UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Rehosting Cloud-to-Cloud di un applicazione su Amazon Web Services

Cloud-to-Cloud Rehosting of an Application on Amazon Web Services

Relatore:
Prof. Mancini Adriano
Correlatore:
Dott. Tonetto Flavio

Laureando:
Dediu Razvan Alexandru

Anno Accademico 2023-2024

Abstract

Il presente lavoro è stato sviluppato in collaborazione con l'azienda **Sinergia EPC** e con il supporto consulenziale dell'azienda **recube S.r.l.**, con l'obiettivo di effettuare la migrazione di una applicazione utilizzando prevalentemente un approccio *rehosting*, affiancato da interventi di *replatforming* per alcuni servizi esistenti. Lo scopo è stato quello di passare da una architettura poco gestita ad una più moderna e sicura, in grado di garantire maggiore scalabilità ed efficienza operativa.
Dopo un'accurata analisi dei fornitori disponibili, tenendo conto del budget e della varietà dei servizi offerti, è stata presa la decisione di adottare AWS, riconosciuto per la sua affidabilità e scalabilità.

Inizialmente, è stata progettata una rete privata virtuale che definisse l'architettura della nuova infrastruttura, permettendo di isolare le risorse e migliorando complessivamente la sicurezza. Sono stati identificati e selezionati i servizi AWS da integrare, tra cui Amazon EC2 per il calcolo, Amazon RDS per la gestione dei database, e Amazon S3 per l'archiviazione dei dati.
Successivamente, si è proceduto alla migrazione e integrazione delle applicazioni esistenti in un nuovo ambiente AWS, utilizzando un approccio che garantisce la minimizzazione dei rischi durante il trasferimento.

Un aspetto fondamentale del progetto è stata l'implementazione delle metodologie più sicure per lo sviluppo cloud, che hanno incluso la gestione del traffico sulle singole risorse. Sono state adottate *best practices* per garantire un accesso controllato e sicuro, ottimizzando le politiche di accesso e i gruppi di sicurezza, e assicurando che tutte le comunicazioni tra le risorse avvenissero in modo protetto.

Per monitorare le prestazioni e i costi, sono stati utilizzati servizi che hanno consentito di ottimizzare le spese per l'infrastruttura e di identificare tempestivamente eventuali anomalie ed errori provenienti dalle istanze.

I risultati ottenuti hanno consentito all'azienda di avere non solo un cloud moderno e all'avanguardia, ma anche un ambiente sicuro, scalabile e altamente disponibile per l'applicazione migrata, oltre che per lo sviluppo e la produzione di nuovi software.
Questo progetto costituisce un importante progresso nell'ottimizzazione delle operazioni aziendali e nel potenziamento delle capacità tecnologiche dell'azienda, creando così le condizioni ideali per affrontare le sfide future e migliorare la propria competitività nel mercato.

Indice

Capitolo 1

Introduzione

Il cloud computing è una delle innovazioni più rivoluzionarie emerse negli ultimi anni, avendo profondamente impattato il mondo tecnologico e il modo in cui le aziende operano. Il principale vantaggio offerto da questa tecnologia è la possibilità di scalabilità, sia orizzontale che verticale, consentendo alle imprese di adattare le risorse IT in modo dinamico e senza vincoli infrastrutturali.

La scalabilità orizzontale (*scale out*) permette di aggiungere ulteriori macchine al sistema per distribuire il carico di lavoro, aumentando così la capacità di gestione senza modificare i server esistenti.

Figura 1.1: Scalabilità orizzontale

Al contrario, la scalabilità verticale (*scale up*) consente di potenziare le prestazioni di una singola macchina, aggiungendo risorse come CPU o memoria, senza cambiare l'infrastruttura generale. Questo approccio rende il cloud estremamente flessibile, consentendo di adeguarsi rapidamente a picchi di utilizzo o alla crescita dell'azienda.

Figura 1.2: Scalabilità verticale

In passato, le imprese erano solite sostenere ingenti investimenti per acquistare hardware e costruire le proprie infrastrutture di *Information Technology* (IT) per crescere. Dovevano acquistare server, allestire data center e avere personale esperto per gestire tutto. Questo comportava costi iniziali molto alti e un grosso rischio: se compravano troppa capacità, rischiavano di sprecare risorse; se ne compravano troppo poca, potevano avere problemi di servizio e non riuscire a gestire la domanda crescente.
Con il cloud computing, questo modello è cambiato drasticamente.In questo modo non devono più possedere o gestire infrastrutture fisiche, poiché possono accedere a risorse tecnologiche attraverso internet, pagando solo ciò che effettivamente utilizzano. Questo ha eliminato le barriere d'ingresso per molte startup e piccole imprese, permettendo loro di accedere a tecnologie avanzate senza la necessità di grandi investimenti.

Per le imprese, il cloud computing ha introdotto un cambiamento significativo. Le aziende possono ora operare in modo più agile, accelerare i tempi di sviluppo di nuovi prodotti e servizi e adattarsi rapidamente ai cambiamenti del mercato. Inoltre, con la capacità di espandere o ridurre le risorse su richiesta, le imprese possono rispondere immediatamente a picchi di domanda o cali imprevisti, migliorando l'efficienza operativa e la gestione dei costi.

I principali provider di servizi tecnologici, mostrati nella figura 1.3 hanno sfruttato l'enorme potenziale del cloud computing e hanno dedicato una parte sostanziale del loro business alla creazione di piattaforme cloud aperte agli utenti. Grandi aziende come Amazon Web Services (AWS), Microsoft Azure e Google Cloud hanno costruito vasti ecosistemi di servizi cloud, offrendo una gamma completa di soluzioni per il calcolo, l'archiviazione, l'analisi dei dati e molto altro.
Questi provider non solo forniscono l'infrastruttura necessaria per supportare le operazioni aziendali, ma hanno anche reso accessibili piattaforme e strumenti per sviluppare applicazioni, gestire i dati e sfruttare tecnologie emergenti come l'intelligenza artificiale e l'Internet of Things (IoT).

Figura 1.3: Principali providers sul mercato

Nelle seguenti sezioni verrà descritto un caso pratico in cui l'azienda Sinergia EPC, presso la quale ho svolto il tirocinio, ha scelto di adottare servizi cloud sia per le sue applicazioni esistenti che per i progetti futuri.

Lo scopo dell'azienda è stato quello di creare un ambiente altamente scalabile e sicuro utilizzando tecnologie innovativa e moderne.
Migrando gli applicativi su cloud AWS l'azienda ha potuto gestire le risorse in modo più efficiente sfruttando il metodo *pay-as-you-go* (pagamento basato sul consumo) ed ha potuto automatizzare alcuni processi che precedentemente richiedevano l'intervento di un tecnico.

Inoltre, L'utilizzo del cloud ha permesso all'azienda di modernizzare anche le applicazioni esistenti in modo più semplice, iniziando a progettare una containerizzazione con Docker e un'orchestrazione basata su software come Kubernetes. Questo approccio non solo facilita la gestione e il deployment delle applicazioni, ma consente anche di ottenere una maggiore scalabilità e resilienza. Grazie a Docker, le applicazioni possono essere isolate e facilmente distribuite, mentre Kubernetes automatizza la loro gestione, garantendo un ambiente più efficiente e reattivo.

Capitolo 2

Cloud computing

Il cloud computing rappresenta un modello che permette agli utenti di accedere facilmente, ovunque si trovino e in qualsiasi momento, a risorse informatiche condivise attraverso una rete. Queste risorse includono elementi fondamentali come reti, server, capacità di archiviazione, applicazioni e vari tipi di servizi. Una delle caratteristiche principali del cloud è che queste risorse possono essere fornite in modo rapido e flessibile, rispondendo immediatamente alle esigenze dell'utente. Inoltre, possono essere rilasciate o disattivate con altrettanta rapidità, senza richiedere interventi complessi da parte dell'utente o lunghe interazioni con il fornitore di servizi.

Il modello del NIST (National Institute of Standards and Technology) è uno dei più utilizzati e riconosciuti a livello globale per definire il cloud computing. La sua popolarità deriva dal fatto che fornisce una definizione chiara, completa e standardizzata delle caratteristiche, dei modelli di servizio e di distribuzione del cloud.
Questo modello menziona cinque caratteristiche essenziali del cloud computing, che sono:

- *Broad access network*: le risorse e le capacità del cloud sono accessibili tramite una rete internet e possono essere utilizzate da una varietà di dispositivi.L'accesso avviene attraverso meccanismi standard, come protocolli e interfacce, che facilitano la connessione senza la necessità di configurazioni complesse. Inoltre, questa caratteristica consente l'uso da parte di piattaforme eterogenee, quindi sia dispositivi leggeri, sia dispositivi più potenti possono accedere alle risorse del cloud.

- *Resource pooling*: le risorse del provider sono gestite per servire più consumatori contemporaneamente. Queste risorse vengono assegnate e riassegnate dinamicamente in base alla domanda del consumatore. Inoltre, esiste un senso di indipendenza dalla posizione: ciò significa che il cliente generalmente non ha il controllo né la conoscenza precisa di dove si trovano fisicamente le risorse che sta utilizzando.

- *Rapid elasticity*: le risorse possono essere fornite in modo elastico, talvolta anche in automatico, permettendo di scalare rapidamente verso l'esterno e verso l'interno in base alla domanda. Per il consumatore, le capacità disponibili sembrano spesso illimitate e possono essere utilizzate in qualsiasi quantità e in qualsiasi momento.

- *Measured service*: i sistemi cloud controllano e ottimizzano automaticamente l'uso delle risorse sfruttando una capacità di misurazione a un certo livello di astrazione, appropriato per il tipo di servizio (ad esempio, archiviazione e elaborazione).

- *On-demand self-service*: un "consumatore" può autonomamente fornire capacità di calcolo,in base alla necessità ed in modo automatico, senza avere alcuna interazione con il fornitore di servizi.

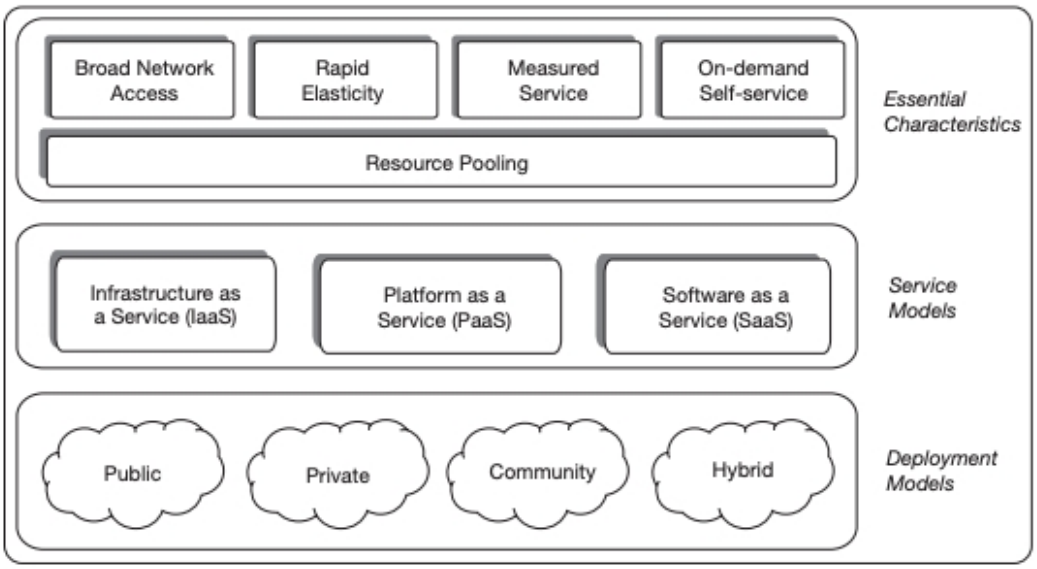


Figura 2.1: Modello di cloud computing del NIST [1]

2.1 Servizi di cloud Computing

Nel contesto del cloud computing, il termine "as-a-Service" (aaS) designa modelli di servizio che forniscono risorse informatiche, piattaforme e applicazioni attraverso Internet. Questi modelli includono Infrastructure as a Service (IaaS), Platform as a Service (PaaS) e Software as a Service (SaaS), ciascuno progettato per rispondere a specifiche esigenze degli utenti.
L'adozione di soluzioni "as-a-Service" consente alle organizzazioni di accedere a risorse scalabili e flessibili, eliminando la necessità di investimenti in infrastrutture fisiche e promuovendo un approccio più agile e cost-effective nella gestione delle tecnologie informatiche.
Nelle prossime sezioni, verranno analizzati i tre servizi principali messi a disposizione da qualsiasi piattaforma di cloud computing.

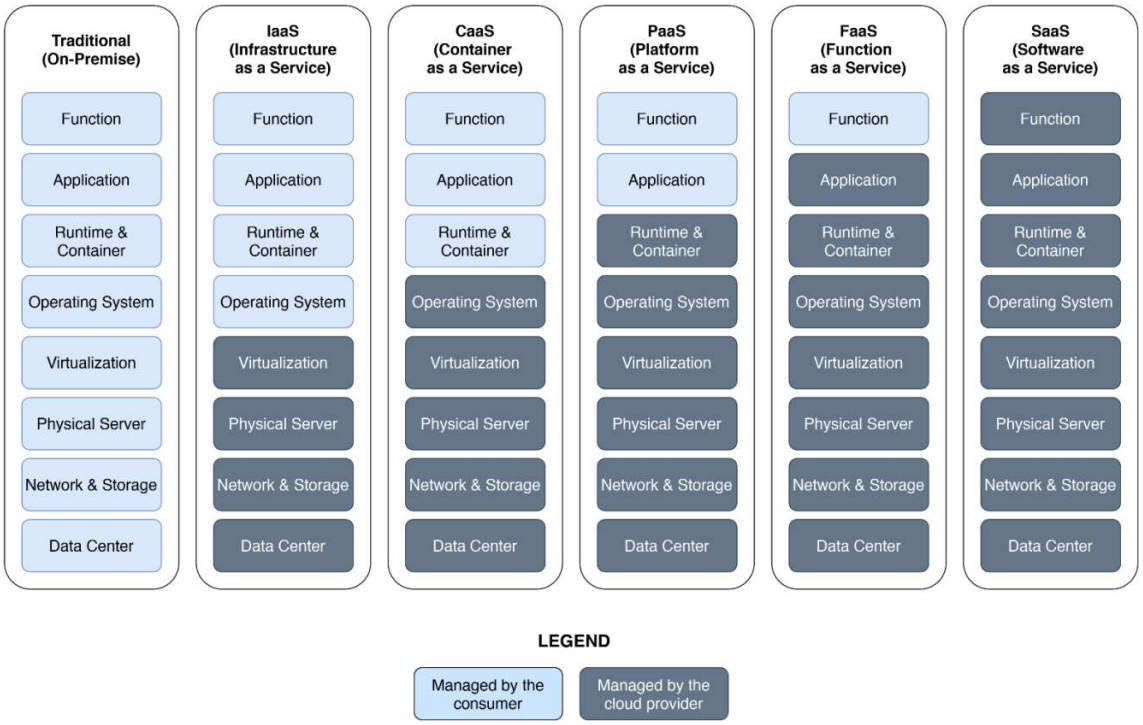


Figura 2.2: Astrazione dei servizi cloud [2]

2.1.1 Software as a Service

con i servizi Software as a Service, gli utenti non gestiscono né controllano l'infrastruttura cloud sottostante, che comprende rete, server, sistemi operativi e archiviazione.
A differenza dei servizi tradizionali, che richiedono una gestione diretta delle risorse e un'installazione locale, il SaaS offre un accesso immediato a software e applicazioni via internet, eliminando la necessità di installazioni complesse e manutenzioni hardware.

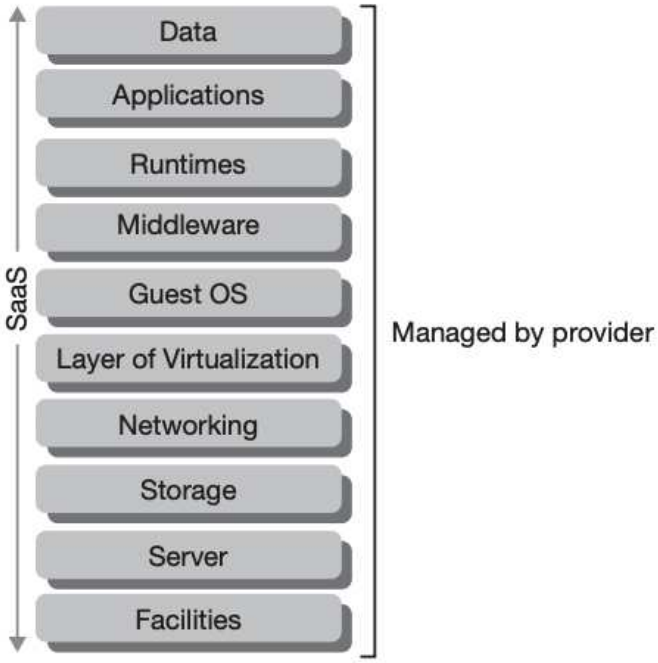


Figura 2.3: Componenti Saas [1]

2.1.2 Platform as a Service

si riferisce al sistema sottostante su cui possono essere installate e sviluppate applicazioni software.Il PaaS si distingue dal SaaS in quanto fornisce non solo l'accesso a software applicativo, ma anche un ambiente di sviluppo completo.

Mentre il SaaS consente agli utenti di utilizzare applicazioni gestite interamente dal fornitore,il PaaS offre strumenti e servizi per costruire e implementare nuove applicazioni. Gli utenti del PaaS hanno il controllo sui propri progetti di sviluppo, mentre il fornitore gestisce l'infrastruttura, il sistema operativo e le risorse hardware necessarie.

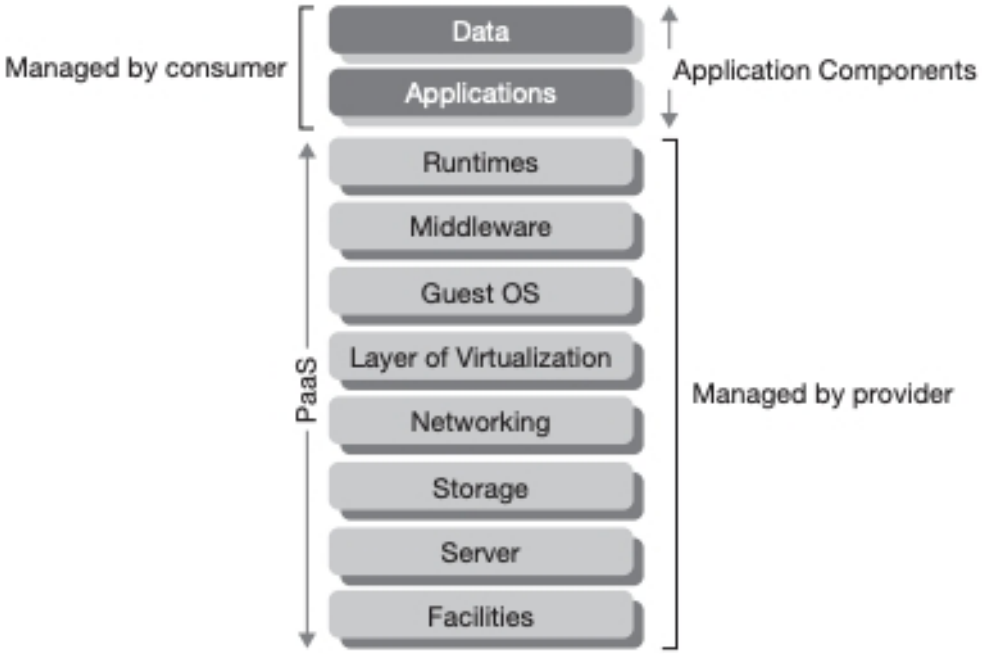


Figura 2.4: Componenti PaaS [1]

2.1.3 Infrastructure as a Service

L'Infrastructure-as-a-Service (IaaS) offre ai consumatori risorse hardware virtualizzate, consentendo loro di utilizzare a distanza risorse virtuali come processori, memoria, archiviazione e rete. Queste risorse possono essere impiegate come se fossero hardware fisico per configurare qualsiasi tipo di infrastruttura informatica.

Anche se il consumatore non gestisce direttamente l'infrastruttura cloud sottostante, mantiene il controllo sui sistemi operativi, sull'archiviazione e sulle applicazioni distribuite. In alcuni casi, ha anche un controllo limitato su componenti di rete, come i firewall degli host. È importante notare che l'hardware è condiviso tra più utenti, consentendo una gestione più efficiente delle risorse.

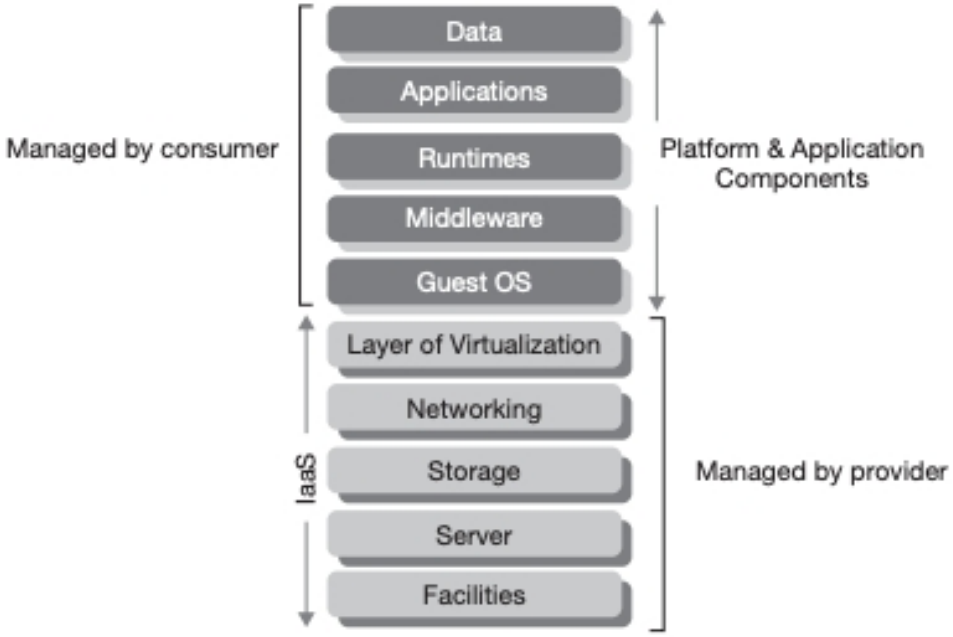


Figura 2.5: Componenti IaaS [1]

2.2 Tipi di Cloud Computing

Esistono diversi tipi di cloud computing, ognuno progettato per soddisfare specifiche esigenze aziendali. I principali sono il cloud pubblico, che offre risorse condivise accessibili tramite Internet; il cloud privato, dedicato esclusivamente a un'unica organizzazione; e il cloud ibrido, che combina le caratteristiche dei due modelli precedenti. Queste diverse tipologie consentono alle aziende di scegliere la soluzione più adatta alle loro necessità.

2.2.1 Cloud pubblico

Il cloud pubblico è un modello di cloud computing in cui le risorse informatiche, come server, archiviazione e applicazioni, sono fornite da un provider esterno e condivise tra più utenti o aziende. Questo modello è accessibile tramite Internet e non richiede investimenti iniziali significativi in hardware o infrastrutture da parte dell'utente finale. Gli utenti possono accedere alle risorse da qualsiasi luogo, purché abbiano una connessione Internet, rendendolo ideale per aziende che operano in diverse località o per lavoratori remoti.

Una delle principali caratteristiche del cloud pubblico è la scalabilità: offre la possibilità di aumentare o diminuire le risorse in base alle esigenze.
Se un'azienda ha bisogno di più spazio di archiviazione o potenza di calcolo, può facilmente richiedere risorse aggiuntive senza dover effettuare investimenti costosi in nuove attrezzature.Inoltre, il provider di cloud pubblico gestisce tutta l'infrastruttura, il che significa che le aziende non devono preoccuparsi della manutenzione, degli aggiornamenti o della sicurezza dei server.
Nonostante i vantaggi del cloud pubblico, esistono anche alcuni problemi che possono insorgere quando una azienda decide di adottare questo modello:

- **Condivisione delle risorse e privacy**: le risorse del cloud pubblico sono condivise tra più clienti, il che potrebbe sollevare preoccupazioni su accessi non autorizzati. Anche se i provider implementano misure per isolare i dati, esiste sempre il rischio che un cliente possa accedere ai dati di un altro. Per questo motivo è importante utilizzare ulteriori metodi di sicurezza informatica per mitigare i rischi.
- **Disponibilità e inattività**: anche se i provider di cloud pubblico mirano a garantire un'alta disponibilità, ci possono essere momenti di inattività o interruzioni del servizio. Le aziende devono pianificare come gestire queste situazioni per minimizzare l'impatto sulle loro operazioni.
- **Gestione dei costi**: sebbene il cloud pubblico possa sembrare conveniente, i costi possono aumentare rapidamente a seconda dell'uso delle risorse. È importante monitorare e gestire attentamente l'utilizzo per non avere costi eccessivi.

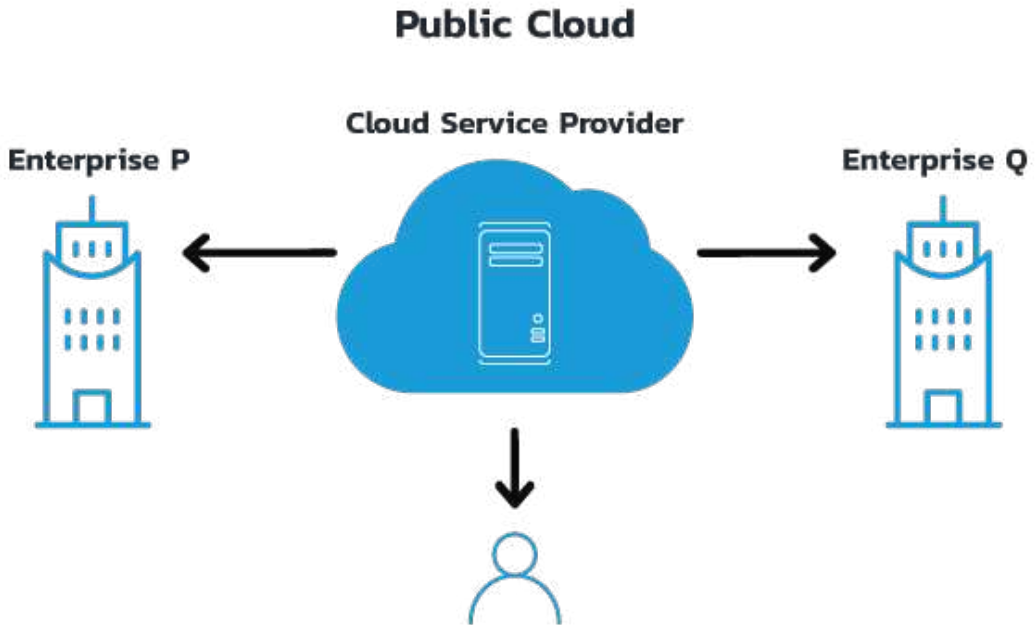


Figura 2.6: Cloud pubblico [3]

2.2.2 Cloud privato

Il cloud privato è un modello di cloud computing in cui le risorse informatiche, come server, archiviazione e applicazioni, sono dedicate esclusivamente a un'unica organizzazione. A differenza del cloud pubblico, in cui le risorse sono condivise tra più clienti, il cloud privato offre maggiore controllo e personalizzazione, poiché le aziende possono configurare l'infrastruttura in base alle proprie esigenze specifiche.

Una delle principali caratteristiche di questo cloud è la sicurezza.
Poiché le risorse non sono condivise con altri, come nel cloud pubblico, le aziende possono implementare misure di sicurezza più rigorose e avere un maggiore controllo sulla protezione dei dati sensibili.

Esistono diversi tipi di cloud privato, ognuno progettato per soddisfare specifiche esigenze aziendali, i principali sono:

- **Cloud privato on-premise**: In questo modello, l'infrastruttura cloud è ospitata all'interno dell'organizzazione, generalmente nei data center aziendali. Le aziende gestiscono e mantengono completamente le risorse, il che offre un alto livello di controllo e sicurezza.
  Lo svantaggio è che richiede significativi investimenti iniziali e competenze tecniche specifiche, questo potrebbe essere problematico per aziende di medie o piccole dimensioni.

- **Cloud privato comunitario (Community Cloud)**: In questo modello, l'infrastruttura cloud è condivisa tra più organizzazioni che hanno esigenze simili, come ad esempio aziende di un settore specifico. Le risorse sono gestite da una terza parte o dalle stesse organizzazioni partecipanti e sono progettate per soddisfare le esigenze comuni di sicurezza.

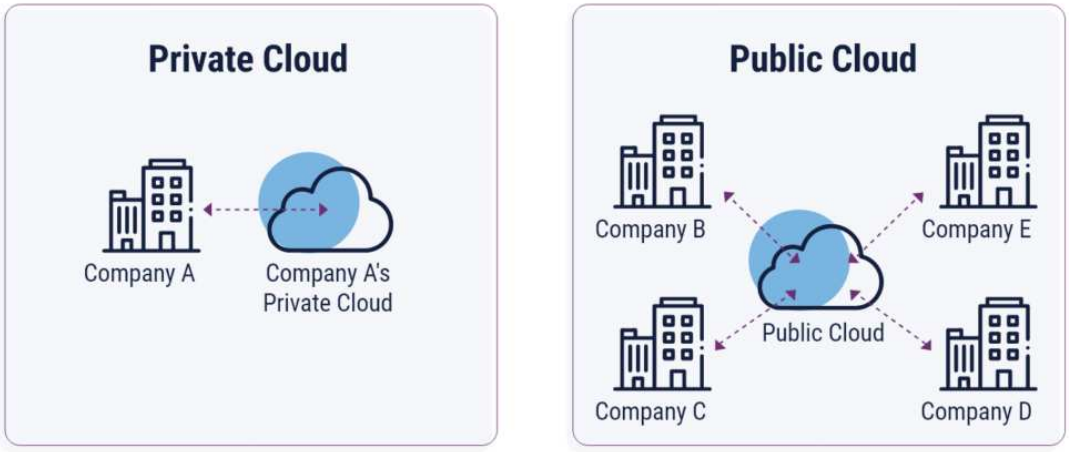


Figura 2.7: Differenza tra cloud privato e pubblico [4]

2.2.3 Cloud ibrido

Il cloud ibrido è un modello che combina elementi di cloud pubblico e cloud privato, consentendo alle aziende di sfruttare i vantaggi di entrambi. In un ambiente cloud ibrido, le organizzazioni possono mantenere alcune risorse e applicazioni nel cloud privato, mentre altre possono essere ospitate nel cloud pubblico. Questa flessibilità consente di bilanciare la sicurezza e il controllo con la scalabilità e i costi contenuti.
Una delle principali caratteristiche del cloud ibrido è la capacità di gestire e spostare i carichi di lavoro tra il cloud pubblico e quello privato in base alle esigenze aziendali. Ciò significa che le aziende possono allocare risorse nel cloud pubblico per gestire picchi di domanda, mantenendo al contempo i dati sensibili e le applicazioni critiche nel cloud privato.
Il modello ibrido offre una maggiore resilienza e continuità operativa, poiché consente alle aziende di implementare strategie di disaster recovery più efficaci. Grazie alla possibilità di replicare dati e applicazioni tra i diversi ambienti, le organizzazioni possono garantire

un recupero rapido in caso di problemi. Inoltre, questo approccio è anche più economico, poiché le aziende possono gestire autonomamente queste strategie senza dover investire in costose infrastrutture secondarie.

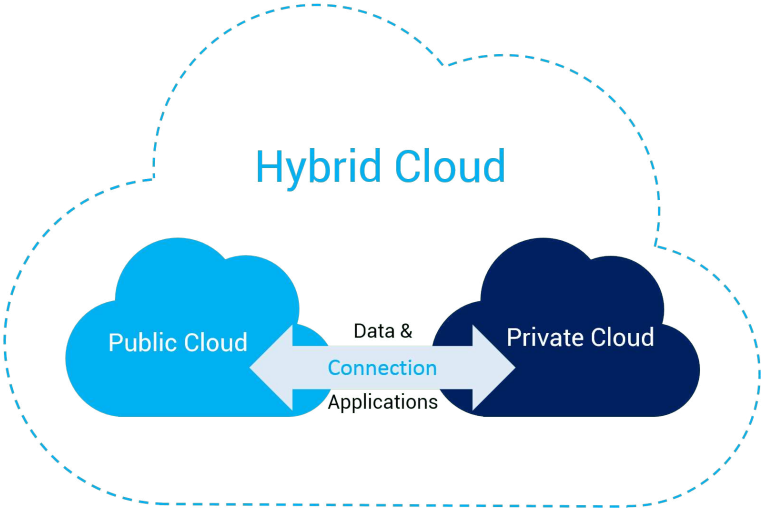


Figura 2.8: Cloud ibrido [5]

2.3 Sicurezza del cloud

a sicurezza nel cloud coinvolge la gestione di vari aspetti cruciali, tra cui la protezione dei dati, il controllo degli accessi e la sicurezza delle reti. In questa sezione, l'attenzione sarà rivolta alla gestione degli utenti e alla sicurezza della rete, elementi fondamentali per garantire un accesso sicuro e controllato alle risorse cloud.

2.3.1 Gestione delle identità e degli accessi

La gestione delle identità e degli accessi (*Identity Access Management*) è un insieme fondamentale di controlli di sicurezza che si concentra sulla verifica delle identità e sulla gestione dei loro accessi alle risorse. Un'analogia appropriata per comprendere IAM è il sistema di badge utilizzato per accedere a un edificio.
Per entrare, bisogna presentare un badge (identità) a una guardia (sistema di autenticazione) che verifica l'identità. Una volta confermata, si ottiene l'accesso all'edificio, ma non necessariamente a tutte le aree. L'accesso specifico a determinate aree o risorse (autorizzazione) dipende dal tipo di badge e dalle politiche del sistema di sicurezza.
È importante dare una definizione di gestione dell'identità e dell'accesso:

- **Identità**: in un contesto IAM, rappresenta un'entità univoca, come un utente, un amministratore o un sistema.
- **Accesso**: si riferisce al permesso concesso a un'identità di interagire con specifiche risorse o di eseguire determinate azioni. L'accesso è concesso in base all'identità autenticata e alle policy configurate nel sistema.

Il ciclo di vita di IAM (Identity and Access Management), come mostrato in figura 2.9, comprende diverse fasi. Sebbene l'autenticazione e l'autorizzazione siano spesso al centro dell'attenzione, sono solo una parte del ciclo di vita IAM. Le principali fasi del ciclo sono:

- **Richiesta**: Un'entità richiede l'accesso e la richiesta dovrebbe in genere essere autenticata.Le richieste comuni includono la creazione di identità, l'eliminazione di identità, la concessione di accesso e la revoca dell'accesso

- **Approvazione**: funge da checkpoint per garantire che solo le richieste di accesso legittime vengano elaborate. Sebbene alcuni sistemi possano concedere l'approvazione automaticamente (con l'utilizzo di CAPTCHA o convalida dell'e-mail), in particolare per le richieste di base o di basso impatto, le richieste più sensibili richiedono in genere un'approvazione esplicita.

- **Crea/Elimina/Concedi/Revoca**: Dopo l'approvazione, l'azione richiesta viene eseguita, come la creazione di un'identità o la concessione dell'accesso.

- **Autenticazione**: l'autenticazione riguarda la verifica dell'identità, assicurandosi che l'entità sia chi dice di essere.Ciò può essere ottenuto tramite vari metodi, come password, certificati digitali o autenticazione Multi Fattoriale (*Multi-Factor Authentication*).

- **Autorizzazione**: una volta autenticata, l'entità viene autorizzata a eseguire determinate azioni in base alle policy e ai ruoli assegnati.

- **Convalida**: La convalida implica la revisione e la conferma periodica che gli utenti e l'automazione abbiano ancora le autorizzazioni corrette. Questo passaggio è importante sia negli ambienti tradizionali che in quelli cloud. La convalida può essere basata su parametri come la data di scadenza dell'accesso o su un giudizio umano.

- **Riassegnazione/Revoca**: infine, l'accesso può essere riassegnato a una nuova identità, revocato quando non è più necessario o quando l'identità non necessita più di accesso.

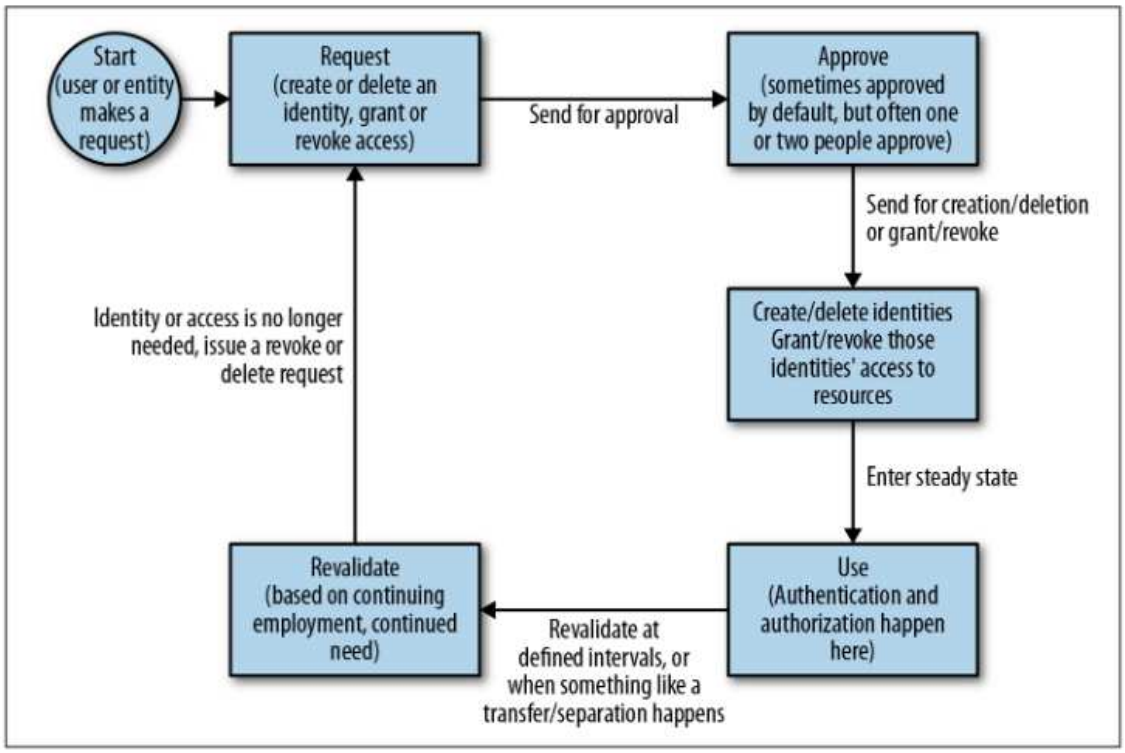


Figura 2.9: Ciclo di vita IAM [6]

Questo ciclo di vita assicura che solo le entità autorizzate possano accedere alle risorse, e che l'accesso venga revocato quando non è più necessario.
È importante sottolineare che i processi specifici del ciclo di vita IAM possono variare a seconda del fornitore di cloud o dell'applicazione utilizzata, ma i principi di base restano invariati.
Nella tabella 2.1 sono stati inseriti i servizi dei provider cloud più utilizzati.

Provider	**Cloud identity System**
Amazon Web Services	Amazon IAM
Microsoft Azure	Azure Active Directory B2C
Google Compute Cloud	Cloud Identity
IBM Cloud	Cloud IAM

Tabella 2.1: Tabella Provider di servizi IAM

2.3.2 Sicurezza della rete: Firewall e Segmentazione

Per garantire la sicurezza di una architettura cloud due delle pratiche più efficaci per garantire un ambiente sicuro sono l'implementazione di firewall e la segmentazione della rete.

Le soluzioni di firewall nel cloud possono includere funzionalità avanzate come il filtraggio del traffico, la prevenzione delle intrusioni e la protezione dalle vulnerabilità. Attraverso la configurazione di regole di accesso rigorose, i firewall possono bloccare il traffico non autorizzato e consentire solo le comunicazioni necessarie tra i servizi e gli utenti, garantendo

così una protezione efficace contro gli attacchi informatici.

Provider	Servizio di Firewall
Amazon Web Services	AWS WAF
	AWS Firewall Manager
Microsoft Azure	Azure Firewall
	Azure Application Gateway
Google Cloud Platform	Google Cloud Armor
IBM Cloud	IBM Cloud Internet Services
Oracle Cloud	Oracle Cloud Firewall

Tabella 2.2: Tabella dei principali servizi di firewall nei provider cloud

La segmentazione della rete consiste nel suddividere una rete in segmenti più piccoli e isolati, ognuno dei quali può avere le proprie politiche di sicurezza e controlli di accesso. Questa pratica limita il movimento laterale degli attaccanti all'interno della rete, poiché un'intrusione in un segmento non implica necessariamente l'accesso a tutti gli altri.

Provider	Servizio di Segmentazione della Rete
Amazon Web Services	Amazon VPC
	Security Groups
	Network ACLs
Microsoft Azure	Azure Virtual Network
	Network Security Groups (NSGs)
Google Cloud Platform	GCP VPC
	Firewall Rules
IBM Cloud	IBM Cloud VPC
	Security Groups
Oracle Cloud	Oracle Cloud Infrastructure VCN
	Security Lists

Tabella 2.3: Tabella dei principali servizi di segmentazione della rete nei provider cloud

Capitolo 3

Rehosting su ambiente AWS

In questo capitolo verranno descritte le strategie per la migrazione dell'applicazione Sinergia Cloud, la creazione di un'istanza per l'applicazione in fase di sviluppo Insideep, e l'implementazione di alcuni nuovi servizi.

L'infrastruttura sfrutta Amazon VPC per la gestione delle subnet in cui collocare le applicazioni, Amazon EC2 come macchine virtuali per l'hosting degli applicativi, e Amazon RDS per la gestione dei database, sia esistenti che nuovi.

Le funzioni AWS Lambda sono state implementate principalmente per gestire i log delle applicazioni e del database, nonché per effettuare i backup di RDS. Tutte le istanze attive vengono monitorate tramite messaggi di log utilizzando il servizio Amazon CloudWatch.

I file e i backup vengono archiviati su Amazon S3, un servizio di storage scalabile. Per garantire la sicurezza e l'efficienza, sono stati gestiti il traffico in ingresso e in uscita, insieme alla configurazione delle subnet in cui devono essere collocate le risorse.

Infine, nella sezione iniziale, sono stati introdotti i principali strumenti di containerizzazione e orchestrazione che sono stati studiati durante lo sviluppo dell'infrastruttura.

3.1 DevOps

DevOps è un approccio che combina lo sviluppo software (Development) con le operazioni IT (Operations) allo scopo di migliorare il processo di produzione del software. Si tratta di una metodologia che mira a integrare i team di sviluppo e quelli operativi per lavorare in modo collaborativo e continuo, rompendo le barriere tradizionali tra le due aree. L'obiettivo è accelerare il ciclo di sviluppo, migliorare la qualità del software e garantire che il prodotto finale sia rilasciato in modo più rapido ed efficiente.

I fondamentali del DevOps si basano su una serie di principi e pratiche che hanno come obiettivo l'integrazione tra sviluppo e operazioni per migliorare l'efficienza del ciclo di vita del software. I punti chiave sono i seguenti:

- **Collaborazione continua**: promuove una stretta collaborazione tra sviluppatori, operatori IT e altri stakeholder per abbattere i silos tra i team. Questa collaborazione permette di condividere responsabilità e obiettivi comuni, migliorando la comunicazione e il coordinamento

- **Automazione**: si automatizzano attività come la costruzione del software, il testing, il deployment e il monitoraggio per rendere i processi più veloci e affidabili, riducendo al minimo l'errore umano e aumentando l'efficienza.
- **Continuous Integration**: questo principio consiste nell'integrare frequentemente le modifiche al codice in un repository condiviso.
- **Continuous Delivery**: mira a rendere il software pronto per essere rilasciato in qualsiasi momento. Il codice viene continuamente distribuito e testato in ambienti di sviluppo o produzione, riducendo i tempi tra i rilasci e migliorando l'affidabilità dei rilasci.

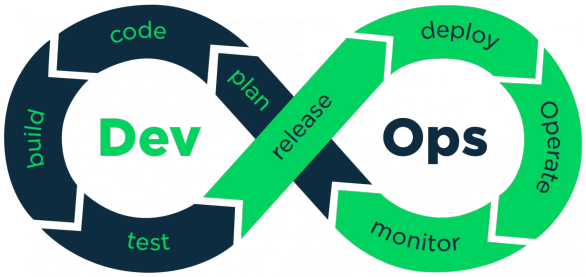


Figura 3.1: DevOps [7]

3.1.1 Macchine virtuali e container

Le macchine virtuali (VM) e i Container rappresentano due tecnologie di virtualizzazione profondamente diverse, ognuna con caratteristiche e finalità distinte. La principale differenza tra queste risiede nella loro architettura: mentre una macchina virtuale esegue un intero sistema operativo sopra un hypervisor, Docker utilizza i container, che condividono il kernel del sistema operativo host ma operano in ambienti isolati.

Le macchine virtuali emulano un'intera infrastruttura hardware, inclusi processore, memoria, dischi e schede di rete, e ogni VM include un proprio sistema operativo completo. Ciò comporta un maggiore utilizzo di risorse e tempi di avvio più lunghi, in quanto è necessario avviare non solo l'applicazione, ma anche il sistema operativo stesso.
I container, al contrario, contengono solo le librerie e le dipendenze strettamente necessarie per eseguire un'applicazione, appoggiandosi al kernel del sistema operativo dell'host. Questo li rende più leggeri e rapidi da avviare rispetto alle VM, riducendo significativamente il consumo di risorse.

In termini di isolamento, le macchine virtuali forniscono un grado elevato di separazione, poiché ogni VM opera in maniera completamente autonoma con il proprio sistema operativo. Questo consente di eseguire sistemi operativi differenti su una singola macchina fisica. I container, invece, offre un isolamento a livello di applicazione, condividendo però il kernel del sistema operativo host, il che limita la possibilità di eseguire container con sistemi operativi diversi da quello dell'host.

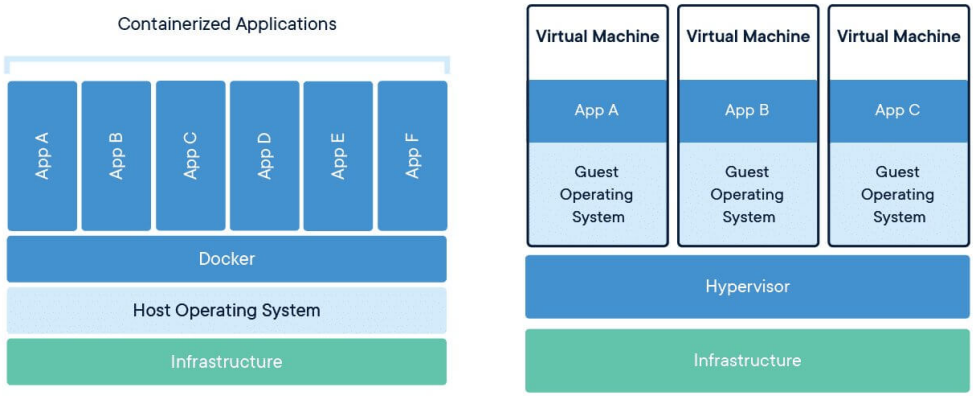


Figura 3.2: Differenza tra macchine virtuali e container [8]

3.1.2 Docker

Docker è una piattaforma open-source che consente di automatizzare la distribuzione di applicazioni all'interno di container, che sono ambienti isolati e leggeri che contengono tutto il necessario per eseguire un'applicazione. Un container può essere visto come una versione ridotta e portatile di una macchina virtuale, ma molto più efficiente in termini di risorse.
le componenti principali di Docker sono:

- **Docker engine**: è il cuore di Docker, il motore che consente di creare e gestire container. È composto da un daemon (processo che esegue in background) che gestisce container, immagini e reti.
- **Immagini**: un'immagine è un pacchetto che contiene tutto il necessario per eseguire un'applicazione, come il codice, le librerie, le variabili di ambiente e altre dipendenze. Le immagini sono create da un file chiamato Dockerfile.
- **Container**: un container è l'istanza di un'immagine, un'applicazione in esecuzione all'interno di un ambiente isolato. Può essere avviato, fermato e distrutto indipendentemente dall'ambiente ospitante.
- **Registry**: è un repository dove vengono memorizzate e condivise le immagini Docker. Il più comune è Docker Hub, che offre immagini pronte all'uso.

La creazione di immagini e container, così come il download delle immagini da Docker Hub, avviene tramite i seguenti comandi:

- *docker run*: questo comando serve a creare ed eseguire un container partendo da un'immagine.
- *docker build*: questo comando viene utilizzato per scaricare un'immagine Docker da un repository (come Docker Hub) alla tua macchina locale.
- *docker pull*: questo comando viene utilizzato per scaricare un'immagine Docker da un repository (come Docker Hub) alla tua macchina locale.

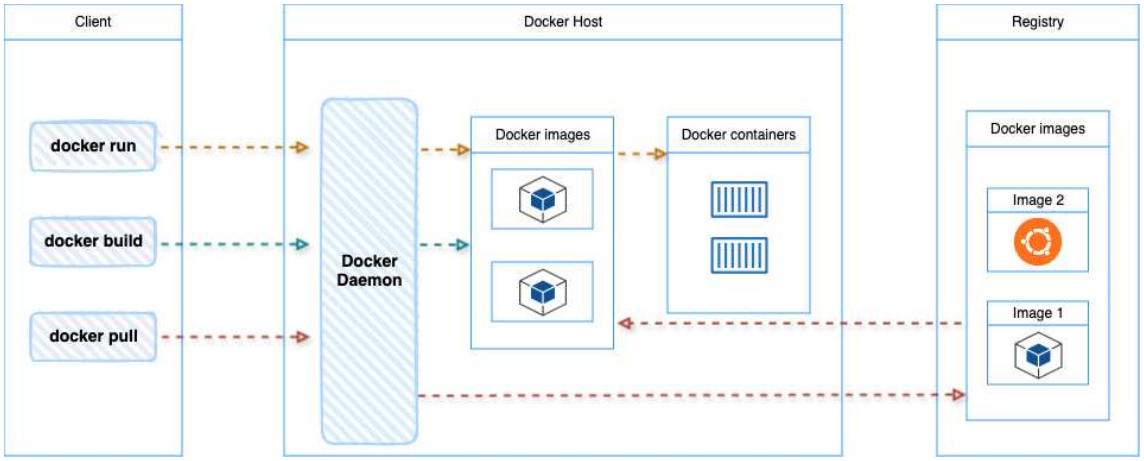


Figura 3.3: Architettura docker [9]

Una delle principali sfide nello sviluppo software è garantire che il codice funzioni correttamente in ambienti diversi. Docker affronta questa problematica attraverso l'uso di immagini "containerizzate". Ogni immagine Docker è immutabile e può essere "versionata", il che significa che è possibile mantenere diverse versioni dello stesso software senza conflitti. In particolare, le immagini permettono:

- *Rollback*: se una nuova versione di un'applicazione presenta problemi, Docker consente di eseguire un rollback rapido a una versione precedente semplicemente eseguendo il comando docker run con l'immagine della versione precedente.
- *Testing e CI/CD*: Docker si integra perfettamente con strumenti di CI/CD, facilitando la creazione di pipeline automatizzate che testano e distribuiscono le nuove versioni del software. Ogni volta che viene effettuata una modifica nel codice, un nuovo container può essere creato e testato automaticamente, garantendo che ogni versione sia sempre pronta per la produzione

3.1.3 Problema Docker Daemon

l principale problema di Docker è la dipendenza dal Docker Daemon, che richiede privilegi di `root` per eseguire container. Questo presenta potenziali rischi di sicurezza, poiché il daemon, se compromesso, può dare accesso amministrativo all'intero sistema.
Inoltre, essendo un processo centrale, un malfunzionamento del daemon può influenzare tutti i container in esecuzione, rendendo il sistema meno resiliente rispetto ad alternative come Podman, che non necessitano di un daemon centralizzato.

3.1.4 Podman

Podman (Pod Manager) è uno strumento open-source per la gestione di container che si propone come un'alternativa a Docker. Podman permette di eseguire, gestire e orchestrare container e pod (gruppi di container) senza la necessità di un daemon centrale. Il suo design è stato sviluppato per essere compatibile con l'ecosistema dei container e supportare gli stessi flussi di lavoro di Docker.
Sebbene Docker e Podman siano entrambi strumenti di gestione dei container, presentano alcune differenze fondamentali:

- **Assenza di Daemon**: l'assenza di un daemon significa che non esiste un processo centrale in background (come il Docker Daemon) che gestisce tutti i container. Invece, ogni container in Podman è eseguito come un processo separato e indipendente.
- **Esecuzione senza privilegi di root**: permette di eseguire container come utente non privilegiato, il che aumenta la sicurezza del sistema. Gli utenti possono eseguire container senza ottenere accesso root, riducendo i rischi di vulnerabilità.
- **Gestione di insiemi di container**: gestisce i pod direttamente, consentendo di raggruppare più container in una singola unità che condivide risorse come rete e storage.

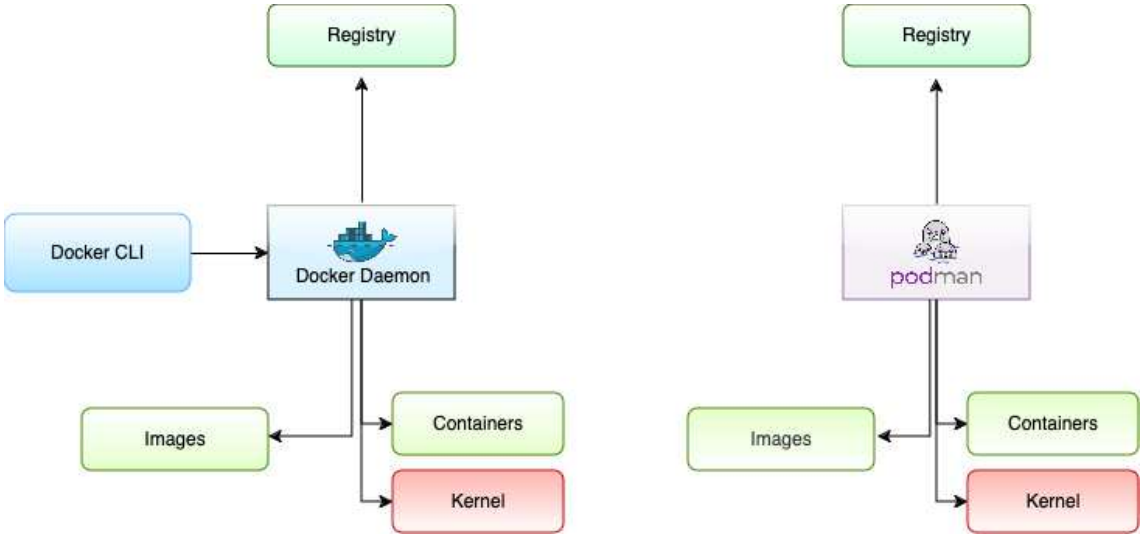


Figura 3.4: Differenza tra Podman e Docker

3.1.5 Orchestrazione

L'orchestrazione dei container si riferisce alla gestione automatizzata del ciclo di vita dei container, che include il loro provisioning, il deployment, la scalabilità, il bilanciamento del carico e la gestione degli errori. Quando si lavora con applicazioni "containerizzate", specialmente in ambienti di produzione su larga scala, è necessario coordinare centinaia o migliaia di container che devono funzionare in modo coerente e scalabile. L'orchestrazione dei container facilita questo processo, garantendo che i container siano distribuiti in modo efficiente su più nodi di un'infrastruttura e che possano essere monitorati e gestiti automaticamente. I software più utilizzati per l'orchestrazione sono:

- *Kubernetes*: È la piattaforma di orchestrazione più popolare e completa. Sviluppata da Google e ora mantenuta dalla Cloud Native Computing Foundation (CNCF), Kubernetes gestisce il deployment, la scalabilità e la gestione dei container su cluster. È particolarmente indicata per applicazioni distribuite complesse, grazie alle sue potenti funzionalità di gestione automatica del carico di lavoro, bilanciamento del traffico e autoscaling.
- *Docker Swarm*: Questo strumento è nativo di Docker e offre una soluzione di orchestrazione più semplice rispetto a Kubernetes. Docker Swarm consente di gestire cluster di Docker e di distribuire e scalare container in maniera automatizzata. Nella figura 3.5 è possibile vedere una rappresentazione dell'architettura.

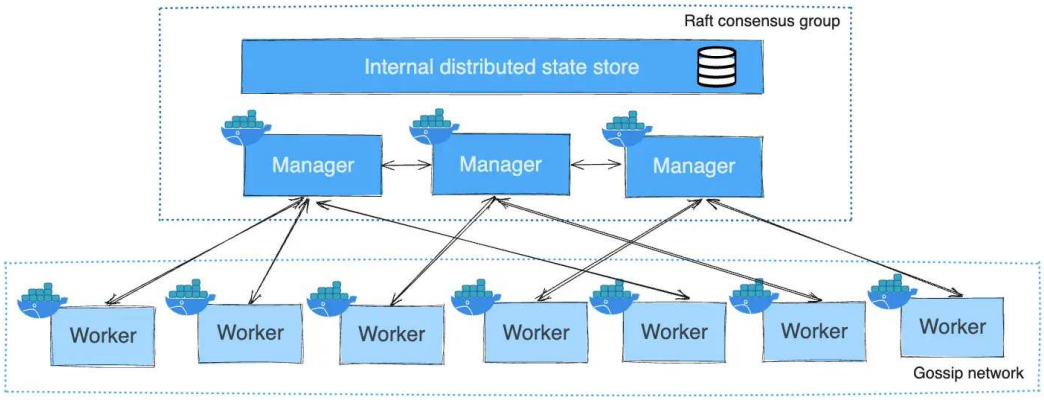


Figura 3.5: Docker Swarm [10]

3.2 Introduzione della migrazione e dello sviluppo su AWS

Il progetto si articola in due parti principali.
La prima parte riguarda la migrazione di server e database da un vecchio cloud a uno nuovo, dove è stata sviluppata una nuova architettura. Questa include una VPC con subnet pubbliche e private, una VPN per la connettività e una gestione della sicurezza per le funzioni serverless, il collegamento a storage S3 e l'interfacciamento tra le istanze EC2 e il database RDS. Inoltre, sono state implementate delle funzioni serverless tramite AWS Lambda, che garantiscono uno storage persistente contenente tutte le versioni dei dati presenti in RDS.

Si è optato per un approccio *lift&shift* di migrazione,che consiste nel trasferire le applicazioni e i dati associati al cloud con modifiche minime o inesistenti. Le applicazioni vengono essenzialmente "sollevate" dagli ambienti esistenti e "spostate" così come sono in una nuova infrastruttura.
Di conseguenza, non vengono apportate modifiche significative all'architettura dell'applicazione, al flusso dei dati o ai meccanismi di autenticazione esistenti.

La seconda parte del progetto prevede la creazione di un cloud per la gestione di un nuovo applicativo aziendale, accessibile anch'esso tramite VPN. L'autenticazione e l'autorizzazione per il nuovo applicativo sono state gestite attraverso Amazon Cognito, che consente di gestire sia i dati degli utenti che l'accesso ad API e servizi di terze parti tramite chiavi e credenziali.
Le infrastrutture create risultano molto simili tra loro. Pertanto, nelle sezioni successive vengono descritti i servizi di una sola parte. Tuttavia, per quanto riguarda l'autenticazione e l'autorizzazione, è stata necessaria una descrizione separata, poiché per il nuovo applicativo della seconda parte del progetto è stato necessario implementare un sistema differente.

3.3 Migrazione AWS

L'esigenza dell'azienda è stata quella di migrare i servizi verso un nuovo cloud più strutturato e sicuro, al fine di ottimizzare le operazioni e migliorare la gestione delle risorse. La decisione di adottare un'infrastruttura AWS più robusta è stata guidata dalla necessità

di garantire maggiore affidabilità, scalabilità e sicurezza, oltre a facilitare l'integrazione di nuove tecnologie e servizi.

La migrazione dell'istanza EC2 e del database RDS è stata realizzata utilizzando due strategie simili, ma con l'impiego di servizi differenti.
Per l'istanza EC2, è stata creata un'immagine (AMI) che è stata successivamente condivisa con il nuovo account. Per quanto riguarda il database RDS, è stato eseguito un backup, salvato in un bucket S3, e successivamente condiviso con un bucket appositamente creato nel nuovo account.

La figura 3.8 illustra i passaggi eseguiti per la migrazione, insieme alla nuova architettura, che verrà descritta in dettaglio nelle sezioni successive.

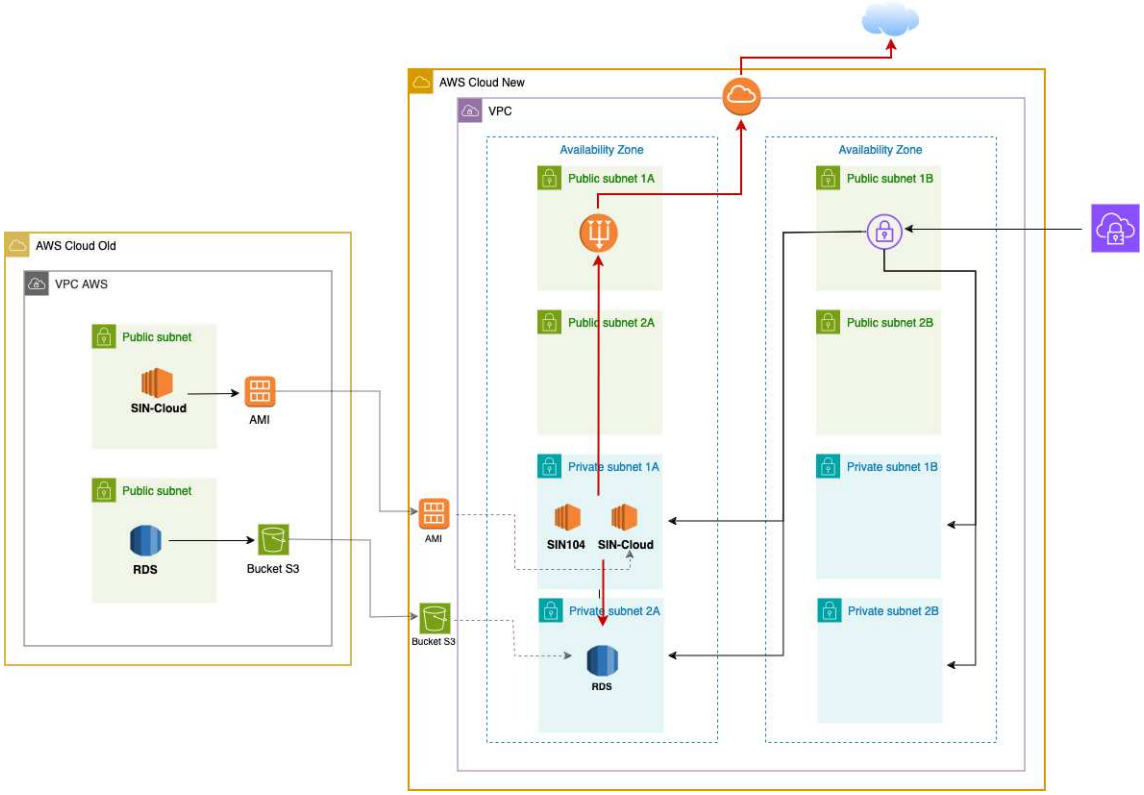


Figura 3.6: Architettura del nuovo cloud

3.4 Gestione della sicurezza in AWS

Per la gestione della sicurezza nel cloud sono stati utilizzati due tipi di sistemi di protezione:

- **Security Groups**: gruppi di regole per gestire la sicurezza a livello di istanza.
- **Access Control List**: regole utilizzate per gestire la sicurezza a livello di subnet.

3.4.1 Security groups

I Security Groups sono un meccanismo di sicurezza utilizzato per controllare il traffico in entrata e in uscita dalle risorse in una rete. In generale, questi gruppi agiscono come un

firewall virtuale, regolando quali connessioni sono consentite verso o dalle risorse, basandosi su regole definite dall'utente. Queste regole possono specificare il tipo di traffico, le porte, i protocolli e gli indirizzi IP di origine o destinazione.

In Amazon Web Services, i Security Groups rappresentano uno dei pilastri della sicurezza per le risorse all'interno di una Virtual Private Cloud. Essi svolgono un ruolo simile a un firewall virtuale, limitando il traffico in entrata e in uscita dalle risorse associate, come istanze EC2, database RDS, load balancer e molti altri servizi. Attraverso l'uso di regole configurabili, i Security Groups consentono agli amministratori di controllare in maniera granulare l'accesso alle risorse, mantenendo un alto livello di sicurezza .
Le caratteristiche principali di questo metodo di sicurezza sono:

- **Associazione a risorse**: i security groups in AWS sono strettamente legati alle risorse all'interno di una VPC. Ogni risorsa può essere associata a uno o più gruppi, ciascuno con il proprio set di regole per il controllo del traffico. Questo significa che le risorse possono condividere lo stesso security group oppure avere configurazioni dedicate, a seconda delle necessità di sicurezza del sistema.
- **Filtraggio del traffico**: le regole possono consentire o bloccare il traffico proveniente da specifici indirizzi IP, da intere subnet o da altre risorse all'interno della stessa VPC. All'interno di questi gruppi è possibile specificare protocolli, porte o indirizzi IP su cui devono agire.
- **Funzionamento Stateful**: se una connessione in entrata è consentita, anche la connessione di risposta verrà automaticamente permessa. Non è necessario creare regole distinte per il traffico in uscita associato a una richiesta in entrata.

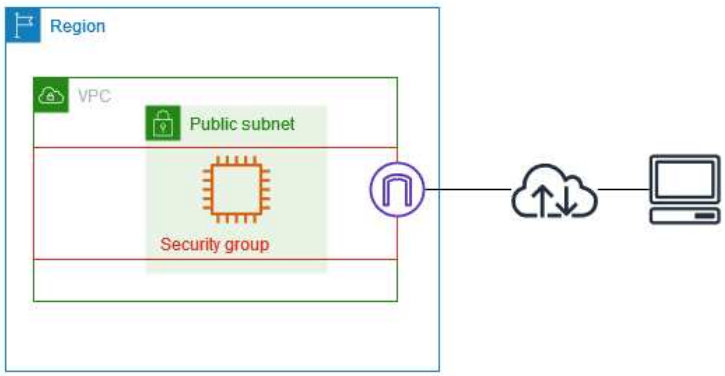


Figura 3.7: Funzionamenti dei Security Groups [11]

3.4.2 Access Control List (ACL)

Le Network ACL (Access Control Lists) in AWS sono un altro strumento di sicurezza utilizzato all'interno di una VPC. A differenza dei Security Groups, che operano a livello di istanza, le ACL operano a livello di subnet e forniscono un controllo stateless sul traffico di rete in entrata e in uscita per tutte le risorse all'interno della subnet.

Le ACL supportano sia regole di allow (per consentire il traffico) che di deny (per bloccare il traffico), in questo caso è necessario specificare sia le regole di entrata che di uscita.

Queste regole possono quindi essere utilizzate per bloccare tipi specifici di traffico o indirizzi IP, come ad esempio negare l'accesso da determinati intervalli di IP o bloccare protocolli specifici

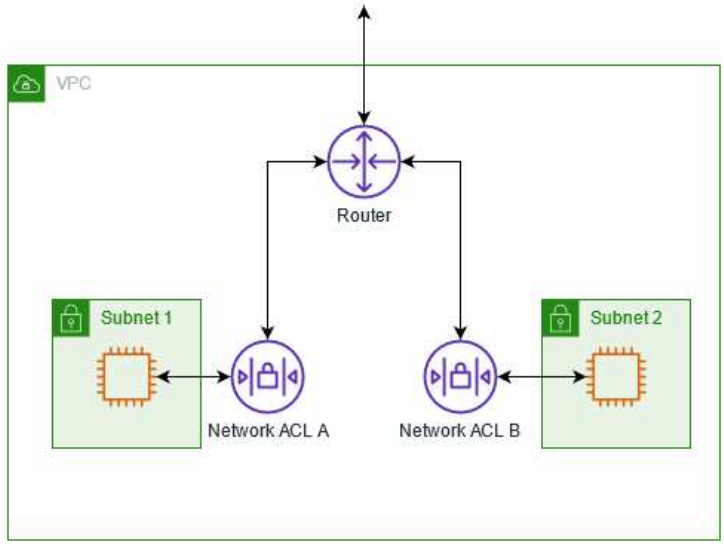


Figura 3.8: Funzionamento ACL in AWS [12]

3.5 Amazon VPC

Il primo passo nella creazione della nuova infrastruttura è stato realizzare una Virtual Private Cloud (VPC) utilizzando il servizio offerto da Amazon Web Services (AWS). Una VPC è una rete virtuale privata all'interno del cloud AWS, progettata per fornire un ambiente isolato e sicuro in cui distribuire risorse come istanze EC2, database e altri servizi del cloud. La VPC permette di avere il controllo completo sulla configurazione della rete, consentendo di definire come le risorse comunicano tra loro e con il mondo esterno.

La creazione di una VPC inizia con l'assegnazione di un intervallo di indirizzi IP privati utilizzando il formato CIDR (Classless Inter-Domain Routing). Questo sistema permette di specificare un blocco di indirizzi IP, come ad esempio 10.0.0.0/16, che definisce la gamma di indirizzi IP disponibili all'interno della VPC.

All'interno della VPC si creano delle subnet,come quella presente nella figura 3.9, che sono segmenti più piccoli della rete in cui possono essere distribuite le risorse.

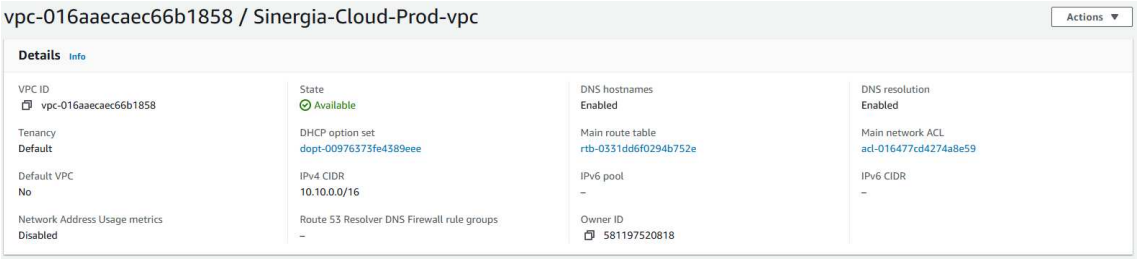


Figura 3.9: VPC con indirizzi 10.10.0.0/16

Le subnet possono essere di tue tipi:

- **Subnet pubbliche**: segmenti di rete hanno accesso diretto a internet tramite un internet gateway e che permetto l'accesso verso l'esterno alle risorse collocate in tali subnet.
- **Subnet private**: segmenti di rete isolati che possono accedere a internet solo indirettamente, solitamente attraverso un gateway NAT, come mostrato in figura 3.10.

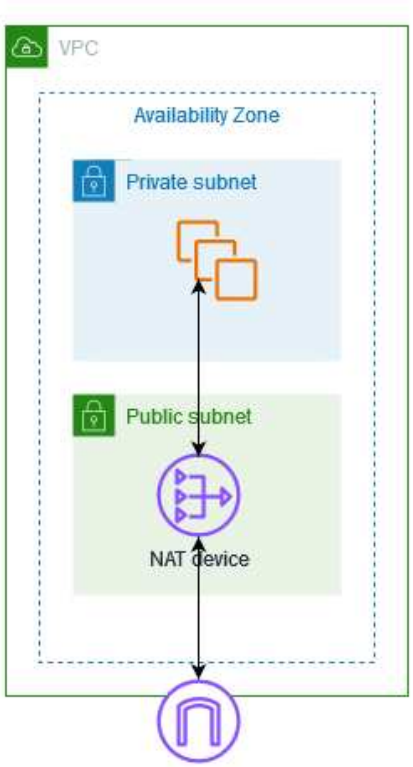


Figura 3.10: gateway NAT per subnet private

Nella figura 3.11 è possibile notare che le subnet sono distribuite in 2 *Availability Zone* (AZ), che sono data center fisici separati all'interno di una regione AWS. A differenza delle subnet, che sono entità logiche della VPC, le AZ rappresentano l'infrastruttura fisica di AWS.
Ogni subnet appartiene a una sola zona di disponibilità, il che significa che le risorse in quella subnet sono fisicamente allocate in un data center specifico.

Distribuire risorse su più subnet in diverse AZ aumenta la disponibilità e la tolleranza ai guasti, perché in caso di problemi in una delle zone, le risorse in un'altra AZ continueranno a funzionare.

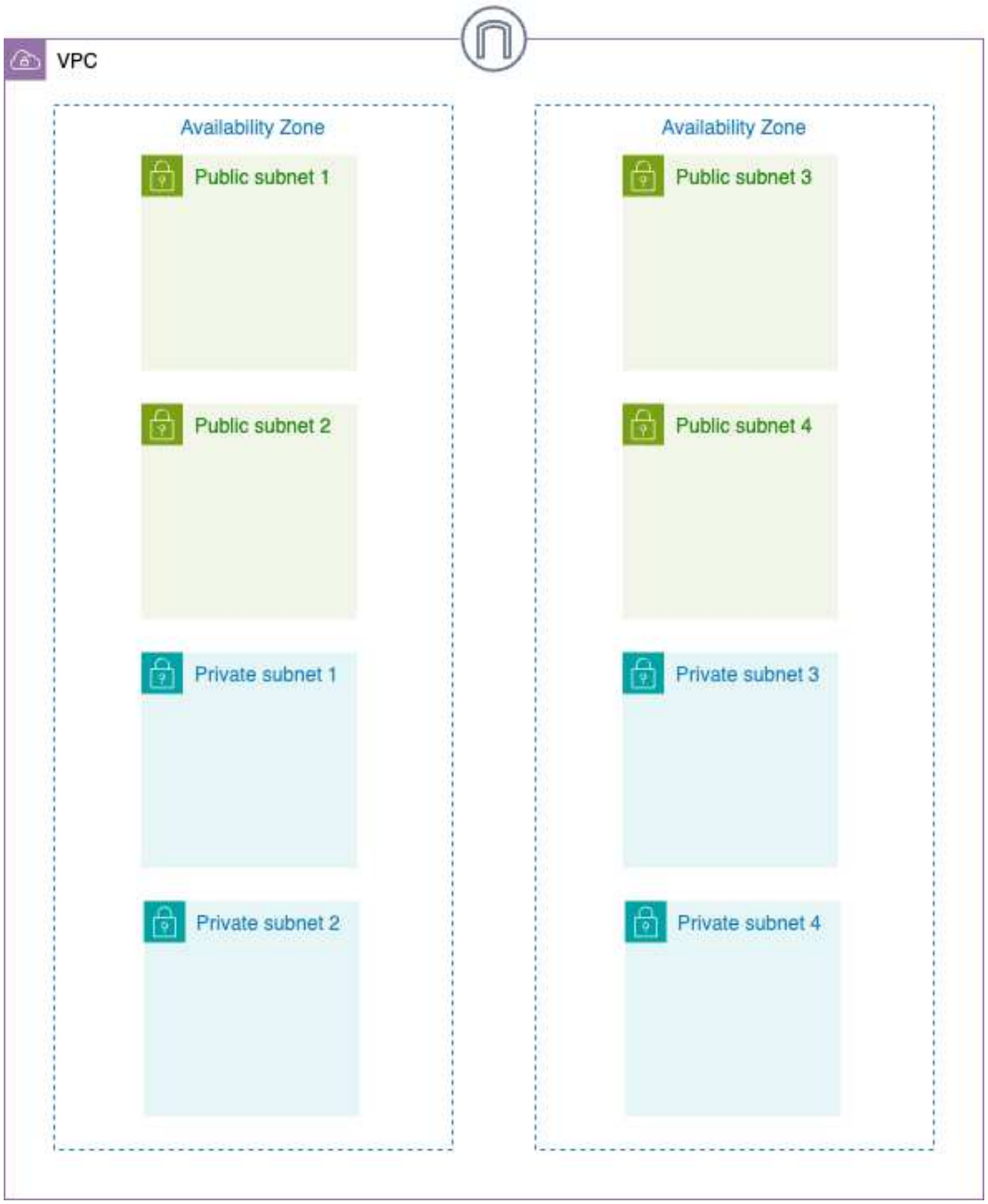


Figura 3.11: Architettura VPC

Ogni subnet è associata a una tabella di routing, che definisce le regole su come il traffico deve essere instradato all'interno della VPC (local) e verso l'esterno.
Le subnet private avranno regole che limitano il traffico all'interno della VPC o attraverso un gateway, garantendo che il traffico possa uscire verso l'esterno senza rendere pubbliche le risorse.

Le tabelle di routing in una VPC di Amazon sono un elemento fondamentale per la gestione del flusso di traffico all'interno della rete. Una tabella di routing contiene un insieme di regole chiamate route, che definiscono dove deve essere inviato il traffico di rete in base alla destinazione assegnata agli indirizzi IP.
Le regole della tabella di routing includono:

- *Destination*: specifica l'intervallo di indirizzi IP (utilizzando il formato CIDR) verso cui il traffico è destinato.
- *Target*: indica dove deve essere inoltrato il traffico destinato a quell'intervallo di IP. Questo può essere, ad esempio, un gateway internet (per traffico pubblico verso internet), un NAT Gateway (per traffico in uscita da subnet private verso internet), un'altra subnet nella stessa VPC o una Virtual Private Gateway (per traffico destinato a una connessione VPN).

Nella figura 3.12 sono mostrate le tabelle create per la nuova infrastruttura AWS. La figura evidenzia in particolare le subnet associate a ciascuna delle tabelle di routing

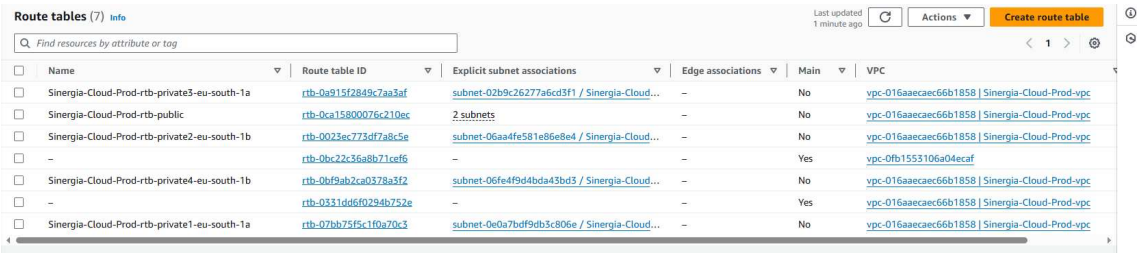


Figura 3.12: Tabella di routing

Il servizio Amazon VPC mette a disposizione una mappa in cui è possibile vedere tutte le risorse e le configurazioni assegnate alla propria VPC. La figura 3.13 illustra alcuni dettagli come le subnet, le tabelle di routing e i gateway creati per la comunicazione interna ed esterna.

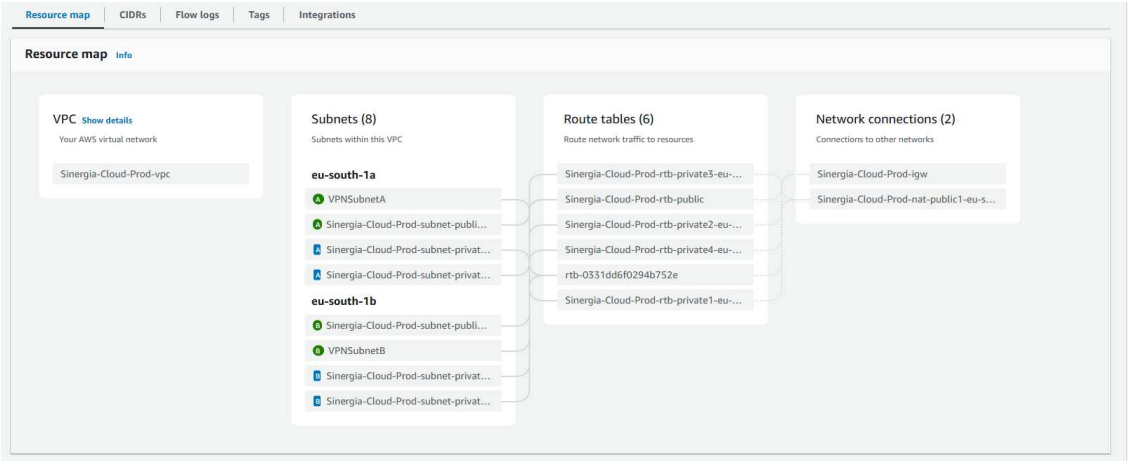


Figura 3.13: Resource map VPC

Una tabella di routing di una subnet pubblica, ad esempio, conterrà una regola che instrada il traffico verso l'internet gateway, permettendo alle risorse nella subnet di comunicare con l'esterno. Una tabella di routing di una subnet privata, invece, potrebbe indirizzare il traffico verso un NAT Gateway, consentendo connessioni in uscita verso internet senza esporre direttamente le risorse interne.Un internet gateway è necessario per fornire connettività pubblica alle risorse in una subnet pubblica, consentendo loro di ricevere traffico da e verso internet.

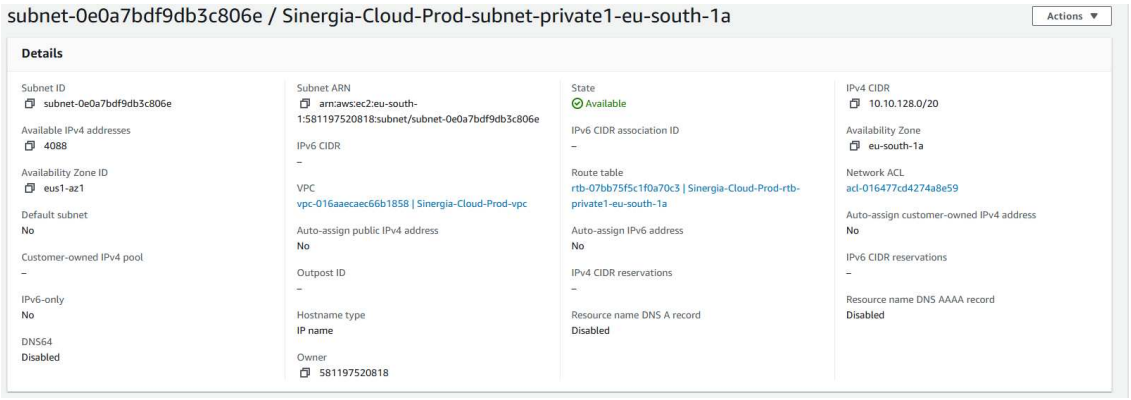


Figura 3.14: Esempio di subnet privata

Abbiamo scelto la regione *eu-south* per tenere i dati il vicino possibile, in quanto l'applicazione è principalmente rivolta ai clienti. Questa scelta riduce la latenza e migliora le prestazioni complessive del servizio. In questo modo, possiamo garantire risposte rapide e un'esperienza utente migliore, rendendo tutto più efficiente.
La figura seguente è una tabella riassuntiva delle subnet, delle regioni in cui sono posizionate le zone di disponibilità, dei nomi delle tabelle di route utilizzate e degli indizzi IPv4 utilizzati.

3.6 AWS Client VPN

Una Virtual Private Network (VPN) è una tecnologia che permette di stabilire una connessione sicura e criptata su una rete pubblica o non protetta, proteggendo i dati trasmessi e nascondendo l'indirizzo IP dell'utente.

Durante lo sviluppo della VPN, è stato necessario scegliere tra i due seguenti tipi:

- *Client-to-site*: è pensata per permettere a singoli utenti, come dipendenti remoti, di connettersi in modo sicuro a una rete aziendale utilizzando un qualsiasi dispositivo, da qualsiasi luogo. In questo modo, possono accedere a risorse interne come file, server o applicazioni aziendali, mantenendo alti livelli di sicurezza anche su reti pubbliche.

- *Site-to-site*: è utilizzata per connettere due o più reti fisicamente separate, come gli uffici di una stessa azienda situati in diverse città o paesi. Questa tipologia di VPN crea un tunnel sicuro tra le reti, permettendo la condivisione di risorse e la comunicazione interna come se tutte le reti fossero parte di un'unica infrastruttura.

È stata preferita l'implementazione di una VPN client-to-site in quanto meglio risponde alle necessità dell'azienda. Dato che alcuni dipendenti lavorano da remoto o necessitano di accedere alle risorse aziendali mentre si trovano fuori sede, questa soluzione consente di garantire una connessione sicura per ogni singolo utente.
La dashboard presente nella figura 3.15 mostra la configurazione del Client VPN Endpoint presente nel servizio VPC. L'endpoint permette di effettuare l'accesso in modo sicuro alle subnet del cloud.

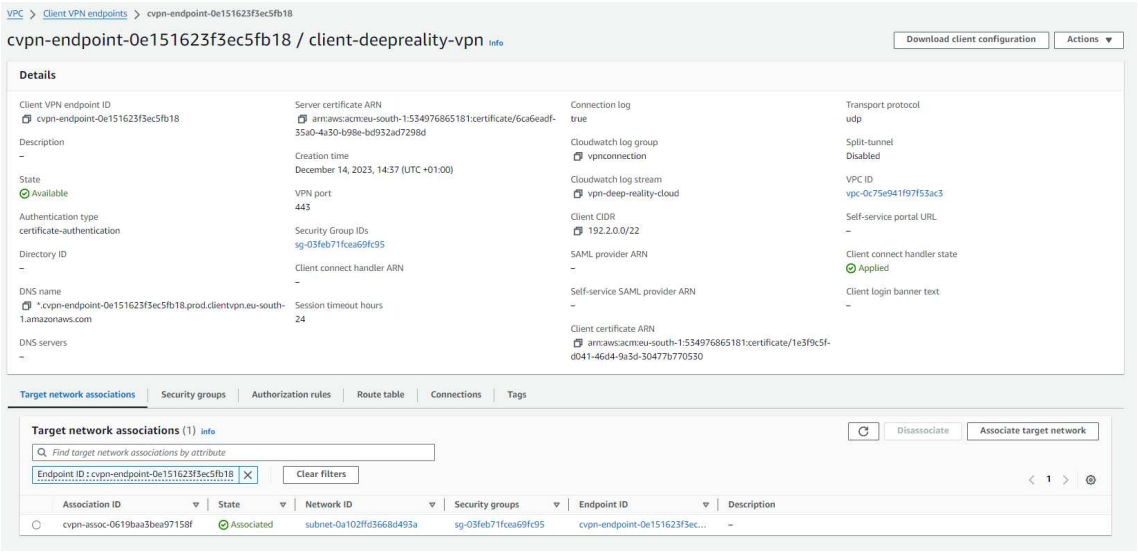


Figura 3.15: Dashboard AWS Client VPN

Nella figura 3.16 uno schema che illustra l'integrazione del servizio VPN all'interno dell'infrastruttura da sviluppare. Questo diagramma fornisce una rappresentazione visiva delle interazioni e delle connessioni tra i vari componenti della rete, evidenziando come il servizio VPN si inserisca nell'architettura complessiva.

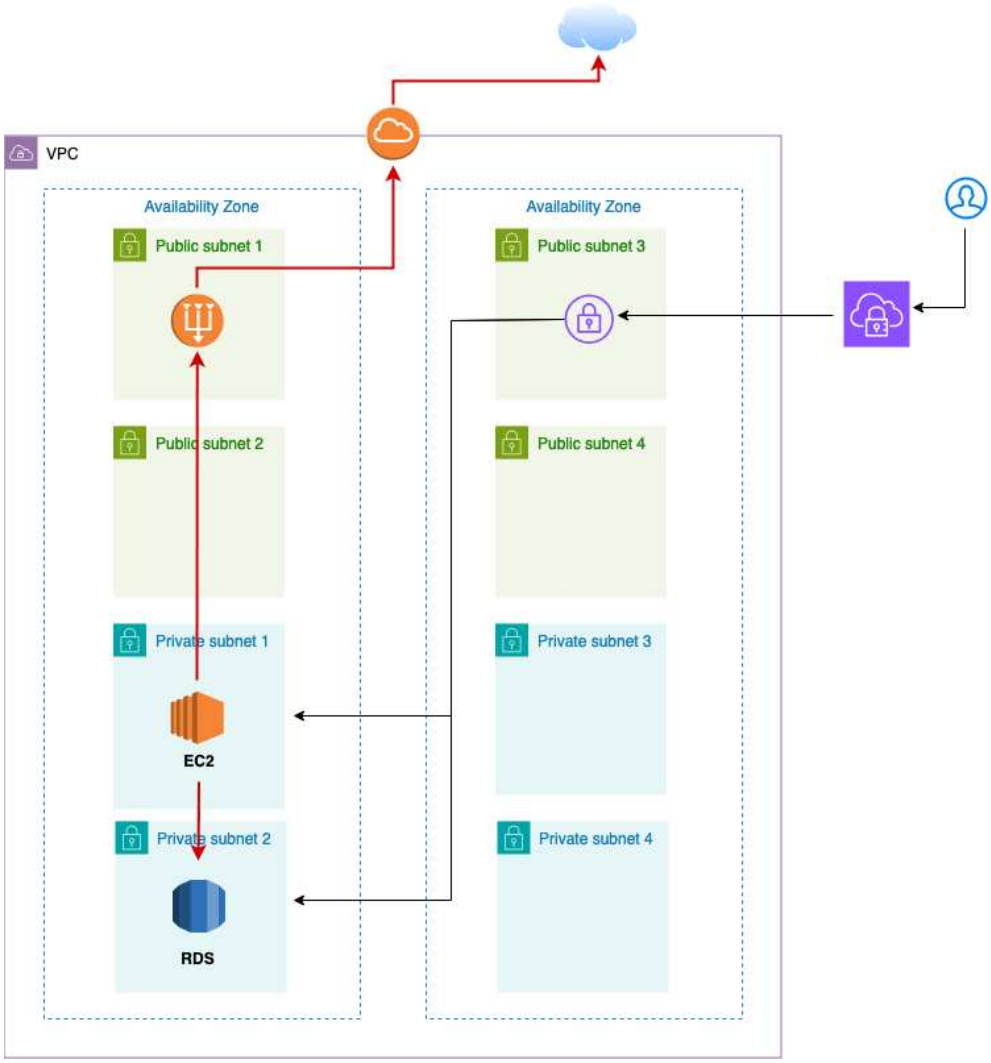


Figura 3.16: Schema VPN Client-To-Side

3.6.1 Mutua autenticazione

AWS VPN permette la mutua autenticazione, ovver un tipo di autenticazione in cui client e server si autenticano a vicenda. l server utilizza i certificati per autenticare i client quando tentano di connettersi tramite VPN. Per svolgere una autenticazione corretta è necessario generare sia per client che per server un certificato e una chiave.
È necessario caricare il certificato del server su AWS Certificate Manager ed utilizzarlo quando si crea un Client VPN endpoint. È necessario caricare il certificato client su ACM solo quando l'autorità di certificazione del certificato client è diversa da quella del certificato server.

I certificati e le chiavi per server e client sono stati generati da Easy-RSA 3.X e poi sono stati caricati in ACM (AWS Certificate Manager).

Per iniziare il processo è necessario scaricare Easy-RSA, uno strumento che permette di creare una Public Key Infrastructure (PKI) e una Certificate Authority (CA) privata.
Il software è stato installato ed utilizzato su sistema operativo Windows, quindi i comandi utilizzato sono per Powershell (estensione .bat).

Una volta completato il download, si deve eseguire il seguente comando all'interno della cartella contenente il software:

```
.\EasyRSA-Start.bat
```

Questo comando avvia l'ambiente Easy-RSA, permettendo l'esecuzione dei successivi comandi necessari per la generazione dei certificati.
Per generare il certificato e la chiave per ciascun utente, si utilizza il comando sottostante. In questo esempio, user1.example.com rappresenta il nome del client (non reale) per il quale si sta generando il certificato:

```
./easyrsa build-client-full user1.example.com nopass
```

Per il server, si devono eseguire i seguenti comandi per generare il certificato e la chiave corrispondente. In questo caso, viene specificato un Subject Alternative Name (SAN) di tipo DNS:

```
./easyrsa --san=DNS
build-server-full server nopass
```

I file generati, aventi estensione *.crt* e *.key*, vengono quindi caricati su AWS Certificate Manager utilizzando i seguenti comandi. Il primo comando è destinato al server:

```
aws acm import-certificate
--certificate fileb://server.crt \ --private-key fileb://server.key
--certificate-chain fileb://ca.crt
```

Il secondo comando è specificamente per il client e utilizza il certificato e la chiave appropriati:

```
aws acm import-certificate
--certificate fileb://client1.domain.tld.crt
--private-key fileb://client1.domain.tld.key
--certificate-chain fileb://ca.crt
```

Questi comandi completano il processo di importazione dei certificati nel servizio AWS ACM.

3.6.2 Configurazione client

Ogni client ha bisogno di un file di configurazione con estensione *.ovpn* da utilizzare con client *OpenVPN*.
Il file di configurazione è scaricabile nella dashboard dell'endpoint Client VPN. All'interno del file è necessario inserire due tag per il certificato e la chiave

```
<cert>
 Contenuto del file certificato (.crt)
</cert>

<key>
 Contenuto del file chiave (.key)
</key>
```

Inoltre, è necessario modificare la riga in cui è presente il nome DNS dell'endpoint Client VPN anteponendo una stringa casuale

`random_string.cvpn-endpoint-0102bc4c2eEXAMPLE.prod.clientvpn.us-west-2.amazonaws.com`

Una volta che il file .ovpn è stato correttamente configurato, sarà possibile utilizzarlo con qualsiasi client VPN. In questo caso specifico, si è scelto di utilizzare OpenVPN Connect piuttosto che altre alternative. Questa scelta è motivata dalle funzionalità avanzate offerte da OpenVPN Connect, che migliorano l'esperienza complessiva dell'utente e semplificano la gestione delle connessioni VPN.

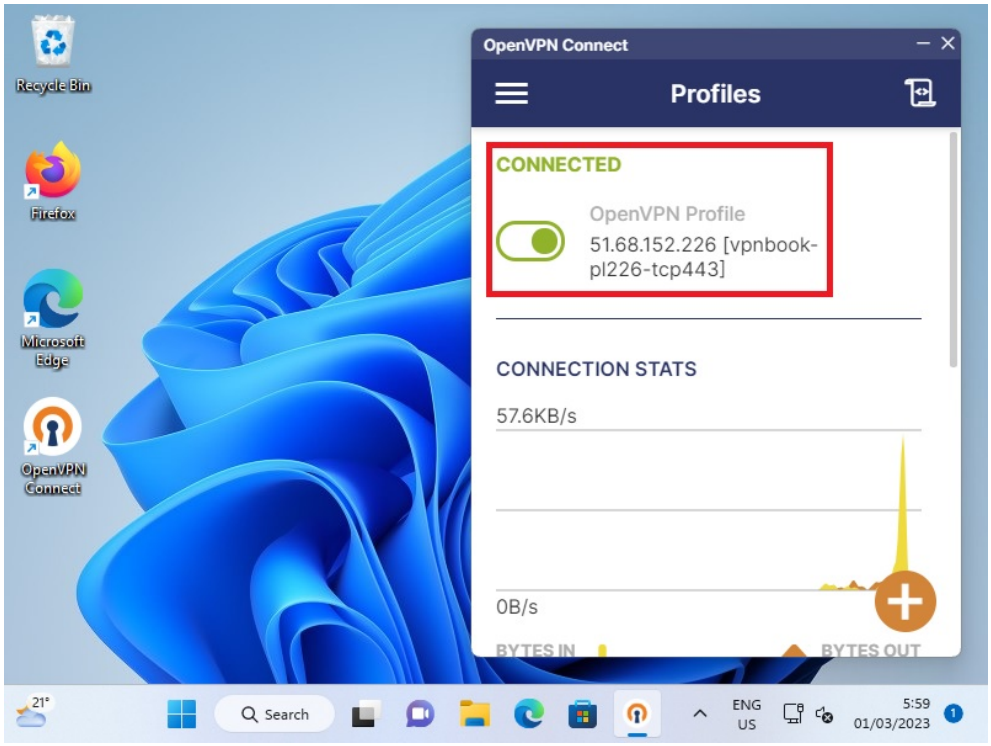


Figura 3.17: OpenVPN Connect [13]

3.7 Amazon EC2

Le istanze Elastic Compute Cloud in AWS sono macchine virtuali scalabili che forniscono capacità di calcolo nel cloud. Funzionano su un'infrastruttura virtualizzata basata su hypervisor, che gestisce la separazione tra hardware fisico e le istanze. Ogni istanza è configurata con specifiche risorse di CPU, memoria, storage e rete, personalizzabili in base alle esigenze. Queste macchine virtuali possono essere avviate e fermate a seconda del carico di lavoro, pagando solo per l'uso effettivo, rendendo EC2 flessibile per applicazioni di ogni tipo, dallo sviluppo ai servizi in produzione.

Per lo sviluppo di questa infrastruttura sono state migrate due istanze dal vecchio cloud e sono state configurate nel nuovo account creando modelli di lancio e gestendo la sicurezza.

3.7.1 Migrazione di istanze EC2

Per trasferire un'istanza EC2 da un vecchio account AWS a un nuovo account, è stata necessaria una migrazione. Il primo passo in questo processo è stato creare un'Amazon Machine Image (AMI) dell'istanza EC2 esistente. Questa AMI funge da snapshot completo dell'istanza, comprendente il sistema operativo e le applicazioni installate.

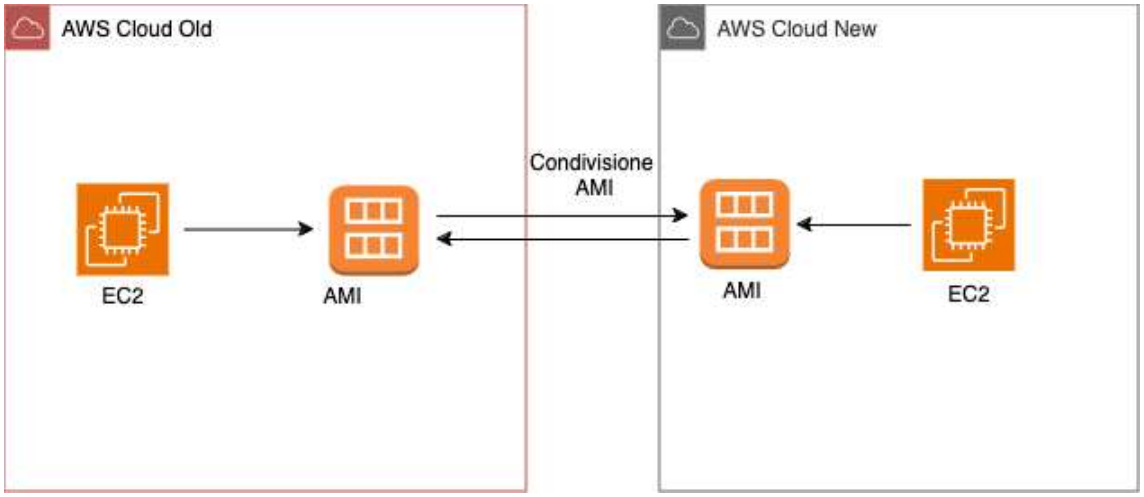


Figura 3.18: Condivisione AMI tra due Cloud AWS

Dopo aver creato l'AMI, è stato fondamentale configurare le autorizzazioni per permettere al nuovo account di accedere all'immagine.
Per fare ciò, è necessario accedere alla console di gestione di AWS e navigare alla sezione delle immagini AMI.
Qui, selezionando l'AMI desiderata, è possibile accedere alle impostazioni di condivisione. Nella sezione "Edit Permissions", è possibile inserire l'ID dell'account del nuovo Cloud, il quale consente di condividere l'immagine con quel specifico account. La modifica dei permessi è illustrata nelle figure 3.19 e 3.20.

Figura 3.19: Percorso per modifica permessi

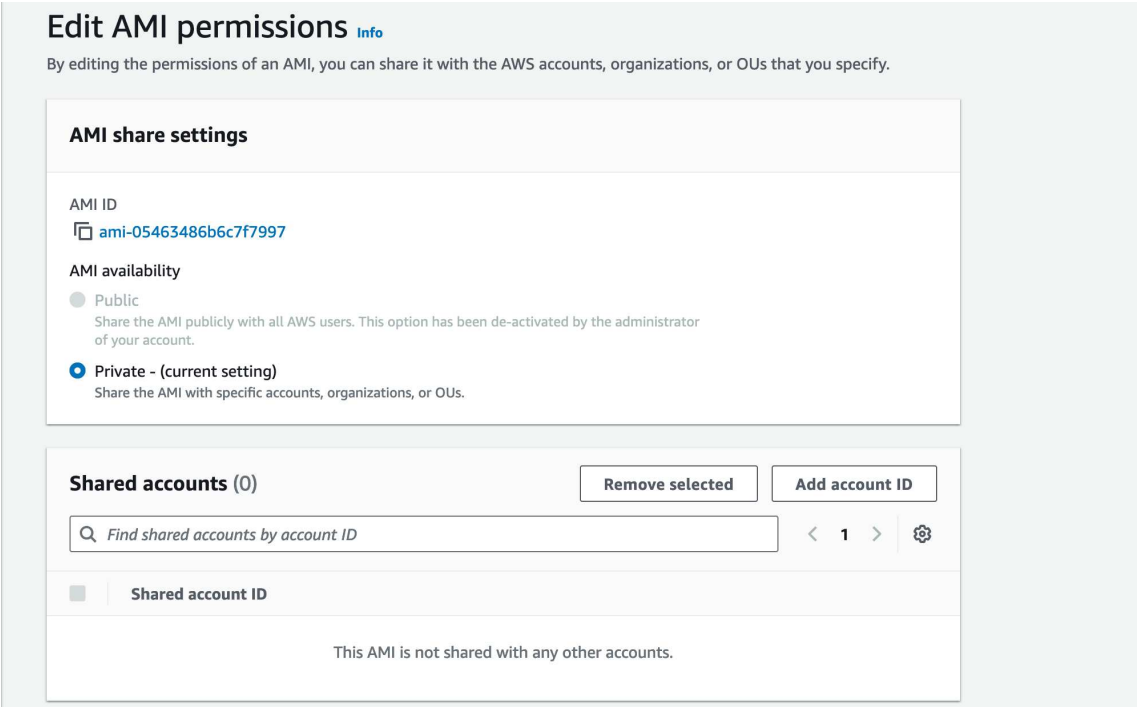


Figura 3.20: Condivisione AMI cross-account

Dopo aver creato la nuova istanza partendo dall'immagine condivisa, è stato generato un *Launch Template*, un modello che include una AMI e la configurazione di rete dell'istanza. Questo consente di automatizzare la creazione delle istanze, evitando la necessità di riconfigurarle a ogni avvio. Grazie a questo modello, tutte le impostazioni, come il tipo di istanza, la rete e i dettagli dell'AMI, vengono salvate e possono essere riutilizzate.
La figura 3.21 mostra il modello di lancio dell'istanza SIN104, lo stesso modello è stata replicato per la seconda istanza chiamata Sinergia-cloud.

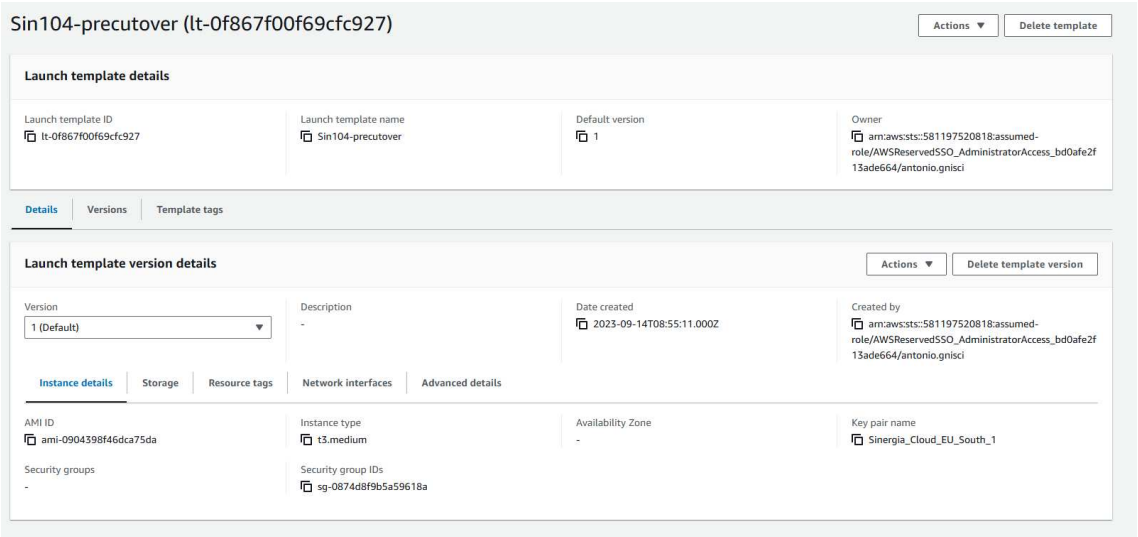


Figura 3.21: Launch Template SIN104

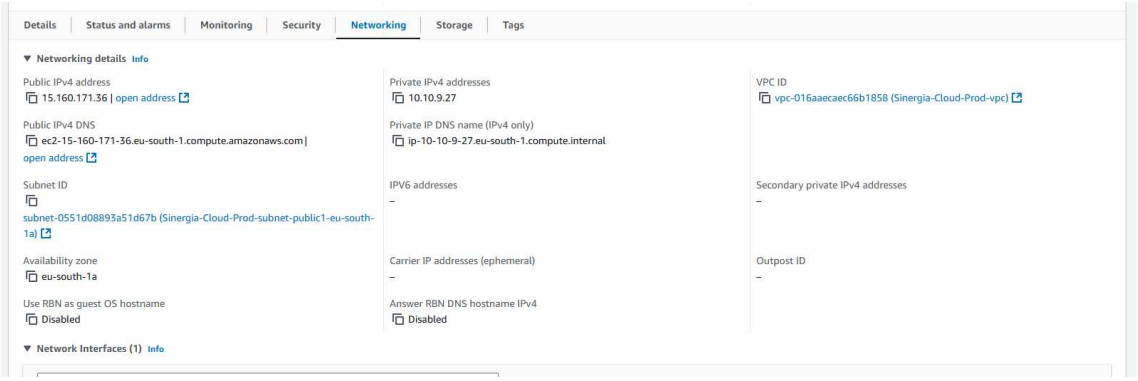


Figura 3.22: Configurazione Launch Template

La figura seguente mostra la configurazione dell'istanza EC2 SIN-Cloud. È possibile osservare la VPC in cui l'istanza è collocata, insieme alla configurazione della Elastic Network Interface (ENI), che include due indirizzi IPv4.
Nella figura 3.22 è possibile vedere gli indirizzi IP e la configurazione di rete dell'istanza Sinergia-cloud.

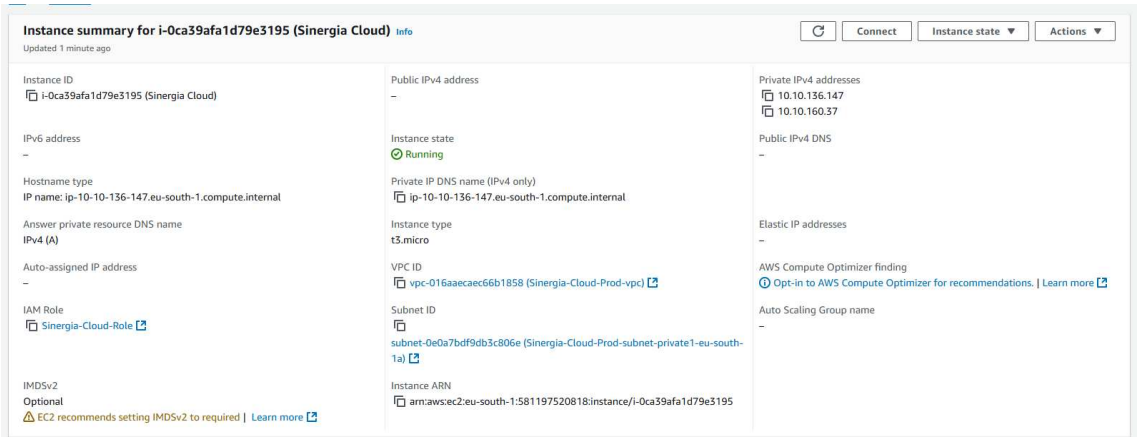


Figura 3.23: Dashboard istanza Sinergia Cloud

3.7.2 Sicurezza EC2

Per l'istanza EC2, la sicurezza è stata gestita utilizzando i security groups e le access control lists (ACL). I security groups agiscono come firewall a livello di istanza, consentendo o bloccando il traffico in ingresso e in uscita in base a regole specifiche per porte, protocolli e indirizzi IP. Le ACL, invece, operano a livello di subnet e forniscono un ulteriore livello di controllo sul traffico che attraversa la rete, regolando l'accesso in base a regole che definiscono quali pacchetti possono entrare o uscire dalla subnet.

La figura 3.24 mostra il traffico in entrata gestito dai security groups. Le regole implementate sono:

- **TCP** (Custom): consente il traffico TCP sulla porta 8172 proveniente da qualsiasi istanza nella VPC 10.10.0.0/16.
- **All traffic**: consente tutto il traffico in entrata da qualsiasi risorsa che appartiene allo stesso security group (sg-0874d8f9b5a59618a). Questo permette alle istanze appartenenti allo stesso gruppo di comunicare tra di loro.

- **MSSQL**: consente il traffico TCP sulla porta 1433, per comunciare con il database RDS, proveniente da qualsiasi istanza nella VPC 10.10.0.0/16.
- **RDP** (Remote Desktop Protocol): consente il traffico TCP sulla porta 3389, utilizzata per accedere all'istanza tramite RDP, da qualsiasi istanza nella VPC 10.10.0.0/16

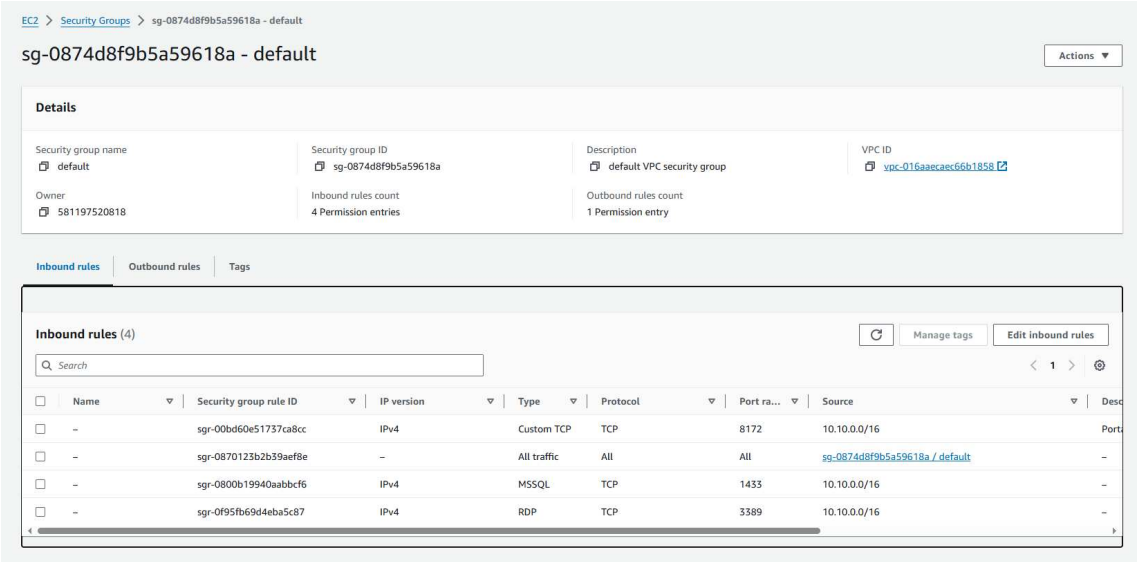


Figura 3.24: Security group in EC2

Le Network ACL illustrate nella figura 3.25 mostrano le regole implementate per i Launch Template di entrambe le istanze. In particolare, è possibile notare che tutto il traffico è stato autorizzato sia entrata che in uscita da tutti gli indirizzi (0.0.0.0/0).
Inoltre, è presente una regola di default, denotata con l'asterisco, che negano il traffico in ingresso e uscita dagli altri indirizzi non gestite dalla regola precedente.

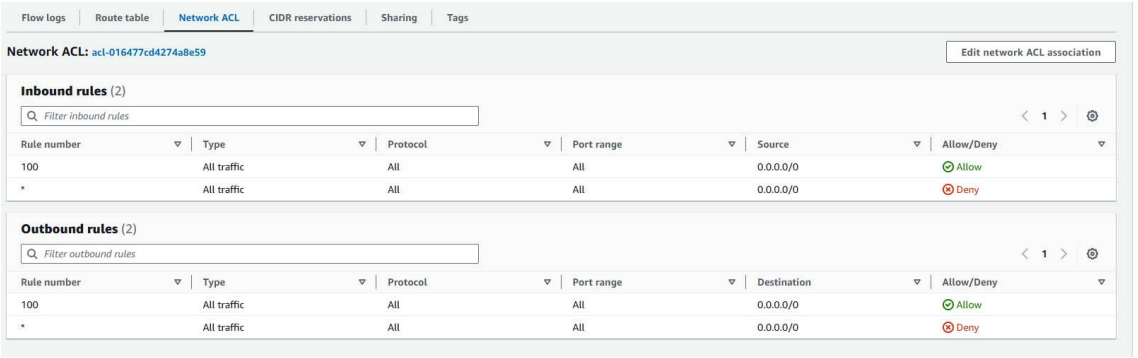


Figura 3.25: Network ACL in EC2

3.8 Amazon RDS

Amazon RDS (Relational Database Service) è un servizio gestito da Amazon Web Services che semplifica l'impostazione, la gestione e la scalabilità di database relazionali nel cloud. RDS supporta diversi motori di database, tra cui MySQL, PostgreSQL, MariaDB, Oracle e Microsoft SQL Server.

Per la migrazione di un'istanza RDS da un ambiente AWS a un altro, è stata adottata una strategia ben definita e articolata, composta da vari passaggi fondamentali. Questa strategia è stata concepita per garantire una migrazione non solo efficace, ma anche sicura, minimizzando il rischio di perdita di dati e downtime.

Nelle sezioni successive, verrà fornita una descrizione dettagliata del processo di migrazione attuato per i database presenti nel vecchio account, evidenziando i passaggi specifici intrapresi e le considerazioni tecniche necessarie.

3.8.1 Creazione di una nuova istanza RDS

Prima di procedere, è stata condotta una valutazione approfondita dell'istanza RDS esistente. Durante questa fase, si è esaminata la configurazione dell'istanza corrente, inclusa la versione del database utilizzata, il tipo di base di dati, le configurazioni (come i gruppi di sicurezza e la VPC) e le politiche di backup e ripristino.

L'obiettivo era comprendere se fosse opportuno trasferire solo i database necessari o includere anche eventuali configurazioni specifiche dell'istanza. In questa fase è stato inoltre redatto un piano di migrazione dettagliato, con l'intento di minimizzare i tempi di inattività e mitigare i rischi associati.

Anziché migrare l'intero database dell'applicazione presente in Sinergia-cloud è stata scelta la creazione di una nuova istanza RDS nel nuovo cloud. Questa decisione ha permesso di ottenere un'istanza pulita, senza portare con sé potenziali problemi derivanti da configurazioni obsolete o dati non più necessari.
Sono state inoltre implementate impostazioni di sicurezza migliorate, attraverso una gestione ottimizzata della VPC e dei gruppi di sicurezza. Le politiche di backup e ripristino sono state riviste per rispondere alle nuove esigenze aziendali.

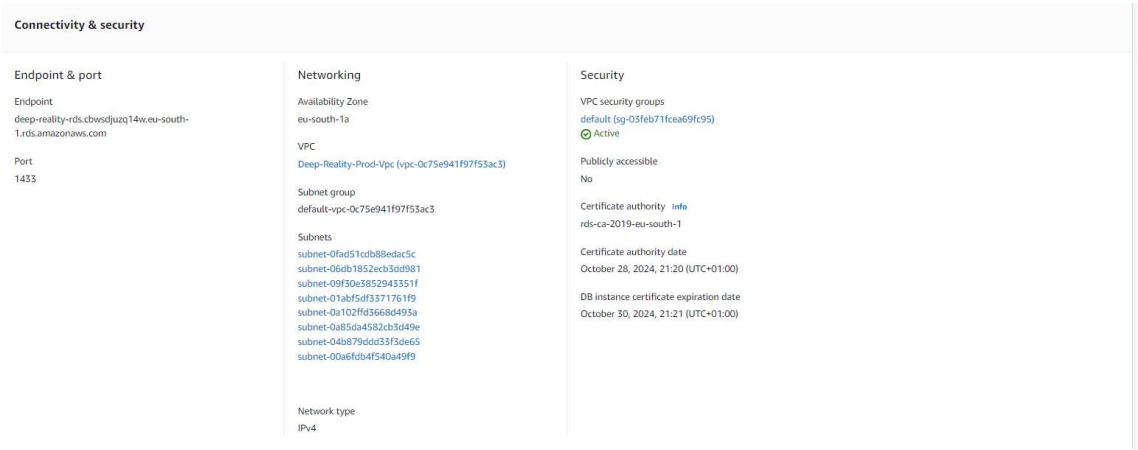


Figura 3.26: Connettività e sicurezza della nuova istanza RDS

Come mostrato nella figura 3.27, abbiamo scelto di utilizzare l'engine SQL Server Express Edition. Questa decisione è stata presa perché SQL Server è la tecnologia che l' azienda ha sempre utilizzato, garantendo così una continuità nelle nostre operazioni e una familiarità con l'ambiente di lavoro.

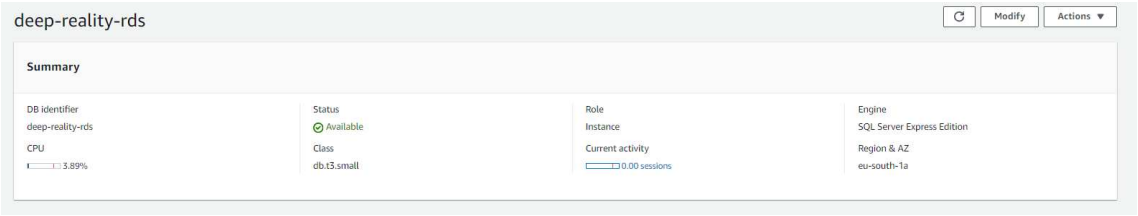


Figura 3.27: Informazioni generali RDS

Nella figura 3.28, sono presenti i dettagli relativi alla nuova istanza RDS. Sono inclusi informazioni importanti come l'endpoint per accedere al database, il tipo di istanza creata e le subnet su cui è collocato. Questi elementi sono fondamentali per comprendere le specifiche della nuova configurazione.

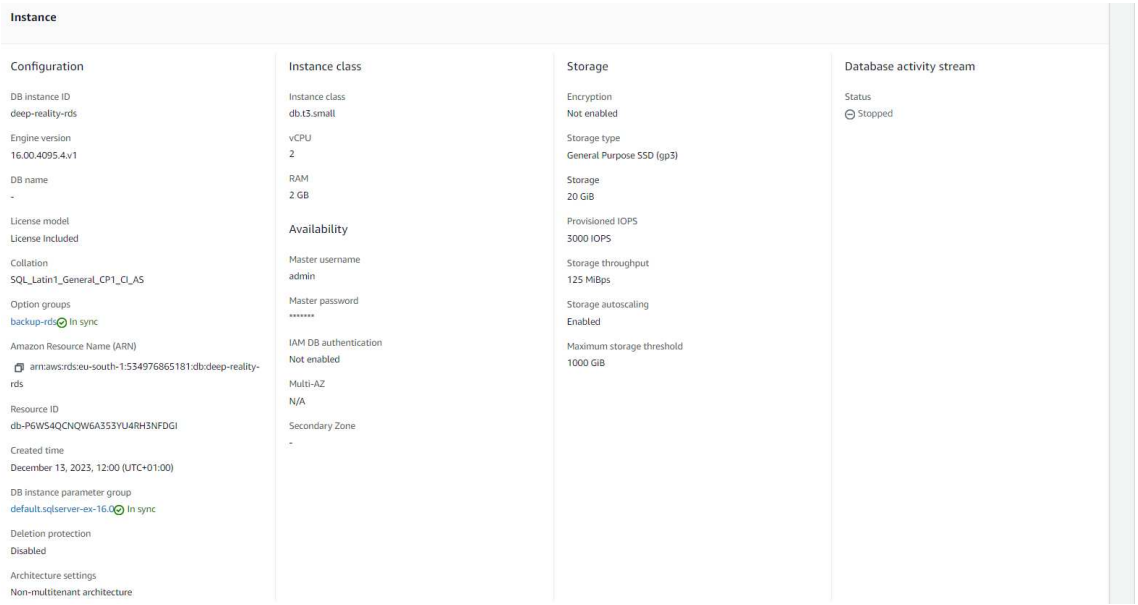


Figura 3.28: Informazioni nuova istanza RDS

3.8.2 Migrazione dei database

La migrazione dal vecchio account AWS al nuovo è stata effettuata attraverso un processo strutturato che ha sfruttato Amazon S3 come soluzione di storage temporaneo e punto di trasferimento. Il primo passo è stato quello di eseguire backup completi dei database esistenti sul vecchio account.
Nelle impostazioni del bucket sono stati inseriti i permessi e l'identificativo del nuovo account per poter effettuare il passaggio dei backup nel nuovo bucket.
Una volta creati, i backup sono stati caricati sul nuovo Bucket S3 chiamato *sinergia-rds-backups*.
Ed infine sono stati stati ripristinati nelle nuove istanze RDS.

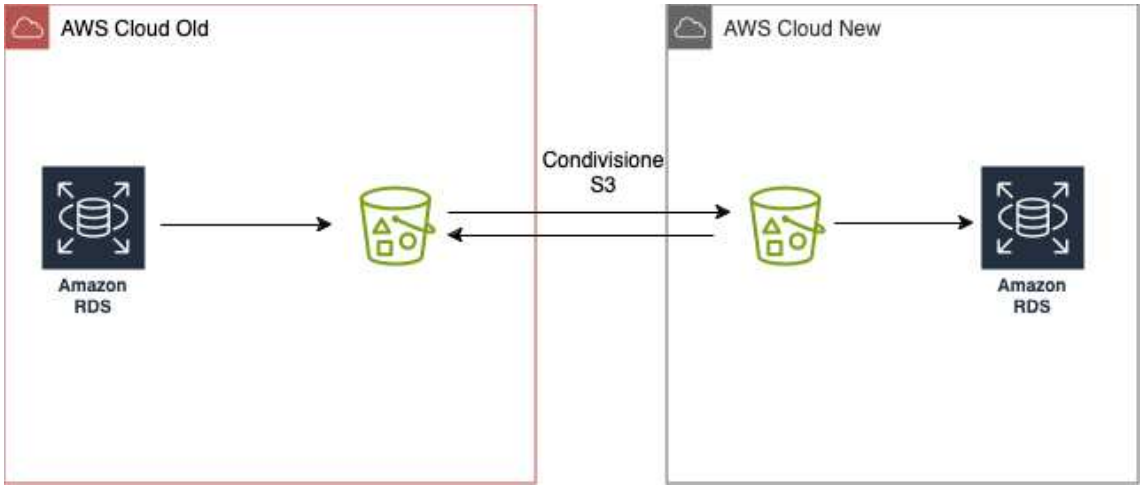
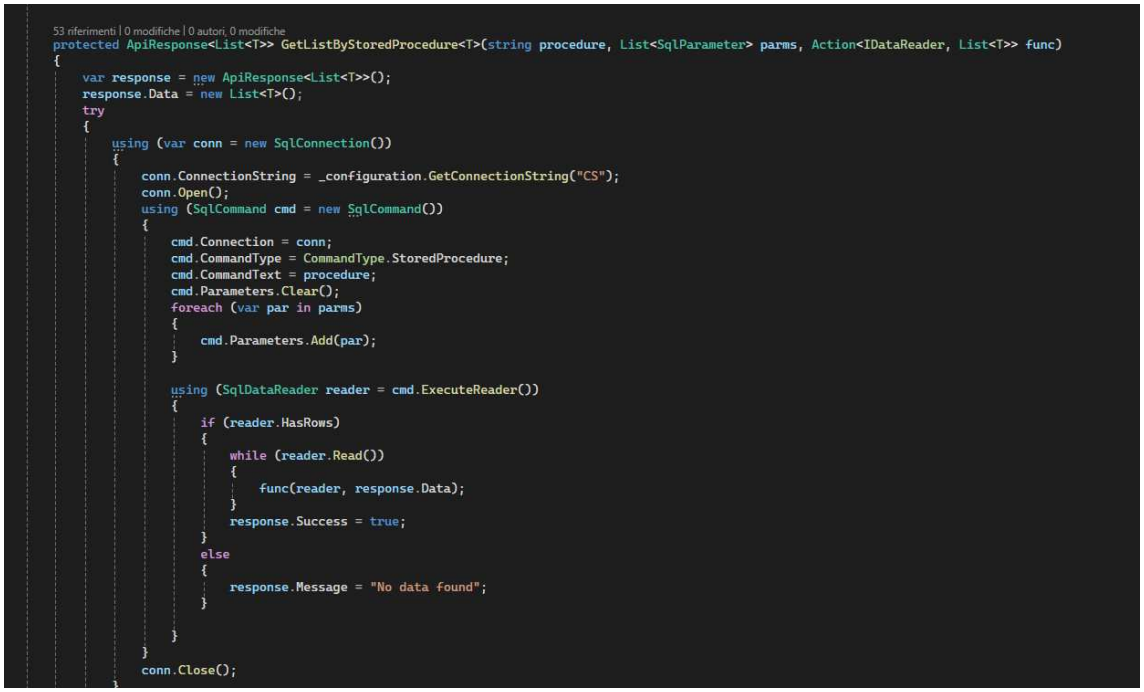


Figura 3.29: Trasferimento Backup da Aws ad Aws

3.8.3 Accesso e Gestione di Amazon RDS

Per effettuare delle interrogazioni al database è stato creato un servizio in linguaggio C# contenente delle funzioni per effettuare l'accesso ad Amazon RDS.
Il codice è stato progettato per eseguire una stored procedure in un database SQL Server, recuperarne i risultati e mapparli in una lista di oggetti di tipo generico T. Questo metodo agisce come un generico interfaccia tra il codice C# e il database, permettendo di eseguire diverse query SQL in modo standardizzato.

nelle figure 3.30 e 3.31 è presente il codice per ottenere una lista di oggetti dal database SQL, il metodo restituisce un oggetto di tipo *ApiResponse<List<T»* che contiene informazioni sulla riuscita dell'operazione (successo/fallimento) e i dati recuperati dalla stored procedure.

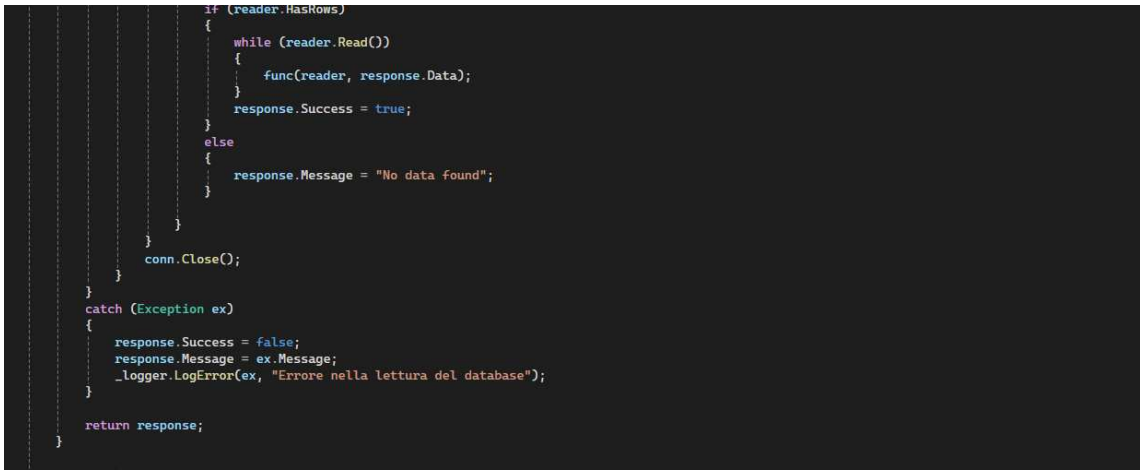


```
53 riferimenti | 0 modifiche | 0 autori, 0 modifiche
protected ApiResponse<List<T>> GetListByStoredProcedure<T>(string procedure, List<SqlParameter> parms, Action<IDataReader, List<T>> func)
{
    var response = new ApiResponse<List<T>>();
    response.Data = new List<T>();
    try
    {
        using (var conn = new SqlConnection())
        {
            conn.ConnectionString = _configuration.GetConnectionString("CS");
            conn.Open();
            using (SqlCommand cmd = new SqlCommand())
            {
                cmd.Connection = conn;
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.CommandText = procedure;
                cmd.Parameters.Clear();
                foreach (var par in parms)
                {
                    cmd.Parameters.Add(par);
                }

                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            func(reader, response.Data);
                        }
                        response.Success = true;
                    }
                    else
                    {
                        response.Message = "No data found";
                    }

                }
            }
            conn.Close();
        }
```

Figura 3.30: Selezione di una lista di oggetti



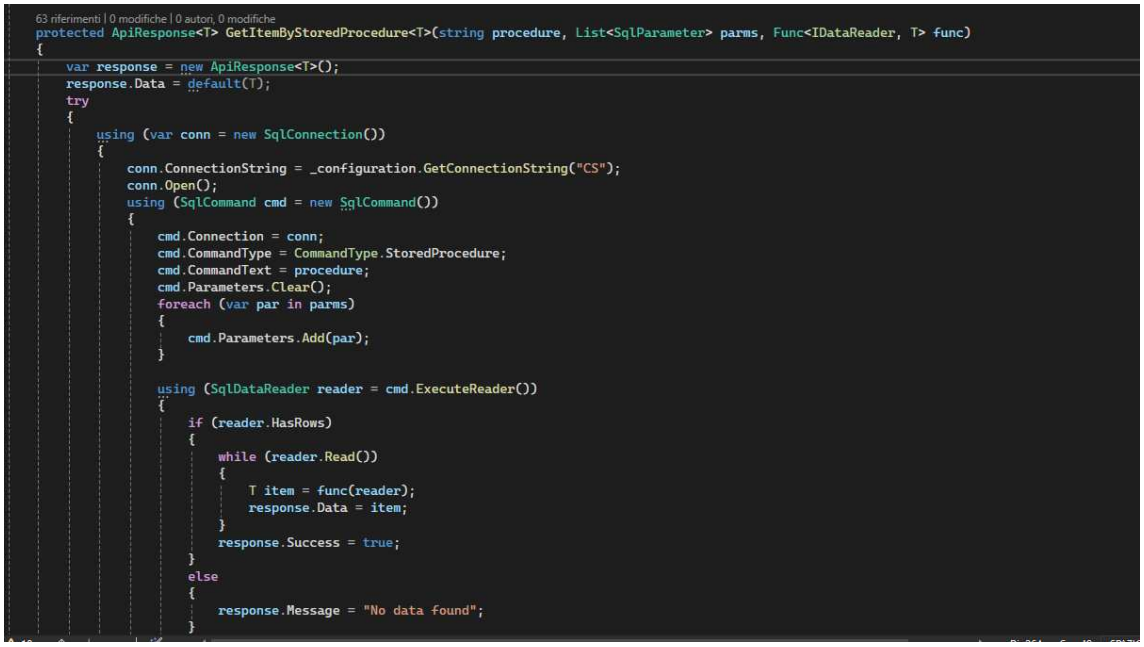
```
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            func(reader, response.Data);
                        }
                        response.Success = true;
                    }
                    else
                    {
                        response.Message = "No data found";
                    }

                }
            }
            conn.Close();
        }
    }
    catch (Exception ex)
    {
        response.Success = false;
        response.Message = ex.Message;
        _logger.LogError(ex, "Errore nella lettura del database");
    }

    return response;
}
```

Figura 3.31: Selezione di una lista di oggetti - 2

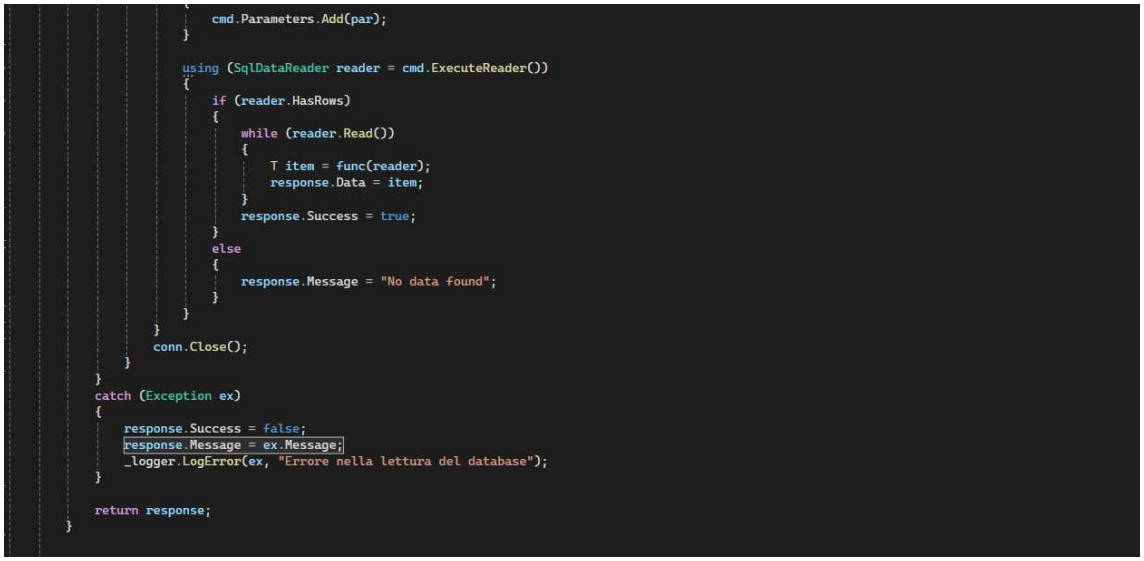
Nelle figure 3.32 e 3.33 è presente il codice che utilizza una stored procedure che permette di interagire con il database SQL per ottenere uno specifico oggetto di tipo $ApiResponse<T>$, contenente informazioni sulla riuscita dell'operazione e i dati ottenuti da RDS.



```
63 riferimenti | 0 modifiche | 0 autori, 0 modifiche
protected ApiResponse<T> GetItemByStoredProcedure<T>(string procedure, List<SqlParameter> parms, Func<IDataReader, T> func)
{
    var response = new ApiResponse<T>();
    response.Data = default(T);
    try
    {
        using (var conn = new SqlConnection())
        {
            conn.ConnectionString = _configuration.GetConnectionString("CS");
            conn.Open();
            using (SqlCommand cmd = new SqlCommand())
            {
                cmd.Connection = conn;
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.CommandText = procedure;
                cmd.Parameters.Clear();
                foreach (var par in parms)
                {
                    cmd.Parameters.Add(par);
                }

                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            T item = func(reader);
                            response.Data = item;
                        }
                        response.Success = true;
                    }
                    else
                    {
                        response.Message = "No data found";
                    }
```

Figura 3.32: Selezione di un singolo oggetto



```
                    cmd.Parameters.Add(par);
                }

                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            T item = func(reader);
                            response.Data = item;
                        }
                        response.Success = true;
                    }
                    else
                    {
                        response.Message = "No data found";
                    }
                }
                conn.Close();
            }
        }
        catch (Exception ex)
        {
            response.Success = false;
            response.Message = ex.Message;
            _logger.LogError(ex, "Errore nella lettura del database");
        }

        return response;
    }
```

Figura 3.33: Selezione di un singolo oggetto - 2

3.8.4 Sicurezza in RDS

Come per le istanze EC2, anche per Amazon RDS sono state implementate regole di sicurezza utilizzando ACL e Security Group. Nella figura 3.34 vengono mostrate le regole di questi security group, che gestiscono il traffico in ingresso e in uscita sulle porte 8172, 1433 e 3389, utilizzate rispettivamente per HTTP, SQL Server e RDP.
La colonna *Source* nella figura indica l'origine del traffico autorizzato, specificando chi può accedere alle risorse. In questo caso, il traffico è limitato esclusivamente a connessioni provenienti dall'interno della stessa VPC, garantendo che solo le risorse all'interno di questo ambiente di rete possano comunicare con il database.

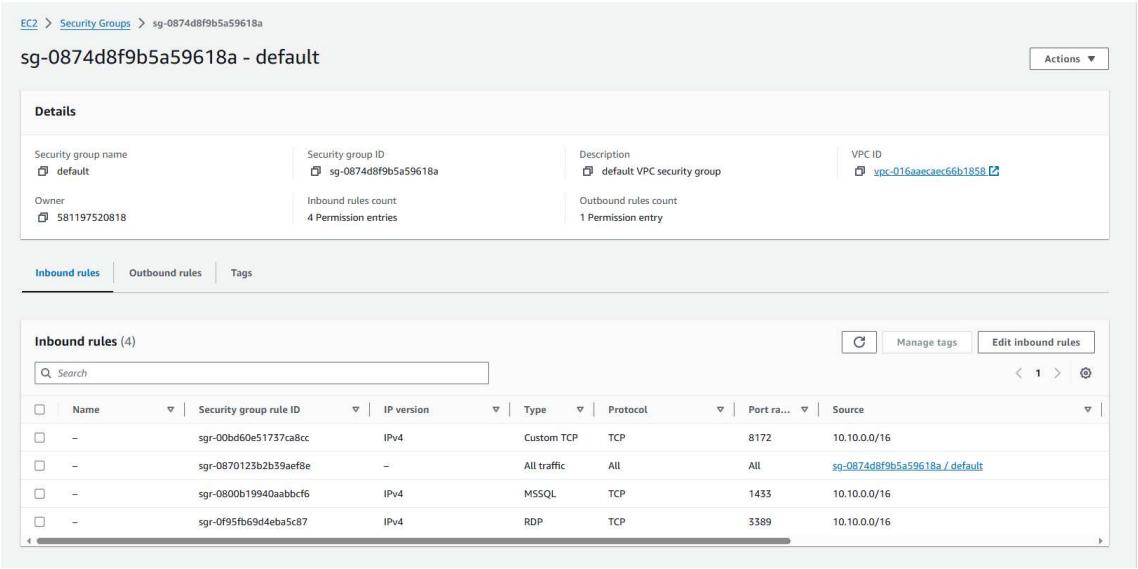


Figura 3.34: Security group in RDS per ingressi

3.9 Amazon S3

Amazon S3 (Simple Storage Service) è un servizio di archiviazione nel cloud che permette di salvare e recuperare dati di qualsiasi dimensione in modo sicuro e scalabile. È progettato per garantire alta disponibilità e offre funzionalità come la gestione delle versioni, il controllo degli accessi e la crittografia dei dati.

I bucket S3 sono contenitori utilizzati per organizzare i dati all'interno di questo servizio. Ogni bucket funge da spazio di archiviazione per file, chiamati oggetti, e può contenere un numero illimitato di essi. Ogni bucket è identificabile tramite un nome unico e può essere configurato con diverse impostazioni di accesso e sicurezza.

Nel progetto, il bucket è stato utilizzato per gestire i backup dell'istanza RDS del database utilizzato dall'applicativo in Sinergia Cloud, come mostrato in figura 3.35.

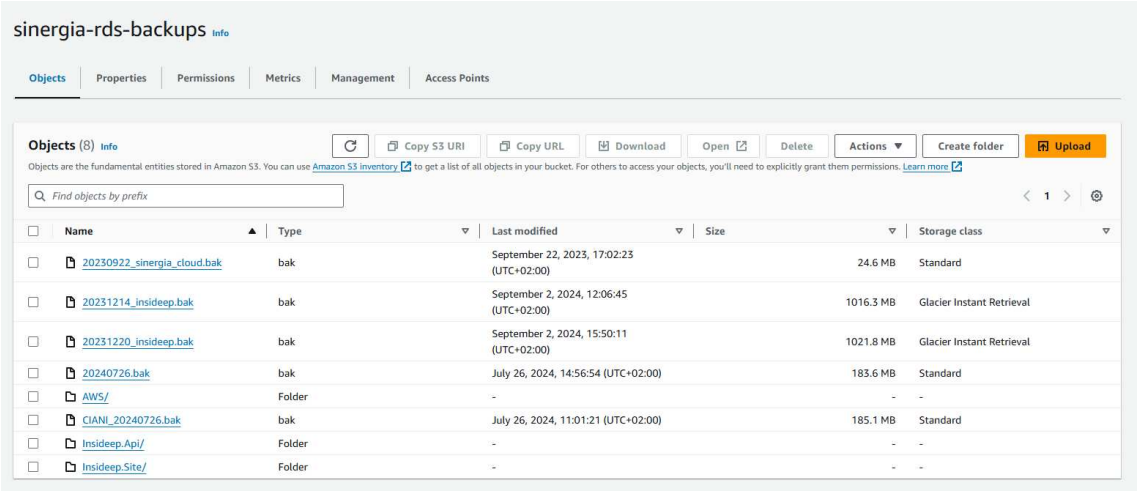


Figura 3.35: Bucket S3 per i backup RDS

È stato creato un ulteriore bucket per gestire i file multimediali della nuova applicazione Insideep, come illustrato nella figura ??.
Fino a questo momento, gli sviluppi e la migrazione descritti riguardano l'applicazione Sinergia Cloud. Tuttavia, Insideep rappresenta il nuovo prodotto dell'azienda, per il quale è stato necessario replicare tutti i servizi precedentemente menzionati.
La decisione di descrivere Amazon S3 anche per questa applicazione è stata presa per evidenziare i diversi utilizzi dei bucket S3 nello sviluppo di questo progetto.

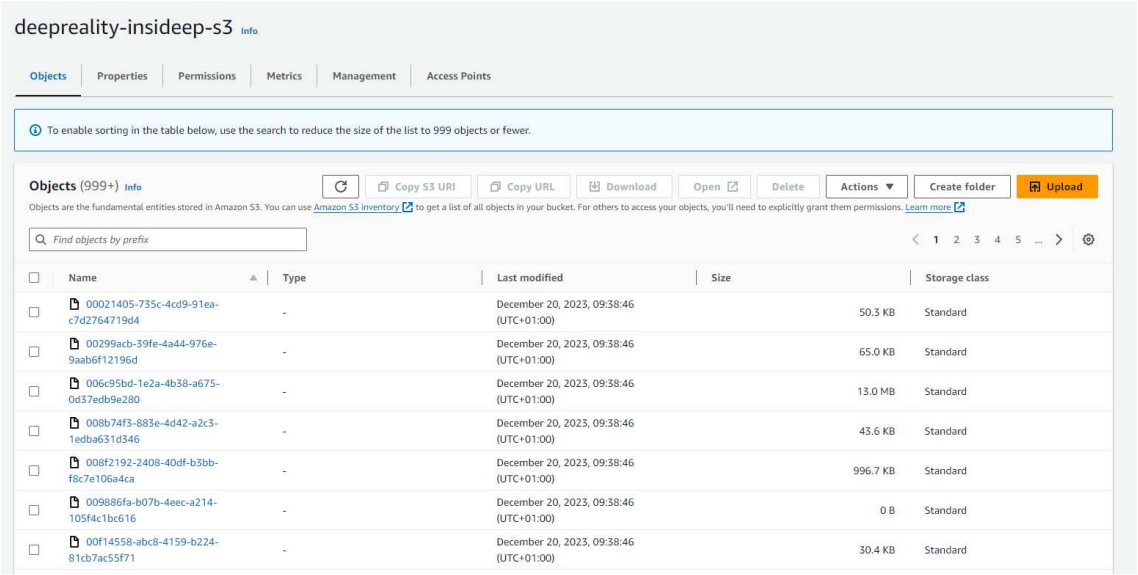


Figura 3.36: Bucket S3 per file multimediali

La gestione dei backup avviene tramite funzioni serverless, mentre le operazioni CRUD (CREATE, READ, UPDATE, DELETE) sui file multimediali sono gestite dal backend dell'applicazione, scritto in C# e .NET. In particolare, è stato creato un servizio che consente di interfacciarsi con S3 utilizzando una chiave di accesso al bucket.

Nella figura 3.37 è possibile vedere la classe realizzatain C# per semplificare le operazioni di gestione dei file multimediali.

Questo servizio è stato progettato per semplificare le operazioni di gestione dei file multimediali, consentendo di eseguire operazioni come il caricamento, il recupero, l'aggiornamento e la cancellazione di oggetti all'interno del bucket.
Le funzioni presentate sono di natura asincrona, il che consente di eseguire le operazioni senza bloccare il thread principale dell'applicazione, migliorando così la reattività e l'efficienza. Inoltre, il codice interagisce con l'API di S3, gestendo le autenticazioni tramite chiavi di accesso e garantendo la corretta manipolazione dei dati.

```
using Amazon;
using Amazon.S3;
using Microsoft.Extensions.Options;
using Amazon.S3.Transfer;
using Amazon.Runtime.Internal.Util;
using Microsoft.Extensions.Logging;
using Amazon.S3.Model;
using Microsoft.AspNetCore.Mvc;
using Stripe;
using Sinergia.Cloud.Model.Dto.DR;

namespace Sinergia.Cloud.Application.Services
{
    6 riferimenti | Elvi Billano, 162 giorni fa | 1 autore, 1 modifica
    public class AwsS3Service : IAwsS3Service
    {
        private readonly AWSS3Option _awsS3Option;
        private readonly IAmazonS3 _awsS3Client;
        private readonly ILogger<AwsS3Service> _logger;
        0 riferimenti | 0 modifiche | 0 autori, 0 modifiche
        public AwsS3Service(IConfiguration config
            , IAmazonS3 awsS3Client
            , IOptions<AWSS3Option> awsS3Option
            , ILogger<AwsS3Service> logger
            )
        {
            _awsS3Option = awsS3Option.Value;
            _awsS3Client = awsS3Client;
            //new AmazonS3Client(_awsS3Option.AwsAccessKey, _awsS3Option.AwsSecretAccessKey, RegionEndpoint.GetBySystemName(_awsS3Option.Region));

            _logger = logger;
        }
```

Figura 3.37: Servizio che utilizza le API AWS

La figura 3.38 illustra il codice per cancellare un file dal bucket S3. La funzione restituisce un messaggio di conferma in caso di eliminazione del file e gestisce i casi di eccezione creando dei messaggi di log.

```
5 riferimenti | 0 modifiche | 0 autori, 0 modifiche
public async Task<ApiResponse<bool>> DeleteFileAsync(string bucketName,string key, string? versionId = null)
{
    var result = new ApiResponse<bool>();
    try
    {

        var deleteRequest = new DeleteObjectRequest()
        {
            BucketName = bucketName,
            Key = key
        };
        if (!string.IsNullOrEmpty(versionId)){
            deleteRequest.VersionId = versionId;
        }

        var response = await _awsS3Client.DeleteObjectAsync(deleteRequest);
        if (response.HttpStatusCode == HttpStatusCode.NoContent)
        {
            result.Success = true;
            result.Data = true;
        }
    }
    catch (AmazonS3Exception ex)
    {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Delete File to S3 Bucket");
    }
    catch (Exception ex)
    {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Delete File to S3 Bucket");
    }
    return result;
}
```

Figura 3.38: Cancellazione dei file dal bucket

Il codice nella figura 3.39 mostra la funzione *DownloadFileAsync*, progettata per scaricare un file da un bucket Amazon S3.
La funzione restituisce un oggetto che contiene il file scaricato in formato byte array e un flag indicante se l'operazione è andata a buon fine.

```
9 riferimenti | 0 modifiche | 0 autori, 0 modifiche
public async Task<ApiResponse<byte[]>> DownloadFileAsync(string bucketName, string key, string? versionId=null)
{
    ApiResponse<byte[]> result = new ApiResponse<byte[]>();
    try
    {
        var getObjectRequest = new GetObjectRequest()
        {
            BucketName = bucketName,
            Key = key,
            VersionId = versionId
        };
        Console.WriteLine(DateTime.Now.ToString() + "Request to s3");
        using (var response = await _awsS3Client.GetObjectAsync(getObjectRequest))
        {
            Console.WriteLine(DateTime.Now.ToString() +  "Response arrived");
            if (response.HttpStatusCode == HttpStatusCode.OK)
            {
                using (var ms = new MemoryStream())
                {
                    Console.WriteLine(DateTime.Now.ToString()  + " converting to memory stream");
                    await response.ResponseStream.CopyToAsync(ms);
                    Console.WriteLine(DateTime.Now.ToString() +  "converted to memory stream");
                    result.Success = true;
                    result.Data = ms.ToArray();
                }


            }
        }
    }
    catch (AmazonS3Exception ex)
    {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Download File to S3 Bucket");
    }
    catch (Exception ex)
    {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Download File to S3 Bucket");
    }
    return result;
}
```

Figura 3.39: Download del file dal bucket

Il codice presente nella figura 3.40 mostra una funzione creata per gestire sia array di byte che stream. Tale funzione è stata progettata per caricare un file su un bucket Amazon S3.

```
4 riferimenti | 0 modifiche | 0 autori, 0 modifiche
public async Task<ApiResponse<string>> UploadFileAsync(string bucketName, string key, string contentType, byte[] content)
{
    using (var ms = new MemoryStream(content))
    {
        return await UploadFileAsync(bucketName,key,contentType, ms);
    }
}

10 riferimenti | 0 modifiche | 0 autori, 0 modifiche
public async Task<ApiResponse<string>> UploadFileAsync(string bucketName, string key,string contentType, Stream content)
{
    ApiResponse<string> result = new ApiResponse<string>();
    try
    {
        var putObjectRequest = new PutObjectRequest()
        {
            BucketName = bucketName,
            Key = key,
            ContentType = contentType,
            InputStream = content,
        };

        var response = await _awsS3Client.PutObjectAsync(putObjectRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            result.Success = true;
            result.Data = response.VersionId;
        }
    }
    catch (AmazonS3Exception ex)
    {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Upload File to S3 Bucket");
    }
    catch (Exception ex) {
        result.Message = ex.Message;
        _logger.LogError(ex, "Unable to Upload File to S3 Bucket");
    }

    return result;
}
```

Figura 3.40: Upload dei file nel bucket

Il codice presente nella figura 3.41 è stata progettata per elencare tutti gli oggetti presenti

in un bucket S3. Questa funzione invia una richiesta al servizio S3 per ottenere l'elenco degli oggetti e restituisce un risultato quando l'operazione è completata, senza bloccare l'esecuzione del resto del codice.

```
2 riferimenti | Elvi Billano, 162 giorni fa | 1 autore, 1 modifica
public async Task<ApiResponse<List<S3Object>>> ListObjectsAsync(string bucketName)
{
    ApiResponse<List<S3Object>> result = new ApiResponse<List<S3Object>>();
    result.Data = new List<S3Object>();
    try
    {
        var request = new ListObjectsV2Request
        {
            BucketName = bucketName,
            MaxKeys = 1000,
        };

        ListObjectsV2Response response;

        do
        {
            response = await _awsS3Client.ListObjectsV2Async(request);

            response.S3Objects
                .ForEach(obj => result.Data.Add(obj));
            //Console.WriteLine($"{obj.Key,-35}{obj.LastModified.ToShortDateString(),10}{obj.Size,10}");
            // If the response is truncated, set the request ContinuationToken
            // from the NextContinuationToken property of the response.
            request.ContinuationToken = response.NextContinuationToken;
        }
        while (response.IsTruncated);

        result.Success = true;

        //return result;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error encountered on server. Message:'{ex.Message}' getting list of objects.");
        result.Success = false;
        return result;
    }catch(Exception ex)
    {
        Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'");
        result.Success = false;
        return result;
    }

    return result;
}
```

Figura 3.41: Lista di file multimediali dal bucket

3.10 AWS Lambda

Le funzioni serverless sono un modello di architettura software che consente di eseguire codice in risposta a eventi senza gestire direttamente i server.
A differenza delle funzioni classiche, dove gli sviluppatori devono gestire l'infrastruttura, la scalabilità e i costi di server, le funzioni serverless si concentrano solo sul codice, con il provider cloud che si occupa della gestione e della scalabilità automatica.
Questo modello prevede un pricing basato sul consumo, pagando solo per il tempo di esecuzione e le risorse utilizzate. Il servizio AWS che offre queste funzioni è AWS Lambda, che consente di creare funzioni attivate da eventi specifici.
Le figure 3.43 e ?? mostrano rispettivamente, la dashboard con le funzioni lambda implementate e una panoramica della funzione lambda per gestire i backup del database RDS.

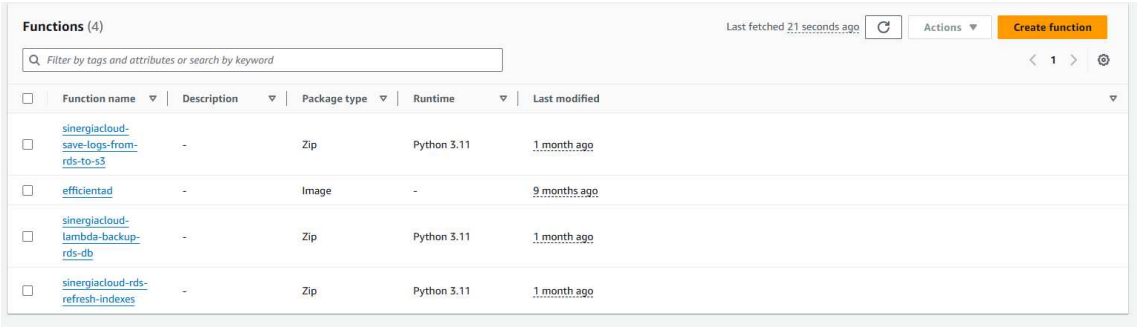


Figura 3.42: Funzioni Lambda sviluppate

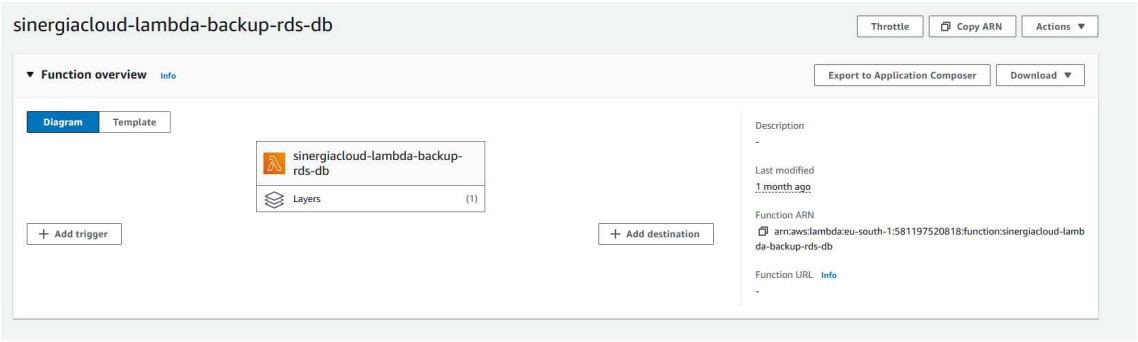


Figura 3.43: Funzione Lambda per la gestione dei backup

Tra le diverse funzioni serverless sviluppate, la più rilevante riguarda la gestione dei backup. Per questa funzionalità, è stato implementato un codice in Python che esegue un backup su Amazon RDS e deposita il file di backup nel bucket S3 precedentemente configurato. Nella figura 3.44 è mostrato il codice del file *apppy* in cui è presente la funzione *lambda_handler*, ovvero il punto di ingresso di una funzione Lambda, ovvero la funzione principale che viene eseguita automaticamente ogni volta che AWS Lambda viene invocata. Viene utilizzata per gestire l'input ricevuto, elaborarlo ed eventualmente restituire un output.
Inoltre, sono state caricate le variabili di ambiente per accedere al database e al bucket.

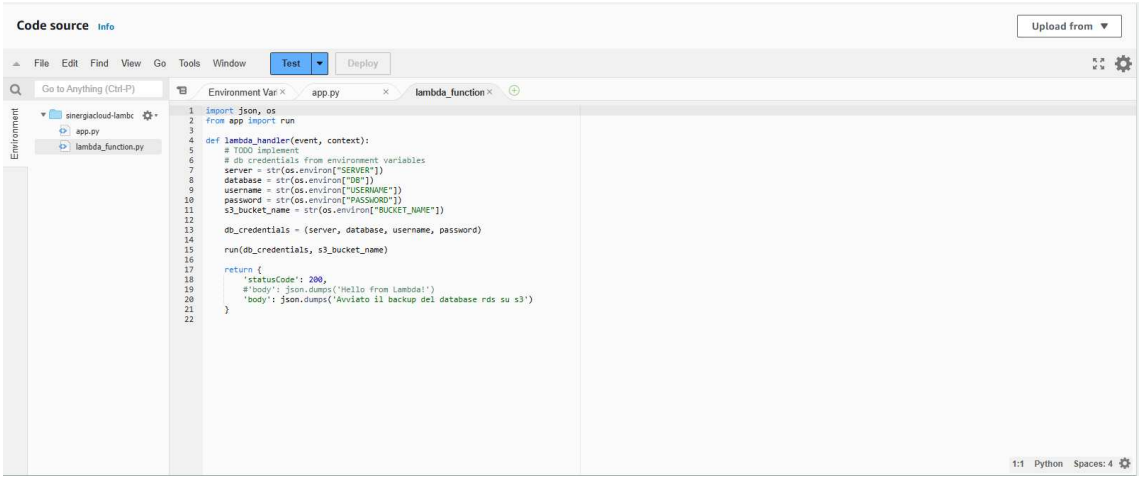


Figura 3.44: Codice della funzione principale

la funzione presente nella figura 3.45 si occupa di estrarre dati da un database SQL Server e caricarli su Amazon S3.quest codice agisce come un processo automatizzato che periodicamente copia i dati da un database in un archivio cloud.

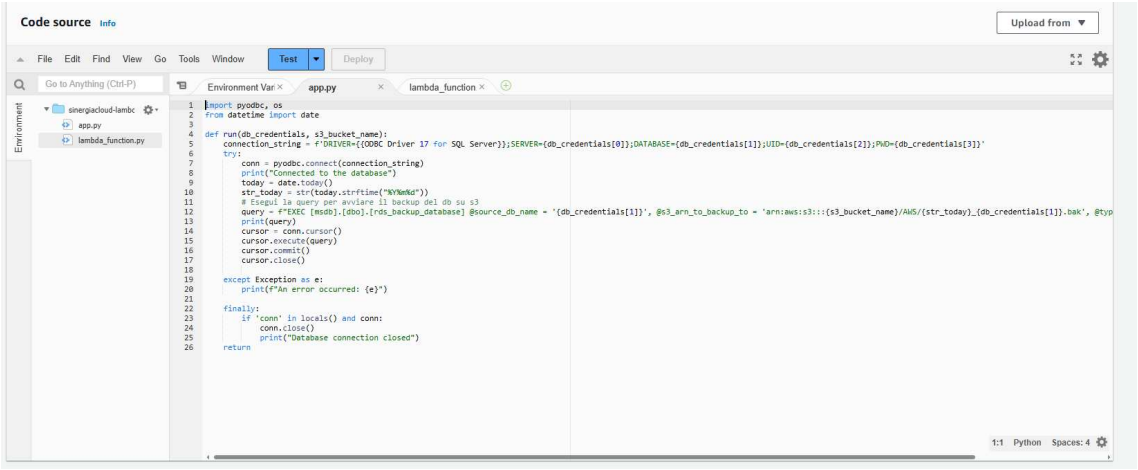


Figura 3.45: Codice per la creazione dei backup

3.10.1 Sicurezza delle funzioni Lambda

La sicurezza delle funzioni Lambda è gestita tramite Security Group, utilizzando regole per il traffico in ingresso e uscita.
Nella figura 3.46 viene mostrata la lista dei gruppi di regole implementate per potersi interfacciare con il database SQL Server.

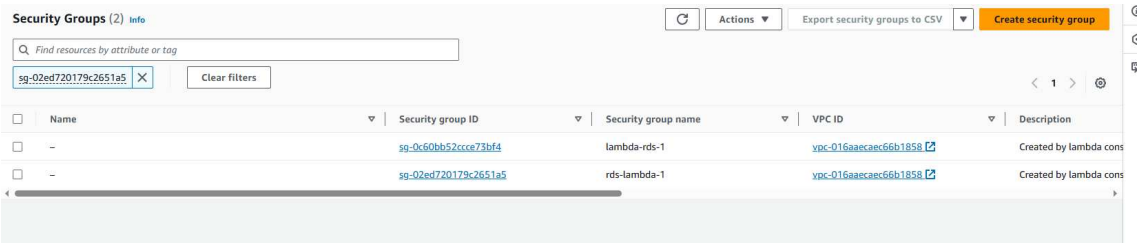


Figura 3.46: Security group per i backup con funzioni Lambda

La figura 3.48 mostra un Security Group su AWS chiamato rds-lambda-1, creato tramite la console Lambda.In questo gruppo è presente una regola per gestire l'ingresso. Questa regola consente il traffico MSSQL sulla porta TCP 1433 utilizzata per Microsoft SQL Server. Il traffico può arrivare solo da un altro Security Group, in particolare quello associato alla risorsa lambda-rds-1 (sg-0c60bb52ccce73bf4).

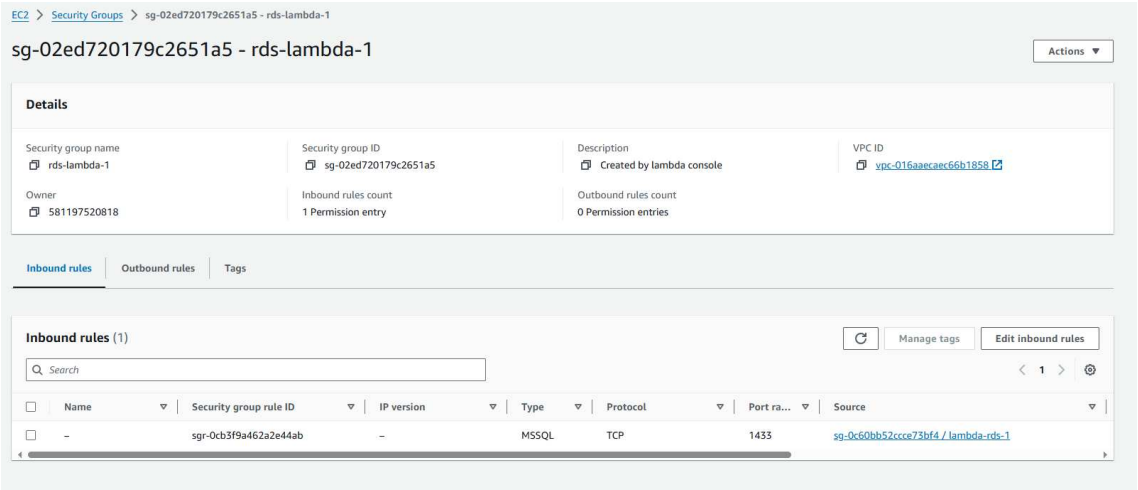


Figura 3.47: Security group per i backup con funzioni Lambda

La figura 3.48 mostra le regole in uscita verso il database RDS.Questo Security Group non ha regole di ingresso, quindi non consente connessioni in entrata. Tuttavia, ha due regole di uscita. La prima regola permette al traffico MSSQL sulla porta TCP 1433 di raggiungere un altro Security Group associato a un'istanza RDS con Microsoft SQL Server. La seconda regola consente tutto il traffico verso un endpoint AWS, che sembra essere collegato a un bucket S3 nella regione eu-south-1.

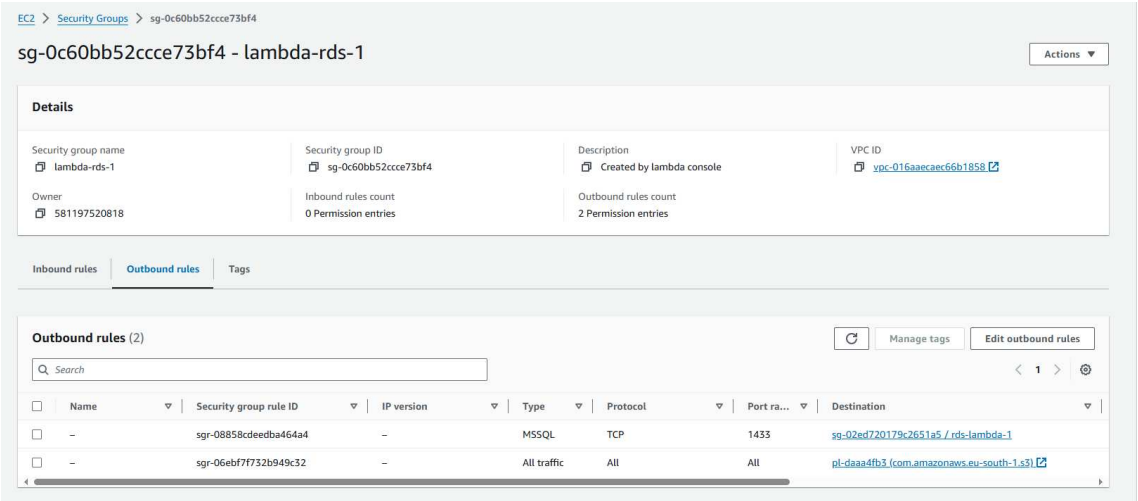


Figura 3.48: Codice per la creazione dei backup

3.11 Amazon Cognito

Amazon Cognito è un servizio dedicato alla gestione dell'autenticazione e dell'autorizzazione degli utenti nelle applicazioni. Questo strumento è composto da due elementi principali: User Pools e Identity Pools.

I pool di utenti fungono da directory per gli utenti, offrendo un sistema completo per la gestione dell'autenticazione. Consentono agli sviluppatori di configurare la registrazione, il login e il recupero delle credenziali. Attraverso User Pools, gli utenti possono registrarsi utilizzando nome utente e password, o accedere tramite account di terze parti come Google

e Facebook. Una caratteristica vantaggiosa è l'interfaccia predefinita per la registrazione e l'accesso, che può essere personalizzata per adattarsi al design dell'applicazione. In questo caso è stato creato un CSS personalizzato per modificare la pagina di accesso e di registrazione.

Al contrario, gli Identity Pools si concentrano sull'autorizzazione, fornendo accesso alle risorse AWS. Mentre i User Pools gestiscono l'autenticazione, gli Identity Pools generano credenziali temporanee per consentire l'accesso ad altri servizi AWS come S3.
Per il software dell'azienda, abbiamo optato per User Pools per la gestione degli utenti, come mostrato nella figura 3.49.

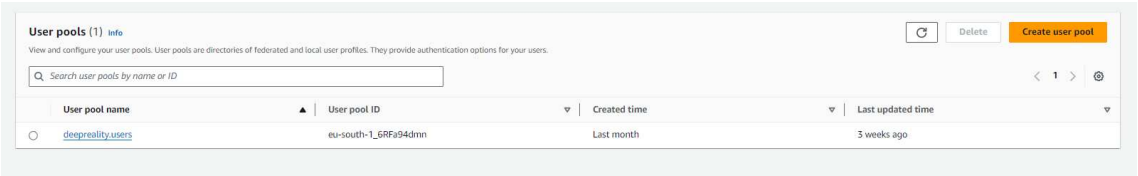


Figura 3.49: User pool Deep Reality

È stata implementata la registrazione e l'accesso tramite un'interfaccia personalizzata, realizzata modificando il layout predefinito con il proprio CSS. Le schermate di accesso e registrazione sono mostrate nelle figure 3.50 e 3.51.

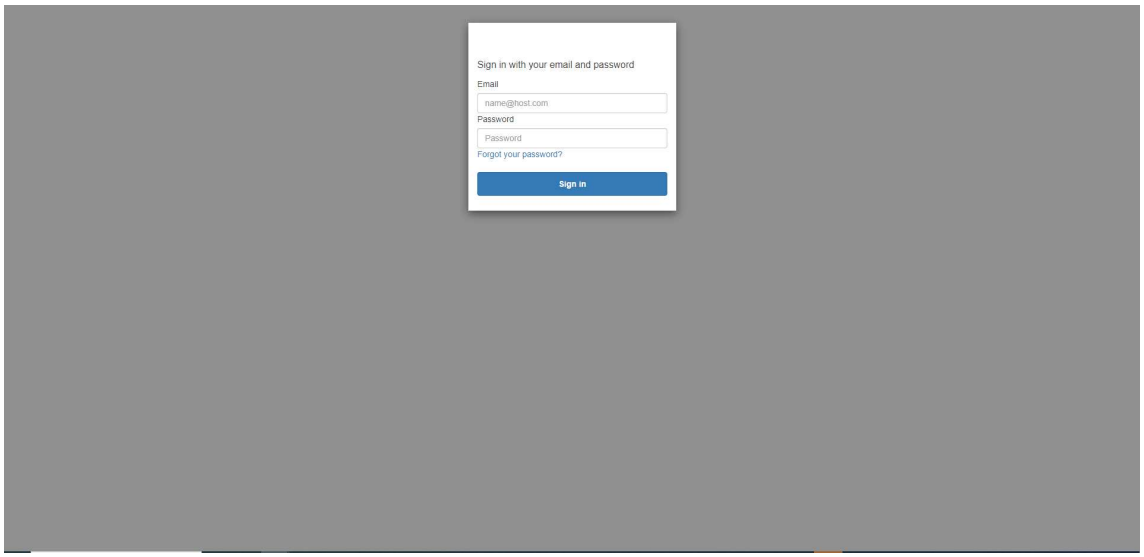


Figura 3.50: Login Insideep

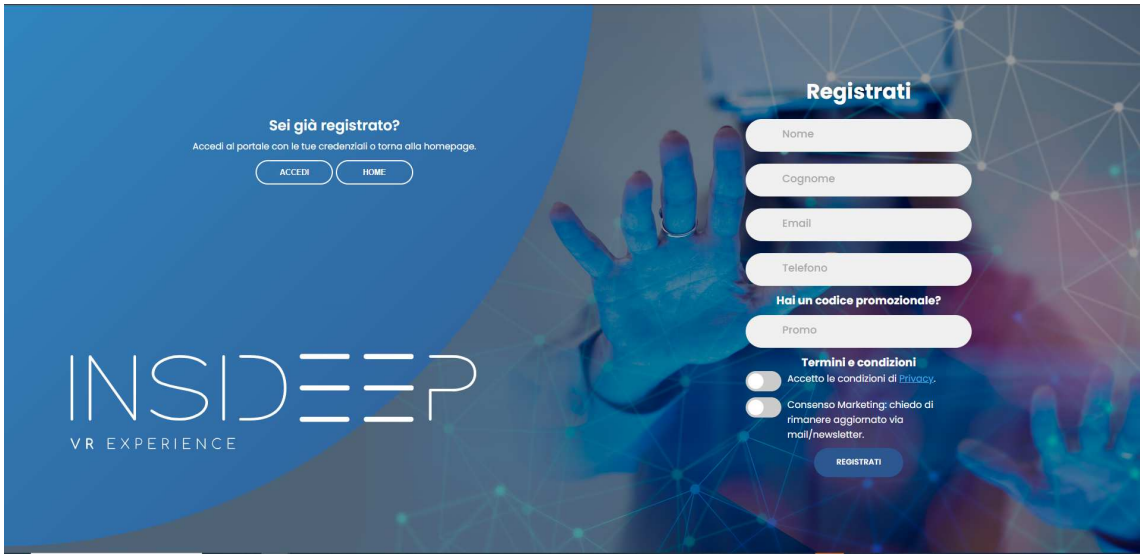


Figura 3.51: Registrazione Insideep

I pool di utenti offrono funzionalità avanzate di sicurezza, come l'autenticazione multifattore (MFA), permettendo agli utenti di utilizzare app di autenticazione o altri servizi per aumentare la protezione. Sono disponibili anche opzioni per il cambio della password e la verifica dell'email, semplificando la gestione delle credenziali e mantenendo la sicurezza. Nella figura 3.52 è possibile vedere alcuni dettagli sull'autenticazione del pool di utenti.

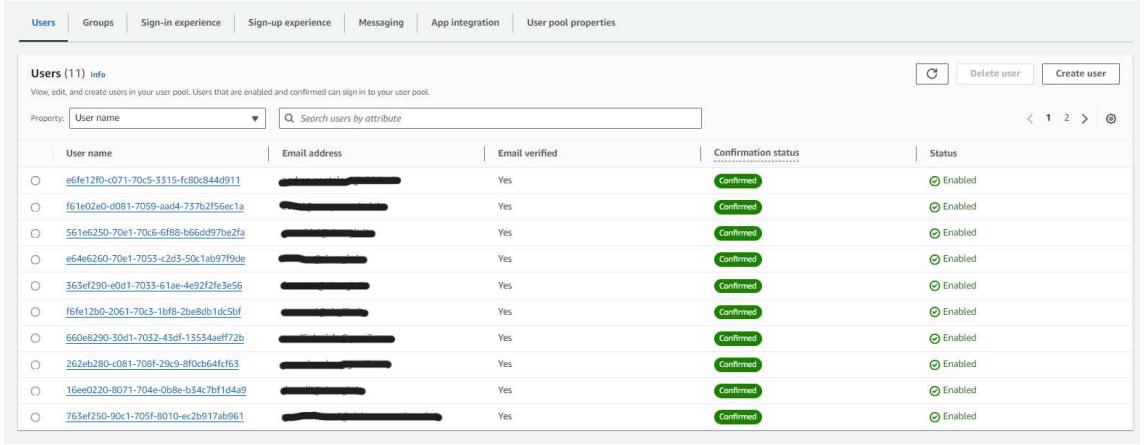


Figura 3.52: Gestione degli utenti

Abbiamo utilizzato i DNS dell'azienda per facilitare un accesso diretto, gestendo le porte necessarie per consentire l'accesso alla pagina di accesso dell'applicazione.
Amazon Cognito permette di definire delle applicazioni per gestire l'autenticazione e l'autorizzazione, come illustrato nella figura 3.64

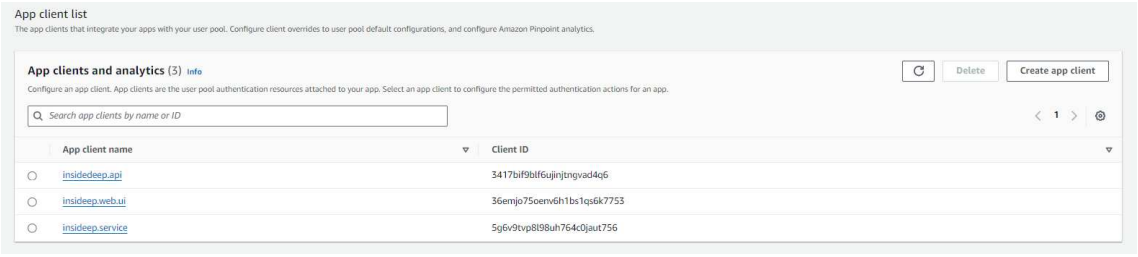


Figura 3.53: Gestione dei client

Il client *insidedeep.api* permette di gestire l'autenticazione tramite Client ID e Secret ID alle API messe a disposizione su istanza EC2. I dettagli del Client sono visibili nelle figure 3.65 e 3.65

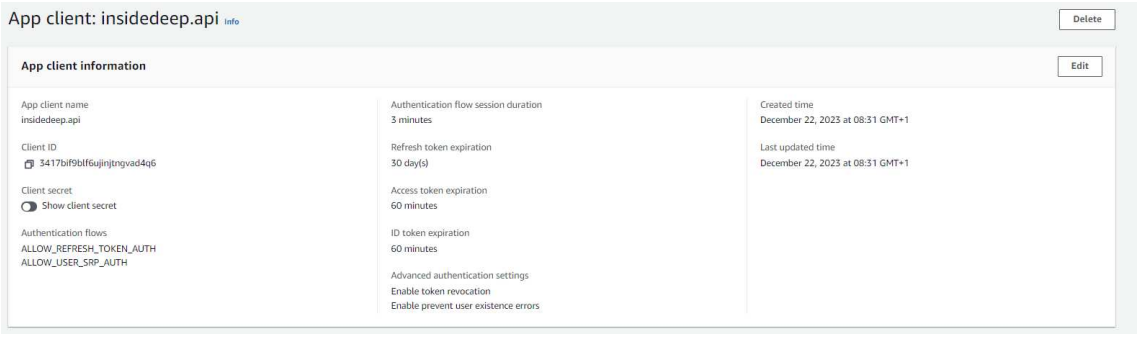


Figura 3.54: Dashboard client API

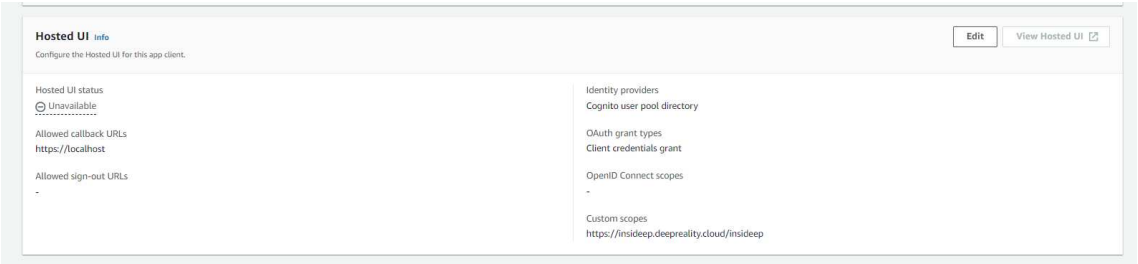


Figura 3.55: Dettagli client API

Nella figure 3.56 e 3.57 è illustrato il Client che permette di effettuare i re-indirizzamenti dalla pagina di accesso alle pagine di Registrazione e Home. L'autenticazione degli utenti avviene tramite username e password.

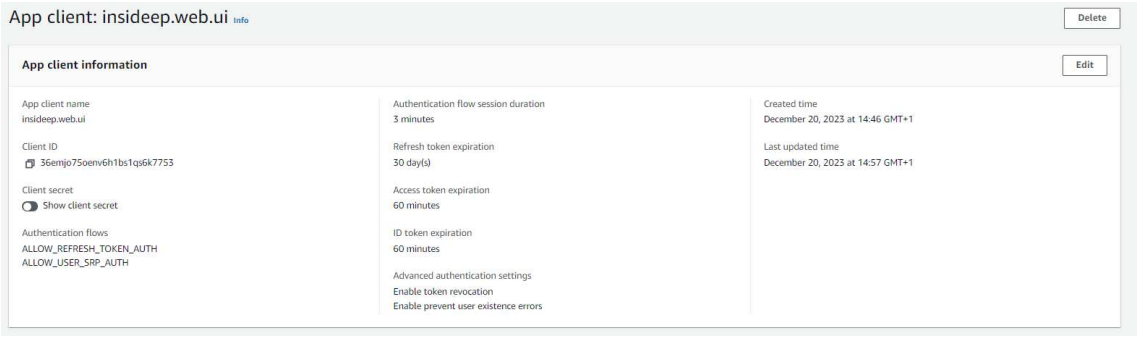


Figura 3.56: Dashboard client web UI

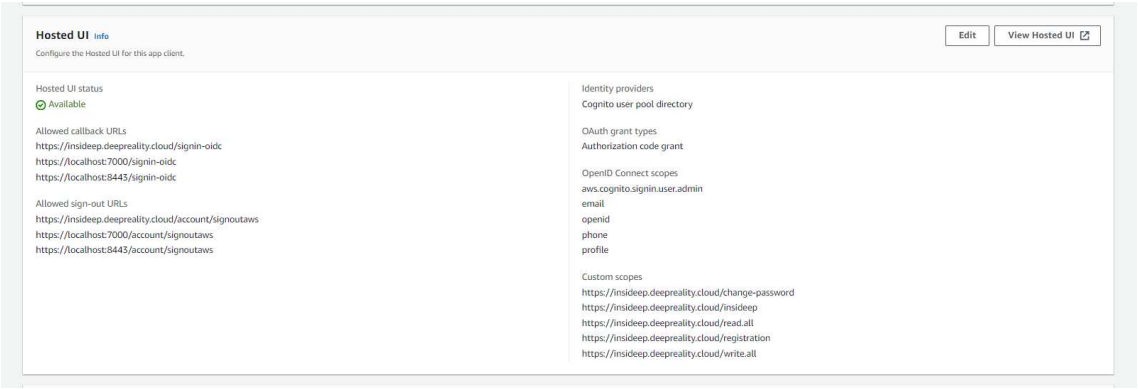


Figura 3.57: Re-indirezzamenti interfaccia grafica

L'applicazione si interfaccia con delle API di terze parti e gestisce l'autenticazione con il Client *insideep.service*. Nella figure 3.58 e 3.59 è possibile vedere i dettagli del Client.

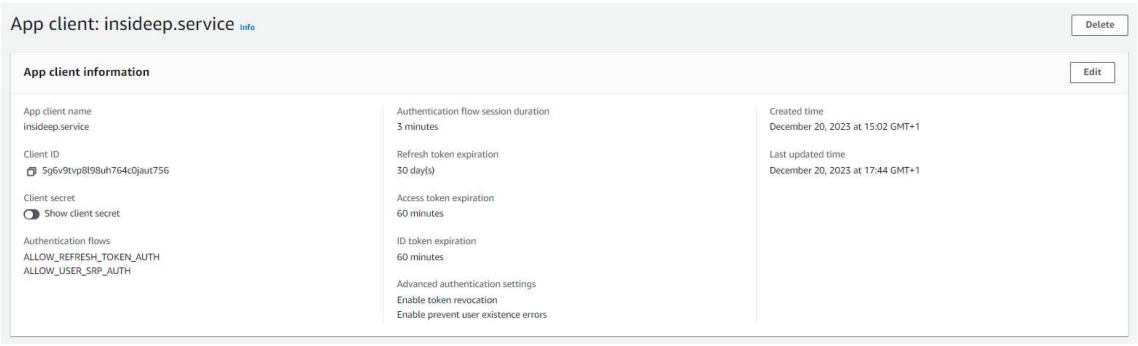


Figura 3.58: Dashboard client service

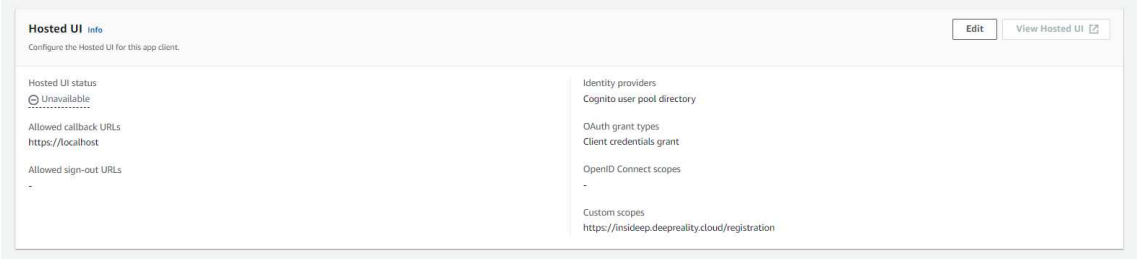


Figura 3.59: Re-indirizzamento registrazione

arte di un'applicazione che interagisce con Amazon Cognito per rinnovare un token di accesso utilizzando un token di refresh. In pratica, questo processo permette all'applicazione di continuare ad accedere a risorse protette senza dover chiedere nuovamente all'utente le credenziali di autenticazione

3.11.1 Gestione di Autorizzazione e Autenticazione

Le figure 3.60 e 3.61 descrivono la parte di un'applicazione che interagisce con Amazon Cognito per rinnovare un token di accesso utilizzando un token di refresh. In pratica, questo processo permette all'applicazione di continuare ad accedere a risorse protette senza dover chiedere nuovamente all'utente le credenziali di autenticazione.

La funzione asincrona *RefreshTokenAsync* crea un client *var identityProvider = new AmazonCognitoIdentityProviderClient()* per interagire con il servizio Amazon Cognito Identity Provider, un oggetto *CognitoUserPool userPool = new CognitoUserPool()* che rappresenta il pool di utenti ed una richiesta di autenticazione *var req = new AdminInitiateAuthRequest()*.

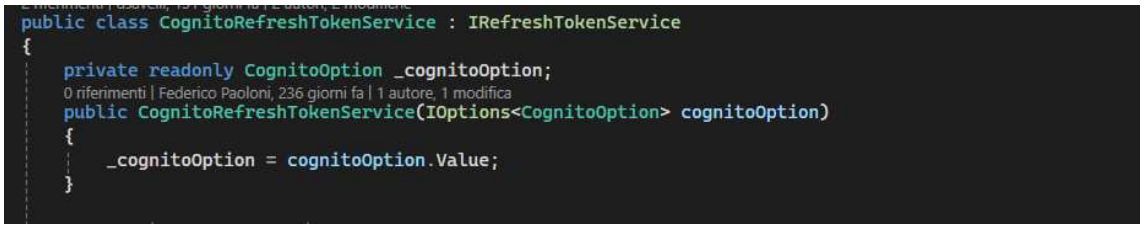


Figura 3.60: Interfaccia refresh token

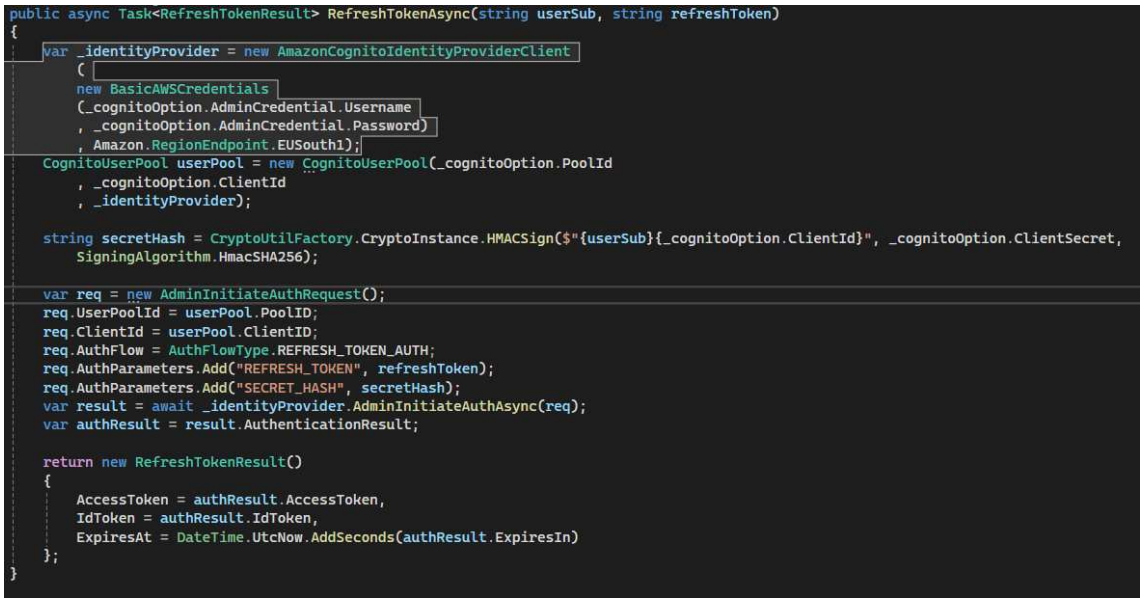


Figura 3.61: Funzione asincrona per gestione del refresh token

Le figure 3.62 e 3.63 illustrano una funzione che interagire con Amazon Cognito per creare un nuovo utente. In particolare, questa funzione, chiamata *AdminCreateUserAsync*, è responsabile di inviare una richiesta al servizio Cognito per creare un nuovo utente con le credenziali specificate.

```
public async Task<CreateUserResult> AdminCreateUserAsync(
string username,
string password,
List<AttributeType> attributeTypes)
{
    CreateUserResult result = new CreateUserResult();
    try
    {
        AdminCreateUserRequest adminCreateUserRequest = new AdminCreateUserRequest
        {
            Username = username,
            TemporaryPassword = password,
            UserPoolId = _cognitoOption.PoolId,
            UserAttributes = attributeTypes
        };
        AdminCreateUserResponse adminCreateUserResponse = await _adminAmazonCognitoIdentityProviderClient
            .AdminCreateUserAsync(adminCreateUserRequest)
            .ConfigureAwait(false);
```

Figura 3.62: Funzione per la creazione di utenti

```
        AdminUpdateUserAttributesRequest adminUpdateUserAttributesRequest = new AdminUpdateUserAttributesRequest
        {
            Username = username,
            UserPoolId = _cognitoOption.PoolId,
            UserAttributes = new List<AttributeType>
            {
                new AttributeType()
                {
                    Name = "email_verified",
                    Value = "true"
                }
            }
        };

        AdminUpdateUserAttributesResponse adminUpdateUserAttributesResponse = await _adminAmazonCognitoIdentityProviderClient
            .AdminUpdateUserAttributesAsync(adminUpdateUserAttributesRequest);

        result.Success = true;

        result.ExternalUserId = adminCreateUserResponse.User.Attributes.Where(w => w.Name == "sub").First().Value;
    }
    catch (Exception ex)
    {
        result.Message = ex.Message;
        //result.Message = "L'utente è già registrato";

    }

    return result;
}
```

Figura 3.63: Funzione per la creazione di utenti - 2

La figura 3.64 mostra una funzione asincrona che si occupa di eliminare un utente con specifiche credenziali. Oltre all'eliminazione e alla creazione sono state implementate anche le funzioni di modifica e di lettura dei dati degli utenti, queste risultano essere molto simili alle due funzioni precedentemente descritte.

```
4 riferimenti | Federico Paoloni, 231 giorni fa | 1 autore, 1 modifica
public async Task<DeleteUserResult> AdminDeleteUserAsync(string username)
{
    DeleteUserResult result = new DeleteUserResult();
    var deleteUserRequest = new AdminDeleteUserRequest { Username = username
        ,UserPoolId = _cognitoOption.PoolId,
    };

    var adminDeleteResult = await _adminAmazonCognitoIdentityProviderClient.AdminDeleteUserAsync(deleteUserRequest)
        .ConfigureAwait(false);

    result.Success = (int)adminDeleteResult.HttpStatusCode >= 200 && (int)adminDeleteResult.HttpStatusCode <= 299;

    return result;
}
```

Figura 3.64: Funzione per eliminazione utenti

La figura 3.65 descrive la funzionalità di cambio password di un utente. Questa funzione asincrona effettua una richiesta di cambio password utilizzando gli strumenti messi a disposizione da *AWS SDK*.

```
2 riferimenti | Federico Paoloni, 174 giorni fa | 1 autore, 1 modifica
public async Task<ChangePasswordResult> ChangePasswordUserAsync(string accessToken,string oldPassword, string newPassword)
{
    ChangePasswordResult result = new ChangePasswordResult();
    var changePasswordRequest = new ChangePasswordRequest
    {
        AccessToken  = accessToken,
        PreviousPassword = oldPassword,
        ProposedPassword = newPassword
    };


    var changePasswordResult = await _adminAmazonCognitoIdentityProviderClient.ChangePasswordAsync(changePasswordRequest)
        .ConfigureAwait(false);

    result.Success = (int)changePasswordResult.HttpStatusCode >= 200 && (int)changePasswordResult.HttpStatusCode <= 299;

    return result;
}
```

Figura 3.65: Funzione per il cambio di password

3.12 Application Load Balancer

L'Application Load Balancer (ALB) di Amazon è una componente essenziale per consentire alle istanze EC2 di interfacciarsi con Internet in modo sicuro e controllato. L'ALB è progettato per gestire il traffico in entrata verso una o più istanze EC2, instradandolo solo verso le porte e i servizi specifici che si desidera esporre, garantendo così una maggiore sicurezza e controllo. Attraverso l'ALB, è possibile definire regole precise per il bilanciamento del carico, distribuendo uniformemente le richieste tra le istanze EC2 per migliorare l'affidabilità e la scalabilità dell'applicazione.

La figura 3.66 mostra due componenti fondamentali dell'Application Load Balancer:

- **Listeners**: un listener è un processo che controlla le richieste di connessione utilizzando un protocollo e un numero di porta specifici. Quando l'ALB riceve una richiesta, il listener ascolta il traffico in entrata e lo inoltra in base alle regole definite.
- **Rules**: le regole determinano come l'Application Load Balancer instrada il traffico. Le regole sono composte da una combinazione di condizioni e azioni.

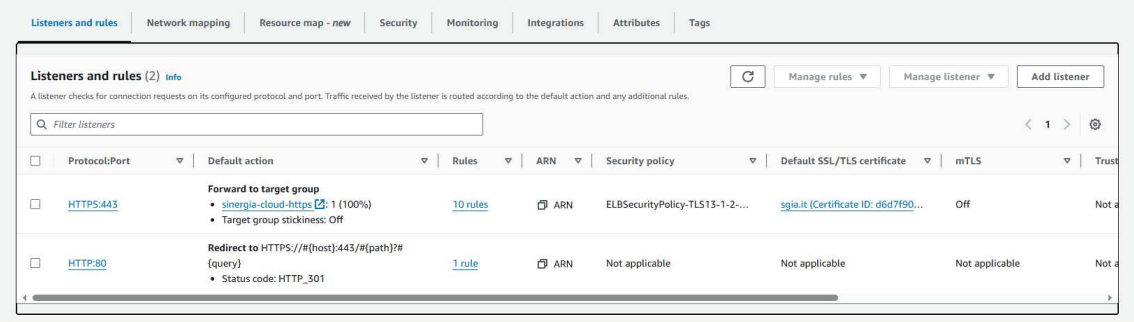


Figura 3.66: Regole ALB

Un aspetto cruciale del processo è la gestione sicura del traffico attraverso l'utilizzo di certificati SSL/TLS. Per garantire che il traffico tra l'ALB e i client esterni sia crittografato, si utilizzano certificati SSL emessi e gestiti dal servizio AWS Certificate Manager (ACM). AWS ACM semplifica la gestione dei certificati, permettendo di generare, rinnovare e collegare certificati SSL direttamente all'ALB. Questo garantisce che tutte le comunicazioni tra i client e l'ALB avvengano tramite connessioni sicure HTTPS, proteggendo i dati durante il transito e prevenendo attacchi.

Nella tabella 3.1 sono indicate le porte esposte per i servizi AWS attivi nell'infrastruttura.

Target Group	Port	Protocol
sgia	80	HTTP
sgia-dr	80	HTTP
sinergia-cloud-https	443	HTTPS
sinergia-cloud-tg	8080	HTTP
insideep-api	8090	HTTP
insideep-https	8443	HTTPS
sgia-impronta	9080	HTTP
sgia-sinergia	9081	HTTPS
sgia-epc	9082	HTTP
www-sinergia-cloud	9083	HTTP

Tabella 3.1: Tabella ALB

3.12.1 Sicurezza Application Load Balancer

La sicurezza dell'Application Load Balancer è stata gestita utilizzando i Security Group. Come per le istanze descritte nelle sezioni precedenti, anche in questo caso agiscono come firewall virtuali che controllano il traffico in entrata e in uscita.
Come descritto nelle figure 3.67 e 3.68, abbiamo configurato il security group in modo tale da consentire l'accesso in entrata solo su determinate porte, garantendo così che l'ALB sia accessibile solo attraverso le porte necessarie al corretto funzionamento dell'applicazione.

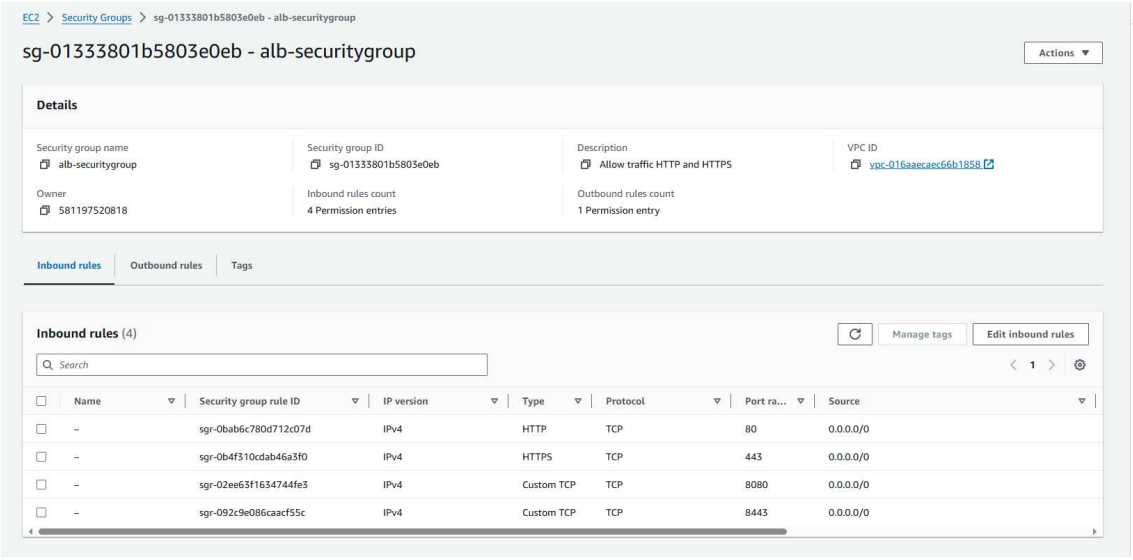


Figura 3.67: Security group per traffico in ingresso ALB

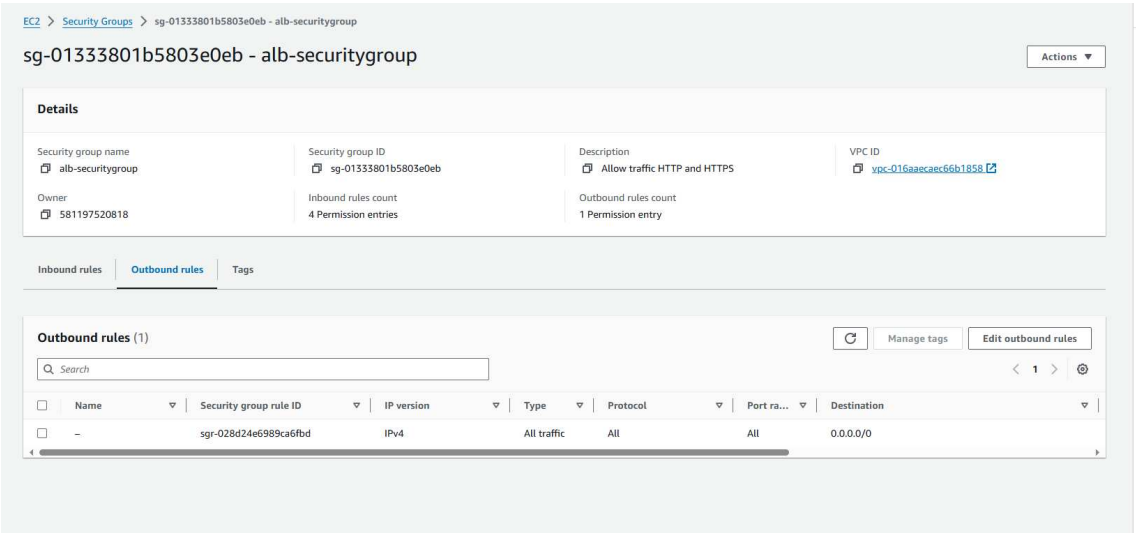


Figura 3.68: Security group per traffico in uscita ALB

3.13 Amazon CloudWatch

Amazon CloudWatch è un servizio di monitoraggio e osservabilità offerto da AWS, progettato per raccogliere, visualizzare e analizzare metriche, log e eventi provenienti da risorse AWS e applicazioni. L'obiettivo principale di CloudWatch è fornire visibilità sulle performance delle infrastrutture cloud e aiutare a gestire le operazioni.
Questo servizio raccoglie metriche in tempo reale da una vasta gamma di servizi AWS. Le metriche monitorate possono includere parametri relativi all'utilizzo della CPU, memoria, disco, traffico di rete, numero di richieste API o errori.
Inoltre, è possibile definire e monitorare metriche personalizzate create dall'utente, fornendo un quadro dettagliato delle prestazioni e dello stato delle risorse .

Nella figura 3.69 è possibile vedere la dashboard di CloudWatch che monitora il processo di

creazione di backup dei database attraverso le Lambda Functions. Questa dashboard offre una panoramica in tempo reale delle operazioni, consentendo di monitorare l'andamento e le prestazioni delle funzioni Lambda durante la creazione dei backup.

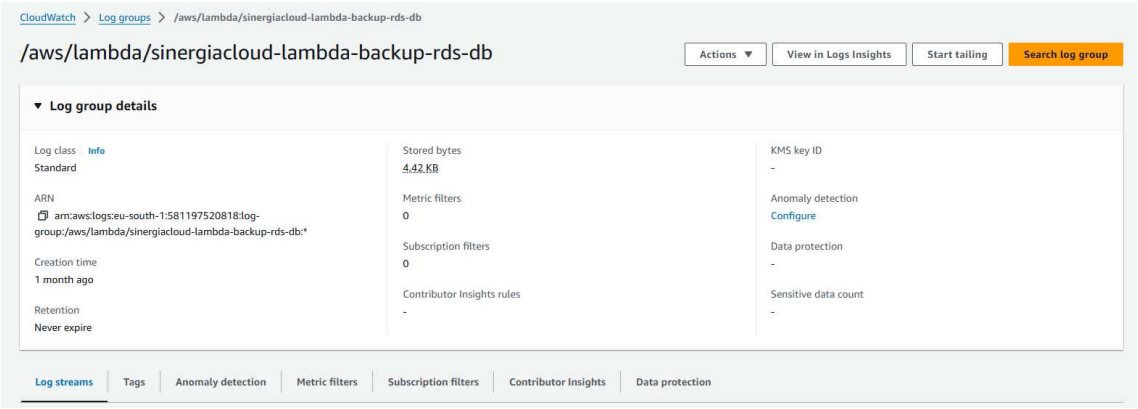


Figura 3.69: Dashboard CloudWatch

Nella figura ?? sono visualizzati i gruppi di log creati per facilitare l'accesso ai messaggi specifici generati dalle applicazioni e dai servizi. Ogni gruppo di log raccoglie le informazioni in base a specifiche categorie, rendendo più semplice la ricerca e la gestione dei log per l'analisi e la risoluzione dei problemi.

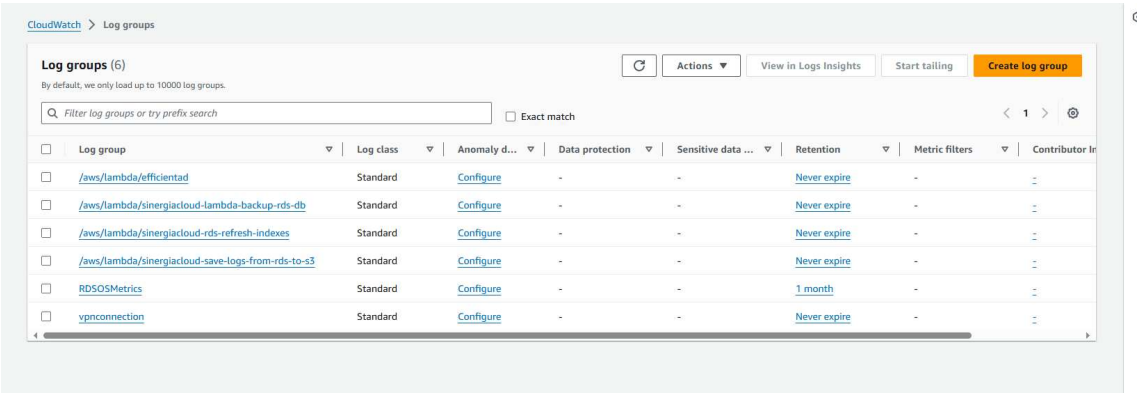


Figura 3.70: Gruppi di log in CloudWatch

Infine, la figura 3.71 mostra un esempio di log stream, ovvero un flusso continuo di eventi di log provenienti da una risorsa specifica, come una funzione Lambda o un'istanza EC2. Questo consente di visualizzare i dettagli di ciascun evento in sequenza, facilitando il monitoraggio delle attività e la diagnosi di eventuali errori o anomalie

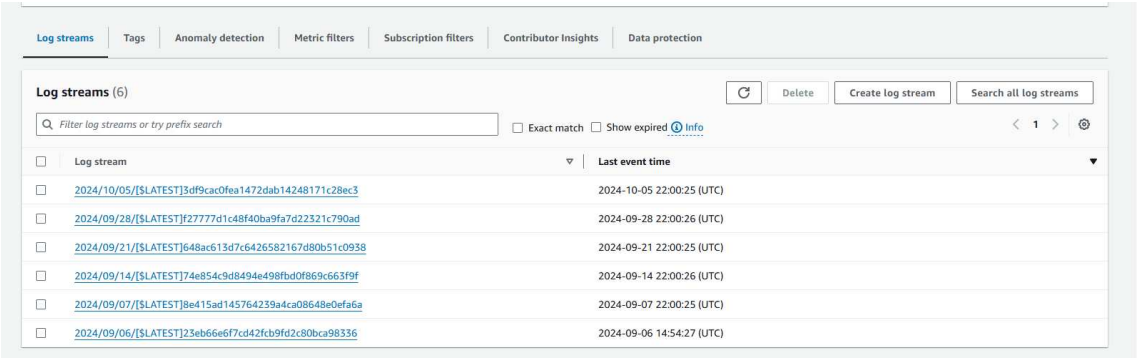


Figura 3.71: Log stream in CloudWatch

3.14 Amazon SES

Amazon Simple Email Service (SES) è un servizio di invio e ricezione di email altamente scalabile e affidabile.
Durante la configurazione iniziale, è necessario verificare il dominio o l'indirizzo email da cui si intende inviare le comunicazioni. Questo passaggio è fondamentale per garantire la legittimità delle email e prevenire l'invio di spam. Per dimostrare la proprietà del dominio o dell'indirizzo email, Amazon SES richiede una verifica, che può avvenire tramite l'aggiunta di record DNS o cliccando su un link di verifica inviato all'indirizzo email.

Questo servizio offre API RESTful e supporta SMTP (Simple Mail Transfer Protocol), consentendo agli sviluppatori di integrare facilmente il servizio nelle loro applicazioni.
Una volta configurato, è possibile inviare email in vari formati, tra cui testo semplice e HTML. Amazon SES supporta anche l'invio di email in massa, consentendo alle aziende di raggiungere grandi elenchi di destinatari senza comprometterne le prestazioni.
Nella figura 3.72 è possibile vedere la Dashboard del servizio, configurato per l'applicazione Insideep.

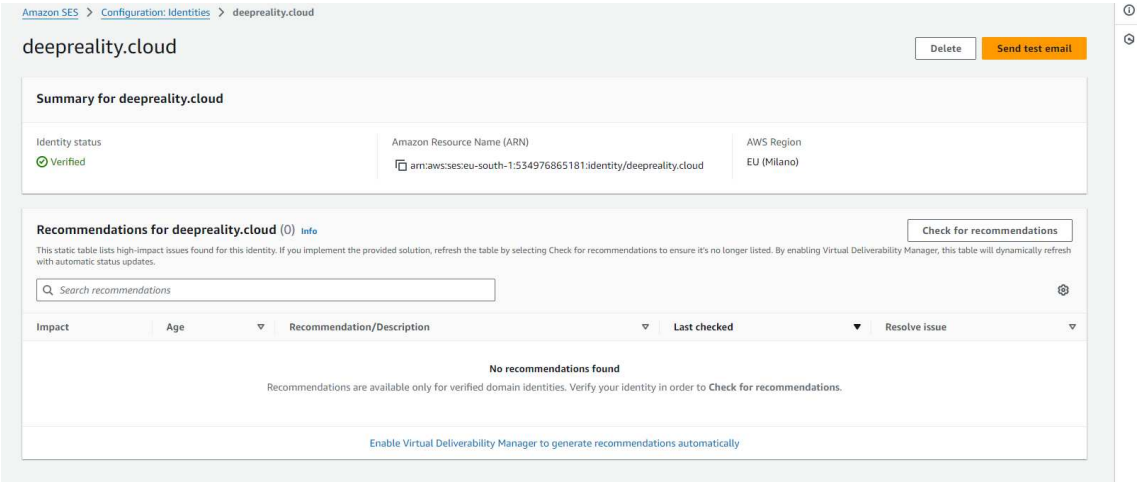


Figura 3.72: Dashboard Servizio Amazon SES

Capitolo 4

Conclusione

Durante il mio tirocinio presso Sinergia EPC, ho avuto l'opportunità di approfondire diverse tecniche di migrazione, come Rehosting e Replatforming, e di utilizzare e integrare i servizi più conosciuti di AWS.
È stata sviluppata una Virtual Private Cloud (VPC) e gestite le subnet all'interno di essa. Inoltre, è stata effettuata la migrazione delle istanze EC2 e RDS da cloud esistenti, cercando di apportare il minor numero possibile di modifiche alle istanze. È stato utilizzato CloudWatch per monitorare le istanze e Amazon S3 per garantire la persistenza di backup e file multimediali.
È stato impiegato anche il servizio VPN di AWS per consentire un accesso sicuro alle applicazioni presenti nell'infrastruttura e per gestire la sicurezza di tutti questi servizi.

Gli obiettivi stabiliti per la creazione dell'infrastruttura durante la fase di progettazione sono stati quasi completamente raggiunti. La fase non completamente sviluppata è stata quella di DevOps, che include lo spostamento delle applicazioni in container Docker, l'utilizzo di un software di orchestrazione come Kubernetes e la creazione di pipeline per l'integrazione e la distribuzione continua del software. Questo approccio renderà le applicazioni più flessibili e scalabili, migliorando significativamente la loro gestione e il processo di rilascio.

In seguito alla conclusione del tirocinio,si è ritenuto opportuno sostituire alcuni servizi con soluzioni più flessibili e meno costose. Un esempio significativo di questa revisione è rappresentato dalla transizione dalla VPN precedentemente utilizzata a un'istanza EC2, la quale ora funge da server VPN. Questa modifica ha permesso non solo di semplificare la gestione dell'infrastruttura, ma ha anche contribuito a una significativa riduzione dei costi operativi.

L'intero processo di sviluppo ha avuto un impatto significativo sul mio percorso, consentendomi di seguire da vicino e di contribuire attivamente alla realizzazione di un'infrastruttura moderna, capace di gestire carichi di lavoro di varia natura. Ho acquisito competenze pratiche e teoriche, imparando a implementare soluzioni scalabili e adatte a soddisfare le esigenze dinamiche del business.
Anche se non è stato possibile sviluppare tutto ciò che inizialmente avevamo pianificato, è stato comunque altamente formativo poter approfondire argomenti come la creazione di container, la distribuzione delle applicazioni e l'integrazione continua.

In aggiunta, ho avuto l'opportunità di approfondire le migliori pratiche in materia di si-

curezza informatica, fondamentali per la protezione delle istanze e delle reti. La sicurezza delle informazioni rappresenta un aspetto cruciale nella progettazione di qualsiasi infrastruttura IT, e l'adozione di strategie adeguate permette di mitigare i rischi associati a potenziali vulnerabilità.

In conclusione, il mio tirocinio e l'attività di tesi collegata unitamente con i successivi sviluppi dell'infrastruttura cloud mi hanno offerto una comprensione approfondita delle sfide e delle opportunità legate alla gestione di servizi in ambienti cloud. L'adozione e l'ottimizzazione delle risorse cloud richiedono un approccio flessibile e dinamico, dove la capacità di adattarsi rapidamente ai cambiamenti tecnologici e di ottimizzare continuamente l'uso delle risorse è fondamentale per mantenere l'efficienza operativa, la scalabilità e la sostenibilità economica delle soluzioni implementate.

Nelle sezioni seguenti, saranno illustrati due servizi che rappresentano gli sviluppi futuri previsti dall'azienda. Questi servizi saranno utilizzati principalmente per implementare un sistema di CI/CD e per la gestione dei container.

4.1 AWS CodePipeline

La gestione di un software richiede una struttura scalabile, sicura e che consenta l'integrazione e la distribuzione continua (CI/CD) delle modifiche apportate dagli sviluppatori. A tal fine, Amazon Web Services mette a disposizione il servizio AWS CodePipeline, il quale permette di orchestrare le fasi principali dei due processi.

Questi processi si articolano in *Pipeline*, ovvero in una sequenza di fasi (*stages*) utilizzate per automatizzare il ciclo di vita dello sviluppo software. Tale approccio garantisce il rilascio di un software stabile, sicuro e pienamente funzionante in ambiente di produzione.

In generale, le fasi che una modifica del codice deve attraversare sono le seguenti:

- *Build*: in questa fase il codice sorgente viene compilato e tutte le dipendenze necessarie vengono risolte. Questo passaggio serve a garantire che il codice possa essere trasformato in un formato eseguibile.
- *Test*: vengono eseguiti una serie di test, sia automatici che manuali, sul codice compilato. L'obiettivo è quello di verificare che il software funzioni correttamente e che non ci siano errori, bug o comportamenti inattesi.
- *Deploy*: una volta superati con successo i test, il codice viene distribuito (*deployed*) in ambienti predefiniti, solitamente ambienti di test o di staging. Questo consente di testare ulteriormente il software in un contesto che replica l'ambiente di produzione.
- *Release*: infine, il codice che ha superato tutte le verifiche viene rilasciato in ambiente di produzione, diventando disponibile agli utenti finali. Questa fase conclude il ciclo di vita del rilascio della modifica e mette il software in condizioni operative.

Il servizio AWS CodePipeline può essere integrato con software di terze parti per eseguire ognuna delle fasi della pipeline. Dato che l'infrastruttura è interamente composta da servizi Amazon, per lo sviluppo di una futura pipeline si è considerato l'utilizzo di servizi AWS, come mostrato in figura 4.1, per eseguire gli stage. In particolare:

- **AWS CodeCommit**: è uno strumento di versioning alternativo ai più conosciuti Github o GitLab.

- **AWS CodeBuild**: è uno strumento che permette la compilazione dell'applicativo e l'esecuzione di test. È quindi possibile eseguire due fasi con lo stesso servizio.
- **AWS CodeDeploy**: questo servizio permette il rilascio in ambienti di produzione, come ad esempio server EC2.

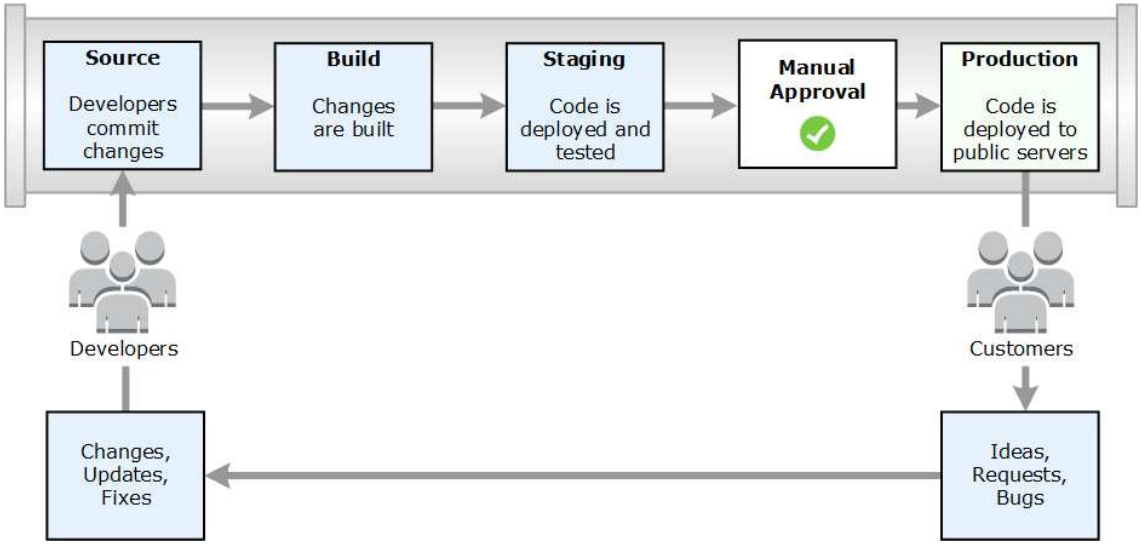


Figura 4.1: Servizi AWS per CI/CD [14]

il servizio AWS CodeBuild utilizza un file in formato *YAML* chiamato *buildspec.yaml*, in cui è possibile definire i diversi test da eseguire sul codice in caso di modifica e rilascio. Essendo il codice utilizzato fino ad ora scritto in linguaggio C# e con l'utilizzo del framework .NET, un semplice file di configurazione yaml per i test e per la generazione di un artefatto è il seguente

```
version: 0.X

phases:
  install:
    runtime-versions:
      dotnet: 6.0 # versione .NET
    commands:
      - echo Installing dependencies...
      - dotnet restore

  pre_build:
    commands:
      - echo Pre-build: Verifying and preparing env...

  build:
    commands:
      - echo Build started on `date`
      - dotnet build --configuration Release
```

```
  post_build:
    commands:
      - echo Running tests...
      - dotnet test --no-restore --verbosity normal
      - echo Build completed on `date`

artifacts:
  files:
    - '**/bin/Release/netX.0/publish/**'
  discard-paths: yes
  base-directory: 'src/MyProjectTest'
```

4.2 Amazon Elastic Container Service

Amazon ECS (Elastic Container Service) è un servizio di orchestrazione per applicazioni containerizzate che permette di eseguire, gestire e scalare container Docker su larga scala senza doversi occupare della gestione manuale dell'infrastruttura. Progettato per offrire un ambiente affidabile e sicuro, ECS facilita la distribuzione di applicazioni sfruttando l'integrazione con i principali servizi del cloud AWS.

Una delle caratteristiche più importanti di Amazon ECS è la flessibilità nella gestione dei container. È possibile scegliere tra due modalità di esecuzione: utilizzando istanze EC2, dove si ha un controllo diretto sulle risorse di calcolo, o tramite Fargate, una soluzione serverless che consente di concentrarsi esclusivamente sulle risorse richieste dai container senza preoccuparsi della gestione dei server sottostanti. Questa doppia opzione permette di adattare il servizio alle necessità specifiche del progetto, sia per piccole applicazioni che per sistemi più complessi.

ECS si integra in modo nativo con altri strumenti all'interno dell'ecosistema AWS, come ad esempio Amazon VPC per la configurazione di rete, Amazon ECR per la gestione delle immagini Docker e CloudWatch per il monitoraggio e il logging delle applicazioni. Questa integrazione semplifica la gestione dei container e consente una maggiore automazione, rendendo ECS particolarmente adatto per ambienti di produzione e applicazioni distribuite.

Nella figura 4.2 è possibile vedere le fasi necessarie per effettuare il deploy di un container su Amazon ECS, utilizzando il servizio Amazon Fargate per l'esecuzione.

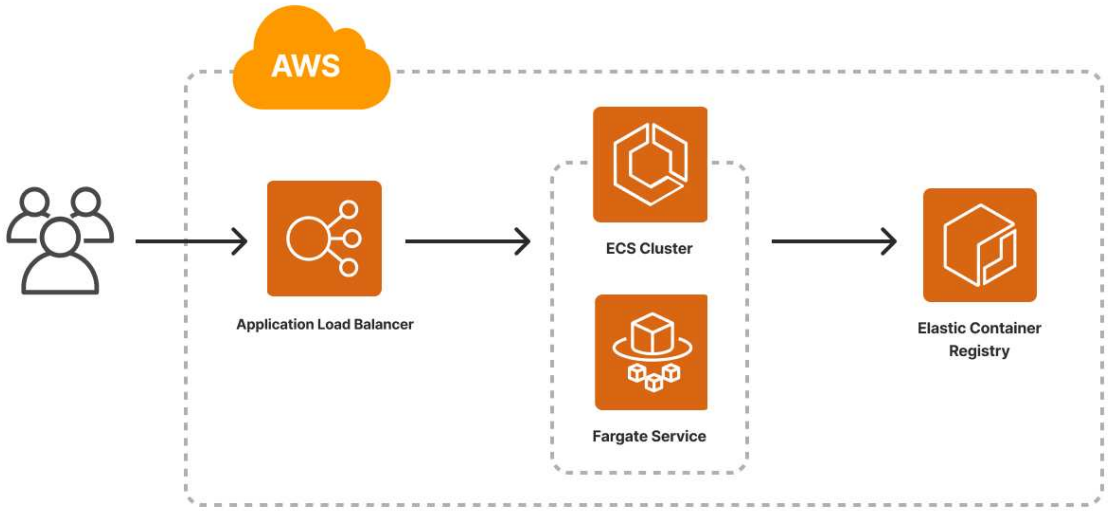


Figura 4.2: Amazon ECS Deploy [15]

Dal punto di vista della sicurezza, ECS offre un controllo preciso grazie all'integrazione con IAM (Identity and Access Management), permettendo di definire autorizzazioni granulari per ogni container. Inoltre, le applicazioni eseguite su ECS possono essere completamente isolate tramite l'uso di Virtual Private Cloud, garantendo la sicurezza delle comunicazioni a livello di rete.

La capacità di scalare in modo automatico è un altro vantaggio fondamentale: ECS permette di adattarsi ai carichi di lavoro variabili, garantendo che le applicazioni siano sempre in grado di gestire il traffico senza interruzioni. Questa scalabilità si accompagna a una gestione semplificata del bilanciamento del carico, migliorando la distribuzione delle richieste tra i container.

Bibliografia

[1] Sandeep Bhowmik. *Cloud Computing*. Cambridge University Press, 2017.

[2] Cloud computing abstraction levels. https://livebook.manning.com/concept/cloud/internet.

[3] Public cloud. https://www.netvault.net.au/cloud/public-cloud/.

[4] Private cloud. https://www.tierpoint.com/blog/private-cloud-architecture/.

[5] Hybrid cloud. https://www.netvault.net.au/cloud/public-cloud/.

[6] Chris Dotson. *Practical Cloud Security*. O'Reilly Media, 2019.

[7] Devops introduction. https://italiancoders.it/introduzione-al-devops/.

[8] Vms vs docker. https://www.openteams.com/getting-started-with-docker-for-machine-learning/.

[9] Documentazione docker. https://docs.docker.com/get-started/docker-overview.

[10] Docker swarm. https://docs.docker.com/engine/swarm/.

[11] Security groups aws. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-groups.html.

[12] Access control lists aws. https://docs.aws.amazon.com/it_it/vpc/latest/userguide/vpc-network-acls.html.

[13] Openvpn connect for windows. https://www.vpnbook.com/howto/setup-openvpn-on-windows11.

[14] Aws codepipeline. https://docs.aws.amazon.com/it_it/codepipeline/latest/userguide/welcome-introducing.html.

[15] Aws elastic container service. https://www.pulumi.com/templates/container-service/aws/.

[16] Nat gateway for private subnet. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat.html.

[17] Heartin Kanikathottu. *AWS Security Cookbook*. Packt Publishing, 2020.

[18] Dan C. Marinescu. *Cloud Computing Theory and Practice*. Morgan Kaufmann, 3rd edition, 2022.

[19] National Institute of Standards and Technology. Nist cloud computing reference architecture. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=909505, 2011. NIST Special Publication 500-292.

[20] National Institute of Standards and Technology. Nist cloud computing standards roadmap. https://www.nist.gov/system/files/documents/itl/cloud/NIST_SP-500-291_Version-2_2013_June18_FINAL.pdf, 2013. NIST Special Publication 500-291, Version 2.

[21] Mutua autenticazione per aws client vpn. https://docs.aws.amazon.com/it_it/vpn/latest/clientvpn-admin/client-auth-mutual-enable.html.

[22] Esportare un database sql server usando bakcup nativi. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/SQLServer.Procedural.Importing.html#SQLServer.Procedural.Importing.Native.Using.

Elenco delle figure

Elenco delle tabelle

Ringraziamenti

Vorrei concludere esprimendo un sentito ringraziamento a tutte le persone che mi hanno accompagnato e sostenuto, direttamente o indirettamente, lungo questo percorso.
Il mio primo ringraziamento va al mio correlatore, Flavio, per avermi permesso di collaborare con il fantastico team di Sinergia e per i preziosi consigli e insegnamenti che mi ha offerto.
Ringrazio anche il Prof. Adriano Mancini per la conoscenza trasmessa negli ultimi anni di università, che ha alimentato la mia passione per l'informatica e le nuove tecnologie. I suoi insegnamenti sono stati preziosi per il mio percorso formativo.
Un ringraziamento speciale va alla mia famiglia per il supporto che mi hanno dato in questi ultimi anni. La loro presenza e il loro incoraggiamento sono stati fondamentali per me.
Un grazie di cuore va alla mia compagna, Siham, il cui sostegno e affetto mi hanno aiutato ad affrontare le sfide. I tuoi preziosi consigli e il tuo continuo incoraggiamento mi hanno permesso di crescere e diventare una persona migliore.
Desidero ringraziare tutti i miei amici per i bellissimi momenti trascorsi insieme, che hanno reso questo percorso ancora più significativo. In particolare, voglio esprimere un grazie speciale a Diego e Michele, la cui compagnia e amicizia hanno reso indimenticabili le esperienze condivise.