virtuali a frazione della potenza di calcolo.

Perché utilizzare i container ? I container offrono un meccanismo logico di confezionamento in cui le applicazioni possono essere astratte dall'ambiente in cui vengono effettivamente eseguite. Le applicazioni basate su container sono distribuite in modo semplice e coerente, indipendentemente dal fatto che l'ambiente di destinazione sia un data center privato, il cloud pubblico o anche il pc di uno sviluppatore. I container offrono la possibilità di creare ambienti prevedibili che sono isolati dal resto delle applicazioni e possono essere eseguiti ovunque. Un controllo più granulare delle risorse, dando all'infrastruttura una maggiore efficienza, che può portare a un migliore utilizzo delle risorse di calcolo.

3.7.2 Terminologia

Images I progetti della nostra applicazione che costituiscono la base dei contenitori. Nella demo di cui sopra, abbiamo usato il comando di docker pull per scaricare l'immagine busybox.

Containers Sono creati da immagini Docker e ci permettono di eseguire l'applicazione real-time. Creiamo un contenitore utilizzando docker run che abbiamo fatto utilizzando l'immagine busybox che abbiamo scaricato. Un elenco dei container in esecuzione può essere visto usando il comando "docker ps".

Docker Daemon Il servizio di background in esecuzione sull'host che gestisce la costruzione, l'esecuzione e la distribuzione di container Docker. Il demone è il processo che viene eseguito nel sistema operativo con cui i client interagiscono.

Docker Client Lo strumento a riga di comando che permette all'utente di interagire con il demone. Più in generale, ci possono essere altre forme di clienti troppo, come Kitematic che forniscono anche una GUI per gli utenti

Docker Hub Un registro delle immagini Docker. Si può pensare al registro come una directory di tutte le immagini Docker disponibili. Se necessario, è possibile ospitare i propri registri Docker e utilizzarli per estrarre le immagini..

3.7.3 Image

Base images Sono immagini che non hanno un'immagine padre, di solito immagini con un sistema operativo come ubuntu, busybox o debian.

Child images Sono immagini che si costruiscono su immagini di base e aggiungono loro funzionalità aggiuntive.

Official images Sono immagini che sono ufficialmente mantenute e supportate dagli utenti ufficiali Docker. Nell'elenco delle immagini qui sopra, le immagini di python, ubuntu, busybox e hello-world sono immagini ufficiali.

User images Sono immagini create e condivise da utenti generici come te e me. Si basano su immagini di base e aggiungere funzionalità aggiuntive. Tipicamente, questi sono formattati come nome utente/immagine.

3.7.4 Come creare un'immagine

1. Sfoglia le directory nel tuo PC e crea una nuova cartella, dove verrà memorizzata l'immagine

```
denisbernovschi@MBP-di-Denis docker-curriculum-master % tree flask-app
flask-app
├── Dockerfile
├── Dockerrun.aws.json
├── app.py
├── requirements.txt
└── templates
    └── index.html

1 directory, 5 files
```

Figura 13: Directory & File Necessary

2. Crea un Docker File

```
denisbernovschi@MBP-di-Denis flask-app % cat Dockerfile
FROM python:3

# set a directory for the app
WORKDIR /usr/src/app

# copy all the files to the container
COPY . .

# install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# tell the port number the container should expose
EXPOSE 5000

# run the command
CMD ["python", "./app.py"]
```

Figura 14: Create a Docker File

- **FROM** Le parola-chiave che ci permette di specificare la nostra base-immagine
- **WORKDIR** Per impostare una directory di lavoro e quindi **COPY** copiare tutti i file per la nostra applicazione
- **RUN ...** Per installare le dipendenze
- **EXPOSE n-port** Dobbiamo specificare il numero di porta che deve essere esposto
- **CMD [...]** Invece dei punti, dobbiamo digitare i comandi run

3. Creazione della nostra applicazione: qui dobbiamo scrivere il codice relativo alla nostra applicazione, nel linguaggio che più preferiamo, Python, Java, C (come nel nostro caso) ...

4. Building della nostra immagine

```
docker build -t yourusername/name-image .
```

5. Ora è possibile eseguire l'immagine

```
docker run -p 8888:5000 yourusername/name-image
```

3.7.5 Caricare la tua immagine su Docker Hub

1. Docker Login

```
docker login
Username: yourusername
Password: ****************
Login Succeeded !
```

2. Pubblicare la tua nostra immagine

```
docker push yourusername/image-name
```

3. Ora ti basterà andare a https://hub.docker.com/r/yourusername/image-name/ per visualizzare la tua immagine. Ora che l'immagine è online, chiunque abbia installato Docker può giocare con l'app digitando un solo comando.

```
docker run -p 8888:5000 yourusername/image-name
```

3.8 AWS Billing and Cost Management

AWS Billing and Cost Management è il servizio che si utilizza per pagare la bolletta AWS, monitorare il vostro utilizzo, e analizzare e controllare i costi.

AWS addebita automaticamente sul metodo di pagamento scelto al momento della registrazione del nuovo account AWS. Gli addebiti compaiono sulla fattura mensile della carta di credito. AWS Billing and Cost Management fornisce strumenti utili per aiutarti a raccogliere informazioni relative ai costi e all'utilizzo, analizzare i driver di costo e le tendenze di utilizzo e agire per budget.

Tramite questo servizio siamo quindi in grado di osservare i costi sostenuti da ciascun servizio. In particolar modo, possiamo discernere le spese sostenute da ciascun servizio. Come noto, AWS è una piattaforma scalabile, e in quanto tale ci permette di utilizzare i servizi a noi più necessari, senza sostenere costi esorbitanti ed inutili alla nostra applicazione, così facendo abbiamo un approccio "you only pay what you use". Siamo inoltre in grado di smentire parte dell'ideologia del momento che sostiene che utilizzando un approccio Cloud Based si rischia di incorrere in costi eccessivi e poco giustificabili.

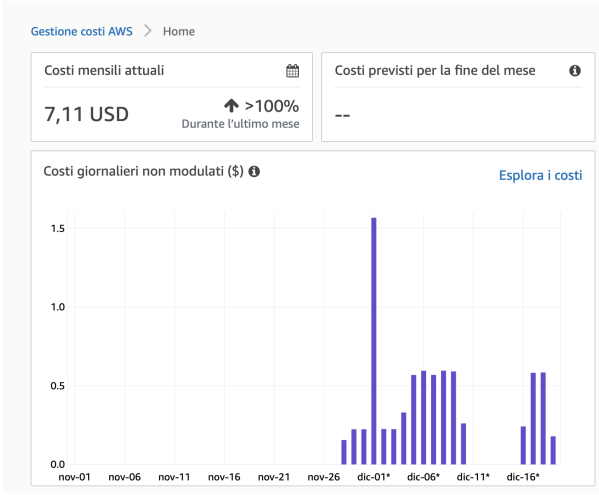


Figura 15: AWS Billing Monthly

Nella (Fig.15) possiamo vedere il costo dell'intera piattaforma AWS con i relativi servizi attivati. In particolare possiamo notare come l'utilizzo non abbia costi esorbitanti, in quanto bisogna osservare che durante la fase di testing, l'infrastruttura richiede un maggior numero di risorse. A regime, possiamo ipotizzare che il costo scenderà di circa il 40%, 50%.

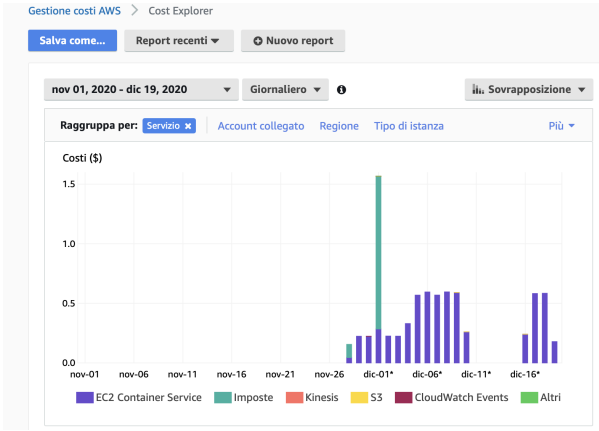


Figura 16: AWS Billing Monthly(Service)

Mentre nella (Fig.16), sono evidenziati i costi giornalieri suddivisi per servizi. Osserviamo quindi che il costo maggior è dato appunto dal ECS (Elastic Container Service) che ospita la nostra applicazione, gli altri servizi connessi hanno costi trascurabili.

Costo per ogni verifica In seguito a diverse esecuzioni e prove sperimentali, è osservabile che il costo di ogni singola operazione è 0,03€circa. Si potrebbe quindi ipotizzare di offrire questo servizio ad un potenziale cliente sotto forma di pacchetto di n. operazioni disponibili sostenendo un importo onnicomprensivo.

4 L'Applicazione

4.1 An IoT Edge-Fog-Cloud Architecture for Vision Based Planogram Integrity

Il planogramma vuole fornire la migliore collocazione dei prodotti sugli scaffali, con l'obiettivo di migliorare l'esperienza e la soddisfazione del cliente, aumentare le vendite e i profitti e gestire meglio i prodotti sugli scaffali. Si utilizza un'architettura di fog computing consisteva di nodi bordo che sono telecamere a basso costo in grado di trasmettere in modalità wireless foto di scaffale ai nodi di nebbia locali nello stesso negozio. Questi ultimi esaminano le immagini provenienti dai bordi e invia i risultati al cloud per ulteriori aggregazioni di dati e analisi.

4.2 Shelf planogram

E' una mappa visiva dettagliata dei prodotti negli scaffali che stabilisce dove i prodotti dovrebbero essere posizionati nel negozio con l'obiettivo di migliorare le vendite e di fornire la posizione migliore per i fornitori. L'importante è sviluppare un planogramma di uno scaffale che rifletta la reale necessità che un prodotto si trovi in una posizione precisa in un particolare lasso di tempo. Elementi (univocamente identificati da un **Stock Keeping Unit, SKU**) sono nelle loro posizioni stabilite, nella loro giusta quantità, nella loro visualizzazione facciale, e con informazioni corrette riguardanti ad esempio il loro prezzo. Un problema è che la conformità è messa continuamente alla prova e quindi è necessario un controllo continuo e l'uso delle risorse. *Il sistema proposto, per mantenere l'integrità del planogramma, utilizza una telecamera frontale installata in un negozio reale, al fine di testare le prestazioni in situazioni reali.*

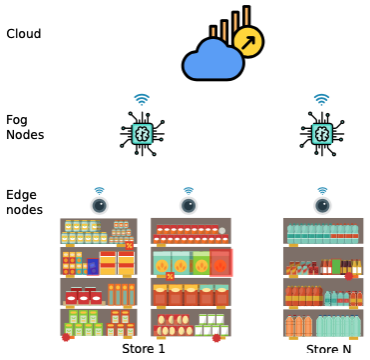


Figura 17: IoT Edge-Fog-Cloud Architecture

4.3 Come funziona ?

Questi brevetti utilizzano anche algoritmi che sono in grado di estrarre alcune caratteristiche dalle immagini di un plano-gramma. Un controllo continuo è molto importante non solo per verificare la conformità del plano-gramma ma anche per gestire lo shelf-out-of-stock, al fine di rendere sempre disponibili tutti i prodotti a scaffale. C'è poi la necessità di introdurre un sistema automatizzato che sia in grado di analizzare il comportamento del cliente, di valutare la conformità del planogramma, e di rilevare shelf-out-of-stock in modo rapido e affidabile.

Durante l'input del sistema responsabile del riconoscimento sono informazioni specifiche del planogramma: numero di scaffali, numero di prodotti e tipo di prodotti

Metodi basati sulla classificazione dopo il rilevamento di oggetti, il lavoro di utilizza in precedenza la trasformazione Hough per il rilevamento di scaffali e quindi supportare algoritmo di macchine vettoriali per classificare le immagini. Nel lavoro dello stesso classificatore è stato utilizzato per la classificazione delle immagini, mentre per il riconoscimento degli oggetti è stato applicato il metodo di trebbiatura HSV. Invece in, è stata implementata una rappresentazione grafica, dove gli oggetti erano associati a nodi e i nodi erano collegati da pesi con un valore calcolato attraverso le occorrenze degli oggetti. Secondo i risultati del confronto, anche se il metodo di riconoscimento dell'oggetto senza il rilevamento dell'oggetto è un metodo veloce e facile, non è molto efficiente. Inoltre, per ottenere un significativo set di dati di immagini è necessario avere alcune precauzioni (Diverse telecamere, diverse distanze, diversi angoli, e diverse condizioni di luce . . .).

4.4 Implementazione

Una telecamera frontale installata sul ripiano opposto riprende l'immagine del planogramma frontale ad intervalli regolari, con una frequenza fissa. Attraverso le tecniche di riconoscimento delle immagini, il software analizza e verifica automaticamente la corrispondenza con il planogramma prestabilito, e quindi valuta la conformità del planogramma. È in grado di rilevare in tempo reale eventuali imprecisioni rilevate come errori di posizionamento o shelf-out-of-stock e inviare un avviso (mail o sms). La fotocamera descritta fornisce immagini per catturare la posizione fisica dei prodotti sugli scaffali. Il sistema di telecamere conosce il planogramma, che è considerato la migliore posizione dei prodotti sugli scaffali.

Procedimento Quindi, per rilevare la conformità al planogramma, il sistema corrisponde automaticamente al planogramma di base, fornito nel server, e le immagini dai negozi. Partendo dall'immagine acquisita dal modulo di acquisizione della smart camera, il sistema implementato abbina il planogramma di base con il planogramma acquisito e fornisce un'immagine di confronto, calcolando le differenze tra le due immagini. L'architettura di fog computing, dove i nodi di bordo sono rappresentati da telecamere a basso costo in grado di trasmettere in modalità wireless le foto di scaffale ai nodi di nebbia locali situati nello stesso negozio. Questi nodi di nebbia analizzano le foto provenienti dai bordi e trasmettono i risultati. L'attuale foto acquisita viene prima rettificata attraverso una trasformazione in prospettiva, perché per prendere l'intero ripiano nella foto la fotocamera deve essere installata sulla parte superiore del ripiano opposto, con un'inclinazione variabile a seconda dell'altezza di installazione.

4.5 Elaborazione dell'immagine

Una finestra scorrevole di dimensioni fisse di un classificatore SVM viene eseguito sulla piramide di HOG di immagini, ottenuto utilizzando funzionalità di HOG [9]. Questo genera una matrice $M_{current}$ di bounding boxes che è confrontato con M_{model} precedentemente generata in fase di configurazione per l'immagine di riferimento del planogramma. Il classificatore di cui sopra è stato addestrato utilizzando solo 5 immagini di scaffale della stessa categoria (shampoo o pannolini) trovate su Internet e manualmente etichettate disegnando i bounding boxes relativi al prodotto. La matrice M_{model} viene utilizzato anche nella fase di configurazione per ritagliare l'immagine in modo da elementi e generare modelli di immagine dei prodotti. Ogni elemento della matrice $M_{current}$ contiene le coordinate dei bounding box. La corrispondenza mira a trovare caselle di delimitazione nell'immagine corrente che sono correlate alle caselle di delimitazione originali

[9] HOG: Histogram of Oriented Gradients

in base alla loro posizione. Ogni volta che uno spazio vuoto (prodotto veramente mancante o mancato rilevamento) viene rilevato tra due bounding boxes consecutivi, il corrispondente elemento M_{model} è inserito in $M_{current}$, questa fase di pre-trattamento trasforma $M_{current}$ al fine di essere comparabile in termini di elementi con M_{model} (stessa struttura e numero di elementi). ogni elemento di $M_{current}$ viene confrontato con tutti i modelli generati durante la fase di configurazione. Questo abbinamento è portato in un ambiente multi-threading. I restanti possibili modelli corrispondenti vengono poi abbinati utilizzando i punti chiave SIFT, generando così una lista ordinata di modelli, dal più simile al meno simile. Dopo una traduzione dello spazio colore, da RGB a HSV, viene calcolato l'istogramma del colore. Un indice di salienza, definito come il numero totale di pixel veri rimanenti dopo aver triplicato la mappa di salienza, della regione di interesse è calcolato per rilevare il prodotto in una posizione non conforme (ruotata o non completamente visibile) o per rafforzare il rilevamento OOS. A questo punto è stato generato un elenco di prodotti/OOS rilevati e deve essere confrontato con il planogramma originale per verificarne la corrispondenza. La lista viene inviata al cloud dai "fog nodes" le telecamere per intenderci.

Classification Results L'algoritmo elabora due immagini in ingresso: l'immagine che rappresenta il planogramma accettato (planogramma di base) e l'immagine acquisita dalla telecamera ad intervalli regolari. Deve stabilire come l'immagine acquisita è diversa dal planogramma accettato come corretto.

Il software rileva correttamente che due facce di prodotti sono stati scambiati con rosso riquadri di delimitazione. Mentre le caselle di delimitazione nero indicano che il software rileva la presenza di alcuni prodotti aggiunti rispetto al numero originale di prodotti per linea. Le caselle di delimitazione blu indicano che alcuni prodotti non sono in una posizione non conforme o sono parzialmente occlusi. Gli unici errori derivano da prodotti estremamente simili. Tre caselle di delimitazione verde rappresentano OOF rilevati dal sistema, ma un OOF è sbagliato a causa di un oggetto posizionato in modo non conforme e erroneamente rilevato. Due oggetti rilevati con caselle di delimitazione blu sono indicati in una posizione non conforme e non molto visibile.

4.6 Conclusione

Il sistema può anche riconoscere diversi casi di oggetti in posizioni non conformi. Può anche rilevare la quantità e la presenza di tutti gli SKU anche se si trovano in posizioni diverse rispetto all'origine. Il rilevamento OOF ha una buona

affidabilità e può essere configurato para-metricamente per essere più o meno rigido, a seconda dei prodotti da controllare

In sintesi ... Nel paper [1] dal titolo "An IoT Edge-Fog-Cloud Architecture for Vision Based Planogram Integrity" si pone l'attenzione sulla disposizione dei prodotti all'interno di uno scaffale, in particolare la nostra applicazione prevede un controllo sulla disposizione dei prodotti su quest'ultimo. L'applicazione è in grado di calcolare la correttezza di un planogramma con buona affidabilità.

- **SKU *(stock keeping unit)*** in OOS *(Out of stocks)*: Il prodotto è totalmente mancante sullo scaffale
- **SKU *(stock keeping unit)*** in OOF *(Out of Facing)*: un prodotto manca sullo scaffale. Un prodotto è ogni singola unità di una SKU, quindi ogni singolo elemento posto sullo scaffale;
- **SKU non compliant**: un prodotto non corrisponde a nessun modello, pertanto non fa parte del planogramma;
- **SKU in a non compliant position**: un prodotto ha un cattivo posizionamento, come rotazione, traduzione e/o occlusione parziale

La verifica della corretta disposizione dei prodotti sul planogramma avviene attraverso delle telecamere posizionate sullo scaffale del lato opposto, così da inquadrarlo integralmente. Ad intervalli regolari viene scattata una foto, che viene poi confrontata con una memorizzata precedentemente con la corretta disposizione dei prodotti.

4.7 Problematiche e obiettivi

Di seguito osserviamo una panoramica delle varie problematiche emerse durante l'analisi:

- Prodotti estremamente simili;
- Oggetto posizionato in modo non conforme e erroneamente rilevato;
- Immagine acquisita è inutilizzabile

Prodotti estremamente simili Prodotti estremamente simili, tendo ad ingannare l'applicativo, creando dei "falsi-positivi". Ovvero diversi prodotti simili, potrebbero essere in posizioni errate, oppure viceversa il prodotto simile pur trovandosi nella posizione corretta viene classificato come errore di posizionamento. E' una problematica molto frequente dovuta al fatto che spesso i

clienti posizionano in maniera errata e/o non corretta i prodotti in seguito ad interazione con essi.

Figura 18: Planogram Base

Figura 19: Planogram Acquired

Oggetto posizionato in modo non conforme e erroneamente rilevato Sono emersi ulteriori problematiche nei casi in cui diversi prodotti si trovano posizionati nella posizione corretta, ma ruotati, capovolti oppure inclinati. Così facendo l'applicativo non è in grado di confrontare la corretta posizione dei prodotti, generando così possibili errori aggiuntivi.

Figura 20: Planogram Base

Figura 21: Planogram Acquired

Soluzioni Per quanto concerne le soluzioni alle problematiche fin qui emerse, non è compito di questa tesi affrontare possibili risoluzioni, ma bensì prevedere alla realizzazione della piattaforma sfruttando i servizi di AWS [10]

Immagine acquisita è inutilizzabile Un ulteriore problematica è dovuta quando l'acquisizione dell'immagine non può essere utilizzata dalla nostra applicazione. Pensiamo ad esempio quando acquisiamo una foto al nostro planogram con ostacolo che impedisce l'intera visualizzazione come ad esempio un soggetto, uno bancale ...

[10] AWS : Amazon Web Services

5 Realizzazione

5.1 Creazione della nostra applicazione di prova

5.1.1 Codice per prelevare un immagine da un S3 bucket

In questa sezione vedremo alcuni snippet di codice che ci permettono di prelevare l'immagine da S3 attraverso delle richieste. Il codice intero è realizzato in linguaggio C.

```
#include <sstream>
#include <opencv2/opencv.hpp>
#include <aws/core/auth/AWSCredentialsProvider.h>
#include <aws/core/utils/StringUtils.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/GetObjectRequest.h>
const Aws::String AWS_ACCESS_KEY_ID = "KEY_ID";
const Aws::String AWS_SECRET_ACCESS_KEY = "ACC";
```

Listing 1: Library

Librerie

- **sstream** Fa parte della Libreria Standard C++. Si tratta di un file header che fornisce modelli e tipi che consentono l'interoperatività tra buffer di flusso e oggetti stringa.
- **opencv** Opencv è una libreria open source di computer vision e software di apprendimento automatico. Opencv è stato costruito per fornire un'infrastruttura comune per le applicazioni di computer vision e per accelerare l'uso della percezione della macchina nei prodotti commerciali. Essendo un prodotto con licenza BSD, Opencv rende facile per le aziende utilizzare e modificare il codice.

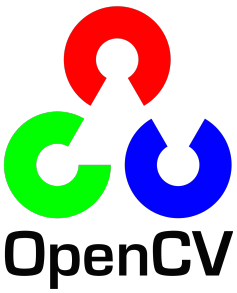


Figura 22: OpenCV

- **aws/core/auth** È una libreria di base AWS utilizzata per operazioni di autenticazione

- **aws/core/utils**È una libreria core AWS utilizzata per lavorare con Amazon Web Services (AWS), inclusi ARN, regioni, stadi, Lambdas, errori AWS, eventi stream, Kinesis, Dynamodb, Documentclients, ecc.
- **aws/s3/S3Client.h** È una libreria AWS utilizzata per interagire con Amazon Simple Storage Service.
- **aws/s3/model/...** È una libreria AWS utilizzata per modellare richieste put, richiesta get , ecc.

Figura 23: AWS Software Development Kit

Constants

- **AWS_ACCESS_KEY_ID** : username
- **AWS_SECRET_ACCESS_KEY_ID** : password

In questo snippet possiamo vedere le configurazioni di impostazione della richiesta e il percorso di archiviazione S3 da dove ottenere l'immagine.

```
int main(int argc, char** argv) {

    Aws::Client::ClientConfiguration config;
    config.scheme = Aws::Http::Scheme::HTTPS;
    config.connectTimeoutMs = 30000;
    config.requestTimeoutMs = 30000;
    config.region = Aws::Region::AP_NORTHEAST_1;

    Aws::S3::S3Client s3Client(Aws::Auth::AWSCredentials(
    AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY), config);
    Aws::S3::Model::GetObjectRequest getObjectRequest;
    getObjectRequest.SetBucket("mybucket");
    getObjectRequest.SetKey("path/to/img.jpg");
```

Listing 2: Main Function

Ora possiamo vedere la richiesta e controllare il risultato di quest'ultima

```
    auto getObjectOutcome = s3Client.GetObject(getObjectRequest);
    if (!getObjectOutcome.IsSuccess()) {
        std::cerr << "File download failed from s3 with error " <<
    std::endl;
```

```
4         std::cerr << getObjectOutcome.GetError().GetMessage() <<
      std::endl;
5         exit(1);
6     }
```

Listing 3: Request Download

Una volta che abbiamo l'immagine, è ora possibile accedere al buffer sottostante. Tuttavia, per gestire l'immagine con OpenCV, è necessario convertire quest'utlima attraverso l'istruzione di **imdecode**

```
1     std::stringstream ss;
2     ss << getObjectOutcome.GetResult().GetBody().rdbuf();
3     std::string str = ss.str();
4     std::vector<char> vec(str.begin(), str.end());
5     cv::Mat img = cv::imdecode(cv::Mat(vec), CV_LOAD_IMAGE_COLOR);
6     cv::imshow("window", img);
7     cv::waitKey(0);
8     return 0;
9 }
```

Listing 4: Converting image

5.1.2 Codice per caricare un immagine un S3 bucket

In questa sezione, possiamo alcuni snippet di codice che ci permettono di salvare l'immagine su un bucket di S3. Come prima partiamo dalle librerie a noi necessarie, successivamente vediamo la richiesta e il risultato finale.

```
1  #include <sstream>
2  #include <string>
3  #include <opencv2/opencv.hpp>
4  #include <aws/core/auth/AWSCredentialsProvider.h>
5  #include <aws/core/utils/StringUtils.h>
6  #include <aws/s3/S3Client.h>
7  #include <aws/s3/model/PutObjectRequest.h>
8  #include <aws/s3/model/Bucket.h>
9
10 const Aws::String AWS_ACCESS_KEY_ID = "KEY_ID";
11 const Aws::String AWS_SECRET_ACCESS_KEY = "ACC";
12 const String fileName = "filename.png";
```

Listing 5: Library & Costant

```
1 int main(int argc, char** argv) {
2    Aws::Client::ClientConfiguration config;
3    config.scheme = Aws::Http::Scheme::HTTPS;
4    config.connectTimeoutMs = 300;
5    config.requestTimeoutMs = 300;
6    config.region = Aws::Region::AP_NORTHEAST_1;
```

```
    Aws::S3::S3Client s3Client
     (Aws::Auth::AWSCredentials
  (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY),
  config);
   Aws::S3::Model::PutObjectRequest putObjectRequest;
   PutObjectRequest.SetBucket("mybucket");
   PutObjectRequest.SetKey("path/to/img.jpg");
```

Listing 6: Main Function to Upload

- **fileName** Il nome del file che vogliamo caricare.

```
auto requestStream = MakeShared<FStream>("PutObjectInputStream",
    fileName.c_str(), ios_base::in | ios_base::binary);
putObjectRequest.SetBody(input_data);
auto putObjectOutcome = s3Client.PutObject(putObjectRequest);
if (!putObjectOutcome.IsSuccess()) {
    std::cerr << "File upload failed from s3 with error " <<
    putObjectOutcome.GetError().GetMessage() << std::endl;
    exit(1);
}
return 0;
}
```

Listing 7: Request Upload

In questo frammento possiamo vedere le configurazioni di impostazione della richiesta e il percorso di archiviazione S3 dove caricare l'immagine. Dopo di che possiamo impostare il corpo della richiesta dove inserire il percorso del file. Ora possiamo effettuare la richiesta e controllare il risultato di questo.

5.1.3 Un pratico esempio utilizzando la libreria OpenCV

```
#include "highgui/highgui.hpp"
#include "imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

Mat rotate(Mat src, double angle)
{
    Mat dst;
    Point2f pt(src.cols/2., src.rows/2.);
    Mat r = getRotationMatrix2D(pt, angle, 1.0);
    warpAffine(src, dst, r, Size(src.cols, src.rows));
    return dst;
}

int main()
```

```
{
    Mat src = imread("1.jpg");
    Mat dst;
    dst = rotate(src, 10);
    imshow("src", src);
    imshow("dst", dst);
    waitKey(0);
    return 0;
}
```

Listing 8: Example Using OpenCV

In questo frammento possiamo vedere una semplice manipolazione dell'immagine. Possiamo ruotare un'immagine usando la funzione **rotate**, dopo l'operazione, nello schermo appariranno due finestre che visualizzano l'immagine prima e dopo la manipolazione.

5.1.4 Nel complessivo

In questa snippet possiamo vedere nel complessivo la nostra applicazione di prova, essere è infatti una fusione dei codici appena visti, in particolare l'workflow della nostra applicazione sarà il seguente:

1. Scaricare l'immagine da S3
2. Manipolare/Trasformare l'immagine
3. Caricare il risultato su S3

Senza entrare nel dettaglio delle diverse righe di codice, mi sono limitato a riportare il codice intero.

```
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>
#include <aws/core/Aws.h>
#include <aws/core/auth/AWSCredentialsProvider.h>
#include <aws/core/utils/StringUtils.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/GetObjectRequest.h>
#include <aws/s3/model/PutObjectRequest.h>
#include <aws/s3/model/Bucket.h>
#include <opencv2/core/types_c.h>


const Aws::String AWS_ACCESS_KEY_ID
  = "******************";
```

```
19 const Aws::String AWS_SECRET_ACCESS_KEY
20    = "************************************";
21
22 using namespace cv;
23 using namespace std;
24 using namespace Aws;
25
26
27 Mat rotate(Mat src, double angle);
28 Mat download(std::string name_bucket_input, std::string path_img,
       std::string filename);
29 void upload(std::string name_bucket_output, std::string path_img,
       std::string filename, cv::Mat);
30
31 int main (int argc, char *argv[])
32 {
33     Aws::SDKOptions options;
34     Aws::InitAPI(options);
35     {
36         std::string name_bucket_input = "bernovschi-tesi";
37         std::string name_bucket_output = "bernovschi-tesi-output";
38         std::string path_img = "/";
39         std::string filename_input = argv[1];
40         std::cout<<"FILE INPUT : " <<filename_input <<endl;
41         std::string filename_output = argv[2];
42         cv::Mat src = download(name_bucket_input, path_img,
       filename_input);
43         //cv::Mat src = cv::imread(filename_input, IMREAD_COLOR);
44         Mat dst;
45         dst = rotate(src, 20);
46         //imshow("src", src);
47         //imshow("dst", dst);
48         waitKey(0); //wait for key press
49         imwrite(filename_output, dst);
50         upload(name_bucket_output, path_img, filename_output, dst);
51     }
52     Aws::ShutdownAPI(options);
53     return 0;
54 }
55
56 cv::Mat download(std::string name_bucket_input, std::string
       path_img, std::string filename) {
57     Aws::Client::ClientConfiguration config;
58     config.scheme = Aws::Http::Scheme::HTTPS;
59     config.connectTimeoutMs = 300;
60     config.requestTimeoutMs = 300;
61     config.region = Aws::Region::EU_WEST_1;
62     Aws::S3::S3Client s3Client(Aws::Auth::AWSCredentials(
       AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY), config);
63     Aws::S3::Model::GetObjectRequest getObjectRequest;
```

```
64      getObjectRequest.SetBucket(Aws::String(name_bucket_input.c_str
        ()));
65      path_img = path_img + filename;
66      getObjectRequest.SetKey(Aws::String(path_img.c_str()));
67      auto getObjectOutcome = s3Client.GetObject(getObjectRequest);
68      if (!getObjectOutcome.IsSuccess()) {
69          std::cerr << "File download failed from s3 with error " <<
        getObjectOutcome.GetError().GetMessage() << std::endl;
70          exit(1);
71      }
72      std::stringstream ss;
73      ss << getObjectOutcome.GetResult().GetBody().rdbuf();
74      std::string str = ss.str();
75      std::vector<char> vec(str.begin(), str.end());
76      cv::Mat img = cv::imdecode(cv::Mat(vec), IMREAD_COLOR);
77      return img;
78  }
79
80  void upload(std::string name_bucket_output, std::string path_img,
        std::string filename, cv::Mat) {
81      Aws::Client::ClientConfiguration config;
82      config.scheme = Aws::Http::Scheme::HTTPS;
83      config.connectTimeoutMs = 300;
84      config.requestTimeoutMs = 300;
85      config.region = Aws::Region::EU_WEST_1;
86      Aws::S3::S3Client s3Client(Aws::Auth::AWSCredentials(
        AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY), config);
87      Aws::S3::Model::PutObjectRequest putObjectRequest;
88      putObjectRequest.SetBucket(Aws::String(name_bucket_output.c_str
        ()));
89      putObjectRequest.SetKey(Aws::String(filename.c_str()));
90      Aws::String filename_aws = Aws::String (filename.c_str());
91      auto requestStream = Aws::MakeShared<Aws::FStream>("
        PutObjectInputStream", filename_aws.c_str(), std::ios_base::in
        | std::ios_base::binary);
92      putObjectRequest.SetBody(requestStream);
93      auto putObjectOutcome = s3Client.PutObject(putObjectRequest);
94      if (!putObjectOutcome.IsSuccess()) {
95          std::cerr << "File upload failed from s3 with error " <<
        putObjectOutcome.GetError().GetMessage() << std::endl;
96          exit(1);
97      }
98  }
99
100 cv::Mat rotate(cv::Mat src, double angle)
101 {
102     Mat dst;
103     cout << "Phase 1\n";
104     Point2f pt(src.cols/2., src.rows/2.);
105     cout << "Phase 2\n";
```

```
106     Mat r = getRotationMatrix2D(pt, angle, 1.0);
107     cout << "Phase 3\n";
108     warpAffine(src, dst, r, Size(src.cols, src.rows));
109     return dst;
110 }
```

Listing 9: "Testing Application for download the image manipulate it and upload the result"

5.2 Creazione dell'immagine

Creare la cartella, dove memorizzeremo l'immagine e i file di cui abbiamo bisogno

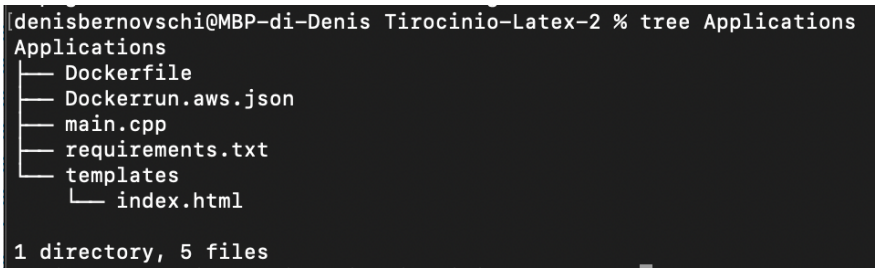


Figura 24: La cartella

5.2.1 Scrittura del Dockerfile

Qui ho deciso di riportare lo snippet del docker file. L'immagine è basata su Ubuntu 20.04.

Figura 25: Ubuntu 20.04

Successivamente si installano le dipendenze necessarie ad OpenCV e ad AWS SDK, configurando il tutto in modo opportuno. Di seguito si copiano tutti i file necessari alla nostra applicazione, ed attraverso un escamotage tramite la keyword **ENTRYPOINT** una volta compilato il codice, durante l'esecuzione, passiamo il nome dei file necessari all'elaborazione.

```
1 #Sistema operativo
2 FROM ubuntu:20.04
```

```
3
4  #Impostazione di Sistema neccessaria ad alcuni pacchetti
5  ENV TZ=Europe/Rome
6  RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /
       etc/timezone
7
8  #Installazione pacchetti necessari a AWS e OPENCV
9  RUN apt-get update
10 RUN apt-get -y upgrade
11 RUN apt-get install -y -f  build-essential g++11 clang
12 RUN apt-get install -y -f  --force-yes qemu-user-static
13 RUN apt-get install -y -f     nano
14 RUN apt-get install -y -f   wget unzip git curl python
15 RUN apt-get install -y -f   python3-numpy python3-dev
16 RUN apt-get install -y -f  --force-yes libcurl4-openssl-dev
17 RUN apt-get install -y -f  --force-yes libssl-dev uuid-dev
18 RUN apt-get install -y -f  --force-yes zlib1g-dev libpulse-dev
19 RUN apt-get install -y -f  --force-yes openssl software-properties-
       common
20 RUN apt-get install -y -f  --force-yes libboost-all-dev zlib1g
       zlib1g-dev
21 RUN add-apt-repository "deb http://security.ubuntu.com/ubuntu
       xenial-security main"
22 RUN apt-get update
23 RUN apt-get install -y -f libjasper1 libjasper-dev
24 RUN apt-get install -y -f cmake git libgtk2.0-dev pkg-config
       libavcodec-dev
25 RUN apt-get install -y -f libavformat-dev libswscale-dev python-dev
        python-numpy libtbb2 libtbb-dev
26 RUN apt-get install -y -f libjpeg-dev libpng-dev libtiff-dev
       libdc1394-22-dev
27 RUN apt-get install -y -f libcurl4-openssl-dev  libtbb-dev libv4l-
       dev
28 RUN apt-get install -y -f libtiff5-dev libeigen3-dev libtheora-dev
       libvorbis-dev
29 RUN apt-get install -y -f libxvidcore-dev libx264-dev sphinx-common
30 RUN apt-get install -y -f libtbb-dev yasm libfaac-dev libopencore-
       amrnb-dev libopencore-amrwb-dev
31 RUN apt-get install -y -f libopenexr-dev libgstreamer-plugins-base1
       .0-dev
32 RUN apt-get install -y -f libavutil-dev libavfilter-dev
       libavresample-dev
33 RUN apt-get install -y -f libgtk-3-dev "libcanberra-gtk*"
34 RUN apt-get install -y -f gfortran openexr libatlas-base-dev
       libgstreamer1.0-dev
35 RUN apt-get install -y -f cmake
36
37 #Installazione OPENCV
38 RUN apt-get update
39 RUN apt-get -y upgrade
```

```
40 RUN apt-get install -y --force-yes libopencv-dev python3-opencv
41
42 WORKDIR /opt/
43 RUN mkdir opencv_build && cd opencv_build
44 WORKDIR /opt/opencv_build
45 RUN git clone https://github.com/opencv/opencv.git
46 RUN git clone https://github.com/opencv/opencv_contrib.git
47 RUN cd /opt/opencv_build/opencv
48 RUN mkdir -p build && cd build
49 WORKDIR cd /opt/opencv_build/opencv/build
50
51 RUN cmake -D CMAKE_BUILD_TYPE=RELEASE \
52 -D CMAKE_INSTALL_PREFIX=/usr/local \
53 -D INSTALL_C_EXAMPLES=ON \
54 -D INSTALL_PYTHON_EXAMPLES=ON \
55 -D OPENCV_GENERATE_PKGCONFIG=ON \
56 -D OPENCV_EXTRA_MODULES_PATH=/opt/opencv_build/opencv_contrib/
       modules \
57 -D BUILD_EXAMPLES=ON /opt/opencv_build/opencv
58
59 RUN make -j4
60 RUN make install
61 RUN ldconfig
62
63 #Installazione AWS (solo pacchetti core & s3)
64 WORKDIR /usr/local/
65 RUN git clone https://github.com/aws/aws-sdk-cpp
66 RUN mkdir build_dir
67 RUN cd build_dir
68 WORKDIR /usr/local/aws-sdk-cpp/
69 RUN cmake . -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/
       local
70 RUN make -j4 -C aws-cpp-sdk-core
71 RUN make -j4 -C aws-cpp-sdk-s3
72 RUN make install -C aws-cpp-sdk-core
73 RUN make install -C aws-cpp-sdk-s3
74 RUN ldconfig
75
76 #Trasferimento codice sorgente ed altre informazioni
77 RUN mkdir /source
78 COPY . /source
79 VOLUME ["/source"]
80 WORKDIR /source
81
82 ENV INPUT_FILE  "a.jpg"
83 ENV OUTPUT_FILE  "a_o.jpg"
84
85 RUN g++ -c $(pkg-config --cflags --libs opencv4) -laws-cpp-sdk-s3 -
       laws-cpp-sdk-core -std=c++11 main.cpp
86 RUN g++ -o main main.o $(pkg-config --cflags --libs opencv4) -laws-
```

```
    cpp-sdk-s3 -laws-cpp-sdk-core -std=c++11
87
88 ENTRYPOINT ./main ${INPUT_FILE} ${OUTPUT_FILE}
89
90 #CMD ["./main" $input_name $output_name]
```

Listing 10: Docker File

Building your image Per quanto riguarda la creazione e l'esecuzione dell'immagine è oppurtuno eseguire le due istruzioni sottostanti nell'ordine riportato

```
1 sudo docker build . -t denis/aws
2 sudo docker run -it --rm denis/aws
```

Listing 11: Terminal commands per il build e l'esecuzione del container

Building Application Per quanto concerne invece l'applicazione, le istruzioni da eseguire sono le seguenti, ed anche qua l'ordine è fondamentale.

```
1 g++ -c (pkg-config --cflags --libs opencv4) -laws-cpp-sdk-s3 -laws-
    cpp-sdk-core -std=c++11 main.cpp
2
3 g++ -o main main.o (pkg-config --cflags --libs opencv4) -laws-cpp-
    sdk-s3 -laws-cpp-sdk-core -std=c++11
4
5 ./ main
```

Listing 12: Commands per la compilazione dell'applicazione di Test

Risultato Il risultato della compilazione e dell'esecuzione dell'applicazione è il seguente. In particolare possiamo vedere che comparire il file "img-o.jpg" frutto appunto dell'esecuzione della nostra applicazione base.

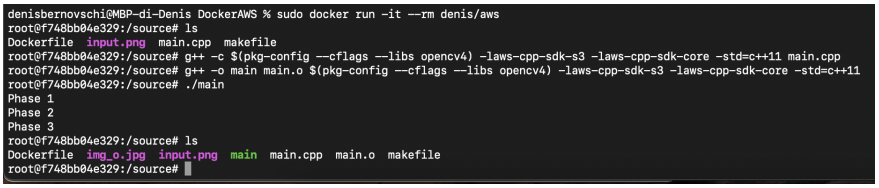


Figura 26: Result of esecution

5.2.2 Caricamento immagine su docker Hub

Per quanto riguardo il caricamento dell'immagine sul docker hub, rifarsi alla sezione 3.7

5.3 Integrazione Fargate & AWS ECS

Per eseguire i container, si hanno due opzioni. È possibile utilizzare ECS [11] tramite delle istanze di container, o è possibile utilizzare Fargate. Entrambe le opzioni di collaborano di base con ECS.

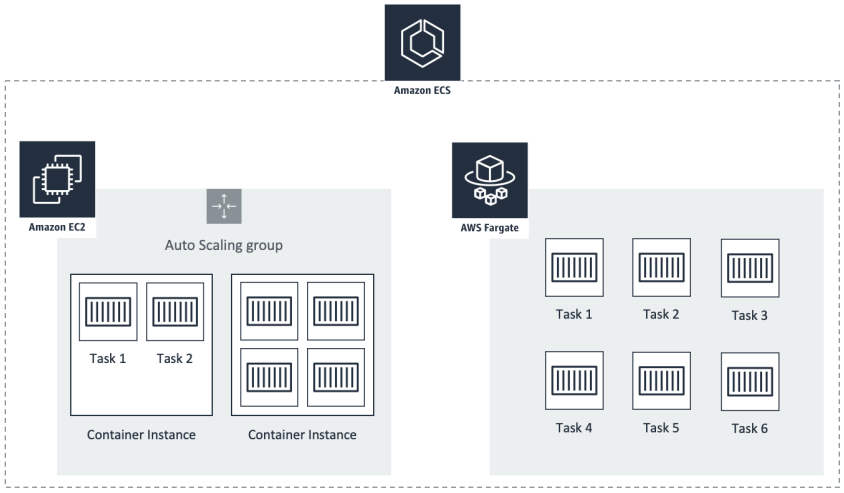


Figura 27: Integrazione Fargate & Elastic Container Service

Utilizzando l'integrazione di Amazon ECS e Amazon EFS ... Nel nostro studio abbiamo utilizzato l'integrazione per poter permettere a Lambda Function di far partire il nostro task, in seguito ad un upload di un file su un S3 bucket. Così facendo possiamo mettere in pratica il nostro workflow.

Workflow

1. Upload dell'immagine su S3
2. S3 triggera la funzione Lambda non appena l'immagine è stata caricata
3. La funzione Lambda avvia ECS Fargate Task tramite gli opportuni parametri
4. ECS Fargate Task esegue il container
   - Si processa l'immagine
   - Si fa l'upload del risultato su S3
5. S3 triggera un ulteriore funzione Lambda non appena il risultato viene caricato sul secondo bucket

[11] ECS: Elastic Container Services

6. La funzione Lambda scrive l'url dell'immagine sul log visibile tramite cloud watch

Il tutto è controllabile tramite AWS Cloud Watch

5.3.1 Le Keywords dell'integrazione

Elastic Container Registry (ECR) Il servizio di registro delle immagini Docker completamente gestito da utilizzare per la memorizzazione della nostra immagine del container.

Elastic Container Service (ECS) Il servizio di gestione container altamente scalabile e performante che utilizzeremo per eseguire l'applicazione container Docker.

Fargate Questo è il motore di elaborazione senza server che ci eliminerà dalla gestione dei server in ECS

API Gateway Il servizio gestito che useremo come porta di accesso al nostro ambiente di backend implementando un'API Restful.

> "AWS Fargate is a technology that allows you to use a containers as a fundamental compute primitive without having to manage the underlying compute instances"

5.3.2 Setting up ECS using Fargate

1. Container Definition
   (a) Seleziona l'opzione Custom e crea un nuovo container definition
   (b) Premi Configura
   (c) Configura il container (nome, image, ...)
   (d) Configura gli aspetti avanzati
   (e) Nella sezione Environment ... settiamo le ENV Variables
   (f) Network Settings ... vuoto
   (g) In the Storage and Logging ... log configuration
   (h) Lascia le sezioni Resource Limits e 'Docker Labels' vuote

   La mia configurazione

   (a) name : bernovschi
   (b) image : denisbernovschi/aws:latest
   (c) port : 8081

(d) Environments Variables : INPUT_FILE, OUTPUT_FILE

(e) Log configuration:

- aws-logs-group - /ecs/bernovschi-task-defnition;
- aws-logs-region - /eu-west-1 ;
- aws-logs-stream-prefix - ecs ;

2. Task Definition

(a) Aggiorna 'Task Definition Name' & Salva

La mia configurazione
bernovschi-task-definition

(b) Controlla la compatibilità con Fargate & Salva

3. Service Definition

(a) Durante la fase di creazione del task-definition, AWS crea automaticamente un 'security group'

(b) Accetta i valori di default & premi Next

4. Configuration the cluster

(a) Accetta i valori di default del 'Cluster name' come default, e nota che AWS automaticamente creerà una VPC e sub-nets come necessario per l'interazione. Ora puoi proseguire, premendo 'Next' .

5. Controlla & Crea

5.3.3 Prerequisiti & Risorse

Per essere in grado di eseguire il nostro compito e supportare l'applicazione, abbiamo bisogno di creare un paio di risorse:

1. Abbiamo bisogno di un bucker S3 in cui verrà caricato l'immagine per la trasformazione
2. Abbiamo bisogno di impostare una policy e IAM Role per i bucket, al fine di permettere le operazioni sugli oggetti contenuti al suo interno.
3. Abbiamo anche bisogno di creare un ulteriore bucket ove memorizzare il risultato

IAM Roles & Policies Per il nostro compito di accedere al bucket S3 che abbiamo specificato dal nostro account AWS. Abbiamo quinidi bisogno di dargli autorizzazioni specifiche. Dal momento che il container sta eseguendo all'interno del contesto ECS, possiamo aggiungere un altro ruolo che avrà le specifiche politiche di accesso S3. Quando abbiamo creato la nostra task-definition, AWS ha creato un ruolo **ecsTaskExecutionRole** per noi, che ha accesso per eseguire il task. Invece di aggiornare direttamente quel ruolo, assegneremo un nuovo.

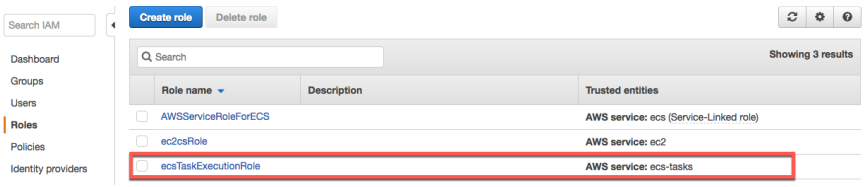


Figura 28: ECS Task Execution Role

Clicca sul ruolo **ecsTaskExecutionRole**. Nella pagina risultante, clicca sul link **Add inline policy**:

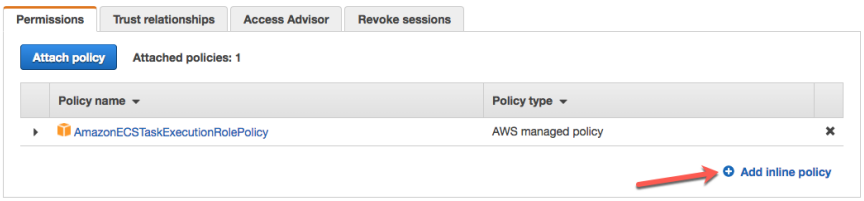


Figura 29: Add inline policy

Nella schermata 'Crea Policy', fai clic sulla scheda JSON. Ho già creato il frammento JSON che incapsula la politica di cui abbiamo bisogno. Basterà incollare il seguente JSON Script nell'area di testo:

```
{
    "Statement": [
        {
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::<your-bucket>"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
```

```
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::<your-bucket-2>/*"
        }
    ],
    "Version": "2012-10-17"
}
```

Listing 13: Policy

Fondamentalmente, la politica di cui sopra permette al nostro task di elencare il nostro bucket e permette inoltre anche di inserire un oggetto nel percorso "bucket/folder". Fai clic sul pulsante 'Review policy" e dai al nuovo ruolo un nome **Uploadtos3rolepolicy**. Fai clic sul pulsante 'Crea policy' per creare la policy e allegala al ruolo **ecsTaskExecutionRole**. Puoi vedere la schermata risultante qui sotto:

Figura 30: Add inline policy

Con tutte le impostazioni oggetto dei prerequisiti completate, ora possiamo iniziare a eseguire il task che abbiamo creato.

5.3.4 Avvio del task-execution

Ora che abbiamo un task impostato, eseguiamolo. Inizieremo eseguendo il compito dalla console AWS. Spunta la casella accanto all'attività, fai clic sul dropdown menu **Actions** e seleziona la voce **Run Task**:

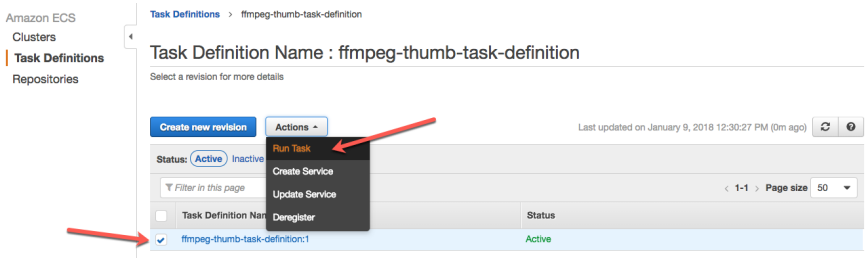


Figura 31: Define task

Nella schermata seguente, è necessario specificare alcune impostazioni che sono richieste dall'attività da eseguire:

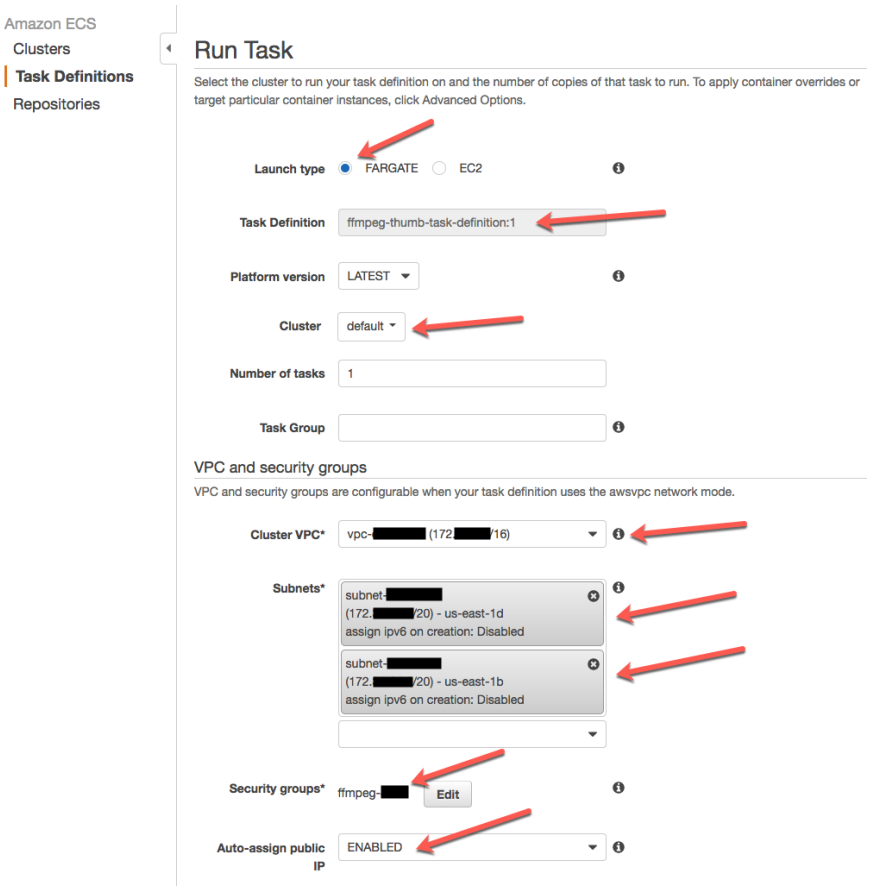


Figura 32: Define task pt.2

Ecco un paio di cose che dobbiamo settare:

- Seleziona 'FARGATE' come 'Launch type'
- Seleziona 'ECS' as 'Cluster'
- Assegna '1' for 'Number of tasks'
- Per 'Cluster VPC', seleziona uno con non-internal IP i.e. 172.x.x.x
- Solo allora si vedrà il drop-down menu riguardo le 'Subnets' . Scegline due...
- Scegli il gruppo di sicurezza predefinito creato da AWS
- Seleziona 'ENABLED' per 'Auto-assign public IP'

Lascia la sezione **Task Overrides** sotto **Advanced Options** così com'è. Nota che un Task Execution Role, **currentecsTaskExecutionRole** è stato creato

automaticamente e assegnato al task. Questo ruolo IAM dà il permesso di eseguire/avviare il task. La sezione 'Override container' ci dà l'opportunità di sovrascrivere qualsiasi impostazione per il container che abbiamo creato. Dovrebbe essere popolato con le impostazioni che abbiamo aggiunto durante la creazione della definizione del task.

5.3.5 Avvio del Task

Qui possiamo vedere la definizione del task, e la sua configurazione

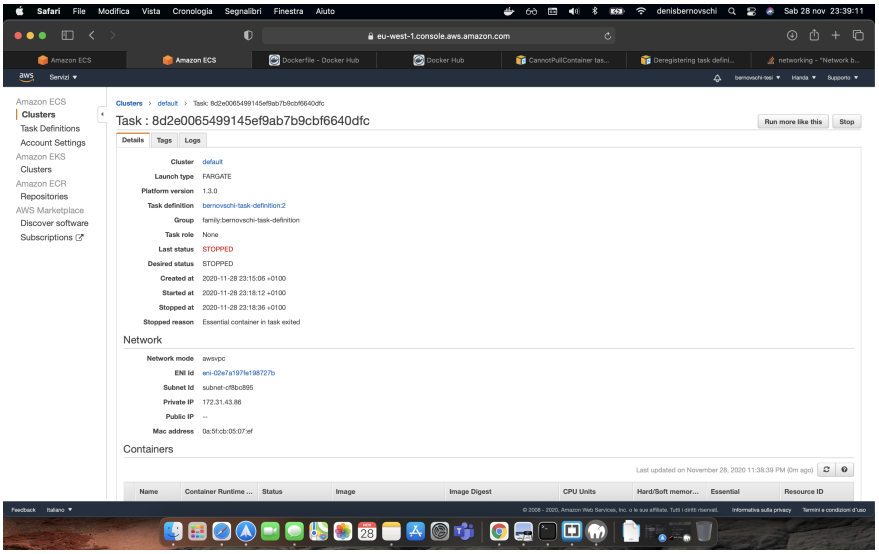


Figura 33: Definizione e Configurazione

Qui possiamo vedere, lo stato del task, in particolare nella sezione **Details** è visibile l'exit code e le Environment Variables, che ci permettono di verificare nuovamente i parametri della nostra applicazione.

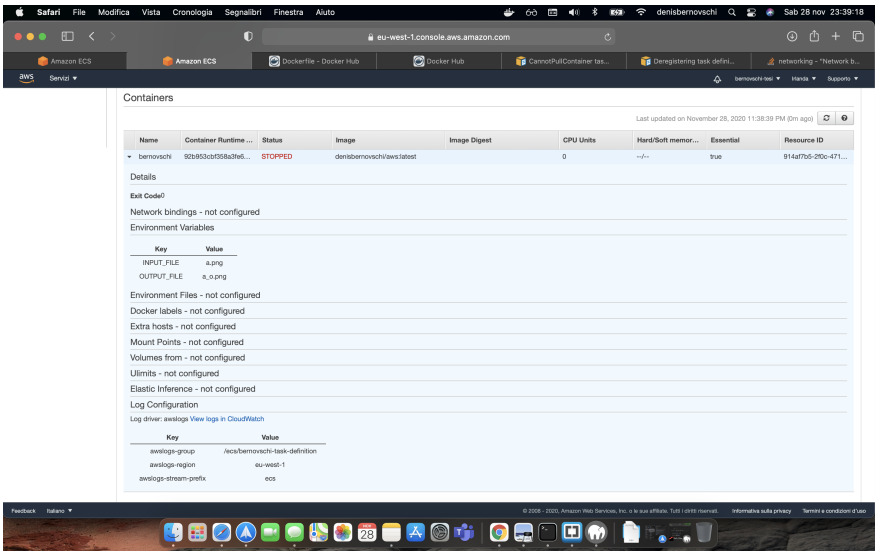


Figura 34: Definizione e Configurazione pt.2

Mentre qui, è messo in evidenza tramite la schermata di Cloud Watch, il corretto funzionamento della nostra applicazione. Tramite questa schermata siamo sicuri che il nostro container e la nostra applicazione hanno funzionato correttamente.

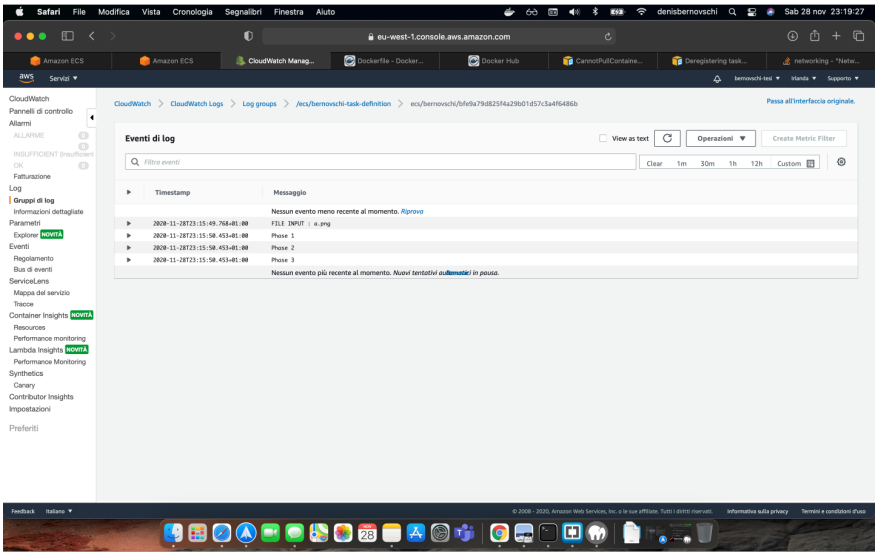


Figura 35: Definizione e Configurazione pt.3

5.3.6 IAM Roles

Per consentire alle funzioni Lambda della nostra applicazione serverless di eseguire determinate azioni, abbiamo bisogno di impostare determinati permessi con AWS. In particolare abbiamo bisogno dei seguenti permessi:

```
iamRoleStatements:
  - Effect: "Allow"
    Action:
      - ecs:RunTask
    Resource: "*"
  - Effect: Allow
    Action:
      - iam:PassRole
    Resource: ${self:custom.execRoleArn}
  - Effect: "Allow"
    Action:
      - s3:GetObject
    Resource: "arn:aws:s3:::${self:custom.bucket}/*"
```

Figura 36: IAM Role per funzioni Lambda

Per permettere alle funzioni Lambda di eseguire e/o avviare le ECS Tasks, assumendo il ruolo definito nell'impostazione **execRoleArn** e permette di ottenere gli oggetti S3 dal bucket che abbiamo definito. Usiamolo, carichiamo un file .png nel S3 bucket configurato.

Ho usato l'AWS-SDK e COGNITO POOL IDENTITY, ho dovuto anche configurare un nuovo ruolo per le persone non autorizzate.

5.4 Problem

Il problema principale era automatizzare il ECS container. In particolare farlo partire quando un nuovo oggetto è caricato e memorizzato nel bucket. Quindi, per questo motivo, uso una funzione Lambda. In particolare, uso due diverse funzioni, perché ogni funzione opera su bucket diversi. Così la roadmap è stata ...

1. Creare i due bucket : "bernovschi-tesi", "bernovschi-tesi-output"

2. Definire le policy e IAM Role, per buckets
3. Definire le policy e IAM Role, per le funzioni Lambda
4. Creare le due funzioni
    - s3UploadFunction
    - triggerOnCreation
5. Creare la pagina web per utilizzare la nostra applicazione di prova

5.4.1 Policy for Buckets & CORS

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:*Object",
            "Resource": "arn:aws:s3:::bernovschi-tesi/*"
        },
        {
            "Sid": "AWSCloudTrailAclCheck20150319",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudtrail.amazonaws.com"
            },
            "Action": "s3:GetBucketAcl",
            "Resource": "arn:aws:s3:::bernovschi-tesi"
        },
        {
            "Sid": "AWSCloudTrailWrite20150319",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudtrail.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bernovschi-tesi/.png/AWSLogs
    /029271510931/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": "bucket-owner-full-control"
                }
            }
        }
    ]
}
```

Listing 14: Policy for Buckets & CORS

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "GET",
            "PUT",
            "POST",
            "HEAD",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "ETag"
        ]
    }
]
```

Listing 15: CORS S3

"It's the same configuration for both buckets"

5.4.2 Definizione delle policy e IAM Role, per le Lambda Function

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*"
        }
    ]
```

```
}
```

Listing 16: Policy for Lambda Function

5.4.3 AWS Lambda - Lambda Function Destination Execution Role

Per la funzione triggerOnCreation

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:eu-west-1:************:
    function:triggerOnCreation*"
        }
    ]
}
```

Listing 17: Lambda Fuction Policy triggerOnCreation

per la funzione s3UploadFunction

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:eu-west-1:************:
    function:s3UploadFunction*"
        }
    ]
}
```

Listing 18: Lambda Fuction Policy s3UploadFunction

5.4.4 ecsTaskExecutionRole

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1512361420000",
            "Effect": "Allow",
            "Action": [
                "ecs:RunTask"
            ],
            "Resource": [
```

```
                "*"
            ]
        },
        {
            "Sid": "Stmt1512361593000",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::************:role/
    ecsTaskExecutionRole"
            ]
        }
    ]
}
```

Listing 19: Policy ecsTaskExecutionRole

5.4.5 Le Upload Policy e i IAM Role

UploadTo2Bucket La policy che ci permette di caricare il risultato sul bucket secondario

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bernovschi-tesi-output/*"
        }
    ]
}
```

Listing 20: Policy UploadTo2Bucket

UploadToS3RolePolicy il ruolo che ci permette di operare con il bucket principale, in particolare ci permette di visionare il contenuto del bucket ...e succesivamente il ruolo che ci permette di operare con il bucket secondario.

```
{
    "Statement": [
        {
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
```

```
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::bernovschi-tesi"
        }
    ],
    "Version": "2012-10-17"
}
```

Listing 21: Policy UploadToS3RolePolicy

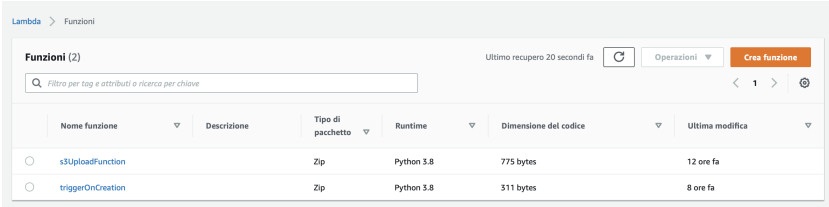
UploadToS3RolePolicyV2

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowListingOfUserFolder",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::bernovschi-tesi-output"
            ]
        },
        {
            "Sid": "HomeDirObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::bernovschi-tesi-output/*"
        }
    ]
}
```

Listing 22: Policy UploadToS3RolePolicyV2

5.5 Lambda Function

Prima di tutto vanno definite le due funzioni necessarie alla nostra applicazione, la prima s3UploadFunction che scatenerà l'ECS task facendo così partire l'esecuzione del container; e triggerOnCreation, che ci permette appunto di verificare se il risultato della manipolazione dell'immagine è stato correttamente caricato sul secondo bucket.



5.5.1 S3 Upload Function

La prima funzione **s3UploadFunction**, mi permette di avviare il contenitore e manipolare l'immagine caricata dal sito

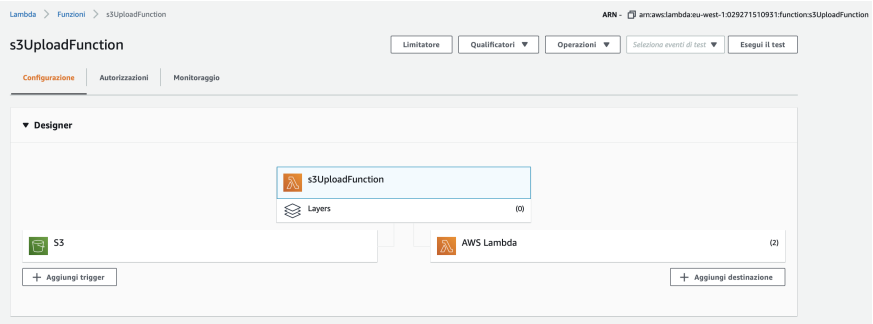


Figura 37: s3UploadFunction

Qui di seguito è riportato la funzione utilizzata ...

```
import json
import os
import boto3

input_namefile = "a.png",
output_namefile = "a_o.png"

def lambda_handler(event, context):
    client = boto3.client('ecs')
    print('## EVENT')
    input_file = (str(event['Records'][0]['s3']['object']['key']))
    input_namefile = (os.path.splitext(input_file)[0])
    output_namefile = input_namefile + "_o.png"
    print('event - input_file ' +input_file + ' output_namefile : '
    + output_namefile)
    response = client.run_task(
        cluster='ECS', # name of the cluster
        launchType = 'FARGATE',
        taskDefinition='bernovschi-task-definition:6', # replace
    with your task definition
        count = 1,
        platformVersion='LATEST',
```

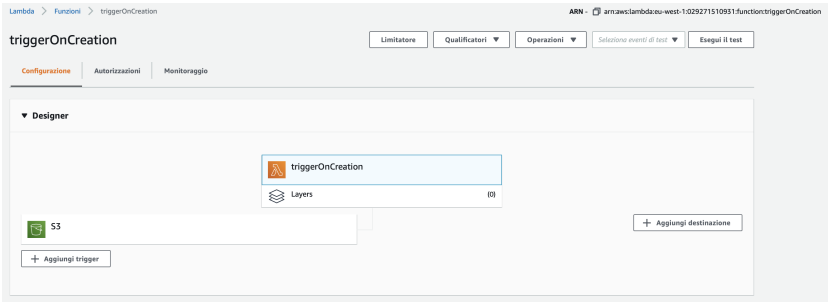
```
21          networkConfiguration={
22              'awsvpcConfiguration': {
23                  'subnets': [
24                      'subnet-76bfb910', # replace with your public
    subnet or a private with NAT
25                      'subnet-cf8bc895', # Second is optional, but
    good idea to have two
26                      'subnet-54ccdd1c' # Third is optional, but good
     idea to have tree
27                  ],
28                  'assignPublicIp': 'ENABLED'
29              }
30          },overrides={
31              'containerOverrides': [
32              {
33                  'name': 'bernovschi-tesi',
34                  'environment': [
35                      {
36                          'name': 'INPUT_FILE',
37                          'value': input_file
38                      },
39                      {
40                          'name': 'OUTPUT_FILE',
41                          'value': output_namefile
42                      }
43                  ]
44              }
45          ]
46          })
47      print('## RESPONSE')
48      print (str(response))
```

Listing 23: Function Lambda to interact through Fargate with ECS

Spiegazione della funzione ... ...

5.5.2 triggerOnCreation

La funzione **triggerOnCreation** mi permette di mettere il risultato della manipolazione dell'immagine, in S3 bucket **bernovschi-tesi-output**



La funzione lambda, stampa il risultato dell'evento in un log, visibile in CloudWatch.

```
import json
def lambda_handler(event, context):
    print('## EVENT')
    print('event 2 ' + str(event))
```

Listing 24: Function Lambda to check the result uploading

5.6 Web Site

5.6.1 Page HTML

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>AWS-S3</title>
    <script src="./aws-sdk-2.799.0.js"></script>
    <script src="./s3_function.js"></script>
  <style>
    body{background-color: #659DBD;}

        h1{margin-left: 30%;
        margin-right: 30%;
        width: 40%;
        color: whitesmoke;
        font-family: courier;
        font-size: 300%;}

        p{font-family: courier;
        font-size: 160%;
        color: whitesmoke;
        text-transform: uppercase;}

        button{
        border-radius: 20px;
        font-weight: bold;
```

```
26          color:black;
27          height:30px; width:100px;
28          cursor:pointer;}
29
30          hr{width: "50%";}
31    </style>
32  </head>
33
34  <body>
35    <h1>AWS - GROTTINI LAB</h1>
36      <hr style="width:50%">
37    <div id="input_image">
38          <p> Scegli l'immagine : <input type="file" id="file" id="
      avatar" name="image" accept="image/png, image/jpeg" onchange="
      loadFile(event)"> </p>
39          <img id="output" width="500px" height="300px" />
40          <p><button id="send_button" for="submit" style="cursor:
      pointer;" onclick="upload()">SEND</button></p>
41      </div>
42      <hr style="width:50%">
43      <div id="check_file" style="font-size: 14px; background-color:
      red; ">
44          <p  style="width:50%" id="check_file_text">FILE NON ANCORA
      PROCESSATO</p>
45      </div>
46      <hr style="width:50%">
47      <p>Risultato :</p>
48      <div id="output_image">
49      </div>
50      <div id="results"></div>
51      <p><button id="get_button" for="submit" style="cursor: pointer;
      " onclick="aggiungiRisultato()">GET RESULT</button></p>
52      <p><button id="reset_button" style="cursor: pointer;"  onclick
      = "location.reload()" >RESET</button></p>
53  </body>
54  </html>
```

Listing 25: The web page

5.6.2 Script

The script with function, for upload, check result and get it.

```
1  var albumBucketName = "bernovschi-tesi";
2  var bucketRegion = "eu-west-1";
3  var IdentityPoolId = "eu-west-1:29c3bee3-6803-46d3-9974-27
       eb55ecdb42";
4
5  AWS.config.update({
6    region: bucketRegion,
```

```
  credentials: new AWS.CognitoIdentityCredentials({
    IdentityPoolId: IdentityPoolId
  })
});

var s3 = new AWS.S3({
  apiVersion: "2006-03-01",
  params: { Bucket: albumBucketName }
});

function upload() {
  var files = document.getElementById("file").files;

  if (!files.length) {
    return alert("Please choose a file to upload first.");
  }

  var file = files[0];
  var fileName = file.name;
  var albumPhotosKey = "";

  var photoKey = albumPhotosKey + fileName;

  // Use S3 ManagedUpload class as it supports multipart uploads
  var upload = new AWS.S3.ManagedUpload({
    params: {
      Bucket: albumBucketName,
      Key: photoKey,
      Body: file,
      ACL: "public-read"
    }
  });

  var promise = upload.promise();

  promise.then(
    function(data) {
        alert("Successfully uploaded photo.");
        deleteButton("send_button");
        fileExist();
    },
    function(err) {
      return alert("There was an error uploading your photo : ",
    err.message);
    }
  );
}

function aggiungiRisultato(){
    var files = document.getElementById("file").files;
```

```
56      var file = files[0];
57      var fileName = file.name;
58      var filename_input = file.name ;
59      var filename_output = filename_input.substring(0,
        filename_input.lastIndexOf("."))+"_o.png";
60      var s3url = "https://bernovschi-tesi-output.s3.amazonaws.com/"
        + encodeURIComponent(filename_output);
61      var img = document.createElement("img");
62      img.setAttribute("width", "500px");
63      img.setAttribute("height", "300px");
64      img.src = s3url;
65      var div = document.getElementById("output_image");
66      div.appendChild(img);
67  }
68
69  function deleteButton(id){
70    var myobj = document.getElementById(id);
71    myobj.remove();
72  }
73
74  setInterval(function fileExist(){
75      var files = document.getElementById("file").files;
76      var file = files[0];
77      var fileName = file.name;
78      var filename_input = file.name ;
79      var filename_output = filename_input.substring(0,
        filename_input.lastIndexOf("."))+"_o.png";
80      var s3url = "https://bernovschi-tesi-output.s3.amazonaws.com/"
        + encodeURIComponent(filename_output);
81
82      var http = new XMLHttpRequest();
83
84      http.open('HEAD', s3url, false);
85      http.send();
86
87      if((http.status != 404) && http.status != 403){
88          document.getElementById("check_file").style.backgroundColor
        ="GREEN";
89          document.getElementById("check_file_text").textContent="
        File processato correttamente";
90      }else{
91          document.getElementById("check_file").style.backgroundColor
        ="RED";
92          document.getElementById("check_file_text").textContent="
        File non ancora processato";
93          document.getElementById("get_button").disable();
94      }
95  }, 3000);
96
97  var loadFile = function(event) {
```

```
98      var image = document.getElementById('output');
99      image.src = URL.createObjectURL(event.target.files[0]);
100 };
```

Listing 26: Javascript functions

6 Risultati

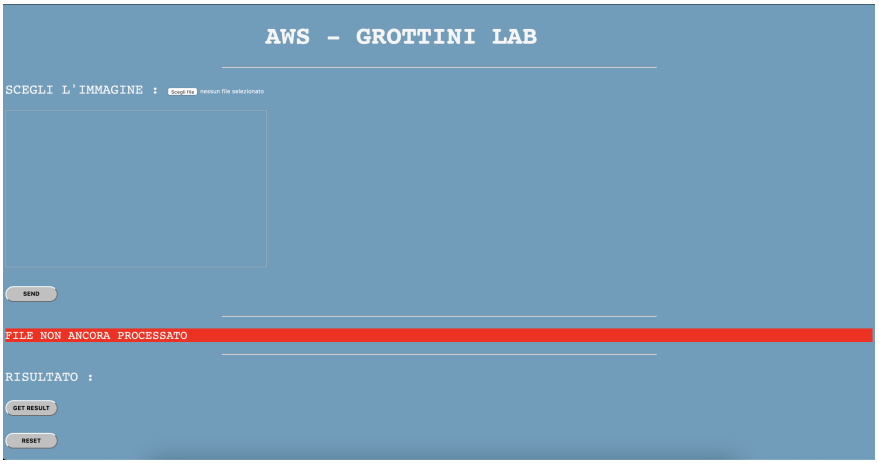


Figura 38: Result

Prima, è necessario scegliere l'immagine da caricare al fine di ottenere la manipolazione. Una volta scelta basterà premere il pulsante **SEND** e attendere il caricamento l'immagine. Quando l'immagine viene caricata con successo, avremo un alert che ci informerà dell'upload eseguito. Ora ci basterà attendere che il file sia elaborato e che il risultato sia stato caricato correttamente nel bucket di output. Per questo sarà sufficiente attendere che la barrà rossa diventi verde. Infine sarà possibile ottenere il risultato, premendo il tasto **GET RESULT**. Per poter processare un'altra immagine, sarà necessario premere il pulsante **RESET** e ricominciare nuovamente la procedura.

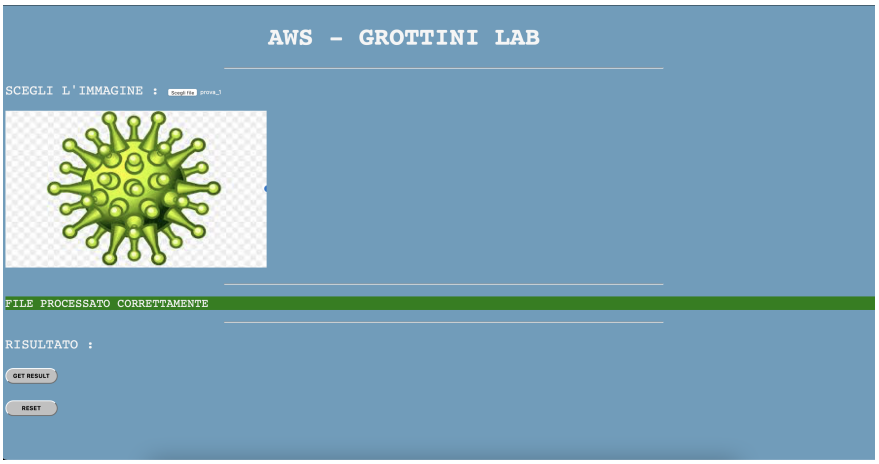


Figura 39: Result pt.2

Figura 40: Result pt.3

7 Conclusioni e sviluppi futuri

Tramite questo elaborato, abbiamo analizzato un fondamentale passaggio di vision per quanto riguarda lo sviluppo di applicazioni cloud computing. Attraverso l'analisi dei costi abbiamo dimostrato che al momento attuale vige l'infondatezza della tesi che sostiene che sia necessario un notevole esborso economico per uno sviluppo cloud computing. Abbiamo poi analizzato i vari servizi di AWS ponendo di volta in volta attenzione non solo sui vantaggi di ciascuno di essi, ma anche sulla sicurezza, aspetto cardine, oggi non più sottovalutabile. Va riservata particolare attenzione sul pilastro della nostra applicazione "IoT Edge-Fog-Cloud Architecture for Vision Based Planogram Integrity" la quale ci permette di analizzare attraverso tecniche di Computer Vision l'integrità del nostro planogramma in tempo reale. Ci permette quindi di analizzare non solo il comportamento dell'utente in ambiente retail, ma anche di osservare quanto una determinata posizione di un determinato prodotto possa influire sulle vendite di quest'ultimo.

7.1 Possibili applicazioni future

Uno tra gli sviluppi futuri più prossimi è il passaggio da telecamere fisse ad un approccio mobile tramite lo sviluppo di un'applicazione multi piattaforma che permetterà l'acquisizione dell'immagine e la successiva elaborazione tramite l'interazione con il mondo AWS. Un ulteriore sviluppo futuro, già oggetto di studio ed elaborazione, è di integrare l'object detection API di Tensorflow per il transfer learning di una rete di detection preallenata, su un dataset custom.
L'obbiettivo sarebbe quindi quello di fare la detection di tutti i prodotti su uno scaffale, più precisamente effetuare una buonding box intorno a ogni prodotto presente sullo scaffale, indipendentemente dalla sua tipologia. Infine queste detection serviranno a regime per confrontare ogni articolo con i modelli nel nostro database al fine di riconoscere l'articolo e analizzare quindi la conformità del planogram.
Il tutto avverrà anche in questo caso nell'ambiente containerazzato su AWS. Soluzioni di questo tipo sono ampiamente sfruttabili in ambienti retail a 360 gradi, dalle semplici self service machines, a supermarket, librerie, etc. . Inoltre, tali implementazioni permettono l'acquisizione di dati estremamente importanti in ottica IoT e Advanced Smart Retail.

Riferimenti bibliografici

[1] R. Pietrini, D. Manco, M. Paolanti, V. Placidi, E. Frontoni, P. Zingaretti, *An IoT Edge-Fog-Cloud Architecture for Vision Based Planogram Integrity*

[2] Rupak Ganguly,*How to use AWS Fargate and Lambda for long-running processes in a Serverless app* https://www.serverless.com/blog/serverless-application-for-long-running-process-fargate-lambda

[3] lobster1234, *Run tasks with AWS Fargate and Lambda* `https://lobster1234.github.io/2017/12/03/run-tasks-with-aws-fargate-and-lambda/`

[4] AWS, *AWS Documentation* `https://docs.aws.amazon.com`