ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA — SCIENZA E INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Analisi dei costi e benefici di Google Cloud Platform per algoritmi di Machine Learning

Relatore:
Prof. Daniele VIGO

Presentata da:
Luca MARINI

SESSIONE UNICA
Anno Accademico 2019 — 2020

«So, ask yourself: If what you're working on succeeds beyond your wildest dreams, would you have significantly helped other people? . . .

If not, then keep searching for something else to work on. Otherwise you're not living up to your full potential.»

Andrew Ng

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

Abstract

Dipartimento di Informatica — Scienza e Ingegneria

Corso di Laurea in Ingegneria e Scienze Informatiche

Laurea

Analisi dei costi e benefici di Google Cloud Platform per algoritmi di Machine Learning

di Luca MARINI

Al giorno d'oggi sempre più aziende richiedono una maggiore flessibilità da poter applicare nell'esecuzione dei propri task computazionali. In questa tesi si vuole svolgere un'analisi dei costi e benefici che Google Cloud Platform (GCP), uno dei leader tra le suite di servizi di cloud computing, offre per l'esecuzione di algoritmi di machine learning. Vengono presentate diverse metodologie con le quali è possibile eseguire questi algoritmi sulla piattaforma in cloud di Google. In particolare, viene approfondito il tema della conteinerizzazione per creare un framework di automatizzazione che riesca ad eseguire algoritmi di machine learning, sfruttando l'efficienza prestazionale dei software container. Si svolge quindi una serie di test ed un confronto, in termini di costi e benefici, tra Google Cloud Platform ed un cloud tradizionale fornito da un provider privato.

Indice

Elenco delle figure

Elenco delle tabelle

Lista delle abbreviazioni

API	**A**pplication **P**rogramming **I**nterface
GCP	**G**oogle **C**loud **P**latform
REST	**RE**presentational **S**tate **T**ransfer
SDK	**S**oftware **D**evelopment **K**it
vCPU	**v**irtual **C**entral **P**rocessing **U**nit
VM	**V**irtual **M**achine

Capitolo 1

Introduzione

Al giorno d'oggi c'è sempre più richiesta dal mercato di flessibilità per l'esecuzione di algoritmi di machine learning.

Il machine learning è un dominio fortemente influenzato dalla rapida evoluzione del cloud computing [1], il quale è un paradigma di elaborazione distribuita su larga scala in cui un pool di calcolatori è messo a disposizione a degli utenti tramite Internet [6]. Le risorse di elaborazione (ad es. potenza di calcolo, archiviazione, software e banda di rete) sono fornite ai consumatori come servizi accessibili su richiesta. Col modello di servizio Infrastucture as a Service (IaaS), i clienti possono specificare lo stack software (ad es. sistemi operativi e applicazioni) da far eseguire sulle proprie macchine virtuali (VM). I requisiti hardware delle VM possono essere regolati anche dai consumatori.

Negli ultimi anni, il cloud computing è diventato un campo emergente nel settore IT. Numerosi fornitori di cloud offrono servizi competitivi di calcolo, archiviazione, rete e hosting di applicazioni, fornendo copertura in diversi continenti, promettendo i migliori prezzi e prestazioni on-demand.

Google Cloud Platform (GCP) è oggi una delle realtà più importanti e in crescita nel mercato del cloud [7]. GCP offre servizi di hosting sulla stessa infrastruttura di supporto che Google utilizza internamente per prodotti come Google Search e YouTube. Google Cloud mette a disposizione delle API per usufruire delle sue funzionalità.

Lo scopo del presente studio è di indagare il potenziale che la piattaforma in cloud di Google offre per eseguire algoritmi di machine learning, analizzando anche i costi e i benefici che questa offre rispetto ad un cloud tradizionale. Con cloud tradizionale si intende un cloud privato gestito, di dimensioni modeste (rispetto ai principali cloud provider come Google, Amazon, Microsoft ecc), completamente gestito da un fornitore esterno. Esso non fornisce delle API con le quali è possibile accedere ai servizi e gestire le risorse. Su Google Cloud Platform sono presenti diverse soluzioni con le quali è possibile eseguire algoritmi di machine learning: ad esempio si può utilizzare AutoML, ovvero una suite di prodotti di machine learning che consente agli sviluppatori con esperienza limitata nel campo del machine learning di addestrare modelli di alta qualità in base alle esigenze aziendali; oppure BigQuery ML, il

quale permette di creare ed eseguire modelli di machine learning utilizzando query SQL standard.

Il focus è stato posto sulla creazione di modelli custom di machine learning a partire da macchine virtuali gestite manualmente sul cloud di Google. In questo modo è stato possibile generare modelli testando diverse configurazioni di VM, le quali sono state successivamente messe in confronto con i tempi e i costi di esecuzione delle macchine virtuali allocate sul cloud tradizionale.

Vengono quindi fatte delle considerazioni su quale delle due tipologie di cloud convenga utilizzare, basandosi sulle seguenti caratteristiche: costo, velocità di esecuzione, servizi aggiuntivi, flessibilità e vincoli orari di utilizzo minimo.

Il lavoro svolto è organizzato come segue. Prima di tutto, nel Capitolo 2, viene introdotto il concetto di cloud computing e vengono elencati i diversi modelli di servizio e di deployment. Nel Capitolo 3 è approfondito il tema della containerizzazione, essenziale per automatizzare e semplificare l'intera esecuzione di una macchina virtuale. Viene esaminata dunque nel Capitolo 4 la piattaforma in cloud di Google, analizzando esempi pratici e ponendo una particolare attenzione su servizi quali Compute Engine, Cloud Storage e Container Registry, con i quali è possibile rispettivamente allocare e gestire manualmente macchine virtuali; archiviare oggetti di qualunque tipo; e prelevare o caricare immagini di software container. Il Capitolo 5 riguarda il benchmarking, e descrive il caso d'uso considerato, le tecnologie e gli strumenti utilizzati, il framework di automatizzazione che è stato ideato per la generazione di modelli di machine learning, un'analisi dei risultati nella quale sono riportati gli esperimenti svolti, ed infine delle considerazioni su quale tipologia di cloud conviene utilizzare in termini di costi e benefici.

Capitolo 2

Cloud

Nella Sezione 2.1 vengono introdotti i concetti di *cloud* e di *cloud computing*, mentre nella Sezione 2.2 sono elencati i principali modelli di deployment in cloud.

2.1 Overview

Il *National Institute of Standards and Technology* (NIST) definisce il *cloud computing* come:

"un ambiente di esecuzione elastico che consente l'accesso via rete e su richiesta ad un insieme condiviso di risorse di calcolo configurabili (ad esempio rete, server, dispositivi di memorizzazione, applicazioni e servizi) sotto forma di servizi a vari livelli di granularità. Tali servizi possono essere rapidamente richiesti, forniti e rilasciati con minimo sforzo gestionale da parte dell'utente e minima interazione con il fornitore" [19].

Il cloud computing è caratterizzato da cinque caratteristiche fondamentali:

1. **Self-service su richiesta**: un cliente può richiedere risorse computazionali, come risorse di elaborazione o altro, in modo automatico e quando necessario, senza richiedere un intervento umano da parte dei fornitori dei servizi stessi.
2. **Ampio accesso alla rete**: le risorse sono disponibili in retee sono accessibili attraverso meccanismi standard che ne permettono l'uso con piattaforme client diversificate ed eterogenee semplici e complesse (ad es. telefoni cellulari, computer portatili, tablet, ecc).
3. **Condivisione delle risorse**: le risorse di calcolo del fornitore sono organizzate per essere disponibili a diversi clienti, utilizzando il modello multi-tenant (modello condiviso), in cui le risorse fisiche e virtuali sono assegnate dinamicamente a seconda delle richieste. Le risorse usufruibili sono indipendenti dalla loro esatta posizione fisica, quindi i clienti generalmente non conoscono dove sono collocate le risorse a lui fornite. Tuttavia, il fornitore potrebbe permettere all'utente di specificare la locazione delle risorse a un certo livello di astrazione, per esempio in

termini di area geografica, paese o data center. Esempi di risorse sono: risorse di archiviazione, di calcolo, di rete e macchine virtuali.

4. **Elasticità**: le risorse possono essere fornite in modo rapido ed elastico, ed in alcuni casi anche automatico, per incrementare e decrementare velocemente le capacità computazionali allocate. Dal punto di vista dell'utente le capacità disponibili appaiono illimitate, e possono essere richieste in qualsiasi quantità ed in qualsiasi momento.
5. **Servizi monitorati**: i sistemi cloud controllano ed ottimizzano automaticamente l'utilizzo delle risorse, sfruttando la capacità di misurarne l'utilizzo delle risorse col livello di astrazione adeguato in base al tipo di servizio (ad es. servizi di archiviazione, di calcolo e di banda di rete). L'utilizzo delle risorse può essere monitorato, controllato e segnalato, garantendo trasparenza sia al fornitore che al cliente che usufruisce del servizio.

2.1.1 Modelli di servizio

Le risorse in cloud vengono fornite come servizio (*as-a-service*) in rete e possono corrispondere a tre livelli diversi [23]:

Software as a Service (SaaS)

In questo caso i servizi offerti al consumatore da parte del fornitore sono applicazioni funzionanti su un'infrastruttura cloud, accessibili da diversi dispositivi tramite una semplice interfaccia. Un esempio di servizio SaaS può essere un'applicazione di email su browser, oppure giochi online. Il consumatore non gestisce, non controlla e quindi non conosce alcun dettaglio dell'infrastruttura cloud sottostante (rete, server, sistemi operativi, memoria, capacità delle singole applicazioni).

Platform as a Service (PaaS)

Il livello PaaS offre all'utente la possibilità di fare il deployment in cloud di applicazioni, utilizzando linguaggi di programmazione, librerie, servizi e strumenti supportati dal cloud provider. Il consumatore non gestisce né controlla l'infrastruttura cloud sottostante (rete, server, sistemi operativi, memoria), ma può gestire e creare delle applicazioni utilizzando un'interfaccia di programmazione (API) messa a disposizione dal fornitore. Esempi di servizi PaaS sono database e web server.

Infrastructure as a Service (IaaS)

Infine il modello di servizio IaaS consente all'utente di acquisire risorse di elaborazione, calcolo, memoria e rete, sui quali possa poi installare ed eseguire del software a lui necessario, come ad esempio dei sistemi operativi

e delle applicazioni. Il consumatore non gestisce l'infrastruttura cloud sottostante, ma controlla appunto sistemi operativi, memoria, applicazioni ed eventualmente alcune componenti di rete (ad es. firewall).

2.2 Modelli di deployment

Un cloud è caratterizzato da una propria architettura o modello di deployment, che riguarda i data center in cui le singole tecnologie sono integrate per creare il cloud. I principali modelli di deployment di un cloud sono:

Private Cloud

In un cloud privato l'infrastruttura è ad uso esclusivo di una singola organizzazione, alla quale possono essere associati molteplici clienti (ad es. filiali). Può essere posseduta, diretta e gestita dall'organizzazione stessa, da una società esterna, o da un insieme delle due.

Community Cloud

In un modello Community cloud l'infrastruttura esistente è condivisa da un insieme di clienti provenienti da aziende, organizzazioni che condividono uno scopo comune (ad es. una stessa missione, uguali requisiti di sicurezza ecc.). L'infrastruttura può essere gestita da una o più organizzazioni appartenenti alla comunità, da un fornitore di servizi esterno, oppure da un insieme delle precedenti.

Public Cloud

I fornitori di servizi in cloud pubblici mettono a disposizione le risorse (di memorizzazione, di calcolo ecc.) dei loro data center a qualsiasi loro cliente. L'intera infrastruttura può essere posseduta e gestita da un'azienda, da un'organizzazione accademica o governativa, oppure da un insieme delle precedenti.

Il punto di forza dei cloud pubblici è dato dal fatto che gli utenti utilizzano le risorse solo per il tempo in cui ne hanno effettivamente bisogno. I clienti quindi hanno un notevole risparmio dei costi sia perché il tempo di utilizzo delle risorse è ottimizzato, sia perché non devono occuparsi della gestione e della manutenzione dell'infrastruttura.

Hybrid Cloud

Il cloud Ibrido è una composizione di due o più modelli di deployment (Private, Community o Public), i quali rimangono separati, ma uniti allo stesso tempo dalla portabilità di dati e applicazioni apportata da tecnologie standard o proprietarie.

Per esempio, un cliente potrebbe decidere di usufruire delle potenzialità di un cloud pubblico ogni qualvolta si dovessero presentare dei picchi inaspettati di lavoro che superano le capacità massimali di quello privato. In un altro contesto, un utente potrebbe utilizzare rispettivamente un cloud privato per gestire dati sensibili e uno pubblico per eseguire i restanti compiti.

Capitolo 3

Containerizzazione

In questo Capitolo vengono inizialmente introdotti i *Software Container* nella Sezione 3.1. Nella Sezione 3.2 è esposto Docker, e la Sezione 3.3 spiega come funziona il servizio di orchestrazione di container Kubernetes.

3.1 I Software Container

I *Software Container* sono costituiti da un intero ambiente isolato: un filesystem, un'applicazione, le sue librerie, file binari e di configurazione necessari per l'esecuzione, raggruppati in un unico pacchetto [21]. Ad esempio possono essere utilizzati quando si vogliono far eseguire su uno stesso hardware più applicazioni che utilizzano diverse versioni di una stessa libreria. Il loro compito è quello di isolare in degli spazi utente separati le applicazioni che hanno esigenze di software tra loro incompatibili, utilizzando un unico sistema operativo. Offrono una soluzione al problema di come far funzionare il software in modo affidabile quando si passa da un ambiente di elaborazione a un altro. Questa tipologia di servizio facilita il *deployment*, visto che i dettagli delle applicazioni basate sui container sono già gestiti all'interno del container stesso [14].

La containerizzazione consente pertanto una maggiore modularità. Anziché eseguire un intero programma all'interno di un singolo contenitore, si può suddividere un'applicazione in moduli (database, front-end, server ecc.). Le applicazioni create in questo modo sono più facili da gestire poiché ogni modulo è relativamente semplice e le modifiche possono essere apportate ad essi in modo isolato. Poiché i container sono leggeri, i singoli moduli possono essere allocati solo quando sono necessari e in modo immediato.

3.1.1 Confronto tra macchine virtuali e container

I container offrono una virtualizzazione più leggera rispetto alle macchine virtuali, infatti non virtualizzano lo stack hardware, ma bensì effettuano la virtualizzazione a livello del sistema operativo (*OS-Level virtualization*). Di fatto i container condividono il kernel del sistema operativo host anziché

crearne uno da zero (sono una partizione isolata di tale sistema operativo), perciò sono più efficienti in termini di:

- velocità, in quanto le istruzioni da effettuare vengono eseguite direttamente sul kernel host e non devono essere interpretate da un hypervisor per poi essere eseguite;
- utilizzo di memoria, poiché non contengono un intero sistema operativo.

Questa loro natura leggera rende possibile l'esecuzione di diversi container sulla stessa macchina.

Nonostante i numerosi pregi, questa innovazione presenta anche dei difetti. Come citato precedentemente, i container condividono il kernel del sistema operativo della macchina sulla quale sono dispiegati, per cui se quest'ultimo diventa vulnerabile, anche tutti i container saranno vulnerabili e potrebbe essere un'applicazione all'interno di un container stesso a causare questa vulnerabilità. Tuttavia questa situazione è più una rarità che una normalità. Infine la virtualizzazione a livello di sistema operativo non può ospitare sistemi operativi differenti, poiché sfrutta l'unico sistema operativo ospitante per creare diverse partizioni.

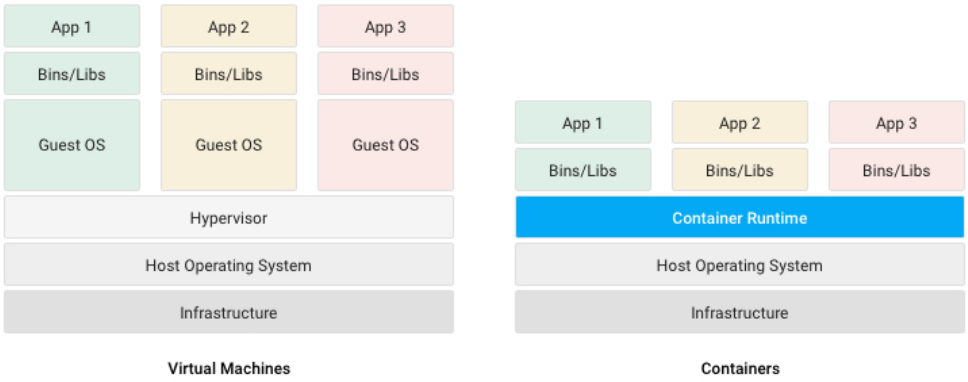


FIGURA 3.1: Differenza tra Macchine Virtuali e Container.

Un deployment basato su hypervisor è l'ideale quando le applicazioni sullo stesso cloud richiedono sistemi operativi diversi o versioni differenti del sistema operativo. Nei sistemi basati su container, le applicazioni condividono un sistema operativo, quindi queste distribuzioni possono avere dimensioni significativamente inferiori.

3.2 Docker

Docker è una delle piattaforme principali per poter creare e gestire i software container e il deployment di moltissimi servizi presenti in cloud è effettuato utilizzando dei Docker container.

3.2.1 Architettura di Docker

Docker è un progetto *open-source* organizzato con un'architettura *client-server* che è composta da tre componenti principali [13]:

- **Docker Daemon o Docker Engine**: è un server, detto anche demone, eseguibile col comando *dockerd*, il quale crea, esegue, e distribuisce container su richiesta dei client.
- **Docker Client**: è un'interfaccia client a linea di comando, utilizzabile col comando *docker*, con la quale si può comunicare con uno o più demoni.

Le due parti possono essere eseguite sullo stesso sistema, oppure un client Docker può interagire con un demone remoto eseguito su un altro host tramite socket o API REST.

3.2.2 Elementi di Docker

Questo servizio comprende diversi elementi fondamentali per il suo utilizzo, di seguito ne vengono elencati i principali:

- **Docker Image**: un'immagine è un *template* utilizzabile per creare, aggiornare ed eseguire un container. Spesso le immagini sono generate richiamando da riga di comando una sequenza di istruzioni contenute in un file di testo chiamato *Dockerfile* [4], le quali permettono di eseguire dei comandi, copiare file dall'host, spostarsi all'interno del filesystem, ecc. La costruzione (build) dell'immagine è quindi automatizzata, ciascuna istruzione aggiunge un nuovo strato e se si vuole modificare il Dockerfile e ricostruire l'immagine, vengono ricreati solo i nuovi livelli e quelli che sono stati modificati. Questa caratteristica rende molto veloce la creazione e la modifica delle Docker Image.
- **Docker Registry**: rappresenta un *repository* remoto che contiene un insieme di immagini Docker già definite e facili da utilizzare. È possibile configurare un registro dal quale poter avviare o prelevare immagini. *Docker Hub* è il registry impostato di default.
- **Docker Container**: un container è un'istanza eseguibile di un'immagine. È possibile crearli, avviarli, arrestarli, spostarli o eliminarli mediante l'utilizzo delle API oppure usufruendo dell'interfaccia da riga di comando. Si può connettere un container a una o più reti e associargli dello spazio di archiviazione. Esso è definito dalla sua immagine e dalle eventuali opzioni di configurazione fornite durante la creazione o l'avvio.

3.2.3 Comandi utilizzabili in un Dockerfile

I comandi che possono essere specificati in un Dockerfile consentono di costruire l'immagine di un container e hanno diverse funzionalità. I principali

sono [13]:

- **FROM**: questo comando deve essere posizionato all'inizio del Dockerfile e specifica un'immagine di partenza dalla quale è possibile partire per costruirne una nuova;
- **MAINTAINER**: è opzionale, può essere utilizzato per indicare chi è il manutentore dell'immagine;
- **RUN**: è necessario per far eseguire comandi durante il processo di creazione dell'immagine Docker;
- **COPY**: serve per copiare file o intere directory dal filesystem della macchina ospitante all'interno del container che sarà costruito;
- **ADD**: ha la stessa funzionalità del comando COPY, ma esiste un'opzione per fare la copia di un file direttamente da un URL, quindi via rete, oltre che dal filesystem locale della macchina host;
- **ENV**: permette di definire una variabile d'ambiente, che potrà poi essere utilizzata e visualizzata dai comandi successivi e dal container;
- **CMD**: consente di specificare qual è il comando che deve essere eseguito quando l'immagine costruita verrà successivamente messa in esecuzione;
- **ENTRYPOINT**: ha la stessa funzione di CMD, ma con una sintassi leggermente diversa;
- **WORKDIR**: permette di spostarsi dentro una cartella e quindi tutte le successive esecuzioni dei comandi RUN, CMD, ENTRYPOINT, COPY e ADD saranno eseguite all'interno di essa;
- **VOLUME**: stabilisce una mappatura tra una directory all'interno del container e un'altra directory contenuta nella macchina che ospita il container.
- **SHELL**: l'istruzione SHELL consente di sovrascrivere la shell di default utilizzata per eseguire i comandi. La shell predefinita su Linux è ["/bin/sh", "-c"]. L'istruzione SHELL deve essere scritta in formato JSON in un Dockerfile.

3.3 Kubernetes

Nella Sezione 3.2 è stato introdotto Docker, con il quale è possibile creare, eliminare o modificare, uno per volta, dei container. Ma quando il numero di container da gestire aumenta, allora conviene utilizzare un orchestratore di

container. *Kubernetes* (noto anche come k8s o "kube") è una piattaforma di orchestrazione di container open source che automatizza molti dei processi manuali coinvolti nella distribuzione, gestione e ridimensionamento delle applicazioni containerizzate [16]. Ciò riduce considerevolmente le tempistiche e gli errori commessi rispetto ad una completa gestione manuale.

Kubernetes è stato originariamente sviluppato e progettato dagli ingegneri di Google e annunciato nel 2014, e con le sue funzionalità principali è possibile:

- orchestrare i container su più host (o su più vm) presenti all'interno di un cluster
- automatizzare il deployment e l'aggiornamento delle applicazioni
- ridimensionare istantaneamente (aggiungendo o rimuovendo container) le applicazioni containerizzate e le loro risorse in base al carico di lavoro. Kubernetes riesce a garantire la sua disponibilità di servizio anche in caso di *fault tolerance*, ovvero in presenza di guasti, perché è in grado di replicare in modo automatico, ridimensionare e riavviare i container che compongono le applicazioni. Questa sua facilità di scalare lo rende ottimo per essere utilizzato in ambienti cloud, perché minimizza il numero di risorse da utilizzare e quindi anche i costi.

Le principali entità di Kubernetes sono [14]:

- **Nodi**

  I Nodi sono gli host, cioè le macchine fisiche o virtuali che costituiscono il cluster, sulle quali poi verrà effettuato il deployment delle applicazioni.

- **Pod**

  Un *Pod* è l'unità di esecuzione di base di un'applicazione Kubernetes ed è formato da un gruppo di uno o più container strettamente correlati. È possibile creare diverse copie un Pod, ma ogni Pod è contenuto in un solo Nodo. I container contenuti all'interno di uno stesso Pod condividono uno stesso indirizzo IP e possono comunicare direttamente tra loro utilizzando le *Inter process Communications* (IPC). Invece se un container1 volesse interagire con un altro container2 in un Pod differente, dovrebbe specificare l'indirizzo IP del Pod a cui appartiene il container2. Le porte esterne di un Pod sulle quali i container possono esporre servizi sono uniche, perciò possono essere utilizzate solo da un container per volta.

- **Deployment**

  Questo termine non va inteso come una distribuzione in senso generale, poiché identifica un gruppo di Pod identici a cui rivolgersi per ottenere una determinata funzionalità. È possibile specificare e ridimensionare il numero di repliche dei Pod; ad esempio potrebbe essere conveniente

aumentarli nel caso in cui il carico di lavoro dovesse crescere. Maggiore sarà la numerosità di richieste dirette verso le repliche, più sarà necessario distribuirle uniformemente per sfruttare nel modo migliore le risorse disponibili. Un *Deployment* è spesso formato da Pod che sono contenuti in Nodi diversi, cosicché se un Nodo per esempio dovesse fallire, allora la funzionalità richiesta potrebbe poi essere fruita da un Pod presente in un altro Nodo.

- **Service**

  Un *Service* (o Servizio) è quell'entità che rende possibile la connessione tra Deployment e il mondo esterno (il resto dell'applicazione). È esso che solitamente si occupa del bilanciamento di carico, fornito di default da Kubernetes, oppure può essere personalizzato con alcune politiche definite dal progettista del sistema. Quando un client si rivolgerà ad un Servizio per ottenere una certa funzionalità, sarà poi quest'ultimo ad inoltrare la richiesta al Deployment e quindi ad uno dei Pod. Un Service serve quindi a distribuire il carico di lavoro ai diversi Pod e a proteggerli, poiché nasconde agli utenti gli indirizzi IP dei Pod. Nel caso in cui diversi Deployment dovessero comunicare tra loro, allora servirebbe un ulteriore Servizio che vada a bilanciare il carico e a proteggere l'altro Deployment. Dunque il bilanciamento di carico avviene non solo tra l'applicazione e l'esterno, ma anche tra diversi Deployment.

3.3.1 Architettura di Kubernetes

Il livello di astrazione più alto dell'architettura è il Kubernetes cluster, ovvero un gruppo di sistemi di elaborazione che eseguono container gestiti da Kubernetes. In un cluster sono presenti uno o più *Worker Node* e almeno un *Control Plane Node*. Quest'ultimo è chiamato anche *Master Node*, è l'entità principale dell'intero sistema e ha il compito di coordinare l'esecuzione dei nodi di tipo Worker, e lo fa esponendo le Kubernetes API tramite un API Server utilizzato anche dagli sviluppatori. Quando il Server API riceve una richiesta, stabilisce se è valida o meno, e in caso positivo procede alla sua elaborazione. Si può accedere all'API con varie modalità: con chiamate REST, tramite l'interfaccia a riga di comando *kubectl* oppure utilizzando strumenti come *kubeadm*.

All'interno del Control Plane Node è presente *etcd*, un database *key-value* (composto da coppie chiave-valore) distribuito che memorizza dati e informazioni di configurazione relativi allo stato del cluster [22]. Predilige la consistenza rispetto alla disponibilità in caso di partizionamento della rete (teorema *CAP*) e fornisce un modo affidabile per archiviare i dati ai quali è necessario accedere da un sistema distribuito come un Kubernetes cluster. Un altro componente del Master è lo *Scheduler*, che seleziona su quale nodo Worker eseguire un nuovo Pod in base alla disponibilità delle risorse. Altri elementi contenuti nel nodo Master sono i *Controller Manager*, ognuno dei quali possiede delle funzionalità tra quelle principali indicate di seguito:

- comunicazione con il server API per creare, aggiornare ed eliminare le risorse che gestisce (ad es. i Pod);
- gestione di replicazione, ridimensionamento e sostituzione dei Pod nel cluster.

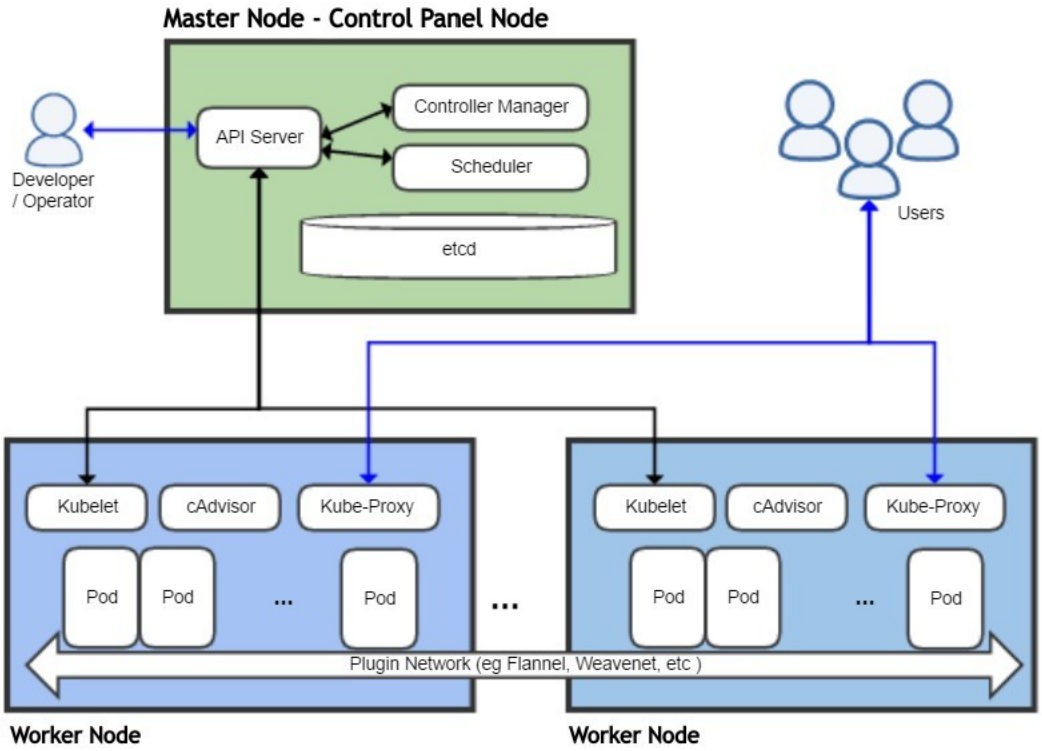


FIGURA 3.2: Diagramma di un Kubernetes cluster

Un Worker Node è una macchina sulla quale vengono eseguiti dei container. Ciascun nodo lavoratore deve eseguire un *Container Runtime* (ad es. Docker).

In aggiunta, ogni Worker Node ha al suo interno *kubelet*, un'applicazione che comunica con il Control Panel Node e verifica che i container vengano eseguiti in dei Pod. Quando il nodo Master deve far eseguire un compito ad un nodo, tale esecuzione viene affidata a kubelet. Tutti i nodi Worker contengono anche *kube-proxy*, un proxy che ottimizza i servizi di rete Kubernetes. La sua funzione è gestire le comunicazioni di rete interne ed esterne al cluster affidandosi al livello di filtraggio dei pacchetti del sistema operativo oppure inoltrando il traffico per proprio conto.

Infine, al cluster viene aggiunto un registry, configurato dall'utente o da terzi, che contiene le immagini dei container che devono essere messi in esecuzione.

Capitolo 4

Google Cloud Platform

Questo Capitolo introduce la piattaforma in cloud di Google, successivamente spiega rispettivamente nelle Sezioni 4.2 e 4.3 come vengono gestite e strutturate le risorse e i costi. La Sezione 4.4 mostra come sono organizzati i servizi di Google nell'omonima piattaforma. Le Sezioni 4.5, 4.6 e 4.7 illustrano i concetti principali dei servizi di GCP utilizzati (Compute Engine, Cloud Storage e Container registry). Nella Sezione 4.8 vengono illustrate diverse soluzioni con le quali è possibile eseguire algoritmi di machine learning su GCP.

4.1 Overview

GCP (Google Cloud Platform) è la piattaforma in *cloud* di Google. Si è scelto di approfondire il servizio cloud di Google per la vastità della sua rete, caratterizzata da un alto *throughput* e latenze bassissime. Questa si interconnette a più di 90 *Internet exchange point* e a più di 100 *Point of presence (PoP)* in tutto il mondo.

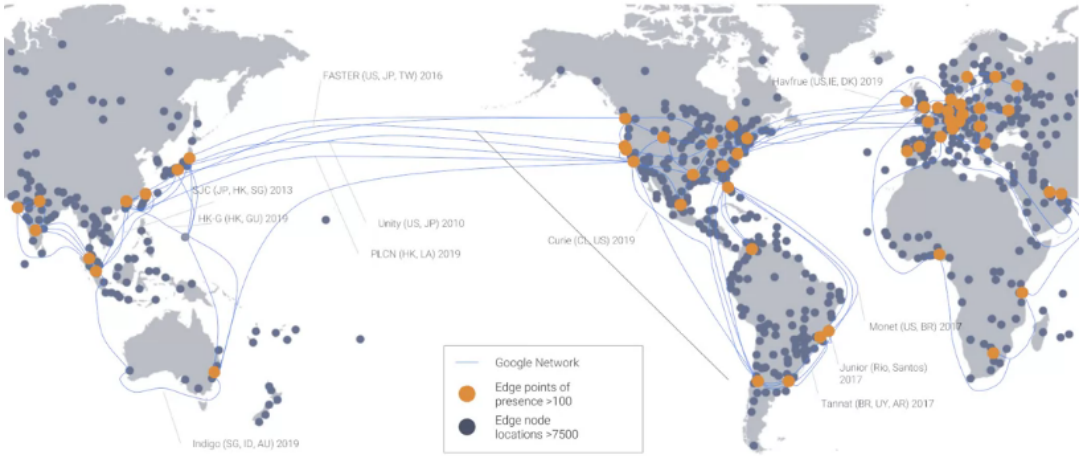


FIGURA 4.1: Rete globale di Google

Questa piattaforma è organizzata in zone, ovvero aree dove poter dispiegare delle risorse (ad es. una macchina virtuale), che possono corrispondere a

una o più strutture fisiche. Esse sono poi raggruppate in Regioni, le quali condividono tra di loro una connettività di rete molto veloce. In particolare il loro *round trip time* (tempo di andata e ritorno di un pacchetto) è inferiore a 5 ms. Le risorse si possono distribuire su più zone in una regione o anche su regioni diverse.

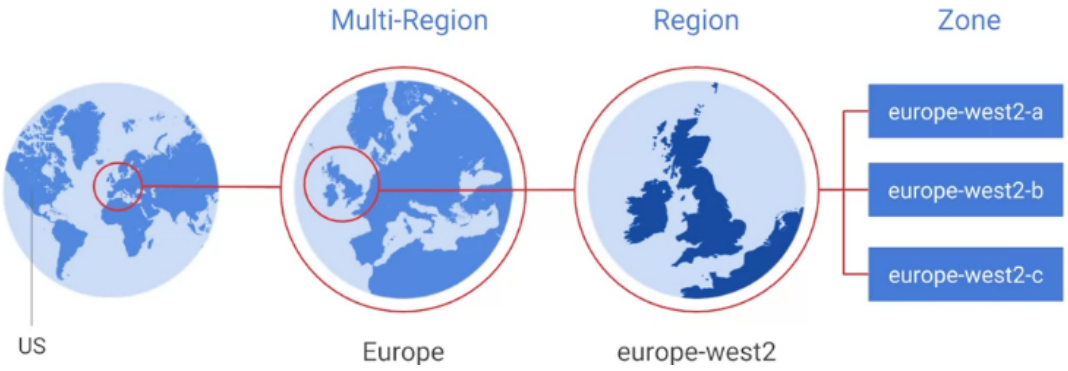


FIGURA 4.2: Zone e Regioni di GCP

4.1.1 Interazione con GCP

Sono presenti quattro modalità per interagire con Google Cloud Platform: la *Console* di Google, il *Software development kit* (SDK) e la *Cloud Shell*, l'applicazione mobile e le *API*.

La prima è un'interfaccia web che permette di vedere e gestire i propri progetti e tutte le proprie istanze. Inoltre si possono abilitare e disabilitare le API dei diversi servizi disponibili.

La Cloud Shell invece consiste in un'interfaccia a riga di comando facilmente accessibile dal proprio browser. Con questa *shell* si possono utilizzare gli strumenti forniti dal software development kit (SDK) della piattaforma, ovvero l'insieme di strumenti che servono per gestire le risorse e le applicazioni su GCP. È possibile anche installare e fare uso dell'intero kit di sviluppo software tramite un'interfaccia a riga di comando in una macchina in locale.

L'applicazione mobile è disponibile sia su sistemi *Android* che *iOS* e offre la possibilità di esaminare e gestire le risorse in cloud, oltre a gestire la fatturazione e a visualizzare i propri progetti.

Infine esistono delle *Representational state transfer application programming interface* (REST API) e delle API relative a diversi linguaggi di programmazione (Python, Java, C#, C++, PHP ecc.) per usufruire delle principali funzioni dei servizi della piattaforma.

4.2 Gestione delle Risorse

La risorse sono strutturate con una gerarchia ad albero. Partendo dal basso, le foglie rappresentano delle risorse vere e proprie, come ad esempio delle macchine virtuali e dei dischi di archiviazione, e sono contenute in progetti. Questi a loro volta possono essere organizzati in cartelle. In cima, il nodo radice identifica un'organizzazione, alla quale possono appartenere diversi progetti e cartelle.

Per ogni progetto, cartella e nodo organizzazione si possono definire regole e permessi sulle risorse e sui servizi. Le regole sono ereditate dall'alto verso il basso nella struttura.

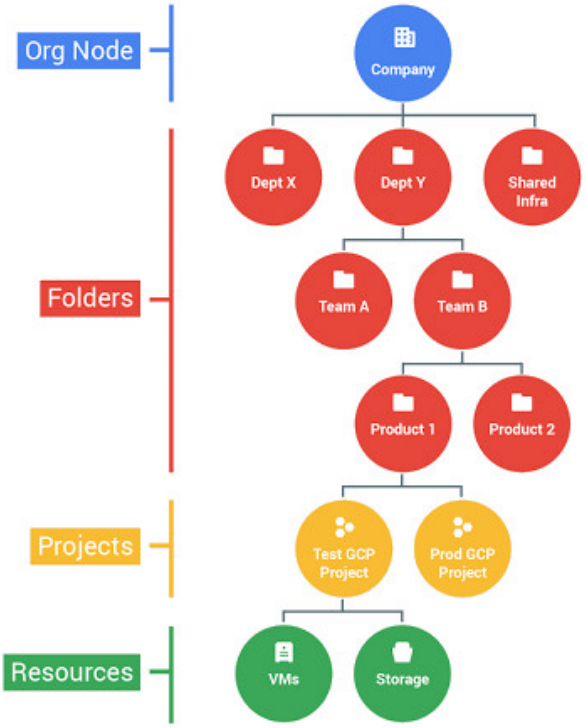


FIGURA 4.3: Esempio di una gerarchia delle risorse

I progetti sono la base per abilitare e utilizzare i servizi GCP come la gestione delle API e della fatturazione, l'aggiunta e la rimozione di collaboratori e l'abilitazione di altri servizi Google. Ogni progetto è un compartimento separato, può avere più proprietari e utenti, e ciascuna risorsa appartiene ad uno e un solo progetto.

Le cartelle possono essere utilizzate per rappresentare dipartimenti diversi, team, applicazioni o settori dell'organizzazione. Per poterle creare è necessaria la presenza di un nodo organizzazione, che serve per avere una visibilità centralizzata su come vengono utilizzate le risorse e applicate le politiche di gestione.

4.2.1 Esempio di organizzazione delle risorse

Nell'esempio della Figura 4.4 non sono presenti cartelle, ma esistono tre progetti, ciascuno dei quali utilizza risorse provenienti da diversi servizi GCP (ad es. una macchina virtuale su Compute Engine e un bucket su Cloud Storage). Le risorse ereditano le regole dal livello superiore. Ad esempio, se viene impostata una norma sul nodo di organizzazione, essa viene ereditata automaticamente da tutti i progetti figli e dalle loro risorse.

D'altro canto è importante sapere che le politiche attuate a un livello superiore in questa gerarchia non prevalgono su quelle ai livelli inferiori. Per esempio, se esiste una regola sul progetto *bookshelf* in Figura 4.4 che stabilisce che l'utente Paolo ha il diritto di modificare un bucket su Cloud Storage, mentre una politica a livello di organizzazione afferma che Paolo può solo visualizzare quel bucket e non modificarlo, allora la prima regola è quella che ha effetto, ovvero quella che scende più nel dettaglio, che è a un livello inferiore.

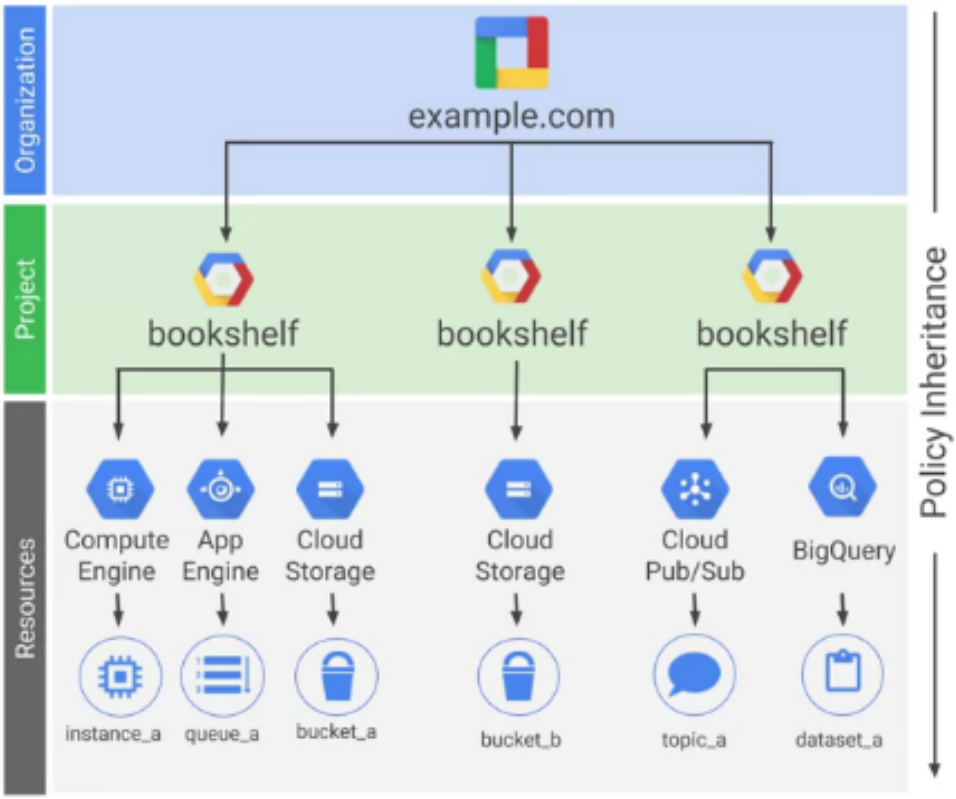


FIGURA 4.4: Esempio di gerarchia IAM

4.2.2 Gestione delle identità e delle autorizzazioni (IAM)

L'*Identity and Access Management (IAM)* è un servizio che permette di gestire gli utenti e le relative autorizzazioni.

Sulla piattaforma cloud di Google i permessi e le risorse sono organizzati a livello di progetto. Solitamente è considerata una buona prassi perseguire il principio del minimo privilegio, che consiste nell'assegnare agli utenti solo quei privilegi necessari per poter svolgere il loro lavoro. Questa politica migliora il livello di sicurezza e previene l'insorgere di futuri errori. Gli amministratori scelgono quali privilegi associare ad ogni utente.

4.2.3 Ruoli dello IAM di Google Cloud

I permessi di un account sono definiti a partire da uno dei ruoli offerti dal gestore delle identità di Google. Ne esistono tre differenti tipologie: primitivi, predefiniti e personalizzati.

Ruoli primitivi

I ruoli primitivi sono quelli meno specifici, una volta applicati a un progetto fanno riferimento a tutte le risorse di quel progetto. Essi sono: proprietario, editor e visualizzatore. Se si è un visualizzatore su una determinata risorsa la si può vedere ma non modificare. L'editor possiede tutti i permessi del visualizzatore, e inoltre può cambiare lo stato delle istanze (avviarle, arrestarle, eliminarle ecc.). Infine il proprietario può svolgere tutte le funzioni appartenenti ad un editor, gestire i ruoli e le autorizzazioni sulle risorse, impostare i budget associati ai progetti, e concedere ad altri account il ruolo di amministratore di fatturazione.

Tuttavia se si lavora con dati sensibili questi ruoli potrebbero essere troppo generici. Per questo il servizio di gestione degli utenti e delle loro relative autorizzazioni di Google fornisce anche la possibilità di definire incarichi più dettagliati.

Ruoli predefiniti

Nello IAM di Google è presente anche un insieme di ruoli predefiniti, caratterizzati da un livello di dettaglio più fine riguardante le operazioni rispetto ai precedenti. Un esempio di ruolo predefinito è *InstanceAdmin*, compreso nel servizio di Compute Engine. Questo titolo permette di effettuare varie azioni sulle macchine virtuali allocate, come elencarle in una lista, visualizzarle, modificare le loro configurazioni, avviarle e arrestarle.

Ruoli personalizzati

I ruoli personalizzati permettono di selezionare le singole funzioni da associare ad ogni utente. Sono utili se si vuole assegnare in modo accurato e ben delineato l'insieme di permessi ai propri dipendenti per prevenire errori

ed aumentare la sicurezza all'interno del proprio sistema informativo. Essi possono essere applicati a livello di progetto o di organizzazione, ma non a livello di cartella. Ad esempio, si potrebbe definire un nuovo ruolo per permette agli utenti di far avviare ed arrestare delle macchine virtuali, ma non di riconfigurarle.

4.2.4 Service Account

Un account di servizio (*Service Account*) è un profilo speciale utilizzato da un'applicazione o da una macchina virtuale, quindi non da una persona specifica, per poter effettuare delle chiamate API autorizzate. Esso è identificato da un indirizzo email univoco e le sue principali differenze rispetto a un account utente sono le seguenti:

- un account di servizio non ha delle password e non può autenticarsi tramite l'utilizzo di *browser* e *cookie*, ma gli vengono associate delle chiavi per poterlo identificare.
- le autorizzazioni Cloud IAM possono essere concesse per consentire ad altri utenti (o altri account di servizio) di rappresentare un account di servizio.

4.3 Gestione dei Costi

Esistono diversi strumenti per monitorare le spese del proprio utilizzo della piattaforma Google. Uno di questi permette di impostare budget legati ad un account di fatturazione o ad un progetto in cloud per non oltrepassare una certa soglia di spesa. Un budget può corrispondere ad una cifra precisa oppure anche ad una percentuale della quota spesa ad esempio in un mese precedente. Inoltre si possono impostare degli avvisi (*alerts*) per farsi notificare quando i costi stanno per raggiungere la quota prefissata (o percentuali di essa).

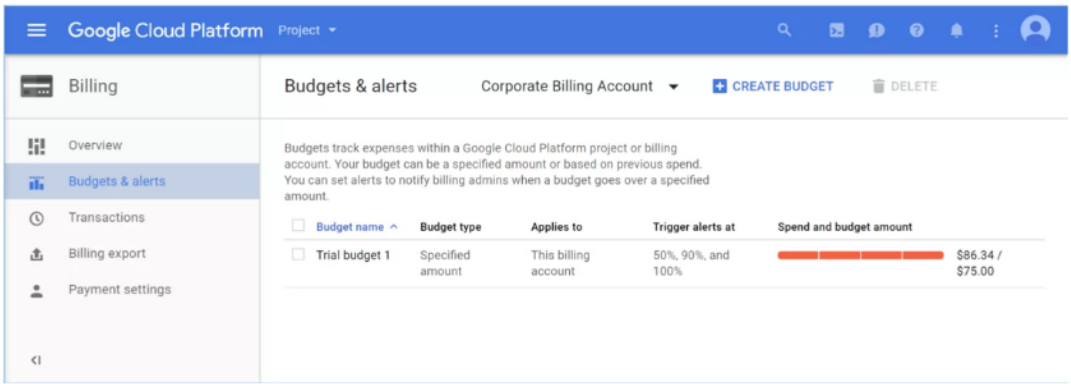


FIGURA 4.5: Esempio di budget e avvisi associati ad un account di fatturazione

Infine si possono visualizzare graficamente le spese con lo strumento *Reports*, che riporta un grafico a linee. Ogni linea rappresenta l'andamento del costo associato ad un progetto.

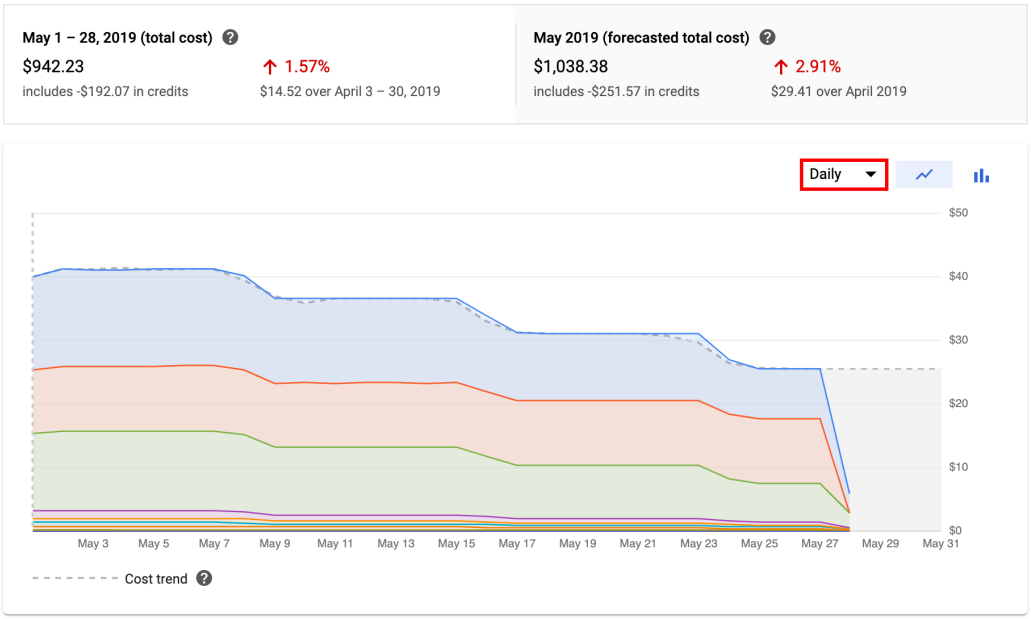


FIGURA 4.6: Visualizzazione di un grafico delle spese generato dallo strumento Reports

Infine Google Cloud Platform rende disponibile l'utilizzo di particolari quote che servono per evitare il consumo eccessivo delle risorse, il quale per esempio può essere dovuto da errori o da attacchi esterni. Ci sono due tipi di quote: quelle tariffarie e quelle di allocazione. Le prime vengono reimpostate dopo un certo periodo di tempo. Per esempio si può impostare un limite di 1000 utilizzi di funzioni API di un servizio ogni 100 secondi. Finito questo tempo viene azzerato il conto dell'impiego e il timer riparte scandendo altri 100 secondi. Invece, le quote di assegnazione hanno lo scopo di limitare il numero di risorse che si possono associare ad uno specifico progetto.

4.4 Organizzazione dei Servizi

GCP offre quattro tipologie di servizi principali:

- **Compute** (calcolo)
- **Storage** (spazio di archiviazione)
- **Big data**
- **Machine learning**

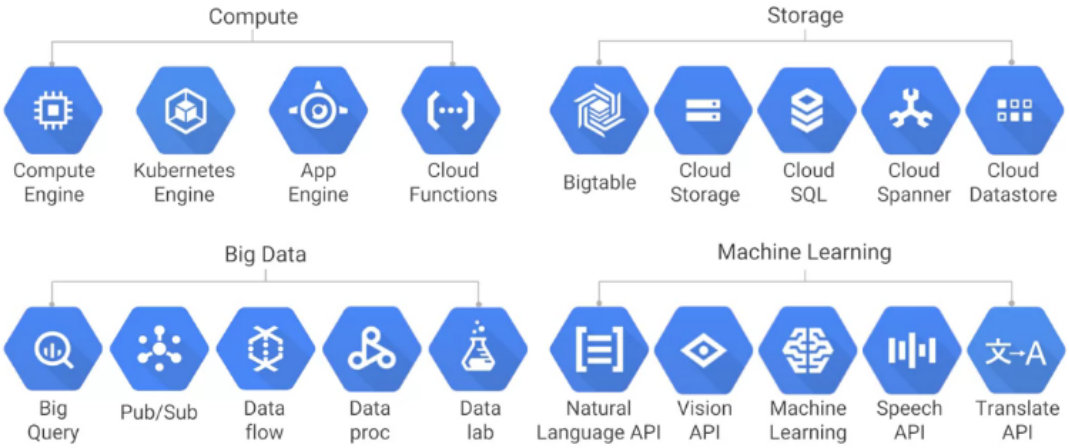


FIGURA 4.7: Principali servizi forniti

4.5 Compute Engine

Il servizio *Compute Engine* (motore di calcolo) consente di allocare e gestire manualmente macchine virtuali situate nei data center di Google. È un servizio di tipo IaaS (*Infrastructure as a Service*) [11].

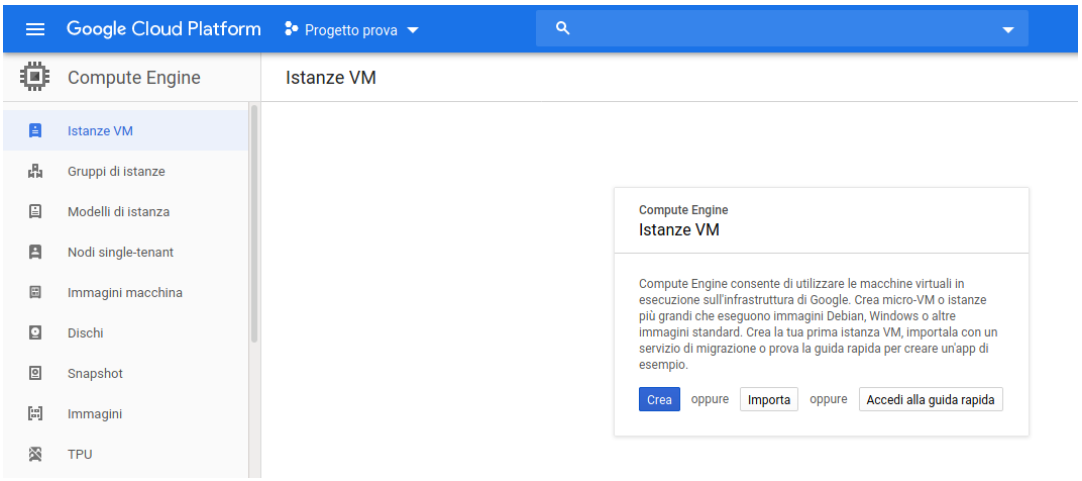


FIGURA 4.8: Schermata principale di Compute Engine

4.5.1 Creazione e avvio di una macchina virtuale

In questa Sezione verrà creata una macchina virtuale utilizzando un'immagine del disco di avvio della distribuzione Debian di Linux. Il servizio di Compute Engine fa partire immediatamente l'istanza dopo che è stata creata. Si possono selezionare immagini di sistemi operativi forniti direttamente da Google, oppure si possono anche importare immagini personalizzate [12].

Una macchina virtuale può essere configurata specificando il numero delle CPU virtuali e la quantità di memoria RAM, utilizzando un insieme di configurazioni predefinite o creandosi la propria tipologia di istanza personalizzata. Dopo aver allocato una risorsa, è possibile riconfigurare le sue proprietà senza doverla eliminare e creare nuovamente.

Inoltre si possono associare uno o più dischi di archiviazione alla propria istanza nel processo di creazione e se ne possono aggiungere altri anche dopo aver allocato la risorsa.

Nelle prossime sezioni verrà descritto come generare una macchina virtuale con i seguenti strumenti:

- **Google Console** (l'interfaccia web)
- **Google Cloud Shell** e il suo **Software Development Kit** (SDK)

VM e Console di Google

Per creare un'istanza dalla Console di Google è sufficiente recarsi nella pagina delle *Istanze VM*, selezionare il progetto che la conterrà, e fare click sul pulsante Crea. Si aprirà un menù nel quale si potranno specificare numerose caratteristiche, partendo dal nome e dalla zona della risorsa stessa.

Nel riquadro *Configurazione macchina* è possibile specificare le principali componenti, ovvero il numero di core appartenenti alla *vCPU* (CPU virtuale) e la quantità di memoria RAM.

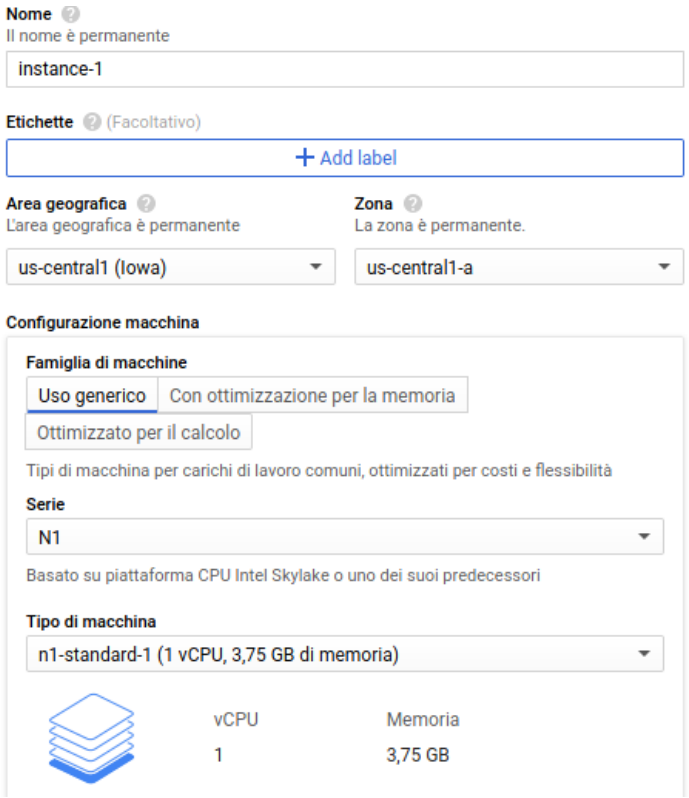


FIGURA 4.9: Proprietà della macchina virtuale

Dopodiché si può decidere quale disco di avvio utilizzare. Può essere un disco permanente standard (più economico) o un disco *solid state drive* (SSD). Ci sono varie famiglie di sistemi operativi disponibili (Debian, Windows Server ecc.) e diverse versioni di queste. In alternativa, il disco di avvio può essere creato a partire da un'immagine personalizzata, da uno *snapshot* o da dischi esistenti.

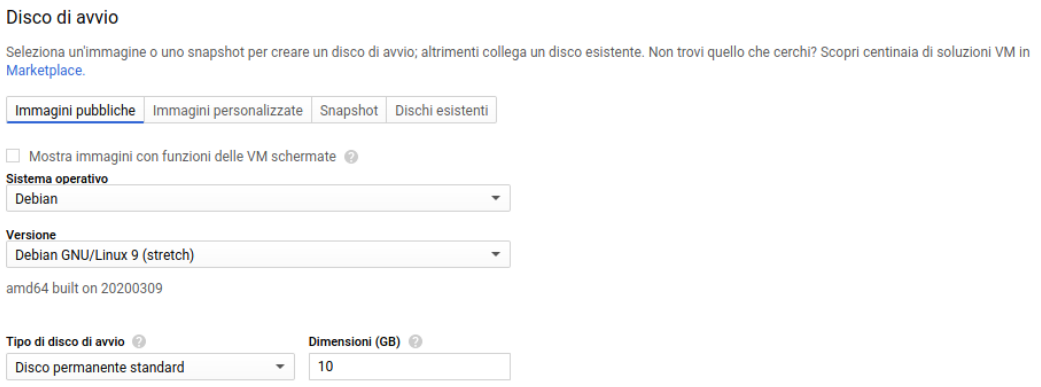


FIGURA 4.10: Scelta del disco di avvio

In termini di *Firewall* è possibile permettere la ricezione di traffico HTTP e/o HTTPS. Mentre una macchina virtuale è in stato di esecuzione si possono aggiungere altri dischi di archiviazione secondari (non di avvio) ad essa.

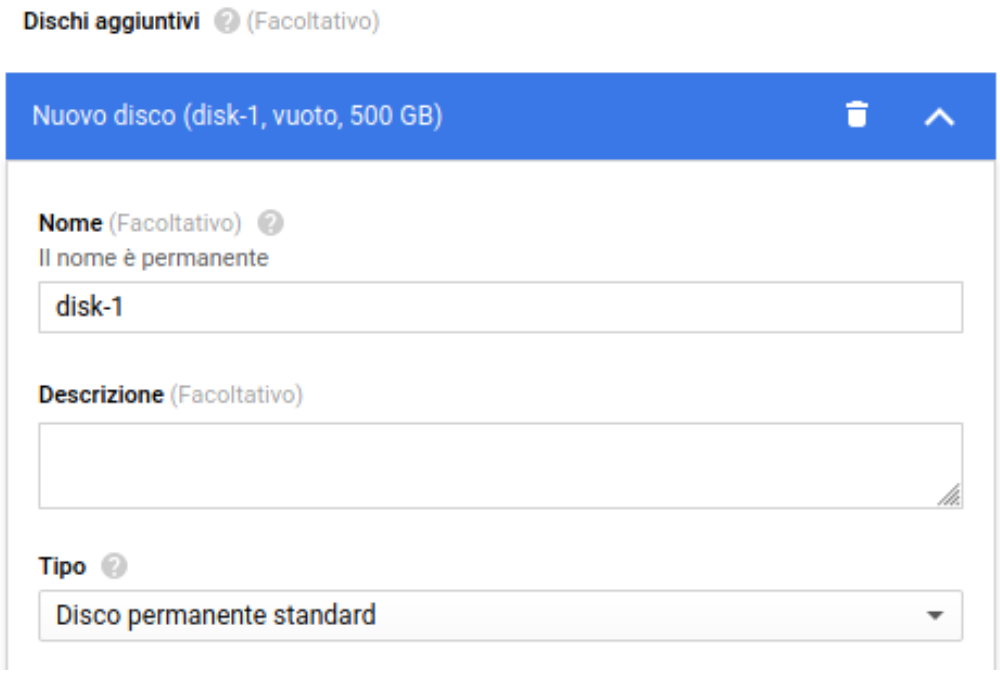


FIGURA 4.11: Dischi di archiviazione aggiuntivi

Infine è possibile associare un proprio script da far eseguire all'avvio dell'istanza.

Dopo aver svolto il *deployment* di una macchina virtuale, essa potrà essere visibile nella lista delle *Istanze VM* nella pagina principale. Qui è possibile arrestare, cancellare, creare, far ripartire e connettersi alle proprie istanze.

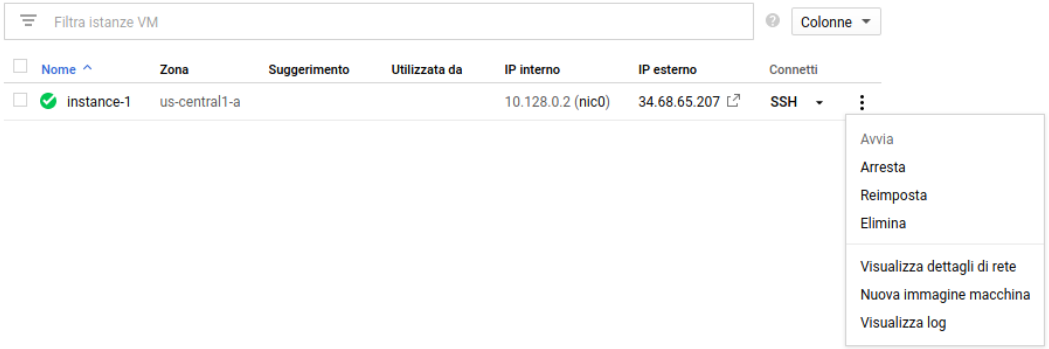


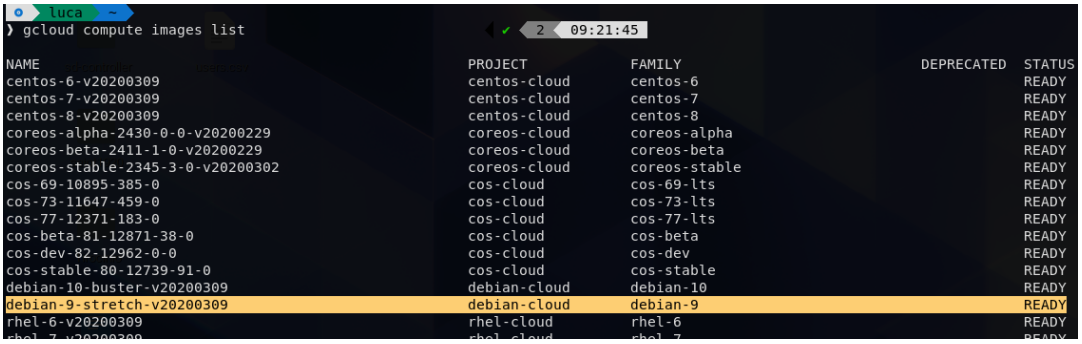
FIGURA 4.12: Lista istanze VM

VM, Google Shell e SDK

In alternativa una macchina virtuale è generabile sfruttando lo strumento gcloud presente nel SDK, utilizzabile tramite la Cloud Shell di Google oppure mediante una shell locale previa installazione del Cloud SDK stesso.

Per vedere la lista delle immagini dei sistemi operativi da utilizzare per il disco di avvio è necessario eseguire il comando:

```
gcloud compute images list
```



Quindi, dopo aver scelto un'immagine, vanno specificate alcune informazioni relative all'immagine stessa (tutte reperibili dall'*output* del comando precedente):

- il nome, nel campo [INSTANCE_NAME]
- la versione, nel campo [IMAGE_FAMILY]
- e l'immagine di progetto, nel campo [IMAGE_PROJECT]

nel seguente comando:

```
gcloud compute instances create [INSTANCE_NAME] \
    --image-family [IMAGE_FAMILY] \
    --image-project [IMAGE_PROJECT]
```

Con il flag `--create-disk` si possono aggiungere eventuali dischi di archiviazione secondari. Infine l'istanza viene creata.

4.5.2 Tipologie di macchine

Compute Engine offre tre macro famiglie di macchine disponibili: *General-purpose*, *Memory-optimized* e *Compute-optmized*.

La tipologia General-purpose è caratterizzata da risorse adatte per carichi di lavoro comuni, ottimizzate per costi e flessibilità. Le varie classi sono: *n1*, *n2*, *n2d* ed *e2*. L'unica differenza tra le serie n1, n2 e n2d è data dal diverso tipo di CPU che incorporano. I tipi di macchine e2 sono VM ottimizzate maggiormente in termini di costi, infatti forniscono la varietà di risorse di calcolo con il prezzo più basso su Compute Engine. Le quattro classi appena citate hanno altre tre sotto-categorie che identificano il tipo di istanza: *standard*, *high-mem* e *high-cpu*. Queste tre sottoclassi rappresentano rispettivamente macchine standard; ad elevata CPU, le quali apportano un moderato aumento delle

vCPU rispetto alla memoria RAM; e ad elevata memoria, che sono l'esatto opposto di quelle precedenti.

La seconda famiglia è quella delle VM ottimizzate per la memoria. L'unica serie disponibile in questo caso è la *m1*, ideale per carichi di lavoro ad alta intensità di memoria, perché offre più memoria per core rispetto ad altri tipi di macchine.

La famiglia Compute-optimized dispone della classe *c2*, la quale offre massime prestazioni per core ed è appunto ottimizzata per carichi di lavoro ad alta intensità di calcoli. Questa serie è generalmente più robusta per i carichi di lavoro *compute-intensive* rispetto ai tipi di macchine n1 high-cpu.

Per ciascuna delle tipologie citate in questa Sezione è disponibile anche la versione *preemptible*. Le macchine virtuali preemptible sono molto più economiche, poiché la loro esecuzione può essere interrotta da Google in un qualsiasi momento. Sono adatte per carichi di lavoro brevi e a tolleranza di errore, e la loro durata massima è di 24 ore.

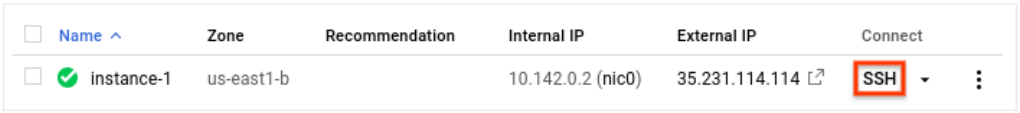
4.5.3 Installazione software sulla macchina virtuale

Esistono due modalità principali per poter installare software esterno sull'istanza allocata:

- tramite una *secure shell* (SSH)
- con uno script di avvio

Secure shell SSH

È possibile connettersi alle proprie macchine virtuali e far funzionare programmi stabilendo delle sessioni remote cifrate tramite interfaccia a riga di comando (ssh). Dal riquadro delle istanze VM si può far partire la connessione ad una risorsa facendo click sul pulsante *SSH*.



Oppure ci si può collegare direttamente da riga di comando dalla Google Cloud Shell o da una bash locale con il Software Development Kit preinstallato, col comando:

```
gcloud compute ssh [INSTANCE_NAME]
```

FIGURA 4.13: Connessione alla macchina virtuale tramite ssh

A questo punto si possono installare programmi esterni da riga di comando.

Script di avvio

In alternativa, del software può essere installato all'atto di creazione dell'istanza se si associa ad essa uno script di avvio, sfruttando diverse modalità:

- specificandone il contenuto mentre si genera la macchina virtuale nella Console di Google

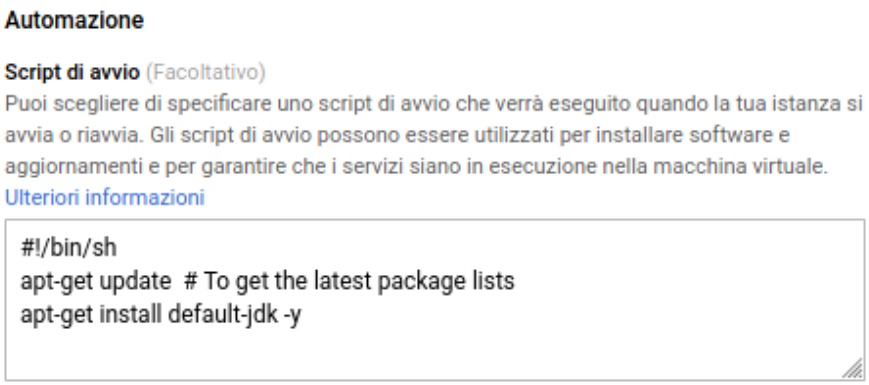


FIGURA 4.14: Esempio di script: installazione del jdk di Java

- con lo strumento gcloud (utilizzabile dalla Google Shell o da una bash locale con Google SDK preinstallato), specificando il file contenente lo script col seguente comando:

```
gcloud compute instances create example-instance \
    --metadata-from-file startup-script=./file.sh
```

- con le *Representational state transfer application programming interface* (REST API) di Google

```
POST https://compute.googleapis.com/compute/v1/projects/
   myproject/zones/us-central1-a/instances
{
  ...
```

```
  "metadata": {
    "items": [
      {
       "key": "startup-script",
       "value": "#! /bin/bash\n\n #Script content"
      }
    ]
  }
  ...
}
```

4.5.4 Container su Compute Engine

Nel servizio di Compute Engine è possibile avviare dei container sulle macchine virtuali con sistema operativo Linux e Windows Server, sulle quali però è necessario installare docker all'avvio. È però disponibile un ulteriore sistema operativo chiamato *Container-Optimized OS*, ottimizzato per la gestione dei container e sul quale è preinstallato docker.

Per poter avviare un singolo container su un'istanza occorre semplicemente specificare il percorso dell'immagine del contenitore al momento della creazione di un'istanza. Questo metodo è valido sia per immagini presenti su Docker Hub che su Container Registry.

4.5.5 Gestione dei Costi di Compute Engine

Compute Engine offre due strumenti per fare un calcolo dei prezzi in maniera efficiente:

- Google Cloud Console,
  che fornisce una stima per le istanze mentre le si stanno creando
- il calcolatore dei prezzi di Google Cloud,
  che è uno strumento con cui si può ottenere una stima dei costi totali di un progetto, poiché considera tutte le risorse richieste nei vari servizi. Nell'esempio in Figura 4.15 è stimato il costo mensile sia di una macchina virtuale sia di uno spazio di archiviazione in *Cloud Storage*.

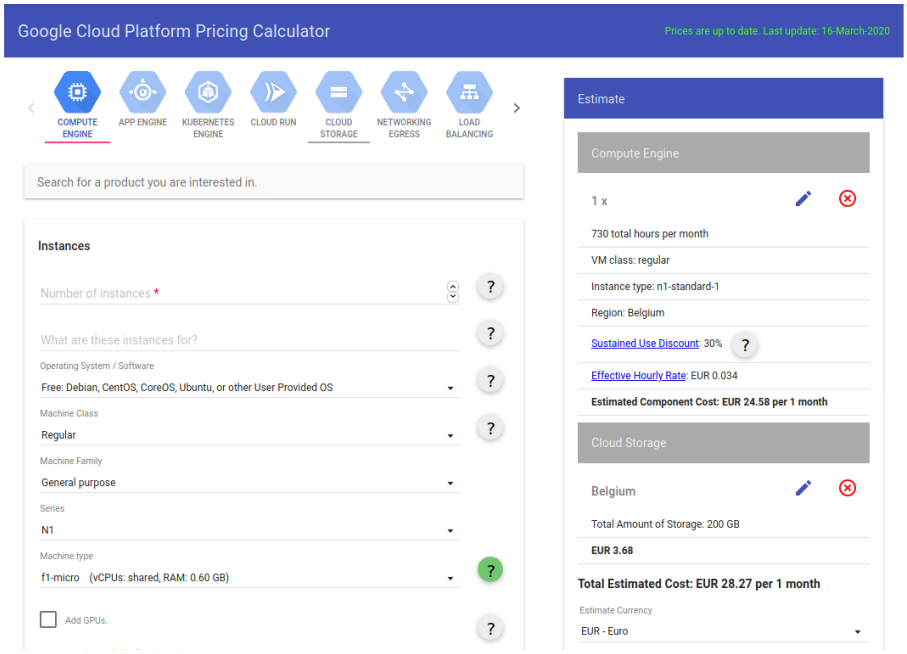


FIGURA 4.15: Calcolatore prezzi di Google Cloud

I costi variano in base alla zona scelta.

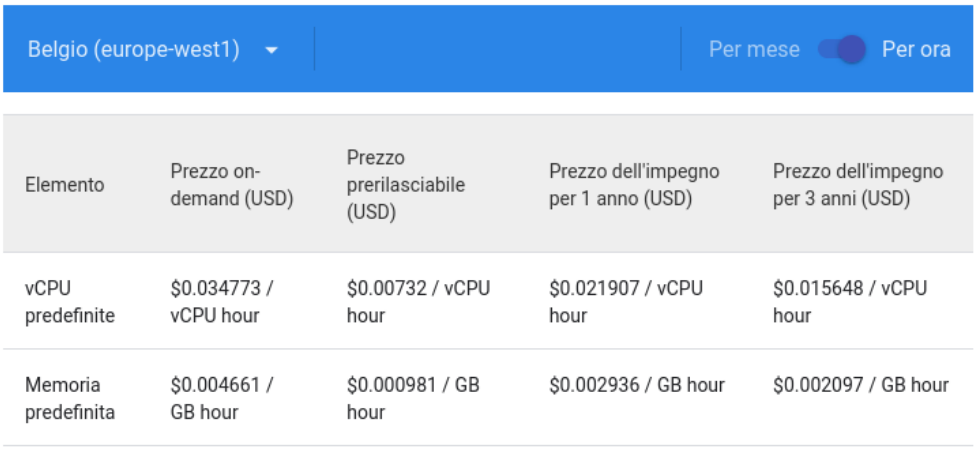
Prezzi delle macchine virtuali

Il modello di fatturazione valido per qualsiasi tipologia di risorsa (vCPU, GPU e RAM) prevede che venga addebitato un costo minimo di 1 minuto di utilizzo. Dopo 1 minuto, il costo delle istanze viene calcolato ad incrementi di 1 secondo.

In particolare, il tempo di attività di un'istanza è dato dal numero di secondi che sono passati dall'avvio all'arresto dell'istanza. Per arresto si intende quando l'istanza è in stato *TERMINATED*. Quindi se un'istanza è inattiva ma in stato di *RUNNING*, allora verrà comunque applicato un addebito. Per visualizzare in che stato attualmente si trovano le proprie macchine virtuali si può eseguire il comando:

```
gcloud compute instances list
```

I prezzi sono basati sulle risorse, ovvero ogni vCPU e ogni GB di memoria su Compute Engine vengono fatturati separatamente anziché visti come un unico insieme all'interno di una macchina. Ogni istanza appartiene a una serie (N1, N2, M1, C2 ecc.), che contraddistingue la specifica piattaforma CPU sulla quale è basata. Il costo varia in base alla serie selezionata.



Elemento	Prezzo on-demand (USD)	Prezzo prerilasciabile (USD)	Prezzo dell'impegno per 1 anno (USD)	Prezzo dell'impegno per 3 anni (USD)
vCPU predefinite	$0.034773 / vCPU hour	$0.00732 / vCPU hour	$0.021907 / vCPU hour	$0.015648 / vCPU hour
Memoria predefinita	$0.004661 / GB hour	$0.000981 / GB hour	$0.002936 / GB hour	$0.002097 / GB hour

FIGURA 4.16: Costo in dollari per vCPU e RAM all'ora per macchine di serie N1 in Belgio

La Google Cloud Console mette anche a disposizione un elenco di macchine standard aventi configurazioni di CPU virtuali e memoria RAM predefinite.

Tipo di macchina	CPU virtuali	Memoria	Prezzo (USD)	Prezzo prerilasciabile (USD)
n1-standard-1	1	3.75GB	$0.0523	$0.0110
n1-standard-2	2	7.5GB	$0.1046	$0.0220
n1-standard-4	4	15GB	$0.2092	$0.0440
n1-standard-8	8	30GB	$0.4184	$0.0880
n1-standard-16	16	60GB	$0.8368	$0.1760
n1-standard-32	32	120GB	$1.6736	$0.3520
n1-standard-64	64	240GB	$3.3472	$0.7040
n1-standard-96 Solo piattaforma Skylake	96	360GB	$5.0280	$1.0560

FIGURA 4.17: Costo all'ora delle tipologie di macchine standard N1

Oltre ai tipi di macchine per uso generale, Compute Engine offre diverse tipologie di istanze: macchine con memoria elevata, con CPU elevata, con vCPU e memoria personalizzate, con memoria ottimizzata, con core condivisi, e infine quelle ottimizzate per il calcolo.

Belgio (europe-west1) | Al mese | All'ora

Elemento	Prezzo on-demand (USD)	Prezzo prerilasciabile (USD)	Prezzo dell'impegno per 1 anno (USD)	Prezzo dell'impegno per 3 anni (USD)
vCPU personalizzate	$0.036489 / vCPU hour	$0.00768 / vCPU hour	$0.021907 / vCPU hour	$0.015648 / vCPU hour
Memoria personalizzata	$0.004892 / GB hour	$0.00103 / GB hour	$0.002936 / GB hour	$0.002097 / GB hour

FIGURA 4.18: Costo all'ora delle vCPU e RAM presenti in macchine personalizzate N1

Paesi Bassi (europe-west4) | Al mese | All'ora

Elemento	Prezzo on-demand (USD)	Prezzo prerilasciabile (USD)	Prezzo dell'impegno per 1 anno (USD)	Prezzo dell'impegno per 3 anni (USD)
vCPU ottimizzate per il calcolo	$0.03738 / vCPU hour	$0.00905 / vCPU hour	$0.02357 / vCPU hour	$0.01495 / vCPU hour
Memoria ottimizzata per il calcolo	$0.00501 / GB hour	$0.00122 / GB hour	$0.00316 / GB hour	$0.00200 / GB hour

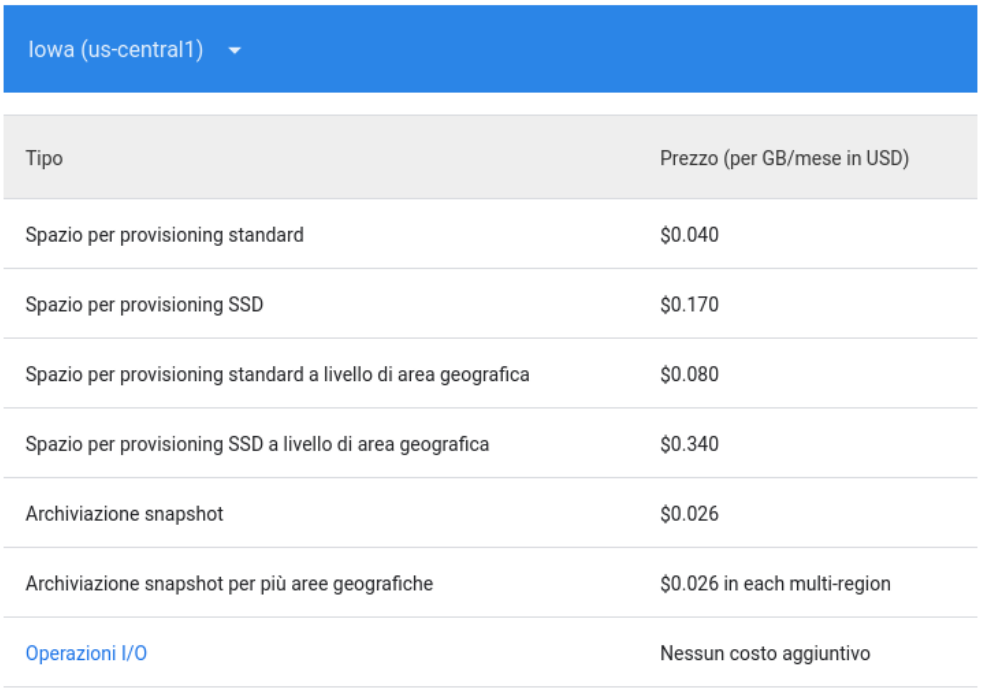
FIGURA 4.19: Costo all'ora delle vCPU e RAM per le macchine ottimizzate per il calcolo (serie C2)

Prezzi delle immagini

La maggior parte dei sistemi operativi disponibili non comportano costi aggiuntivi. Tuttavia le seguenti immagini premium comportano un costo che varia da pochi centesimi di dollaro a mezzo dollaro all'ora: Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Windows Server, SQL Server.

Prezzi dei dischi di archiviazione

A ogni istanza VM è associato almeno un disco (quello di avvio) avente un proprio costo di addebito. L'addebito legato ai dischi di memoria secondaria dipende dalla quantità dello spazio richiesto. Le operazioni di input/output sono incluse nel prezzo e le prestazioni degli spazi di archiviazione persistenti migliorano col l'aumentare del volume allocato.



Iowa (us-central1)

Tipo	Prezzo (per GB/mese in USD)
Spazio per provisioning standard	$0.040
Spazio per provisioning SSD	$0.170
Spazio per provisioning standard a livello di area geografica	$0.080
Spazio per provisioning SSD a livello di area geografica	$0.340
Archiviazione snapshot	$0.026
Archiviazione snapshot per più aree geografiche	$0.026 in each multi-region
Operazioni I/O	Nessun costo aggiuntivo

FIGURA 4.20: Costo in dollari al mese dei dischi persistenti in Belgio

Altri costi

Sono presenti costi di rete, che corrispondono a pochi centesimi per GB di dati trasferiti tra aree diverse (per esempio tra due aree geografiche in Europa si ha un addebito di $ 0,02 per GB).

Si hanno anche costi se si vogliono aggiungere GPU alle istanze, che variano da $0.416 per GPU a $1.80 per GPU all'ora (nella regione europe-west4, ovvero i Paesi Bassi) in base al modello di GPU scelto.

4.6 Cloud Storage

Nonostante sia possibile immagazzinare dati direttamente sulla memoria secondaria delle macchine virtuali allocate col servizio di Compute Engine, spesso è più utile salvare dati altrove, per esempio se si necessita di un maggiore spazio di archiviazione usufruibile anche da diverse risorse.

Per questo Google offre il servizio *Cloud Storage*, che consiste in un archivio di oggetti di qualunque tipo (chiamati anche *Blob*). È un servizio scalabile, quindi non va specificata la capacità totale in anticipo [10].

A ciascun oggetto è associato un *URL* univoco per poterlo visualizzare, scaricare o renderlo facilmente reperibile a diverse tecnologie Web. I Blob sono immutabili, quindi non vengono modificati, ma vengono create nuove versioni. Sono contenuti in *bucket*, ovvero contenitori identificati da un nome univoco globalmente, appartenenti a uno specifico progetto e situati in una zona. A quest'ultimi è conveniente assegnare una zona vicina agli utenti e alle macchine che dovranno utilizzare il servizio per ridurre la latenza.

Dal punto di vista della sicurezza i dati in transito da e verso Cloud Storage sono criptati con HTTPS. Normalmente l'accesso agli oggetti è gestito dallo IAM in Cloud, infatti i bucket ereditano i ruoli dal progetto in cui sono contenuti e gli oggetti a loro volta ereditano i ruoli dai bucket. Infine si può attivare l'*object versioning*, ovvero il controllo di versione degli oggetti per mantenere uno storico delle modifiche.

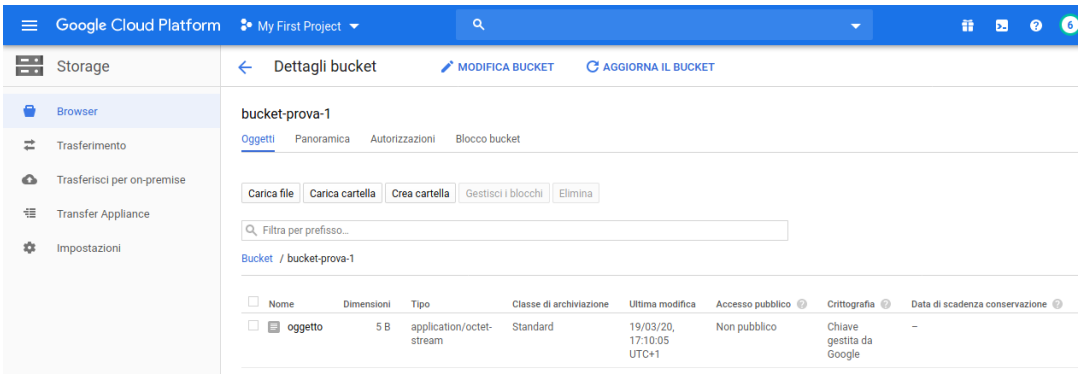


FIGURA 4.21: Bucket contenente un oggetto

Cloud Storage permette di scegliere tra quattro classi differenti di archiviazione: *Regional*, *Multi-regional*, *Nearline*, e *Coldline*. Le prime due rappresentano spazi di archiviazione di oggetti con alte prestazioni di lettura e scrittura, mentre le rimanenti sono ottimizzate per funzioni di backup.

Il tipo di *storage* Region consente di memorizzare i propri dati in una specifica regione di GCP, è più economico del modello Multi-regional, ma offre anche meno ridondanza, infatti quest'ultima opzione replica le informazioni in due zone distanti almeno 160 Km. Nearline è un esempio di storage a basso costo per archiviare dati ai quali si accede raramente, per esempio una volta al mese o anche meno. Infine lo spazio di archiviazione Coldline ha il costo minore in termini di conservazione dati, mentre ha la tariffa maggiore per quanto riguarda l'accesso ad essi. Quindi è perfetto per salvare dati di backup, reperiti però con una frequenza molto bassa di circa una volta all'anno.

4.6.1 Gestione dei Costi di Cloud Storage

Per quanto riguarda i prezzi, tutte le classi di archiviazione comportano un costo per gigabyte di dati archiviati al mese, con il Multi-regional storage avente il prezzo di conservazione più alto (€0.0331452 per GB al mese), Region (€0.018414 per GB al mese), Nearline (€0.009207 per GB al mese) e Coldline il più basso (€0.0036828 per GB al mese).

Oltre a questi addebiti, Nearline e Coldline, essendo spazi di archiviazione di tipo backup, comportano un costo anche per il recupero dei dati (rispettivamente di €0.009207 per GB al mese e €0.018414 per GB al mese). Infine sono presenti altri costi per operazione e per costo di rete, ma considerati irrisori se si considera il totale.

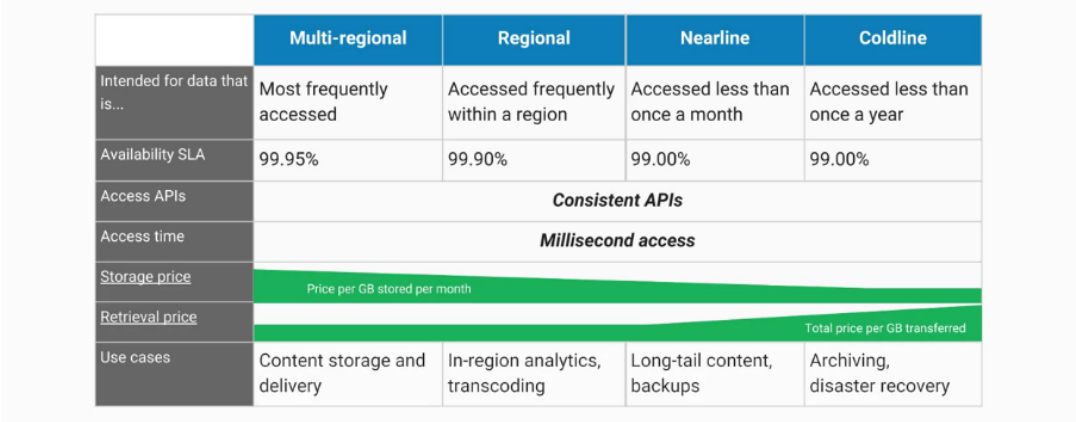


FIGURA 4.22: Confronto delle quattro classi di archiviazione

4.7 Container Registry

Container Registry è uno strumento offerto dalla piattaforma in cloud di Google che consiste in un registro privato, compatibile con Docker, dal quale si possono prelevare o caricare immagini di container.

Per poter caricare un'immagine Docker su questo servizio occorre innanzitutto creare un'immagine sulla macchina locale, per esempio mediante un Dockerfile eseguendo il seguente comando nella directory dove è presente il Dockerfile:

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

Dopo aver generato l'immagine è necessario configurare Docker affinchè possa utilizzare lo strumento a linea di comando *gcloud*:

```
gcloud auth configure-docker
```

Prima di poter effettuare l'upload di un'immagine, occorre contrassegnare quest'ultima con il nome del registro, ovvero la posizione nella quale la si vuole sistemare:

```
docker tag [image-name] [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

Gli ultimi tre paramentri corrispondono rispettivamente alla posizione dell'host (ad es. eu.gcr.io per l'Europa), all'identificatore del progetto che conterrà l'immagine e al nome dell'immagine che sarà visualizzato in cloud.

Infine per caricare la Docker Image su Container Registry è sufficiente eseguire il comando:

```
docker push [HOSTNAME]/[PROJECT-ID]/[IMAGE]
```

4.8 Machine Learning su GCP

Su Google Cloud Platform sono presenti diverse soluzioni con le quali è possibile eseguire algoritmi di machine learning. In questa Sezione ne vengono descritti alcuni.

4.8.1 Cloud AutoML

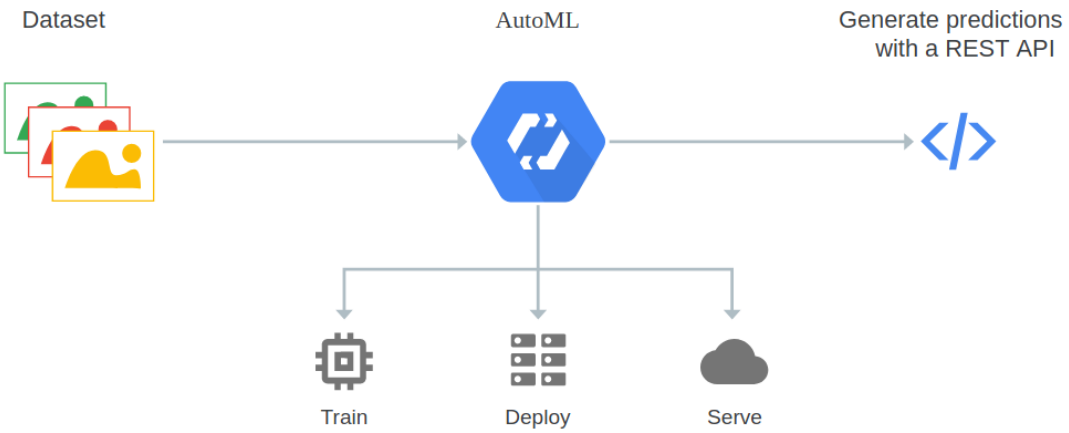


FIGURA 4.23: Funzionamento di Cloud AutoML

Cloud AutoML è un servizio che consente a sviluppatori con esperienza limitata nel campo del machine learning di addestrare modelli automatici specifici per le loro esigenze aziendali [9]. In particolare sono disponibili diversi prodotti con cui creare (ed effettuare il deployment) in modo automatico modelli di machine learning:

- **Natural Language** mette a disposizione tre differenti approcci per lavorare su documenti testuali: Classification, Entity Extraction e Sentiment Analysis. Il primo permette di risolvere problemi di classificazione su un set documenti etichettati. Utilizzando Entity Extraction è possibile identificare un insieme di entità in un testo scritto in lingua inglese.

Con Sentiment Analysis si riescono ad estrapolare opinioni da un testo scritto sempre in inglese (*sentiment analysis*).

- **Tables** consente di costruire modelli di machine learning su dati strutturati per problemi di regressione e di classificazione
- **Translation** offre la possibilità di creare i propri modelli di traduzione personalizzati in modo che le query di traduzione restituiscano risultati specifici per il proprio dominio.
- **Vision** è composto dai sottoprodotti Classification e Object Detection. Il primo serve per classificare le immagini in base alle etichette definite, mentre l'ultimo è fondamentale per la rilevazione ed estrazione di diversi oggetti dalle immagini e inoltre fornisce informazioni su ogni singolo oggetto rilevato, inclusa la sua posizione all'interno dell'immagine.
- **Video Intelligence** comprende i servizi di Classification e Object Tracking, i quali consentono di addestrare modelli di machine learning rispettivamente per classificare riprese nei video in base alle proprie etichette definite; e per seguire oggetti specifici in degli intervalli di tempo nei video.

Prendendo come riferimento AutoML Tables, per generare modelli di machine learning in modo automatico, su dati strutturati (senza dover scrivere una singola linea di codice), è sufficiente utilizzare l'interfaccia grafica web. Prima di tutto è necessario importare un dataset tramite dei file in formato *csv*, oppure utilizzando il servizio *BigQuery*.

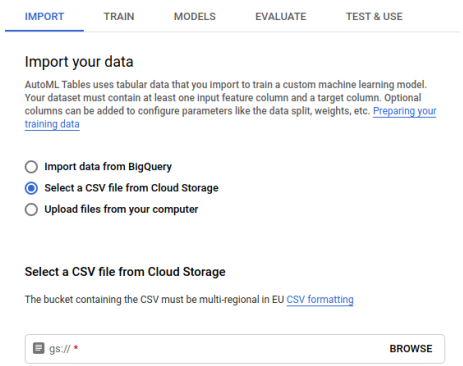


FIGURA 4.24: Import del dataset

Successivamente si può selezionare lo schema dei dati (nomi e tipologia delle feature ecc.) e la variabile da predire.

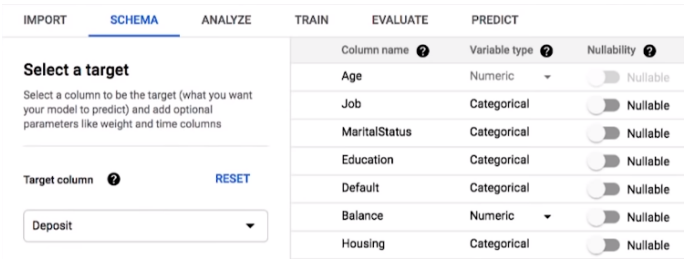


FIGURA 4.25: Definizione dello schema del dataset

Dopodiché verranno fornite delle statistiche sulle feature del dataset.

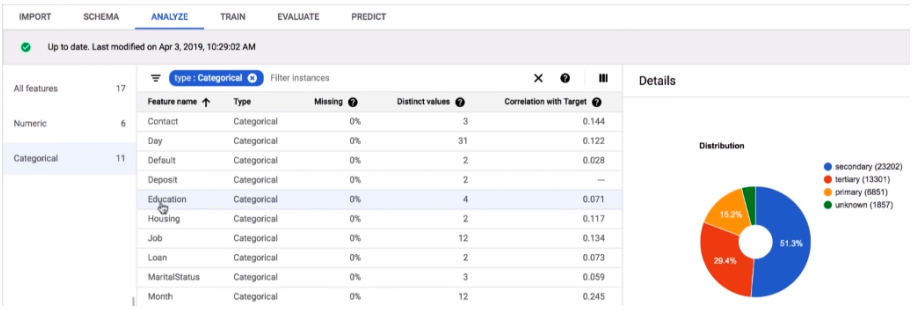


FIGURA 4.26: Analisi statistiche sui dati

A questo punto si esegue il training del modello, specificando il massimo numero di ore per l'esecuzione).

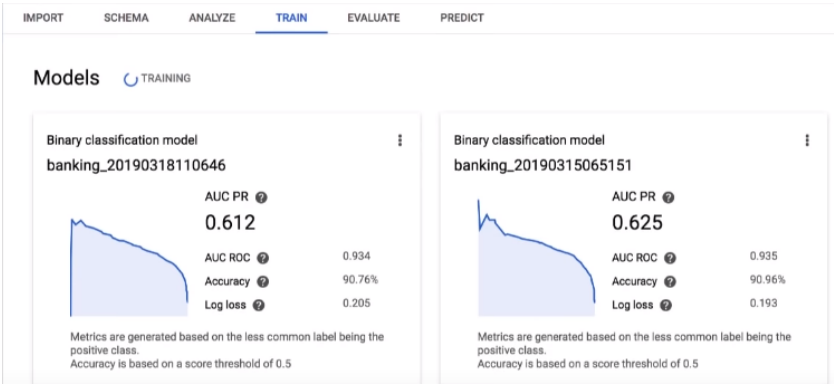


FIGURA 4.27: Esecuzione del training automatico del modello

Quando il training del modello è stato completato, saranno riportate diverse metriche di valutazione (ad es. F1 score , accuratezza, precision, recall e matrice di confusione, in caso di classificazione) per visualizzare le prestazioni del modello e il contributo che dà ogni singola feature.

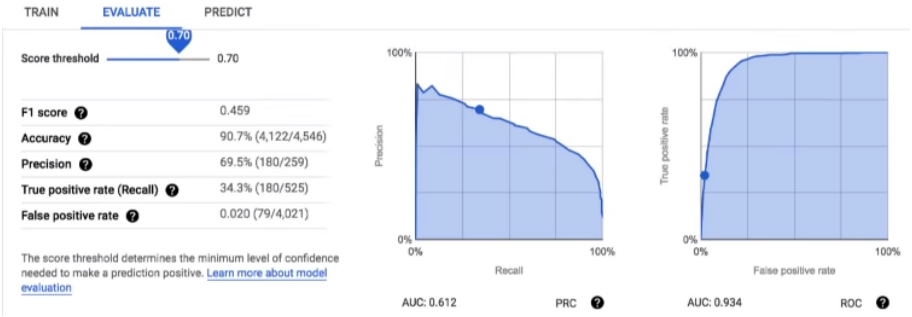


FIGURA 4.28: Grafici e metriche di valutazione del modello

Infine si può effettuare il deployment del modello e in seguito si possono generare delle predizioni direttamente mediante l'interfaccia grafica o con chiamate a delle REST API.

4.8.2 BigQuery ML

BigQuery ML è un servizio col quale creare ed eseguire velocemente modelli di machine learning utilizzando query SQL standard. Supporta algoritmi di regressione lineare, regressione logistica, XGBoost e altri [5].

Prima di tutto va caricato un dataset su BigQuery, sul quale si possono modificare e interrogare dati.

In seguito si esegue training (generazione) del modello col seguente comando SQL:

```
1 CREATE MODEL 'dataset_name.model_name'
2 OPTIONS(model_type='logistic_reg') AS
3 SELECT ...
```

I diversi parametri sono:

- *dataset_name* è il nome del set di dati sul quale creare il modello.
- *model_type* è il tipo di modello, ovvero l'algoritmo che si vuole utilizzare per risolvere il problema.
- dopo *SELECT* va specificata la query che restituirà il sottoinsieme di dati che saranno utilizzati per svolgere il training.

Dopo che l'esecuzione del training ha terminato è possibile valutare le prestazioni del modello:

```
1 SELECT * FROM
2 ML.EVALUATE (MODEL 'dataset_name.model_name')
```

Con questo comando sono riportate le principali metriche di valutazione del modello, come ad esempio precision, recall e F1 score se si ha un problema di classificazione.

Infine si può utilizzare il modello generato per effettuare previsioni su dati sconosciuti:

```
SELECT * FROM
ML.PREDICT (MODEL 'dataset_name.model_name')
AS prediction
```

4.8.3 Modelli custom su VM gestite manualmente

Un'altra modalità che può essere sfruttata per generare un modello di machine learning sulla piattaforma in cloud di Google è quella di far svolgere questo compito a delle macchine viruali allocate manualmente sul servizio Compute Engine (Sottosezione 4.5.1).

Quando queste istanze vengono create, internamente non contengono nessun file, e siccome scrivere da zero del codice per generare un modello di machine learning ogni volta che si crea una VM richiederebbe troppo tempo, ha più senso effettuare l'upload del codice direttamente in un bucket su Cloud Storage.

Questo codice può essere poi scaricato sulla macchina da riga di comando collegandosi ad essa tramite collegamento ssh, può essere reperito in modo automatico dall'istanza specificando dei comandi da farle eseguire in uno script di avvio, o in alternativa la VM può effettuarne il download mediante l'esecuzione di alcuni comandi svolti da un container (allocato al suo interno) per mezzo del suo Dockerfile.

Per generare il modello di machine learning sarà poi necessario far eseguire alla macchina il codice appena scaricato.

4.8.4 Tensorflow su GCP

Sulla piattaforma in cloud di Google si possono creare modelli di machine learning da cima a fondo utilizzando la libreria open-source *Tensorflow* all'interno di notebook generati su *JupyterLab*, i quali vengono eseguiti da delle macchine virtuali di Google Cloud. Jupyter Notebook è un'applicazione web open source che consente di creare e condividere documenti che contengono codice, equazioni, grafici e testo.

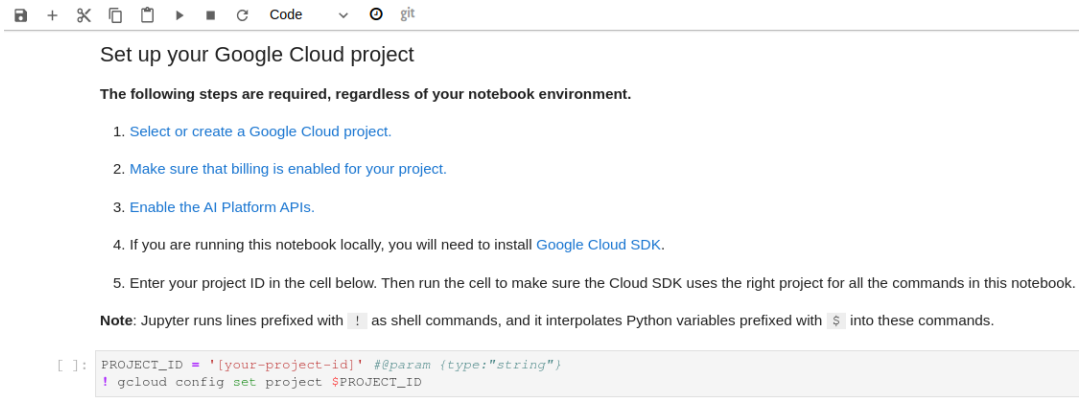


FIGURA 4.29: JupyTer Notebook su GCP

Con l'utilizzo di Tensorflow su GCP è possibile generare modelli custom senza dover gestire manualmente le risorse (VM, dischi di archiviazione, configurazione delle GPU e delle librerie ecc.).

Capitolo 5

Benchmarking

In questo Capitolo viene svolta una vera e propria analisi comparativa. Si procede per step spiegando innanzitutto nella Sezione 5.1 quali sono gli obiettivi di questa analisi. Nella Sezione 5.2 è descritto il caso d'uso considerato per il *benchmarking*. Successivamente la Sezione 5.3 mostra come si raggiungeranno a livello tecnico gli obiettivi proposti. Le due Sezioni 5.4 e 5.5 spiegano rispettivamente come è stato possibile automatizzare tutto il procedimento del caso d'uso e come sono stati analizzati i risultati finali ottenuti. Infine, nella Sezione 5.6 vengono effettuate delle considerazioni aggiuntive sul confronto tra GCP e cloud tradizionale in termini di servizi offerti, vincoli orari di utilizzo minimo e flessibilità.

5.1 Obiettivi

I vari obiettivi da raggiungere nell'analisi dei costi e dei benefici di algoritmi di machine learning su Google Cloud Platform sono:

- valutare i costi e i tempi di esecuzione dell'algoritmo in base al tipo di macchina utilizzato
- confrontare i costi e i tempi delle VM offerte da GCP rispetto ad un cloud tradizionale. Con cloud tradizionale si intende un cloud privato gestito, di dimensioni modeste (rispetto ai principali cloud provider come Google, Amazon, Microsoft ecc), completamente gestito da un fornitore esterno. Esso non fornisce delle API con le quali è possibile accedere ai servizi e gestire le risorse
- capire quale tipologia di macchina conviene utilizzare in base alle proprie esigenze

5.2 Use Case

Il caso d'uso scelto sul quale raggiungere gli obiettivi appena descritti è l'esecuzione del *tuning* di un modello di forecasting di serie temporali storiche

tramite un progetto in Python fornitomi dall'azienda. Le serie temporali fanno riferimento alla domanda termica e alla temperatura. Come meccanismo per vedere come scala l'algoritmo è stata utilizzata una *grid search*.

Grid search è il metodo tradizionale di ottimizzazione degli iperparametri. È una ricerca esaustiva che viene eseguita su un sottoinsieme specificato manualmente dello spazio iperparametrico di un algoritmo di apprendimento. Le grid search sono poco performanti quando ci sono molti parametri, perché provano tutte le combinazioni a prescindere. Essa è facilmente parallelizzabile, poiché esegue sempre ogni combinazione indipendentemente dai risultati intermedi [17]. Una grid search addestra un numero di modelli pari alla cardinalità del prodotto cartesiano degli insiemi di valori dei vari iperparametri. Infine restituisce il modello a cui sono associati gli iperparametri che minimizzano la funzione d'errore. Il modello è anche noto come *estimator* [18].

La grid search è un metodo in certi versi limitato, poiché potrebbe non trovare mai l'ottimo globale e perché il numero di valutazioni della funzione cresce esponenzialmente con il numero di dimensioni. D'altro canto essa è più che adatta per questo caso, siccome è semplice da implementare, permette di controllare il numero di core utilizzati ed è parallelizzabile. Non sono stati utilizzati altri metodi più efficienti, poiché avrebbero aggiunto complessità al benchmarking e perché l'obiettivo non è ottimizzare l'esecuzione dell'algoritmo, ma è quello di testare l'esecuzione di algoritmi di machine learning in cloud per poterne fare un'analisi dei costi e benefici.

I metodi di ottimizzazione Bayesiana sono delle alternative alla grid search molto usate. Gli algoritmi di machine learning richiedono spesso un'attenta regolazione degli iperparametri del modello, dei termini di regolarizzazione e dei parametri di ottimizzazione. Sfortunatamente, queste operazioni richiedono molta esperienza, regole pratiche non scritte o, a volte, una ricerca con forza bruta. L'ottimizzazione Bayesiana consente di sviluppare approcci automatici in grado di ottimizzare le prestazioni di un dato algoritmo di apprendimento per una specifica attività da svolgere [24].

Anche questi metodi necessitano di sapere qual è lo spazio degli iperparametri da testare. Però, a differenza della grid search, non occorre specificare tutte le singole possibilità da sperimentare, dato che hanno semplicemente bisogno di un intervallo di valori per ogni iperparametro (ad es. da 0 a 10). Sarà poi questa ottimizzazione a trovare il giusto valore in quell'intervallo provando varie combinazioni in modo automatico.

Questi metodi sono più efficienti della grid search e possono portare a risultati che superano le prestazioni di un esperto nel campo dell'ottimizzazione degli algoritmi di machine learning [24]. Un esempio di framework di ottimizzazione degli iperparametri open source per automatizzare la ricerca degli iperparametri è *Optuna* [2].

Lo scopo del caso d'uso analizzato è quello di prevedere la domanda termica, in particolare è quello di fare il tuning e scegliere il modello con l'errore minore. Il tuning genera 7875 modelli (esegue 7875 operazioni di fit). Questo numero è dato dalla cardinalità del prodotto cartesiano degli insiemi che contengono i valori assunti dagli iperparametri presenti nella grid search (Script 5.1). Ciascun modello viene creato a partire da un'unica combinazione (tra tutti i modelli testati) di valori degli iperparametri. Infine la grid search si occupa di scegliere il modello a cui sono associati gli iperparametri che minimizzano l'errore.

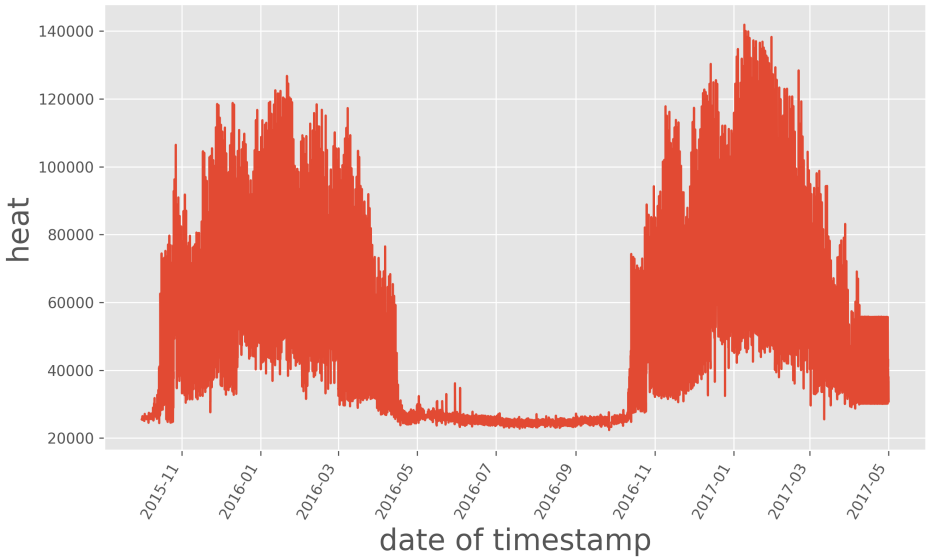


FIGURA 5.1: Serie storica della domanda termica

I due grafici nelle Figure 5.1 e 5.2 mostrano rispettivamente l'andamento della domanda termica e della temperatura nell'intervallo di tempo considerato, ovvero il periodo compreso tra il primo Ottobre 2015 e il 30 Aprile 2017. Ciascuno dei 13872 dati è caratterizzato da un valore (domanda termica nel grafico 5.1, o temperatura nel grafico 5.2) ed un *timestamp* composto da una data e un orario. Per questioni di leggibilità nelle marche temporali dei due grafici sono riportati solamente il mese e l'anno. La domanda di calore è nettamente maggiore nei mesi invernali rispetto a quelli estivi. La temperatura ha un trend che è esattamente l'opposto a quello della domanda termica, e raggiunge il suo picco nel mese di Luglio con 37.1 °C.

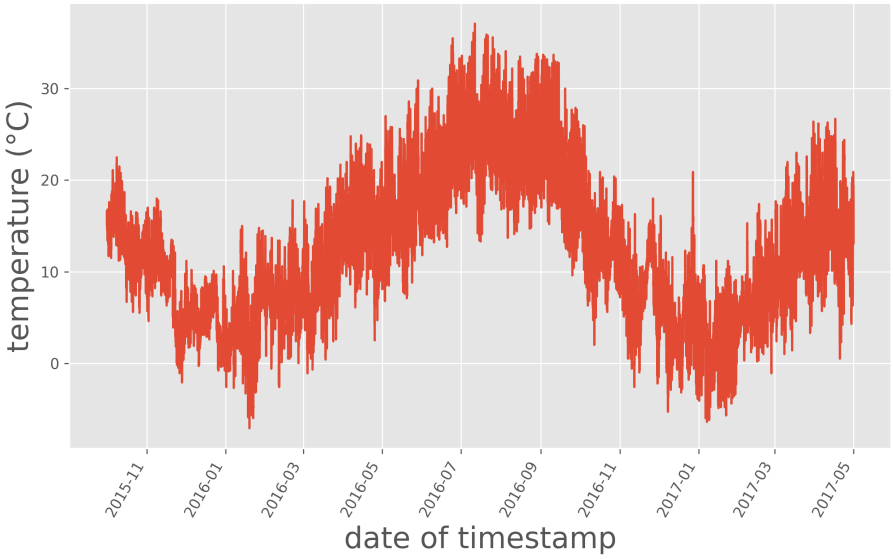


FIGURA 5.2: Serie storica della temperatura

L'esecuzione del tuning è stata svolta utilizzando l'algoritmo *XGBoost* (*Extreme Gradient Boosting*), che è usato per risolvere problemi di supervised learning (apprendimento supervisionato, quando i dati sono già etichettati).

XGBoost utilizza insiemi di alberi di classificazione e regressione (*CART*, *classification and regression trees*) [25]. In Figura 5.3 è mostrato un semplice esempio di un CART che classifica se a qualcuno piacerà un ipotetico gioco *X* per computer basandosi sull'età delle persone fornitegli in input.

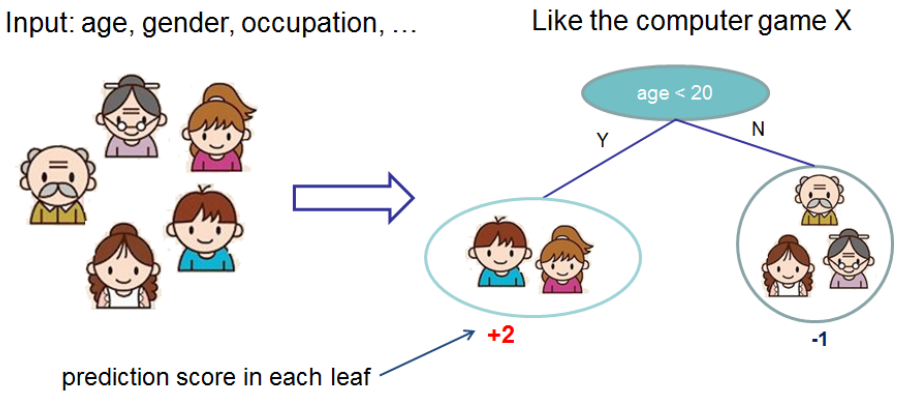


FIGURA 5.3: Esempio di CART

Solitamente però un singolo albero non è abbastanza potente per essere usato nella pratica. Ciò che viene effettivamente utilizzato è un insieme di più alberi, e quindi si considera la somma delle previsioni di ciascun CART. In Figura 5.4 è presente un esempio di un insieme di due alberi (il secondo introduce

il vincolo "utilizza il computer quotidianamente"). Lo score di previsione di ogni singolo individuo è dato dalla somma dei punteggi a lui associati da ciascun albero.

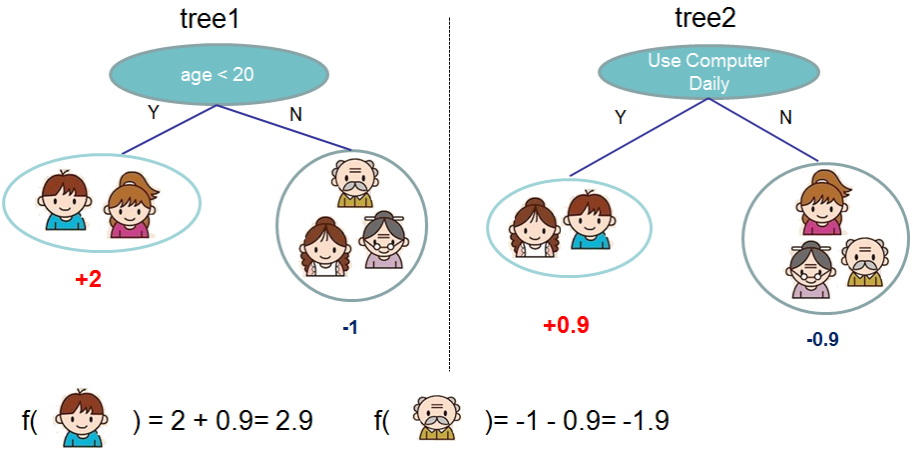


FIGURA 5.4: Somma delle predizioni di più CART

Dal punto di vista matematico si può rappresentare il modello con la Formula 5.1:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{5.1}$$

dove K è il numero degli alberi (in questo caso due), f è una funzione nello spazio di funzioni F, ed F è l'insieme di tutti i possibili CART [8].

L'esecuzione del tuning descritto in precedenza è un problema di regressione, poiché lo scopo è quello di predire una variabile continua (domanda termica). Si è fatto uso dell'implementazione di XGBoost per la regressione, denominata *XGBRegressor*. I parametri della classe XGBRegressor sono contenuti nel dizionario *params* all'interno della grid search (Listing 5.1), e sono stati testati con varie combinazioni di valori. Essi sono [26]:

- **objective**: indica l'attività di apprendimento e il corrispondente obiettivo di apprendimento
- **colsample_bytree**: specifica la frazione di colonne che vengono utilizzate come sotto-campione durante la costruzione di ogni albero.
- **gamma**: definisce la minima riduzione di perdita richiesta per realizzare un'ulteriore partizione su un nodo foglia dell'albero. Maggiore è gamma, più l'algoritmo sarà conservativo.

- **learning_rate (eta)**: questo parametro, nel gradient boosting, ha lo scopo di ridurre l'effetto di ogni albero aggiuntivo sul modello. Viene identificato anche come *shrinkage* (restringimento), ed è un metodo aggiuntivo per prevenire l'*overfitting* (adattamento eccessivo ai dati di training). Minore è eta, più robusto sarà il modello nel prevenire l'overfitting, ma maggiore sarà il tempo computazionale richiesto.
- **max_depth**: è la profondità massima di un albero decisionale, intesa come lunghezza del percorso più lungo da una radice a una foglia. L'aumentare di questo valore renderà il modello più complesso e maggiormente soggetto a overfitting. XGBoost consuma grandi quantitativi di memoria durante l'allenamento di un albero profondo.
- **n_estimators**: numero di alberi generati dall'algoritmo XGBoost.
- **subsample**: indica la frazione delle istanze di training da campionare in modo casuale per ogni albero. Valori più bassi prevengono l'overfitting, ma valori troppo piccoli potrebbero portare a underfitting.

```
xgb_model = xgb.XGBRegressor(objective="reg:squarederror")

    params = {
        "colsample_bytree": [0.1, 0.3, 0.5, 0.7, 0.9],
        "gamma": [0, 0.25, 0.5, 0.75, 1, 1.5, 2],
        "learning_rate": [0.01, 0.05, 0.1, 0.15, 0.2],
        "max_depth": [2, 3, 4, 5, 6],
        "n_estimators": [50, 100, 150],
        "subsample": [0.8, 0.9, 1]
    }

    search = GridSearchCV(xgb_model, param_grid=params, cv=3,
        n_jobs=-1 , return_train_score=True, verbose=1)
```

LISTING 5.1: Parametri utilizzati nella grid search e nell'algoritmo di XGBoost

I parametri passati come argomento alla grid search (Script 5.1) sono [15]:

- **estimator**: è il modello, in questo caso corrisponde alla variabile *xgb_model*. L'estimator deve fornire una funzione di score oppure tale funzione deve essere fornita come parametro.
- **param_grid**: è il dizionario descritto precedentemente, contenente nomi di parametri come chiavi e array di valori come valori. Consente di analizzare qualsiasi combinazione di valori dei parametri.
- **cv**: determina la strategia di suddivisione della *cross validation*. L'assegnazione di un numero intero a questo parametro specifica il numero di *fold* in un k fold stratificato.

- **n_jobs**: definisce il numero di core da utilizzare in parallelo. Il valore -1 ha il significato di usufruire di tutti i core disponibili nella macchina di elaborazione.
- **return_train_score**: questo parametro stabilisce se riportare o meno gli score di training. Il calcolo degli score di training viene utilizzato per ottenere informazioni su come le diverse impostazioni dei parametri influiscono sul compromesso tra overfitting/underfitting.
- **verbose**: controlla la verbosità dei messaggi dati in output: più è grande, maggiore sarà il numero di messaggi restituiti.

5.3 Setup Esperimenti

In questa Sezione viene descritto come si raggiungeranno gli obiettivi proposti nella Sezione 5.1.

In generale sono stati esaminati vari fattori. Innanzitutto, per eseguire il tuning, si è preso in considerazione solo un sottoinsieme di tutte le tipologie di macchine descritte nella Sottosezione 4.5.2, contenente le versioni high-cpu delle macchine virtuali *General-purpose* di GCP (n1, n2, n2d, e2) e le risorse ottimizzate per il calcolo (c2), ottime per carichi di lavoro *compute-intensive*. Invece si è deciso di escludere le VM *memory-optimized*, perché più adatte a carichi di lavoro ad alta intensità di memoria.

Uno degli obiettivi è capire quale tipologia di macchina conviene utilizzare per svolgere il tuning, quindi servirà avere una configurazione con un numero crescente di core della CPU per ogni risorsa esaminata. Le vCPU (intese come core in GCP) dovranno essere confrontabili, vale a dire tutte su una stessa scala. In particolare si è scelto di sperimentare l'esecuzione con i numeri di core compresi nell'insieme {2, 4, 8, 16, 32}. Non tutte le VM messe a disposizione dal Cloud di Google hanno le stesse configurazioni, ma la similarità è comunque alta. Di seguito è riportata la lista delle macchine sulle quali far eseguire il tuning.

machine type	instance type	zone	instance	vCpu	RAM (GB)
n1	high-cpu	europe-west1-b (Belgium)	n1-highcpu-2	2	1.8
			n1-highcpu-4	4	3.6
			n1-highcpu-8	8	7.2
			n1-highcpu-16	16	14.4
			n1-highcpu-32	32	28.8
n2	high-cpu	europe-west1-b (Belgium)	n2-highcpu-2	2	2.0
			n2-highcpu-4	4	4.0
			n2-highcpu-8	8	8.0
			n2-highcpu-16	16	16.0
			n2-highcpu-32	32	32.0
n2d	high-cpu	europe-west4-c (Netherlands)	n2d-highcpu-2	2	2.0
			n2d-highcpu-4	4	4.0
			n2d-highcpu-8	8	8.0
			n2d-highcpu-16	16	16.0
			n2d-highcpu-32	32	32.0
e2	high-cpu	europe-west1-b (Belgium)	e2-highcpu-2	2	2.0
			e2-highcpu-4	4	4.0
			e2-highcpu-8	8	8.0
			e2-highcpu-16	16	16.0
c2	standard	europe-west1-b (Belgium)	c2-standard-4	4	16.0
			c2-standard-8	8	32.0
			c2-standard-16	16	64.0
			c2-standard-30	30	120.0

TABELLA 5.1: Lista delle macchine di Google che eseguiranno il tuning

Si effettueranno tanti esperimenti quante sono le configurazioni disponibili in termini di core (fino a 32) delle tipologie di macchine scelte.

Per ottenere le performance delle macchine di Google si dovranno misurare il tempo per eseguire il tuning e la durata totale di esecuzione, cioè il tempo di tuning sommato al periodo di creazione e distruzione delle risorse in cloud. La scelta di calcolare anche la durata solo del tuning permette di sapere con precisione il tempo di vita della sola macchina virtuale, e quindi di avere una maggiore precisione sul calcolo dei prezzi. Perciò, per sapere quanto costano le macchine di Google, per ciascuna istanza verrà calcolato il costo in euro del tempo di tuning (sia con la macchina normale che preemptible) e il costo orario col calcolatore di prezzi di Google Cloud.

Ciascuna risorsa verrà allocata in Europa per ridurre al minimo il throughput. In particolare verranno tutte distribuite in Belgio tranne quelle di tipo n2d, perché disponibili solo in Olanda.

Infine, per confrontare i costi e i tempi delle VM fornite dalla piattaforma in cloud di Google rispetto ad un cloud tradizionale, il tuning verrà eseguito

anche su delle macchine appartenenti a quest'ultimo, testandole sempre con i numeri di core compresi nell'insieme {2, 4, 8, 16, 32}. Inoltre si svolgeranno dei test su delle macchine del cloud tradizionale aventi le stesse caratteristiche della serie c2 di Compute Engine, così da poter confrontare i costi del cloud tradizionale anche con la serie c2. Infatti questa tipologia ottimizzata per lavori compute-intensive ha dei valori di ram molto più alti rispetto tutti gli altri tipi di VM.

machine type	vCpu	RAM (GB)
traditional cloud	2	2
	4	4
	8	8
	16	16
	32	32
traditional cloud with c2 configuration	4	16
	8	32
	16	64
	30	120

TABELLA 5.2: Lista delle macchine di un cloud tradizionale che eseguiranno il tuning

5.4 Framework di Automatizzazione

In questa Sezione verrà descritto come è stato possibile automatizzare la generazione del modello di *Machine Learning* per fare previsione della domanda di calore utilizzando la containerizzazione sulla piattaforma di Google e un ulteriore script Python eseguibile in locale. Si vuole far generare questo modello di machine learning da parte di un container racchiuso in una macchina virtuale remota in cloud, e poi scaricarlo in locale tramite un unico script di Python eseguito sulla propria macchina.

Gli obiettivi sono:

- *automatizzare l'intero processo* in modo tale che sia richiesta l'esecuzione di un solo comando da terminale
- *minimizzare i costi* e quindi eliminare le risorse (bucket e VM) utilizzate non appena hanno portato a termine il compito

Di seguito sono riportate le varie operazioni svolte dallo script Python principale eseguito in locale:

1. Autenticazione per utilizzare le API di Google Cloud in Python
2. Creazione di un bucket su Cloud Storage

3. Creazione della macchina virtuale su Compute Engine contenente un software container, creato ed eseguito mediante un Dockerfile
4. Caricamento dell'intera cartella contenente il progetto di machine learning su Cloud Storage. Questa fase è eseguita mentre il container è in fase di costruzione. Subito dopo che è stato fatto l'upload del progetto di machine learning, il Docker container dovrà svolgere i seguenti compiti:
   (a) scaricare il progetto che è stato appena salvato in cloud dal pc locale
   (b) eseguire l'algoritmo di machine learning, quindi generare un modello dall'algoritmo stesso
   (c) effettuare l'upload del modello su Cloud Storage, in modo tale che lo script principale possa poi ottenerlo in locale
5. Download del modello (risultato) in locale
6. Cancellazione della macchina virtuale
7. Cancellazione del bucket

Nella Figura 5.5 sono illustrate le interazioni che avvengono tra la macchina locale e i servizi di Google Cloud utilizzati.

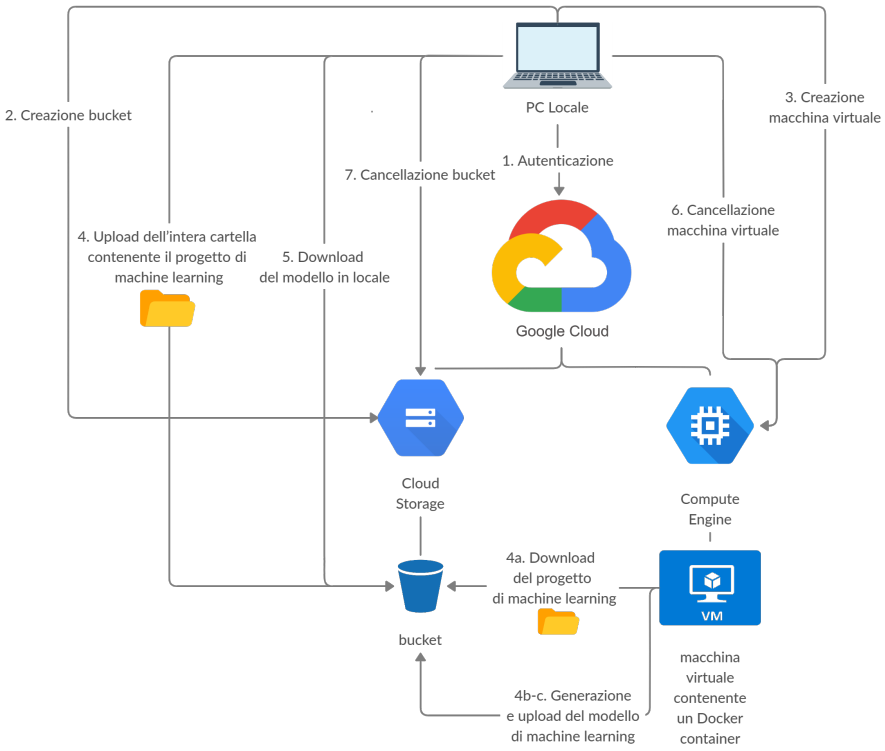


FIGURA 5.5: Comunicazioni tra la macchina locale e Google Cloud

Autenticazione per utilizzare le API di Google Cloud in Python

Per poter utilizzare le API di Google Cloud Platform è necessario autenticarsi col proprio account di servizio. Per questo dovrà essere contenuto un riferimento al percorso del file *json* contenente le credenziali di autenticazione. Il codice riportato nello Script 5.2 crea un *client* a cui sono associate le credenziali per usufruire delle API di Cloud Storage.

```
# Creation of the client to execute operations in Google Cloud
    Storage
storage_client = storage.Client.from_service_account_json(
    json_credentials_file)
```

LISTING 5.2: Autenticazione eseguita mediante un file json

Creazione di un bucket su Cloud Storage

Dopo essersi identificati si procede alla creazione del bucket con la funzione *create_bucket* (Script 5.3), alla quale vengono passati come parametri il nome del bucket (univoco globalmente) e lo *storage client* creato precedentemente.

```
# Creates a new bucket in Google Cloud Storage service
def create_bucket(bucket_name, storage_client):
    """Creates a new bucket."""
    # bucket_name = "your-new-bucket-name"

    bucket = storage_client.bucket(bucket_name)
    bucket.create(location="EU")

    print("Bucket {} created".format(bucket.name))
```

LISTING 5.3: Funzione che crea un bucket su Cloud Storage

Creazione della macchina virtuale su Compute Engine contenente un software container

Si genera la macchina virtuale con la funzione *create_instance* (Script 5.4) passando come parametri l'identificatore del progetto, il nome e la zona dove la si vuole creare. Nella definizione della funzione si specifica l'intera configurazione della macchina: il tipo di macchina, il disco di avvio, e le ulteriori proprietà già discusse nella Sottosezione 4.5.1. Ad esempio, nel codice sottostante viene selezionato il sistema operativo Container-Optimized OS e l'immagine Docker creata a partire dal Dockerfile scritto interamente per questo progetto, e caricata su Container Registry.

```
# Creates a new instance with a container in Compute Engine
    service
def create_instance(compute, project, zone, name):
    # Get the latest Container-Optimized OS image.
```

```
image_response = compute.images().getFromFamily(
    project='cos-cloud', family='cos-stable').execute()
..
      "items": [
        {
          "key": "gce-container-declaration",
          # image of the container
          "value": "spec:
          containers:
          - name: instance-6
          image: 'eu.gcr.io/project-name/tune:latest'
..
        }
```

LISTING 5.4: Funzione che genera una macchina virtuale su Compute Engine

La chiave *image* specifica il percorso all'immagine Docker su Container Registry che sarà utilizzata per creare il container (il nome del progetto non è stato specificato per motivi di privacy).

Caricamento dell'intera cartella contenente il progetto di machine learning su Cloud Storage

Mentre il container è in fase di creazione, il pc locale starà caricando su Cloud Storage l'intera cartella del progetto di machine learning, la quale contiene il codice da far eseguire al Docker container. In seguito la macchina locale attenderà il completamento dell'esecuzione del container controllando ad intervalli periodici (polling) se è stato caricato in cloud il modello di machine learning.

Costruzione del Docker container

La costruzione del software container avviene mentre la macchina locale sta compiendo l'upload del progetto in cloud (fase 4). È stato scritto un intero Dockerfile che potesse contenere tutti i comandi per poter costruire l'immagine Docker e metterla in esecuzione.

La creazione del container parte da un'immagine già creata e disponibile su *Docker Hub*, la quale installa *anaconda*, che è una distribuzione gratuita e open source di Python per il calcolo scientifico che mira a semplificare il deployment, la gestione dei pacchetti e delle librerie [3]. Successivamente viene creata una cartella sulla quale lavorare e ci si sposta dentro. Dopodiché vengono installati vari componenti software (ad es. Google Cloud SDK) e con lo strumento *gsutil* si scarica la cartella contenente il progetto di machine learning precedentemente caricato in un bucket su Cloud Storage.

In seguito il Dockerfile, tramite anaconda, crea e attiva un ambiente Python, ed installa pacchetti (es. pandas, xgboost, cloud storage). Successivamente

esegue lo script che fa l'operazione di *tune* per generare il modello di machine learning e caricarlo su Cloud Storage.

Infine, quando l'esecuzione del container è terminata, la macchina locale potrà quindi ottenere il modello di machine learning generato in cloud ed eliminare tutte le risorse allocate (macchina virtuale con container e bucket).

Download del modello (risultato) in locale

Non appena si ha la certezza che il modello è stato caricato nel bucket, si procede a scaricarlo in locale tramite lo Script 5.5.

```
# Downloads a blob (an object) from the specified bucket
def download_blob(bucket_name, source_blob_name,
    destination_file_name, storage_client):
    # bucket_name = "your-bucket-name"
    # source_blob_name = "storage-object-name"
    # destination_file_name = "local/path/to/file"

    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)

    print(
        "Blob {} downloaded to {}.".format(
            source_blob_name, destination_file_name
        )
    )
```

LISTING 5.5: Funzione che esegue il downolad di un blob contenuto in uno specifico bucket presente in Cloud Storage

Cancellazione della macchina virtuale e del bucket

Infine, vengono chiamate tre funzioni di cancellazione (Script 5.6, 5.7, 5.8). Rispettivamente eliminano l'istanza che ha creato e caricato il file in cloud; tutti gli oggetti presenti nel bucket; e il bucket stesso (è obbligatorio cancellare tutti i blob contenuti in un bucket prima di poter procedere con quest'ultima operazione).

```
# Deletes an instance in Compute Engine
def delete_instance(compute, project, zone, name):
    return compute.instances().delete(
        project=project,
        zone=zone,
        instance=name).execute()
```

LISTING 5.6: Funzione che elimina una specifica macchina virtuale allocata su Compute Engine

```
# Deletes all the blobs (objects) from the specified bucket
def delete_blobs(bucket_name, storage_client):
    """Lists all the blobs in the bucket."""
    # bucket_name = "your-bucket-name"

    blobs = storage_client.list_blobs(bucket_name)

    for blob in blobs:
        blob_name = blob.name
        blob.delete()
        print("Blob {} deleted.".format(blob_name))
```

LISTING 5.7: Funzione che cancella tutti i blob contenuti in uno specifico bucket su Cloud Storage

```
# Deletes a bucket in Google Cloud Storage service
def delete_bucket(bucket_name, storage_client):
    """Deletes a bucket. The bucket must be empty."""
    # bucket_name = "your-bucket-name"

    bucket = storage_client.get_bucket(bucket_name)
    bucket.delete()

    print("Bucket {} deleted".format(bucket.name))
```

LISTING 5.8: Funzione che elimina un bucket contenuto in Cloud Storage

I tempi di esecuzione totali e di tuning sono stati misurati direttamente e rispettivamente mediante lo script in locale e lo script eseguito dal container.

5.5 Analisi dei Risultati

Nella Tabella 5.3 sono riportati i risultati di tutti gli esperimenti descritti nella Sezione 5.3.

machine type	instance type	zone	instance	vCpu	RAM (GB)	tuning time (min)	total time (min)	cost (eur)	cost preemptible (eur)	cost (1 h) (eur)
n1	high-cpu	europe-west1-b (Belgium)	n1-highcpu-2	2	1.8	156.2	161.1	0.18	0.04	0.07
			n1-highcpu-4	4	3.6	80.1	84.5	0.19	0.04	0.14
			n1-highcpu-8	8	7.2	38.6	42.9	0.18	0.04	0.28
			n1-highcpu-16	16	14.4	19.0	22.6	0.18	0.04	0.56
			n1-highcpu-32	32	28.8	9.6	13.4	0.18	0.05	01.11
n2	high-cpu	europe-west1-b (Belgium)	n2-highcpu-2	2	2.0	124.2	129.3	0.15	0.04	0.07
			n2-highcpu-4	4	4.0	60.1	63.3	0.14	0.04	0.14
			n2-highcpu-8	8	8.0	30.1	33.1	0.14	0.04	0.28
			n2-highcpu-16	16	16.0	15.3	18.3	0.14	0.04	0.56
			n2-highcpu-32	32	32.0	7.7	10.7	0.14	0.05	01.13
n2d	high-cpu	europe-west4-c (Netherlands)	n2d-highcpu-2	2	2.0	125.5	130.7	0.13	0.03	0.06
			n2d-highcpu-4	4	4.0	59.6	63.1	0.12	0.03	0.12
			n2d-highcpu-8	8	8.0	29.0	32.2	0.12	0.03	0.24
			n2d-highcpu-16	16	16.0	14.4	17.5	0.12	0.03	0.49
			n2d-highcpu-32	32	32.0	7.3	11.1	0.12	0.04	0.98
e2	high-cpu	europe-west1-b (Belgium)	e2-highcpu-2	2	2.0	155.7	162.4	0.13	0.04	0.05
			e2-highcpu-4	4	4.0	78.1	82.2	0.13	0.04	0.10
			e2-highcpu-8	8	8.0	39.1	42.9	0.13	0.04	0.19
			e2-highcpu-16	16	16.0	19.2	23.0	0.12	0.04	0.39
c2	standard	europe-west1-b (Belgium)	c2-standard-4	4	16.0	53.8	56.7	0.18	0.05	0.20
			c2-standard-8	8	32.0	26.8	30.1	0.18	0.05	0.41
			c2-standard-16	16	64.0	13.5	16.8	0.18	0.06	0.82
			c2-standard-30	30	120.0	8.2	11.6	0.21	0.07	01.54
traditional cloud				2	2	82.9		0.06		0.04
				4	4	41.0		0.06		0.08
				8	8	20.5		0.06		0.18
				16	16	10.3		0.06		0.33
				32	32	5.1		0.05		0.62
traditional cloud with c2 configuration				4	16	41.0		0.13		0.19
				8	32	20.5		0.20		0.58
				16	64	10.3		0.13		0.78
				30	120	5.5		0.13		1.48

TABELLA 5.3: Tempi e costi per ogni configurazione testata

Come già espresso, non tutti i tipi di macchine virtuali dispongono dello stesso numero di vCPU (2, 4, 8, 16, 32), quindi in alcune righe tali valori possono essere leggermente diversi (ad es. le VM di tipo c2 dispongono della configurazione con 30 core, ma non della configurazione con 32 core).

I tempi di esecuzione delle macchine preemptible sono equivalenti a quelli delle macchine normali, poiché hanno le loro stesse identiche caratteristiche. L'unica differenza che le contraddistingue è data dal fatto che queste VM possono essere arrestate in un qualsiasi istante da Google, e quindi se dovessero fallire l'esecuzione, sarebbero poi da riavviare.

Dall'ultima colonna della Tabella 5.3 si può notare come il costo orario scali linearmente all'aumentare del numero delle vCPU. Nella Figura 5.6 sono riportati i costi orari di 4 core di tutte le tipologie di macchine testate. Sono stati presi in considerazione 4 core, perché è il numero più basso di core in comune tra tutte le tipologie di risorse testate.

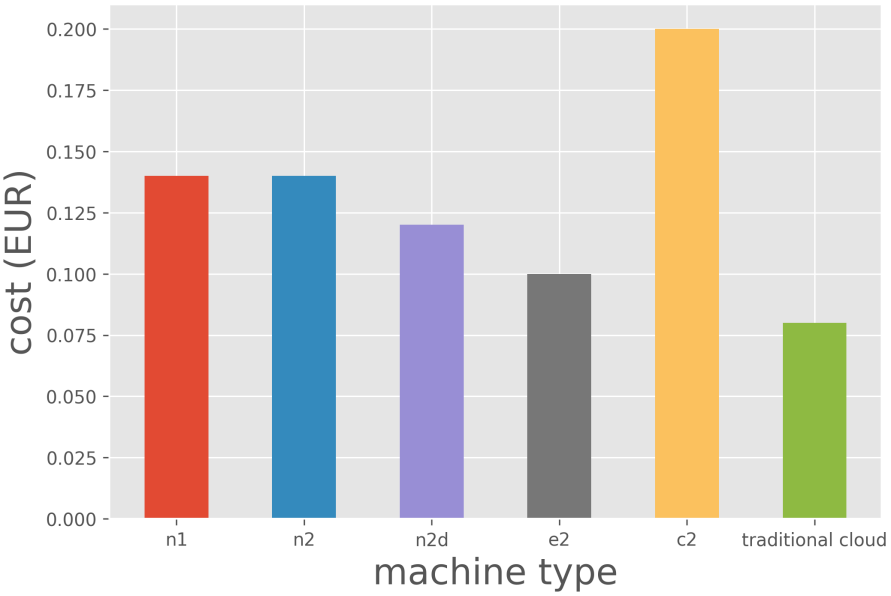


FIGURA 5.6: Costo all'ora delle tipologie di macchine con 4 vCPU

Le VM per utilizzo orario più costose sono quelle compute-optimized (c2), le quali comportano una spesa di €0.20, mentre quelle meno care sono quelle del cloud tradizionale con un prezzo di €0.08. C'è da considerare il fatto che i costi del cloud tradizionale riportati in questa Sezione sono costi approssimati. Infatti, in esso, la tariffazione non è così fluida come sulla piattaforma in cloud di Google, poiché sarebbe presente un vincolo di utilizzo minimo (ad es. un mese) che però non è stato considerato. Come già descritto nella Sottosezione 4.5.2 le macchine di tipo e2 sono quelle più economiche disponibili su GCP. Le rimanenti tipologie (n1, n2 e n2d) hanno un prezzo orario molto simile fra loro che oscilla da €0.12 a €0.14.

La tariffazione delle vm di serie c2 è molto più alta rispetto alle altre anche per via della RAM, la quale è molto superiore rispetto a quella presente in tutte le altre tipologie di macchine (Tabella 5.3). Nel grafico a barre in Figura 5.7 sono riportati i costi orari di tutte le macchine con 4 core, considerando però che in questo caso il cloud tradizionale abbia le stesse identiche configurazioni delle vm di tipo c2. Si può notare che, a parità di numero di core e RAM, i prezzi orari delle macchine c2 e del cloud tradizionale sono quasi uguali, con un costo rispettivo di €0.20 e €0.19.

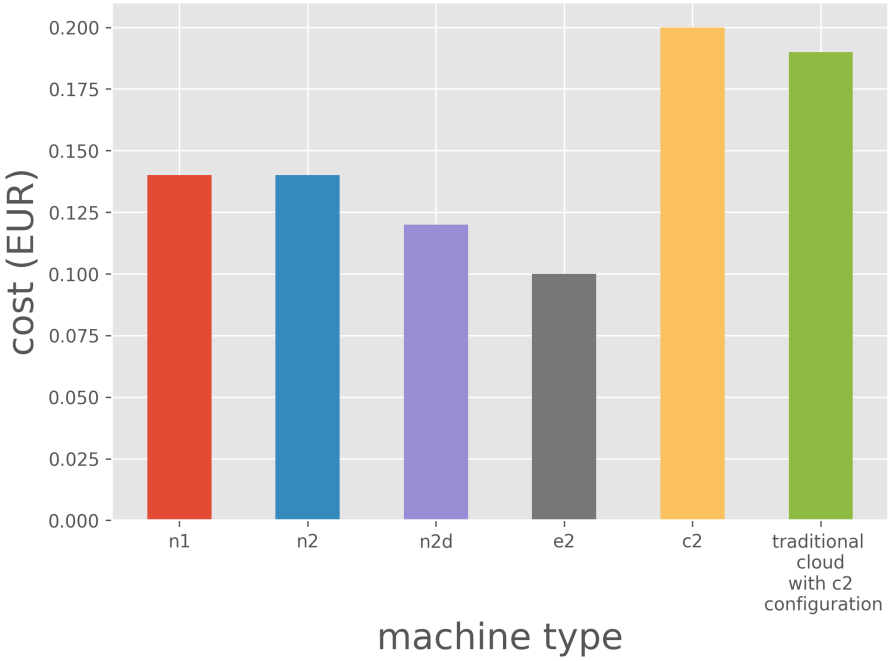


FIGURA 5.7: Costo all'ora delle tipologie di macchine con 4vCPU, con il cloud tradizionale avente le stesse configurazioni delle macchine c2

Nella Figura 5.8 si può osservare che alle macchine di tipo preemptible è associato un costo orario nettamente inferiore rispetto alle VM normali. Sono molto vantaggiose dal punto di vista economico, ma comportano il rischio di un'improvvisa interruzione della loro esecuzione (Sottosezione 4.5.2), la quale comprometterebbe il lavoro svolto dall'operazione di tune fino a quel momento. Anche le macchine preemptible hanno una tariffazione che cresce in maniera lineare rispetto al numero di vCPU e ugualmente le VM di tipo c2 sono quelle più dispendiose, con un prezzo di 5 centesimi all'ora, seguite da quelle di tipo n2 (€0.034/h). Le restanti tipologie (n1, n2d ed e2) hanno un costo equivalente di €0.03 all'ora.

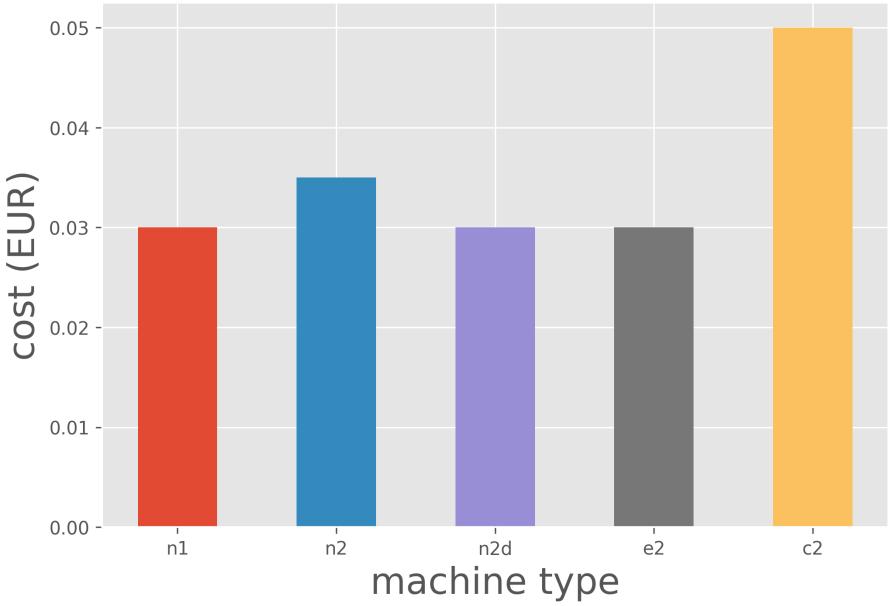


FIGURA 5.8: Costo all'ora delle tipologie di macchine preemptible con 4 vCPU

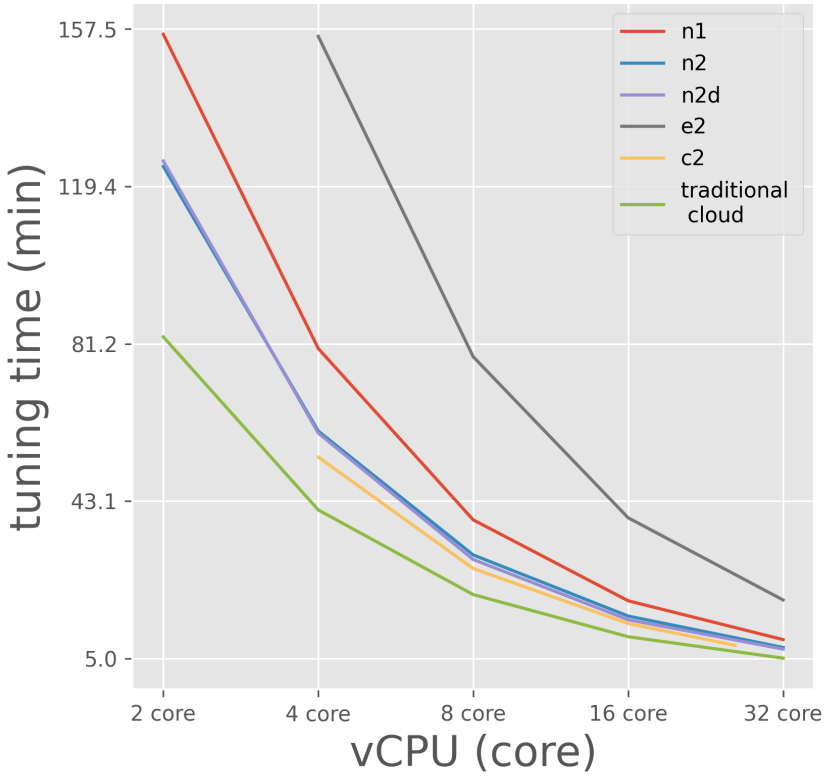


FIGURA 5.9: Tempi di esecuzione del tuning time delle tipologie di macchine testate

Nella Figura 5.9 il grafico a linee riporta i tempi di esecuzione del tuning di tutte le configurazioni di VM testate. Le più veloci sono quelle allocate nel cloud tradizionale, mentre quelle più lente sono quelle aventi serie e2. Le macchine virtuali c2 sono quelle con prestazioni migliori tra tutte le VM esaminate su Google Cloud Platform. In seguito sono presenti le risorse di tipo n2 e n2d con tempi molto simili tra loro e minori rispetto alla serie n1. La macchina che riesce a svolgere il tuning nel tempo minore è quella del cloud tradizionale con 32 core, la quale impiega solo 5.13 minuti, mentre la più performante su GCP è quella di serie n2d avente 32 core, con un tempo di 7.3 minuti.

Gli andamenti illustrati in Figura 5.10 riportano il tempo totale di esecuzione calcolato sulle istanze utilizzate sulla piattaforma in cloud di Google. Il tempo totale di esecuzione è costituito dalla somma fra il tempo di tuning e il tempo di creazione e distruzione delle risorse. I valori del tempo totale di esecuzione hanno lo stesso trend del tempo di esecuzione del tuning mostrato in Figura 5.9, perché, come si può osservare nella Tabella 5.3, il tempo in più richiesto per creare e distruggere le istanze è una costante che varia circa da 3 a 7 minuti.

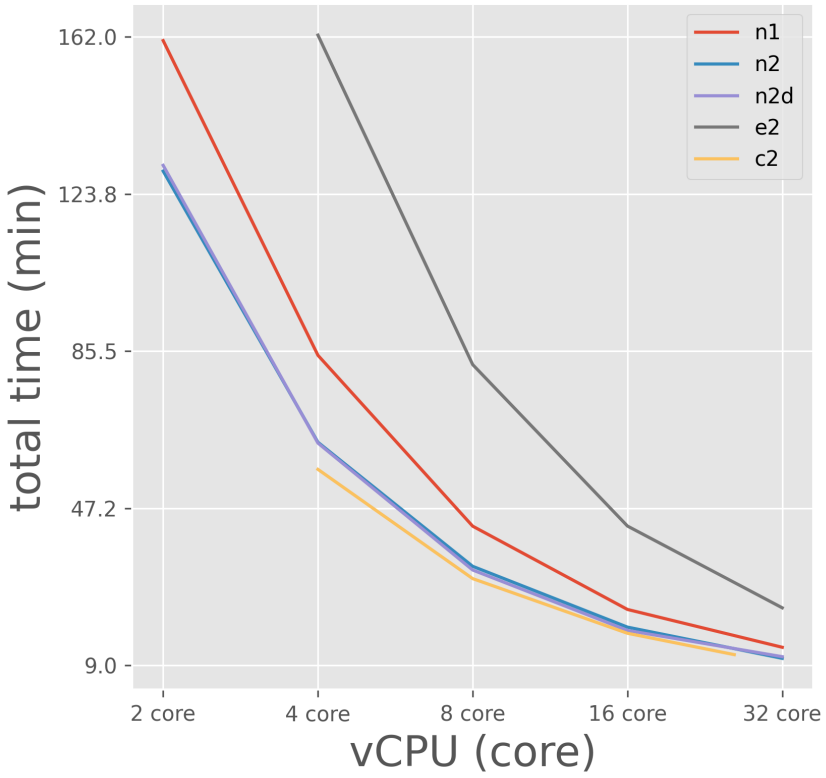


FIGURA 5.10: Tempi totali di esecuzione delle tipologie di macchine testate

La figura 5.11 indica quali sono i costi dovuti all'esecuzione del tuning dei vari tipi di macchine. Le VM del cloud tradizionale, oltre ad avere performance migliori in termini di tempo richiesto per l'esecuzione, sono anche più economiche rispetto alle istanze di Google Cloud, di fatto comportano una spesa che inizialmente è stabile sui 6 centesimi di euro e con 32 core diminuisce leggermente arrivando a €0.05. In questo caso il prezzo cala all'accrescere del numero dei core, perché il tempo di tuning (e quindi il tempo di utilizzo) col massimo numero di core è ridotto al minimo. Le istanze meno dispendiose su GCP per il caso d'uso testato sono quelle di serie n2d, le quali hanno un costo che decresce da €0.13 con 2 core a €0.12 con 4, 8, 16 e 32 core. Anche la serie n2 ha stesso identico andamento della serie n2d, ma con una spesa in più di 2 centesimi. La tipologia e2 (quella più conveniente a parità di utilizzo) ha un prezzo a metà tra quello dei tipi n2 e n2d. Le macchine virtuali più care nell'eseguire il tuning sono quelle appartenenti alle serie n1 e c2. In particolare a queste ultime corrisponde un importo inizialmente stabile di €0.18, il quale poi cresce a €0.21.

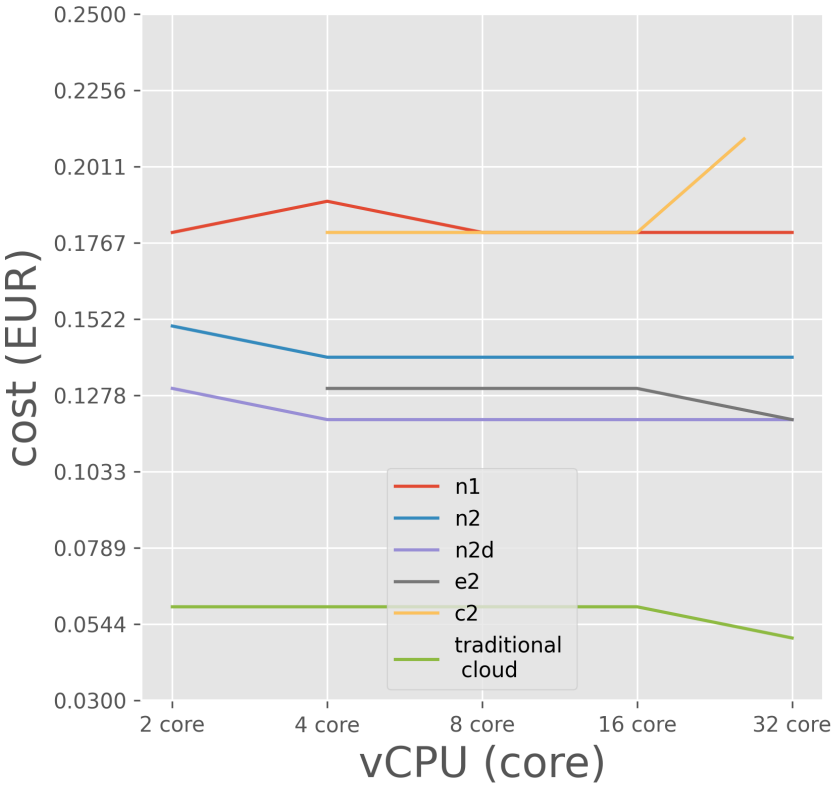


FIGURA 5.11: Costo del tuning delle tipologie di macchine testate

In precedenza è stato fatto presente che l'algoritmo utilizzato scala linearmente, e dunque nel grafico in Figura 5.11 i costi non dovrebbero cambiare

all'aumentare dei core. Le oscillazioni dei prezzi sono dovute ai tempi di esecuzione del tuning che non sono perfettamente deterministici. Probabilmente queste oscillazioni scomparirebbero se si dovessero eseguire numerosi test per ogni configurazione di macchina e se venisse poi calcolata per ciascuna di esse la media dei risultati ottenuti, ma per motivi di costi e tempo è stato svolto un solo test per ogni VM presa in considerazione.

Nella Figura 5.12 sono illustrati i costi derivanti dall'esecuzione del tuning dei diversi tipi di macchine preemptible, quindi non sono presenti le VM del cloud tradizionale. L'asse y ha la stessa scala del grafico in Figura 5.11 cosicché sia maggiormente visibile la differenza di costo tra una macchina preemptible e una normale. Il prezzo minore (€0.03) è dato dal tipo di macchina n2d da 2 a 16 vCPU, mentre la serie c2 è quella più costosa, con un prezzo che varia dai €0.05 ai €0.07. Le classi n1 e n2 hanno lo stesso identico trend, sempre stabile sui €0.04 tranne per la configurazione a 32 core che costa €0.05. La spesa dovuta all'esecuzione del tuning delle macchine preemptible di tipo e2 è di €0.04.

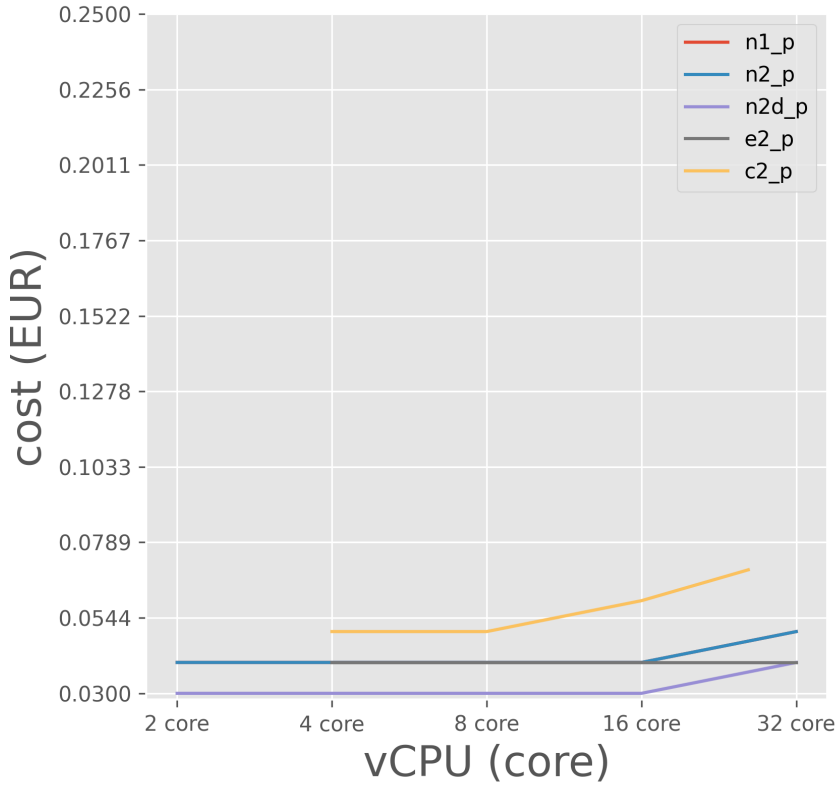


FIGURA 5.12: Costo del tuning delle tipologie di macchine preemptible testate

Il grafico di dispersione riportato in Figura 5.13 mostra le combinazioni di

tempo e costo di esecuzione del tuning per tutte le configurazioni di VM analizzate. I risultati migliori sono nella parte inferiore sinistra del grafico, poiché l'obiettivo è quello di minimizzare il costo e il tempo di esecuzione dell'algoritmo di machine learning. In particolare la migliore configurazione è data dalla macchina virtuale con 32 core allocata nel cloud tradizionale, la quale è in grado di completare il tuning in 5'13", con un costo di €0.05. Per ciascun numero di vCPU il cloud tradizionale offre esiti più vantaggiosi, sia in termini di velocità che di risparmio, rispetto alle macchine testate sulla piattaforma in cloud di Google. Tra le VM esaminate su GCP, quella che impiega il minor tempo per svolgere questo caso d'uso (7'30") è anche la più economica (€0.12), ha 32 vCPU ed è di tipo n2d. Le istanze c2, a parità di core, hanno le prestazioni migliori in termini di durata di esecuzione tra tutte le risorse analizzate su Google Cloud. Però le VM compute-intensive (c2) sono anche quelle che hanno i costi più alti. Come visto in Figura 5.11 l'andamento del prezzo delle macchine di tipo n1 è simile a quelle di tipo c2, ma quelle di tipo n1 svolgono il tuning in tempi più lunghi. Invece le macchine virtuali di serie n2 hanno tempi simili a quelle di categoria n2d (Figura 5.9), ma comportano una spesa maggiore. Considerando lo stesso numero di vCPU, le risorse di tipo n1 ed e2 sono quelle che eseguono il tuning più lentamente. Le VM del cloud tradizionale aventi le stesse configurazioni delle macchine di tipologia c2 impiegano più tempo per eseguire l'operazione di tune, ma hanno un costo minore (tranne per il caso con 8 core).

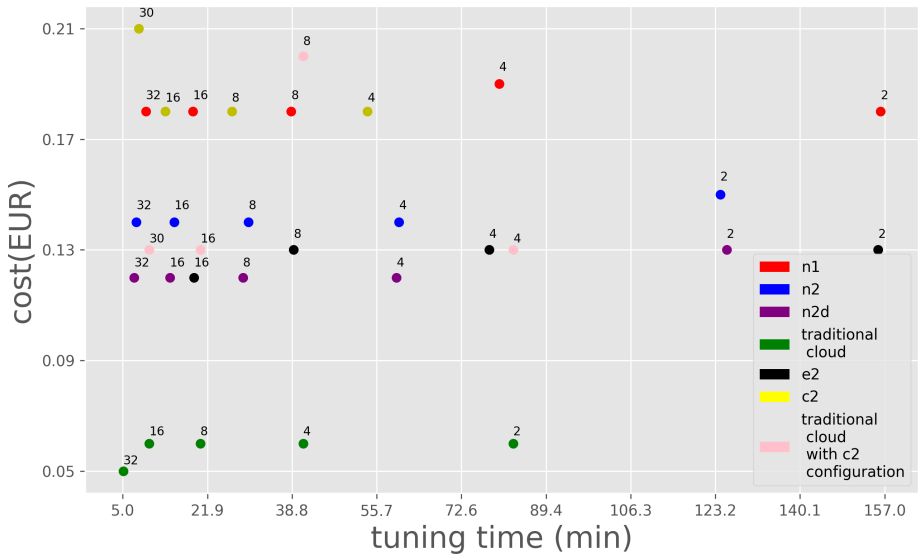


FIGURA 5.13: Relazione tra tempo e costo di esecuzione del tuning delle varie tipologie di macchine testate

La Figura 5.14 illustra qual è la relazione tra il tempo e il costo dello svolgimento del tuning delle macchine virtuali preemptible su GCP e di quelle del cloud tradizionale. Come già descritto all'inizio di questa Sezione, i tempi di

esecuzione delle macchine preemptible di Google Cloud sono equivalenti a quelli delle VM normali, mentre si ha una riduzione sostanziale del costo e il rischio aggiuntivo di un'improvvisa interruzione dell'esecuzione. Le configurazioni più veloci sono quelle già citate nella descrizione della Figura 5.13. In questo grafico di dispersione è il cloud tradizionale ad avere i prezzi più alti (€0.06), ad eccezione del caso con 32 core che è minore rispetto a quello della classe c2. La serie preemptible più economica per svolgere il tuning è n2d, con una spesa che varia dai €0.03 ai €0.04. Come descritto in Figura 5.13 le due tipologie n2 e n2d hanno tempi molto simili e anche in questo caso la prima delle due comporta una spesa leggermente superiore. Come mostrato in Figura 5.12, le serie n1, n2 ed e2 hanno un trend del costo pressoché identico.

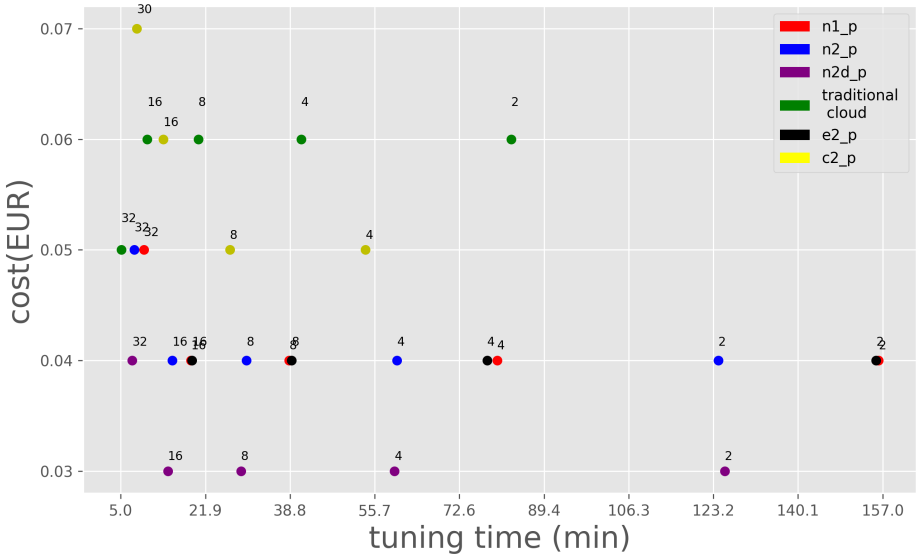


FIGURA 5.14: Relazione tra tempo e costo di esecuzione del tuning delle varie tipologie di macchine preemptible e del cloud tradizionale

5.6 Considerazioni aggiuntive

In questa Sezione vengono fatte delle considerazioni aggiuntive sul confronto tra GCP e cloud tradizionale in termini di servizi offerti (Sottosezione 5.6.1), vincoli orari di utilizzo minimo (Sottosezione 5.6.2) e flessibilità (Sottosezione 5.6.3). Le due tipologie di cloud analizzate (GCP e cloud tradizionale) non sono facilmente confrontabili. Non è sufficiente sapere solo il costo orario delle due piattaforme per riuscire ad ottenere una risposta esaustiva su quale dei due sia meglio utilizzare. Le prossime Sottosezioni spiegano come questa scelta cambi in base a molteplici fattori, come ad esempio la tipologia di compito che si vuole svolgere, il numero di ore di utilizzo al mese,

i servizi richiesti ecc. Non è una decisione banale, ci sono casi in cui conviene uno e casi in cui è maggiormente vantaggioso optare per l'altro.

5.6.1 Servizi

Una sostanziale differenza tra Google Cloud platform e il cloud tradizionale preso come riferimento è data dal fatto che quest'ultimo offre un numero maggiore di servizi aggiuntivi integrati. Per esempio, Compute Engine non fornisce né un'assistenza né una gestione automatica dei backup.

Per il caso d'uso d'interesse (tuning di algoritmi di machine learning) queste funzionalità possono anche non essere necessarie (ad es. il backup), però quando si vuole avere un ambiente che deve stare operativo per lunghi periodi (ad es. quando si hanno algoritmi di produzione che un cliente esegue in continuazione), allora servono certi protocolli (certificazioni, firewall, backup). Il problema di scegliere quale cloud utilizzare non è solo un problema di costi, ma è anche un problema di servizi, perché non è semplice utilizzare la piattaforma in cloud di Google ed avere tutte quelle funzionalità che mette a disposizione un provider privato. Il provider privato fornisce anche un'assistenza, la quale risolve problemi in caso di guasti o malfunzionamenti, e gestisce le configurazioni delle macchine.

Per quanto riguarda il caso d'uso analizzato (Sezione 5.2) il focus è incentrato sul testare algoritmi di machine learning in cloud, quindi probabilmente, in termini di servizi, conviene GCP, però ci sono altri casi nei quali certe funzionalità aggiuntive sono essenziali. Queste considerazioni sui servizi servono per far capire che non bisogna tenere conto solo dei prezzi quando si vuole fare un confronto tra due diverse tipologie di cloud.

5.6.2 Vincoli orari

Osservando il grafico a barre in Figura 5.6 mostrato nella Sezione 5.5, il quale riporta i costi orari delle diverse tipologie di macchine, sembrerebbe ovvio quale cloud scegliere tra l'uno e l'altro per eseguire un certo tipo di lavoro. Infatti il cloud tradizionale ha un costo orario minore in confronto a tutte le altre serie offerte dalla piattaforma in cloud di Google. Tuttavia, nel cloud tradizionale non è possibile allocare una VM per solo un paio d'ore, perché esso impone dei vincoli orari di utilizzo minimo per le risorse che mette a disposizione. Spesso questo vincolo (come lo è stato per il cloud tradizionale contattato) corrisponde a 24 ore. È vero che il cloud tradizionale costa di meno ed è più veloce (per il caso d'uso del tuning) rispetto a GCP, però non offre la possibilità di pagare un prezzo inferiore alla quota di utilizzo minimo (24 h), anche nel caso in cui si faccia uso di una macchina virtuale per solo un paio d'ore. Ad esempio, se si volesse sfruttare una risorsa per 6 ore al giorno tutti i giorni del mese, allora il cloud tradizionale imporrebbe di pagare il prezzo pieno, cioè di 24 ore per tutti i giorni del mese, poiché con esso non è possibile richiedere l'utilizzo di un'istanza per una durata inferiore di un

giorno intero. Invece Compute Engine non impone alcun vincolo di durata minima, infatti permette di creare istanze anche solo per periodi di tempo limitati (poche ore o minuti) e di eliminarle quando lo si vuole, ma ha un costo orario maggiore.

Il grafico a linee in Figura 5.15 riporta la variazione dei prezzi (in euro) delle varie tipologie di macchine testate (con 4 core) in base al numero di ore di utilizzo giornaliero. Si può notare come il costo delle macchine fornite dal cloud tradizionale resti costante a €1.92 (€0.08 x 24 h) per l'intero arco della giornata. Superato il vincolo orario di utilizzo minimo, il cloud tradizionale aumenta di €0.08 per ogni ora richiesta in più. Come detto anche precedentemente, le VM di GCP non hanno una soglia di utilizzo minimo, e comportano una spesa che è direttamente proporzionale al numero di ore di computazioni, anche per carichi di lavoro brevi. Si può notare come il cloud tradizionale, per utilizzi ininterrotti uguali o superiori a 24 ore, abbia una tariffazione minore rispetto a tutte le altre VM presenti su Compute Engine. In particolare la serie c2 diventa più dispendiosa (con utilizzi non interrotti) delle macchine fornite dal cloud tradizionale dopo 10 ore, le tipologie n1 e n2 dopo 14 ore, la serie n2d dopo 16 ore, mentre la tipologia e2 dopo 20 ore.

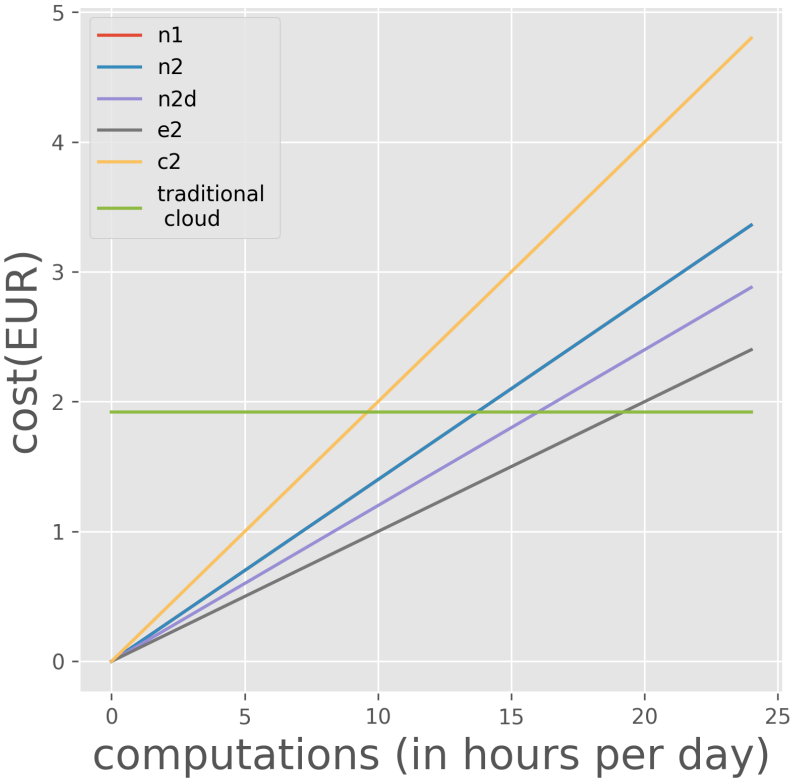


FIGURA 5.15: Costo delle varie tipologie di macchine al crescere del numero di ore di utilizzo giornaliero

Per comprendere meglio quale dei due cloud conviene in base a quanto tempo lo si utilizzi, si è deciso di prendere in considerazione vari esempi, ai quali è associato un diverso numero di ore mensili di utilizzo continuo e non.

Il primo caso considerato è quello in cui un'azienda svolge il training di modelli di machine learning una volta a settimana. Si ipotizza che la durata dell'addestramento dei modelli duri circa un'ora, quindi mensilmente questo lavoro richiede un utilizzo di circa 4 ore di computazioni. Nel grafico a barre in Figura 5.16 sono riportati i costi associati a 4 ore di impiego per ogni tipologia di macchina testata. Il cloud tradizionale comporta una spesa di €7.68. Questa cifra equivale a quattro giornate intere di esecuzione (€1.92 x 4 giorni), è data dal vincolo orario minimo, ed è di gran lunga maggiore in confronto a tutte le altre VM analizzate su Compute Engine. La serie più conveniente di Google è e2, che ha un costo di €0.40, seguita dalle serie n2d, n1 ed n2, e tutte rispecchiano i costi orari indicati in Figura 5.6.

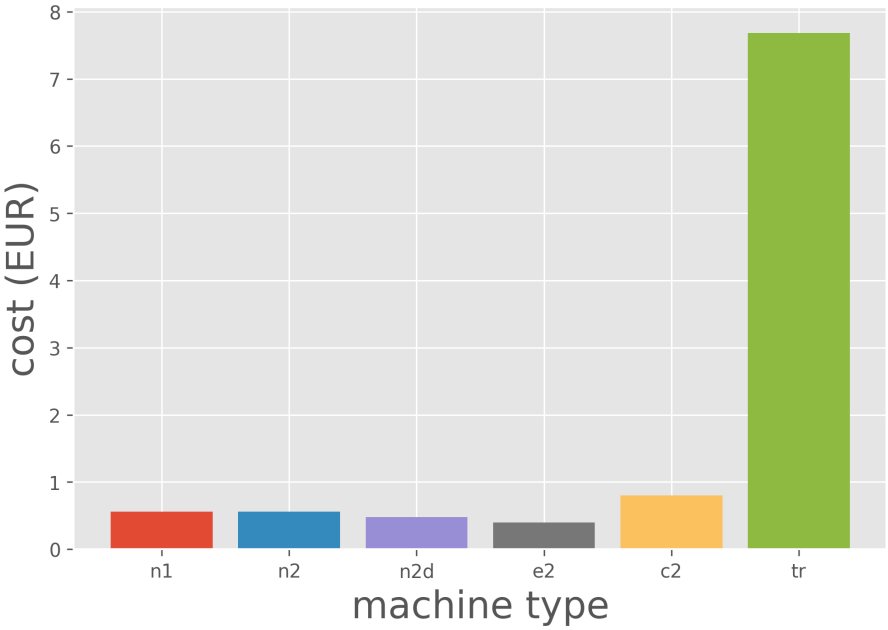


FIGURA 5.16: Costi delle varie tipologie di macchina (con 4 core) per 4 ore di utilizzo

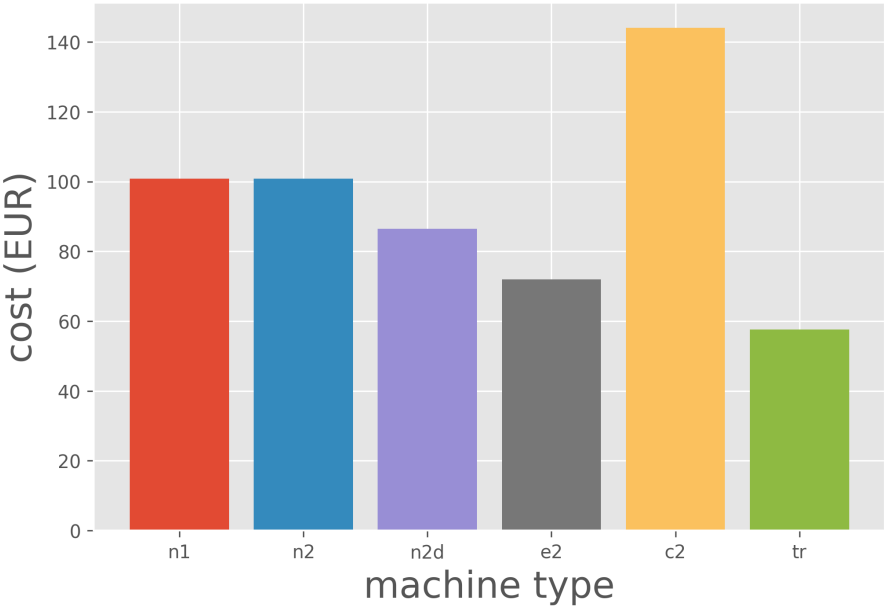


FIGURA 5.17: Costi delle varie tipologie di macchina (con 4 core) per 720 ore di utilizzo

Il secondo caso preso in considerazione è un po' l'opposto del precedente. In questo si assume di dover utilizzare un'istanza per 24 ore al giorno, tutti i giorni del mese. Un esempio è quello in cui un'azienda debba tenere attivo un server che risponda in continuazione alle richieste dei clienti. Il grafico a barre in Figura 5.17 riporta i prezzi delle diverse tipologie di VM analizzate per 720 ore ininterrotte di computazioni. In questo caso conviene di più fare uso delle macchine messe a disposizione dal cloud tradizionale, perché hanno un costo di €57.60, che è inferiore rispetto a quello di tutte le altre istanze testate su GCP. Le serie n1 e n2 comportano una spesa di circa €100.80, e2 di €72.00 e c2 di addirittura €144.00.

Nella Tabella 5.4 sono riportati quattro esempi mensili di impiego delle risorse. Due esempi sono quelli appena analizzati graficamente nelle Figure 5.16 e 5.17, mentre gli altri casi considerano rispettivamente un numero di ore di utilizzo di due giorni consecutivi mensili e di 6 ore al giorno per tutti i giorni di un mese. Per ciascun esempio sono stati riportati i seguenti dati: il massimo numero di ore ininterrotte di computazioni, il numero di ore al mese effettivamente pagate per il cloud tradizionale e i costi delle varie tipologie di macchine.

monthly use case	hours of uninterrupted computations	hours paid for traditional cloud	cost (eur)					
			n1	n2	n2d	e2	c2	traditional cloud
1 hour per week	1	96 (24 x 4)	0.56	0.56	0.48	0.40	0.80	7.68
24 hours per day	720	720 (24 x 30)	100.80	100.80	86.40	72.00	144.00	57.60
2 consecutive days	48	48 (24 x 2)	6.72	6.72	5.76	4.80	9.60	3.84
6 hours per day	6	720 (24 x 30)	25.20	25.20	21.60	18.00	36.00	57.60

TABELLA 5.4: Ulteriori casi d'uso mensili con costi associati

Da questi esempi si è compreso che non esiste una soluzione unica per tutti i casi, ma ogni singolo caso va analizzato singolarmente. In generale, in termini di costi, conviene optare per le istanze fornite dal cloud tradizionale se c'è il bisogno di eseguire un algoritmo di machine learning per periodi maggiori di un paio di giornate consecutive al mese. Mentre è più vantaggioso utilizzare il cloud di Google se l'algoritmo di machine learning (o un altro compito) viene eseguito solo per poche ore al mese, o per utilizzi discontinui così come il primo caso, i cui costi sono riportati in Figura 5.16.

Il cloud tradizionale ha un costo orario basso, ma comporta una spesa alta se lo si vuole sfruttare per brevi periodi di tempo (o per intervalli temporali discontinui), a causa dei vincoli orari di utilizzo minimo. Esso offre il prezzo migliore per utilizzi prolungati (più di 24 ore) rispetto a Google. Compute Engine invece ha una tariffa oraria più alta ed un costo prolungato peggiore; però è ottimale, a livello di risparmio, per task medio-brevi (ovvero aventi durata minore di 24 ore) e discontinui.

5.6.3 Flessibilità

Il cloud tradizionale testato non ha la stessa flessibilità di Google Cloud, di fatto non mette a disposizione delle API con le quali poter eseguire in modo autonomo certe operazioni (ad es. la creazione di una macchina virtuale, come mostrato nella Sottosezione 4.5.1). Per poter allocare delle VM nel cloud tradizionale è necessario contattare (solitamente per mail) il rappresentante dell'azienda che fornisce il servizio, specificando le caratteristiche della macchina richiesta e per quanto tempo la si vuole utilizzare. Questa minore elasticità semplifica la gestione delle risorse, poiché vengono completamente controllate dal fornitore, ma comporta maggiori tempi di attesa per poter svolgere un particolare lavoro.

Le API nella piattaforma in cloud di Google sono vantaggiose nel caso in cui l'azienda cliente abbia personale qualificato che sappia utilizzarle, in caso contrario conviene affidarsi all'assistenza messa a disposizione da un cloud tradizionale.

Se si ha fretta di svolgere un particolare task, allora conviene utilizzare GCP. Infatti essa consente di allocare le risorse con le API in tempi brevissimi, mentre col cloud tradizionale si dovrebbe attendere un feedback da parte del fornitore.

È maggiormente vantaggioso scegliere il cloud di Google anche nel caso in cui non si sa a priori quali caratteristiche richiedere (ad es. core e ram) per le macchine virtuali che dovranno svolgere un certo lavoro. Difatti, col cloud tradizionale risulterebbe dispendioso testare molteplici VM, poiché ciascuna macchina non può essere eliminata istantaneamente dato che ha un impiego di utilizzo minimo (come visto nella Sottosezione 5.6.2), mentre con le API di Google questo problema non si presenta.

Capitolo 6

Conclusioni

In questa tesi sono state studiate le modalità che Google Cloud Platform mette a disposizione per la generazione di modelli di machine learning, ponendo una particolare attenzione sulla creazione di modelli custom a partire da macchine virtuali gestite manualmente. Questo progetto mira a svolgere un'analisi dei costi e benefici che il cloud di Google offre rispetto ad un cloud provider privato di modeste dimensioni.

Nella prima parte sono stati introdotti i concetti di cloud computing e di containerizzazione. Ciò che è stato evidenziato è la migliore efficienza delle risorse dei software container rispetto alle VM, e anche la loro maggiore flessibilità come framework per la gestione di task.

È stata quindi esaminata la piattaforma in cloud di Google, studiando nel dettaglio le funzionalità di Compute Engine, Cloud Storage e Cloud Registry. Questi approfondimenti sono stati essenziali per procedere con l'esecuzione dei test tramite il framework di automatizzazione appositamente ideato per la generazione di modelli di machine learning.

Dai risultati degli esperimenti svolti è stato notato che non esiste una risposta chiara e sempre valida su quale tipologia di cloud conviene scegliere, poiché entrambi hanno dei pregi e dei difetti, spesso complementari. Sono stati esaminati dunque diversi fattori: la flessibilità offerta, i vincoli orari di utilizzo minimo e i servizi aggiuntivi inclusi.

I test sui costi derivanti dal vincolo orario minimo hanno indicato che è più vantaggioso optare per il cloud tradizionale quando si devono svolgere computazioni per periodi ininterrotti con una durata maggiore di 24 ore, mentre invece Google Cloud Platform è maggiormente economico quando si devono eseguire task discontinui e/o di medio-breve periodo di tempo.

La piattaforma in cloud di Google mette a disposizione delle API, fornendo quindi una maggiore flessibilità, con la quale è possibile gestire in modo completamente autonomo le risorse. In questo modo è possibile personalizzare a proprio piacere le macchine virtuali da allocare in cloud, e si evitano quei lunghi tempi di attesa dovuti alla gestione delle VM fornite da un provider privato.

Dal punto di vista dei servizi aggiuntivi è il cloud tradizionale ad avere la meglio, poiché offre una completa assistenza in caso di guasti ed errori, e altre funzionalità incluse come la gestione automatica dei backup.

Visto che questi temi rimangono aperti, sarebbe interessante anche lavorare su altri sviluppi futuri. Qui vengono elencati alcuni di essi:

- Estendere questo confronto con ulteriori provider di cloud, come Amazon Web Services (AWS), Microsoft Azure, Oracle ecc.
- Eseguire gli stessi esperimenti sugli altri servizi di machine learning presenti su GCP (AutoML, BigQuery ML ecc.) per poter ampliare il confronto sui costi e sulla velocità di esecuzione.
- Creare un'interfaccia web che gestisca l'intero processo del framework di automatizzazione (anziché eseguire il tutto con un comando da terminale), in modo da poter fornire le funzionalità di questo progetto a chiunque, anche a chi non è capace di utilizzare le API di Google Cloud.

Bibliografia

[1] Kyriakos N. Agavanakis et al. «Practical machine learning based on cloud computing resources». In: *XIAMEN-CUSTIPEN WORKSHOP ON THE EQUATION OF STATE OF DENSE NEUTRON-RICH MATTER IN THE ERA OF GRAVITATIONAL WAVE ASTRONOMY*. AIP Publishing, 2019. DOI: 10.1063/1.5117023. URL: https://doi.org/10.1063/1.5117023.

[2] Takuya Akiba et al. «Optuna: A Next-generation Hyperparameter Optimization Framework». In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] *Anaconda Documentation*. URL: https://docs.anaconda.com/.

[4] D. Bernstein. «Containers and Cloud: From LXC to Docker to Kubernetes». In: *IEEE Cloud Computing* 1.3 (2014).

[5] *BigQuery ML Documentation*. Google. URL: https://cloud.google.com/bigquery-ml/docs.

[6] S. Chaisiri, B. Lee e D. Niyato. «Optimization of Resource Provisioning Cost in Cloud Computing». In: *IEEE Transactions on Services Computing* 5.2 (2012), pp. 164–177.

[7] S. Challita et al. «A Precise Model for Google Cloud Platform». In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. 2018, pp. 177–183.

[8] Tianqi Chen e Carlos Guestrin. «XGBoost». In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, ago. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145/2939672.2939785.

[9] *Cloud AutoML Documentation*. Google. URL: https://cloud.google.com/automl/docs.

[10] *Cloud Storage Documentation*. Google. URL: https://cloud.google.com/storage/docs.

[11] *Compute Engine Documentation*. Google. URL: https://cloud.google.com/compute/docs.

[12] *Creating and starting a VM instance*. Google. URL: https://cloud.google.com/compute/docs/instances/create-start-instance.

[13] *Docker Documentation*. Docker Inc. URL: https://docs.docker.com/.

[14] Vittorio Ghini. *Docker e Kubernetes - dispense delle lezioni*.

[15] *GridSearchCV Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[16] *Kubernetes Documentation*. Kubernetes. URL: https://kubernetes.io/docs/home/.

[17] Petro Liashchynskyi e Pavlo Liashchynskyi. «Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS». In: (2019).

[18] Farhad Malik. *What Is Grid Search? Explaining How To Obtain Optimal Hyperparameter Values*. URL: https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a.

[19] P M Mell e T Grance. *The NIST definition of cloud computing*. Rapp. tecn. 2011. DOI: 10.6028/nist.sp.800-145. URL: https://doi.org/10.6028/nist.sp.800-145.

[20] Claus Pahl et al. «Cloud Container Technologies: A State-of-the-Art Review». In: *IEEE Transactions on Cloud Computing* 7.3 (lug. 2019), pp. 677–692. DOI: 10.1109/tcc.2017.2702586. URL: https://doi.org/10.1109/tcc.2017.2702586.

[21] Rubens Paul. *What are containers and why do you need them?* Giu. 2017. URL: http://narnia.homeunix.com/~robert/Fall2018/is375/WhatAreContainers.pdf.

[22] RedHat. *Introduction to Kubernetes architecture*. URL: https://www.redhat.com/en/topics/containers/kubernetes-architecture.

[23] Alessandro Ricci. *Cloud Computing - dispense delle lezioni*.

[24] Jasper Snoek, Hugo Larochelle e Ryan P Adams. «Practical Bayesian Optimization of Machine Learning Algorithms». In: *Advances in Neural Information Processing Systems 25*. A cura di F. Pereira et al. Curran Associates, Inc., 2012, pp. 2951–2959. URL: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

[25] *XGBoost Documentation*. URL: https://xgboost.readthedocs.io/en/latest/index.html.

[26] *XGBoostRegressor*. URL: https://xgboost.readthedocs.io/en/latest/python/python_api.html.

Ringraziamenti

Innanzitutto vorrei ringraziare il mio relatore Daniele Vigo, per avermi dato la possibilità di svolgere la tesi presso Optit. Ringrazio il Dott. Simone Graziani, tutor aziendale, per essere stato sempre disponibile nel darmi consigli e chiarirmi dubbi. Ringrazio di cuore la mia famiglia per il sostegno che mi ha offerto durante l'intero percorso di studi. Un grazie speciale lo dedico alla mia ragazza, che è stata sempre presente nei momenti più difficili. Infine, desidero ringraziare i miei amici e colleghi universitari per avermi accompagnato in quest'avventura e per avermi fatto vivere quest'esperienza con una maggiore serenità.