POLITECNICO DI TORINO

Corso di Laurea in Ingeneria Informatica

Tesi di Laurea Magistrale

Cloud Security Assessment

Un approccio procedurale

Relatore
prof. Fulvio Valenza

Laureando
Lorenzo Gigli

Tutor Aziendali
Blue Reply Srl
Luigi Casciaro
Ivan Aimale

a.a 2021-2022

Sommario

In uno scenario in costante evoluzione, i servizi in cloud stanno prendendo sempre di più il sopravvento rispetto al panorama passato. Vista la rapida ascesa di questi nuovi paradigmi di sviluppo e distribuzione, si è venuta a creare una mancanza di esperienza e formazione, con conseguente aumento del rischio di produrre ambienti o applicazioni non sicure, che espongono le aziende ad attacchi informatici sempre più sofisticati. Il lavoro di tesi che segue vuole affrontare la tematica della sicurezza in cloud in maniera orizzontale su quelle che possono essere le tematiche fondamentali. Più precisamente il lavoro è organizzato su due fronti. Il primo prettamente teorico, di analisi e individuazione degli elementi di sicurezza fondamentali riguardanti il cloud computing. Il secondo, più pratico, si concentra sulla realizzazione di un framework per sviluppo di applicazioni in cloud. Questo framework si basa sugli studi preliminari svolti e su quello che risulta essere lo stato dell'arte delle tecnologie migliori per poter integrare la sicurezza in tutto il ciclo di sviluppo, seguendo l'approccio DevSecOps. Gli studi svolti hanno permesso l'individuazione di quattro grandi argomenti di sicurezza. Si è scelto di definirli come i quattro pilastri della sicurezza in cloud: Cloud Application Security, Cloud Data Security, Cloud Container Security e Gestione dell'ambiente cloud. È stata poi eseguita una ricerca di quelli che potessero essere gli strumenti ad oggi utilizzati per lo sviluppo di applicativi sicuri in cloud. Questa ricerca ha portato alla scelta di utilizzare la Pipeline Jenkins. La Pipeline Jenkins permette, grazie alla sua estrema compatibilità, di agganciare svariati strumenti di sicurezza nei vari stage che la caratterizzano. Jenkins è risultato essere pertanto lo strumento più adatto per la realizzazione del framework di sviluppo. Questo framework vuole porsi come strumento di base da cui attingere informazioni e metodologie. Per meglio far comprendere lo scopo di questo lavoro, si è scelto di fornire un caso d'uso esemplificativo. È stata scritta una semplice applicazione di test e la si è inserita in un contesto di sviluppo DevSecOps. In Jenkins sono stati collegati tool per analisi statica del codice come Sonarqube, tool per analisi delle dipendenze come Dependency Check di OWASP, e tool per l'analisi dei Dockerfile e delle immagini Docker come Trivy. Questi strumenti sono a titolo d'esempio e possono essere sostituibili con altri equivalenti o complementari. I risultati di questo lavoro di tesi mostrano come l'integrazione della sicurezza in ogni fase dello sviluppo è di fondamentale importanza, in quanto anche una semplice applicazione di prova come quella proposta in questo progetto è risultata essere non priva di problematiche di sicurezza. Inoltre il vero fine del lavoro era quello di mostrare come avere un framework su cui basarsi per lo sviluppo in cloud possa semplificare di gran lunga lo sviluppo e possa rendere estremamente facile trovare criticità e porre rimedi quando si è ancora in fase di sviluppo e prima che sia troppo tardi.

Indice

Elenco delle figure

Elenco dei listati

Capitolo 1

Introduzione

Il mercato per la trasformazione digitale nel 2021 è stato valutato a 608,72 miliardi di dollari e ci si aspetta che si espanda con un tasso di crescita annuale composto del 23.1% dal 2022 al 2030[1]. Stessa situazione per quanto riguarda il mercato dei servizi in cloud. Nel primo quadrimestre del 2022, la spesa per i servizi in cloud ammontava a 55,9 miliardi di dollari, che con una crescita del 43% risulta essere 14 miliardi di dollari in più rispetto al primo quadrimestre del 2021[2]. Quest'ultimo dato ci indica come sempre più aziende decidono di passare all'utilizzo di servizi in cloud, decidendo di rinunciare all'utilizzo di servizi "on premises"[3]. Questi dati ci indicano come il mercato del cloud computing sia un mercato in grande e costante crescita. Tuttavia, una crescità così rapida può portare con se delle conseguenze, come mancanza di esperienza e mancanza di capacità necessarie ad affrontare questa transizione. L'obiettivo di questo lavoro di tesi sarà quello di affrontare in maniera orizzontale il cloud computing e la relativa messa in sicurezza dei servizi in esso ospitati. Si partirà da un analisi sulle tematiche di sicurezza in generale, si passerà a una individuazione di quattro pilastri ritenuti fondamentali per poi definire un framework che definisca un approccio standard.

1.1 Un approccio standardizzato

L'approccio standardizzato è reso possibile dall'aver seguito la pratica DevSecOps che inserisce la sicurezza in ogni fase del ciclo di vita del software. È stata realizzata una pipeline di sviluppo che presenta una serie di stadi con una serie di controlli e strumenti di sicurezza. Questo sistema può essere considerato standard per via del fatto che il modo di procedere può essere preso come esempio per poter realizzare i propri sviluppi, riadattandolo ai tool che meglio soddisfino le esigenze delle aziende e degli sviluppatori. Di fatto nel capitolo di sviluppi futuri (Capitolo 6) è stato fornito un esempio di possibile integrazione ed espansione del caso d'uso reale realizzato a supporto della tesi.

---

[1] https://www.grandviewresearch.com/industry-analysis/digital-transformation-market

[2] https://www.canalys.com/newsroom/global-cloud-services-Q1-2022

[3] All'interno della propria sede

1.2 Struttura della tesi

La struttutra della tesi è organizzata in questo modo:

- Capitolo 2: presentazione della tematica cloud computing, in maniera generale e da un punto di vista informativo per il lettore.
- Capitolo 3: studio approfondito sulla sicurezza in cloud, con individuazione di quattro tematiche di base che verranno definiti come i quattro pilastri della sicurezza in cloud.
- Capitolo 4: presentazione degli obiettivi pratici della tesi.
- Capitolo 5: presentazione dei vari strumenti e degli ambienti utilizzati per lo sviluppo del progetto.
- Capitolo 6: applicazione degli studi effettuati ad un caso d'uso reale.
- Capitolo 7: eventuali sviluppi futuri e integrazioni.
- Capitolo 8: conclusioni.

Capitolo 2

Il Cloud Computing

In questo capitolo verrà presentato quanto concerne il cloud computing, con alcune digressioni storiche, andando ad analizzare anche quali siano stati i motivi che hanno portato alla nascita e alla diffusione del cloud. L'analisi comprenderà anche una trattazione approfondità su quali siano vantaggi, svantaggi e rischi associati all'utilizzo del cloud. L'obiettivo sarà quello di fornire al lettore una visione di insieme sulla tecnologia trattata così da poter permettere di poter comprendere le motivazioni di questo lavoro di tesi.

2.1 Dalla nascita alla diffusione del Cloud

Si vuole partire facendo una breve digressione sulla nascita e l'evoluzione del cloud computing. Le aziende, in passato, utilizzavano i propri server, all'interno delle proprie organizzazioni, all'interno dei quali erano ospitati i propri servizi. L'approccio base di allora come di adesso, risulta essere quello di avere un'applicazione per server. Nel periodo previrtualizzazine questo significava avere un grande numero di server fisici sottoutilizzati, o quanto meno con ottimizzazione delle risorse subottimale, con costi decisamente elevati. L'introduzione della virtualizzazione ha permesso la realizzazione di server fisici generici, con più server virtualizzati ospitati all'inetrno. In questo modo, oltre alla possibilità di acquisto di hardware generico, si è aperta la possibilità di ottimizzazione delle risorse all'interno della singola macchina fisica (Figura 2.1).

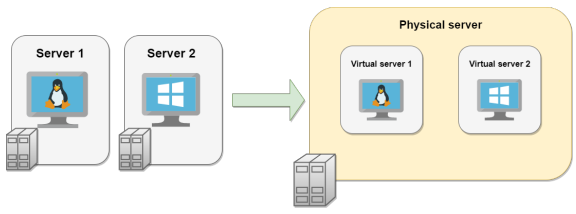

Figura 2.1: Da più server fisici a server fisico unico con più server virtuali

Data la situazione alcune aziende hanno visto la possibilità di iniziare a fornire le proprie risorse, sia hardware che software, dando di fatto inizio a quello che viene comunemente

chiamato cloud computing. Tra queste aziende la prima in assoluto fu Amazon con AWS[1]. Amazon risulta essere il principale fornitore con una quota di mercato del 33%. Le altre due aziende con la maggior presenza sul mercato sono Microsoft e Google con rispettivamente il 21% e 8%[2].

Categorie e modelli di distribuzione dei servizi in cloud

A seconda della domanda degli utenti si sono venuti a creare diversi scenari di infrastrutture e modelli di distribuzione di questi servizi. È possibilie individuare tre principali categorie di servizi in cloud:

- **IaaS** (Infrastructure as a Service): In accordo con quanto descritto in "The NIST Definition of Cloud Computing[3] nella categoria IaaS ciò che viene fornito al cliente è l'infrastruttura fisica, con processori, dispositivi di archiviazione, dispositivi di rete e altre risorse fondamentali, con la possibilità di rilasciare su questa infrastruttura qualsiasi software, compresi i sistemi operativi. Questo tipo di modello garantisce la massima flessibilità all'utente, con conseguente maggiore responsabilità e superficie di lavoro.

- **PaaS** (Platform as a Service): Nella categoria PaaS, ciò che viene fornita è una piattaforma di base su cui poter costruire nuovo software. In questo caso il grado di flessibilità è minore, ma allo stesso tempo anche il grado di responsabilità e superficie operativa sono minori. Questo comporta meno complessità di utilizzo da parte del consumatore.

- **SaaS** (Software as a Service): La categoria SaaS, è quella con il minimo grado di flessibilità. Ciò che si acquista è l'intero software, da fornire agli utenti finali. Le operazioni del cliente sono minime come sono minime le possibilità di personalizzazione del software.

Per quanto riguarda invece i modelli di distribuzione dei servizi in cloud se ne possono individuare principalmente 4:

- **Pubblico**: i servizi sono accessibili dalla rete. Su un'unica infrastruttura possono esserci più clienti che utilizzano questi servizi. Il punto più critico di questa soluzione è la necessità di separare e isolare i vari utenti.

- **Privato**: in opposizione al pubblico, c'è un contratto uno a uno tra fornitore e cliente. Non c'è condivisione con altri utenti.

- **Ibrido**: è un approccio misto tra pubblico e privato, e offre il meglio di entrambi consentendo di avere sia un ambiente "interno" e intrinsecamente più sicuro, sia un accesso a un'infrastruttura pubblica.

[1] Amazon Web Services

[2] https://www.canalys.com/newsroom/global-cloud-services-Q1-2022

[3] https://www.nist.gov/publications/nist-definition-cloud-computing

- **Multi-Cloud**: utilizzato essenzialmente per avere un maggiore grado di affidabilità. Si usano più cloud pubblici, così da avere risorse di backup nel caso fallisse una delle infrastrutture.

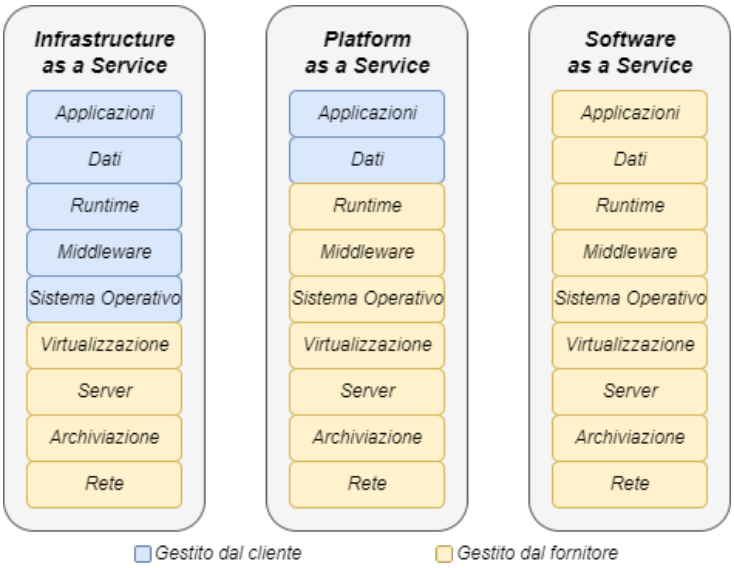

Figura 2.2: Differenza fra le varie categorie di cloud

2.2 I vantaggi del Cloud

In un'analisi condotta da Dell nel 2015, il GTAI[4], è emerso come le compagnie che stessero investendo in tecnologie quali Big Data, Cloud, Security e Mobility avessero una crescita dei ricavi superiore del 53% rispetto alle compagnie che non facessero di questi investimenti una priorità.[5] L'analisi ci permette di comprendere come tecnologie innovative, ad esempio lo stesso cloud permettono alle aziende di avere ricavi decisamente maggiori, e le motivazioni dietro questo sono strettamente correlate alla serie di vantaggi che queste tecnologie apportano.

- Risparmio economico: seppur il passaggio da un'architettura on premises all'utilizzo di servizi in cloud può comportare un iniziale grande investimento, si deve andare a considerare il ROI[6] sul lungo periodo. Inoltre, qualsiasi sia la categoria di infrastruttura, che sia IaaS, PaaS o Saas, il cloud ha il vantaggio di far spendere a seconda dell'utilizzo che si fa dei vari servizi, seguendo il modello Opex[7].

[4]Global Technology Adoption Index

[5]https://www.businesswire.com/news/home/20151013005365/en

[6]Return of Investment

[7]Compensation based operating expenses

- Flessibilità e risorse illimitate: l'utilizzo di servizi in cloud permette l'utilizzo di risorse in maniera efficiente e basato sulle necessità che si presentano. Una qualsiasi risorsa può essere aumentata o diminuita a seconda della domanda. In uno scenario on premises, aumentare la disponibilità di una risorsa potrebbe comportare la necessità di dover rivedere l'intera infrastruttura, con costi non trascurabili. E bene poi considerare l'irreversibilità dell'ampliamento di una infrastruttura. Il cloud permette di usare sempre il quantitativo necessario di una risorsa.

- Mobilità: l'accesso ai servizi in cloud è sempre garantito, da dovunque, da qualsiasi dispositivo, in qualsiasi momento. I fornitori, attraverso SLA[8], si prefiggono di garantire la continuità del servizio.

- Resilienza: secondo RapidScale, il 20% degli utenti di un servizio in cloud dichiara di aver ottenuto un recupero dal disastro in meno di 4 ore; al contrario, solo il 9% di utenti non utilizzatori del cloud può dichiarare lo stesso. La resilienza dipende da una serie di fattori, quali la ridondanza di servizi e la replicazione dei dati. I dati di fatto in cloud non sono archiviati in un unico punto, ma sono sparsi e replicati su diversi data center. Se da un lato questa cosa permette un veloce recupero dal disastro e una prevenzione alla perdita di dati, dall'altro complica decisamente la gestione del ciclo di vita dei dati.

- Aggiornamenti software istantanei: in cloud i servizi possono essere aggiornati e rilanciati automaticamente, senza la necessità di interrompere il servizio e senza la necessità di forzare il dipartimento IT a effettuare aggiornamenti manuali su scala aziendale. Ne consegue un risparmio di risorse e denaro, tenendo anche a mente il fatto che periodi di sospensione del servizio possano comportare ingenti perdite di denaro.

- Sviluppo software semplificato: il cloud computing ha aperto al paradigma di sviluppo software a microservizi grazie all'utilizzo della containerizzazione. L'utilizzo dei container oltre a rendere più agevole lo sviluppo, lo rende estremamente malleabile. Con i microservizi si ha la possibilità di comporre i vari moduli in maniera indipendente l'uno dall'altro, anche dal punto di vista di linguaggi e framework, andando ad aprire alla possibilità di modificare, eliminare, aggiungere un singolo modulo senza dover necessariamente modificare l'intero applicativo. Lo sviluppo a microservizi è senza dubbio consigliato per applicazioni molto complesse. Per applicazioni semplici il vantaggio dell'utilizzo dei microservizi non è percettibile. [Figura 2.3]

- Sicurezza: i servizi in cloud permettono di avere controlli di sicurezza molto più efficienti. Una delle materie su cui le aziende sono maggiormente preoccupate è la possibilità di furto dei dati. La realtà dei fatti è che la maggior parte dei furti avviene proprio internamente alle aziende con errori o azioni volontarie da parte dei dipendenti. Inoltre sempre secondo RapidScale il 94% delle aziende ha dichiarato di aver assistito ad un miglioramento della sicurezza dopo essere passati al cloud e il

[8]Service Level Agreement - strumenti contrattuali attraverso i quali si definiscono le metriche di servizio

91% ha dichiarato come l'utilizzo di servizi in cloud renda più semplice superare i requisiti governativi in materia di sicurezza[9]. Per quanto la sicurezza possa costituire però un vantaggio, è sicuramente uno dei punti su cui prestare maggiore attenzione. Il dettaglio della sicurezza in cloud sarà argomento centrale di questo lavoro di tesi e verrà trattato approfonditamente nei capitoli successivi.

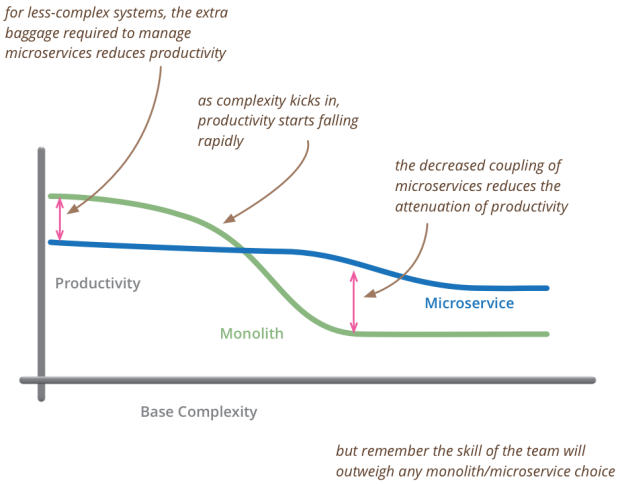

Figura 2.3: Sviluppo a microservizi vs sviluppo monolitico

2.3 I rischi del Cloud

In questa sezione verranno trattati in maniera introduttiva i possibili svantaggi del cloud computing. L'obiettivo è cercare di rendere il più chiaro possibile che per quanto il passaggio al cloud sia conveniente sotto molti punti di vista, non bisogna sottovalutare i rischi di sicurezza impliciti che porta con sè l'utilizzo di queste nuove tecnologie. Bisogna oltremodo tenere a mente le capacità degli attaccanti di escogitare sempre nuove modalità di attacco, sempre più complesse e potenzialmente in grado di causare danni sempre maggiori. Il vero problema della sicurezza in cloud è la presenza di una curva di apprendimento troppo ripida per l'approccio definito DevSecOps, con le organizzazioni che stanno avendo problemi a colmare questo gap. Questo tipo di approccio prevede l'integrazione della sicurezza a partire dalle prime fasi di sviluppo del codice. Nel report di Check Point[10] viene fuori come circa un quarto delle organizzazioni ha subito un incidente correlato alla sicurezza in cloud, e fino al 10% di questi si è verificato lo scorso anno. Questi incidenti sono per la maggior parte correalati a mancanze umane; essenzialmente configurazioni errate di risorse (23%) dati e file condivisi in maniera inappropriata (15%) compromissione dell'account (15%). In poche parole una grande fetta degli incidenti si è verificata per mancanza di coscienza tecnologica e errore umano. Tuttavia non è l'unico elemento di rischio. I rischi associati

[9] https://www.slideshare.net/rapidscale/cloud-computing-stats-security-and-recovery

[10] https://pages.checkpoint.com/2022-cloud-security-report.html

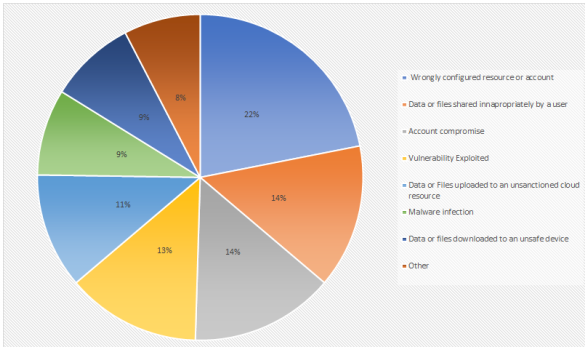

Figura 2.4: Distribuzione degli incidenti in cloud

al cloud possono essere relativi a diversi strati dell'infrastruttura, a diversi componenti, o anche a componenti tipici delle tecnologie impiegate in cloud. Le principali minacce sono:

- Data Breach: con data breach si intende l'evento che porta, accidentalmente o a seguito di un attacco, alla perdita, distruzione o divulgazione di dati privati. I motivi alla base di questo evento, sono strettamente correlati alla natura distribuita del cloud. Basti pensare al numero differente di utenti, alla distribuzione dei dati, alla gestione delle chiavi ai database condivisi. Ciò che comporta tutto ciò è un notevole ampliamento della superficie d'attacco, ma anche un notevole aumento della probabilità di incidenti.

- Data Loss: questa situazione non va confusa con il data breach. In questo caso, con data loss si intende la perdita o la compromissione di informazioni, ad esempio corruzione, sovrascrittura distruzione di informazioni in un ambiente cloud.

- Insecure Interfaces and APIs: l'accesso e la gestione dei servizi in cloud avviene attraverso le cosidette API[11], offerte dai cloud provider. La natura complessa del cloud e l'integrazione di API di terze parti da parte di aziende, rischia di aumentare la complessità delle chiamate a queste interfacce, con il conseguente aumento della probabilità di creazione di interazioni insicure.

- Denial of Service: l'obiettivo degli attacchi di denial of service è quello di rendere inutilizzabile un servizio o una risorsa allo scopo di arrecare danno ad una organizzazione. Essendo i servizi in cloud estremamente basati su concetti quali affidabilità e elasticità, renderli inutilizzabili può comportare un ingente danno per il cloud service provider. L'obiettivo quindi è riuscire a creare ambienti cloud privi di SPOF[12] e in grado di supportare un quantitativo di traffico decisamente maggiore rispetto al traffico base.

- Malicious Insiders: con malicious insider si intende quella figura (un ex o attuale dipendente) che ha accesso a risorse o servizi di un'azienda e decide deliberatamente

[11] Application Programming Interface

[12] Single Point of Failure

di agire maliziosamente, magari per ragioni personali, con il solo scopo di danneggiare l'organizzazione.

- Abuse of Cloud Services: L'accesso conveniente e semplificato a servizi online, con la relativa possibilità di avere risorse che fino a prima erano inaccessibili economicamente apre le possibilità a numerosi attaccanti di sfruttare il cloud per eseguire attacchi complessi con un grande quantitativo di potenza di calcolo, in modo da poter ridurre drasticamente i tempi di attacco.

- Insufficient Due Diligence: uno dei rischi che si corre nel fare una migrazione al cloud computing troppo affrettata è la mancanza di dovuta investigazione e analisi dei rischi a cui un'azienda deve far fronte. Un analisi superficiale dei rischi rischia di causare danni ingenti all'organizzazione.

- Shared Technology Vulnerabilities: la necessità di fornire da parte dei CSP i loro servizi in maniera elastica e scalabile, comporta l'utilizzo di infrastrutture e tecnologie condivise. Il risvolto negativo di questo approccio è che una possibile vulnerabilità rischia di essere condivisa da molti più utenti. Dove possibile infatti i CSP[13] dovrebbero adottare un approccio stratificato, con una strategia di difesa in profondità su tutti gli starti[14].

Rischi e misure di sicurezza in base al modello cloud

A seconda del tipo di infrastruttura cloud in esame, le minacce e i possibili attacchi possono essere diversi, per via della superficie d'attacco o degli asset che si possono raggiungere.

Nel caso **IaaS** va prestata particolare attenzione alla moltitudine di livelli e componenti che caratterizzano questo modello, a partire dall'architettura fino ai componenti virtuali. Un elemento caratterizzante di questo modello è sicuramente la virtualizzazione, l'utilizzo di reti virtuali e l'utilizzo di hypervisor. Le principali metodologie di attacco quindi sono mirate a danneggiare o prendere controllo di questi elementi. Attaccare una macchina virtuale può essere un primo passo per compiere attacchi VM-to-VM, andando a compromettere un numero potenzialmente elevato di macchine. Un attacco alle reti virtuali può compromettere tutto il traffico dati. Un attacco all'hypervisor può essere il più problematico per via del grande controllo che ha sugli strati superiori, sul livello fisico e sulle applicazioni ospitate. L'avvento del cloud inoltre ha portato alla creazione di nuove problematiche prima non esistenti; basti pensare alla natura multi-tenant del cloud, ovvero stessa infrastruttura e più organizzazioni che la utilizzano. In uno scenario del genere si presenta il rischio di data leakage tra le varie organizzazioni, cosi come la possibile compromissione di una macchina virtuale di un'azienda può portare a compromissione di macchine virtuali di altre aziende. Inoltre, più è grande l'intera infrastruttura più è ampia la superficie d'attacco, e più è ampia la superficie d'attacco maggiore è la probabilità di scovare delle vulnerabilità.

Nel caso **PaaS** l'attenzione si sposta su 4 macrocategorie:

[13] Cloud Service Provider

[14] Defense-in-depth strategy

- Isolamento di sistemi e risorse: i tenant in questo caso non dovrebbero avere accesso alle shell dei propri server, anche se virtualizzati. La ragione dietro questa misura è cercare di limitare la possibilità che azioni errate possano danneggiare altri tenant. Ove possibile, operazioni con permessi di amministratore dovrebbero essere eseguite in container isolati.

- Permessi a livello utente: ogni istanza di un servizio dovrebbe avere la propria nozione di permessi a livello utente. Avere inoltre appropriate contromisure e appropriati controlli può ridurre il rischio di acquisizione di permessi non necessari durante il tempo.

- Gestione dell'accesso degli utenti: una ben strutturata gestione dell'accesso degli utenti serve a proteggere meglio la disponibilità, l'integrità e la riservatezza di risorse e asset, assicurandosi che solo coloro i quali abbiano autorizzazione possano accedervi. Essendo il cloud per natura più complesso è stato necessario rivedere i tradizionali metodi di autorizzazione e autenticazione per fornire una granularità più fine, basata su policy aziendali e regole. Diventa perciò una necessità imparare a gestire con precisione questi meccanismi, facendo anche particolare attenzione all'interazione con componenti e ambienti un po' più datati.

- Protezione contro malware, backdoor e trojans: l'installazione di applicativi in cloud può comportare l'installazione di malware e/o backdoors. Pertanto è buona norma che qualsiasi applicativo o software venga installato venga passato sotto attente revisioni durante tutto il suo ciclo di vita, in modo da ridurre significamente il rischio di avere vettori d'attacco potenzialmente disastrosi.

Nel caso **Saas** il focus può essere centrato su tre aree principali.

- Segregazione dei dati: come ripetuto più volte una delle caratteristiche principali del cloud è la multitenancy. Ne consegue che diversi utenti possano archiviare i dati sulla stessa applicazione offerta dal modello SaaS. Dati di diversi utenti si troveranno archiviati in stesse località, oppure dati di un unico utente si troveranno dispersi su più località. Il servizio offerto quindi deve operare un'intelligente segregazione dei dati dei diversi utenti, sia a livello fisico che a livello applicativo.

- Accesso ai dati e relative policy: l'accesso ai dati deve essere accompagnato da un'attenta pianificazione delle regole e della granularità di accesso. Ciò che si vuole evitare è l'accesso non autorizzato a dati sensibili. Al contempo si vuole garantire l'accesso esclusivamente ai dati richiesti dall'utente.

- Sicurezza delle applicazioni web: visto il volume di tenant condivisi e colocati in un ambiente SaaS, qualora una vulnerabilità venisse sfruttata, sia utenti che CSP potrebbero ritrovarsi a dover affrontare conseguenze disastrose. È bene perciò eseguire attente revisioni del codice prima della pubblicazione di un'applicazione, ed è bene eseguire controlli a runtime per avere applicazioni sempre aggiornate.

La responsabilità sulla sicurezza è dipendente dal tipo di modello cloud adottato. A seconda se il modello sia SaaS, PaaS, IaaS ci sarà una responsabilità dell'azienda, una responsabilità condivisa, una responsabilità del Cloud Service Provider (2.5)

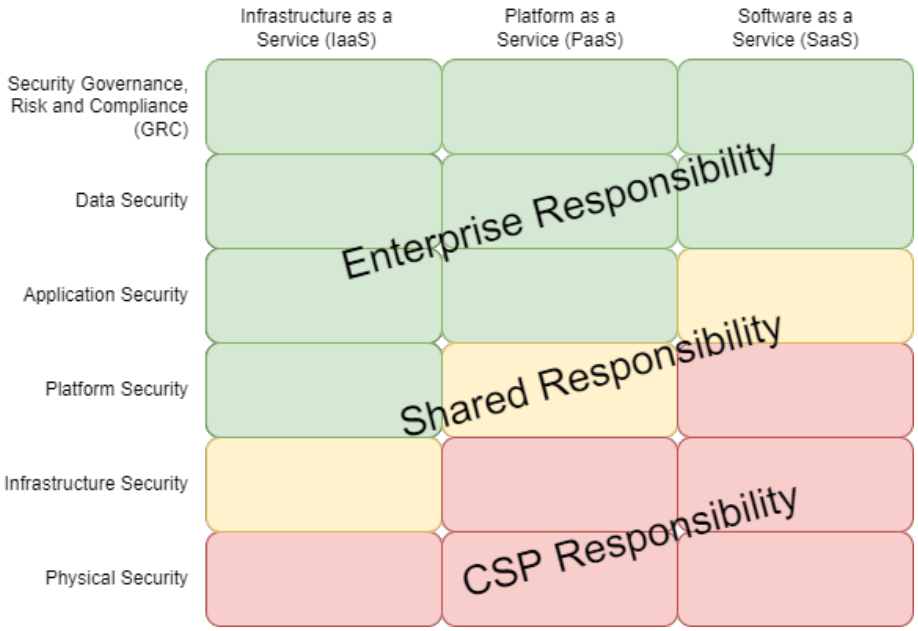

Figura 2.5: Distribuzione della responsabilità tra CSP e azienda

Nel caso **IaaS**, l'utente ha ampio controllo, e grandi responsabilità. Quasi la totalità della gestione della sicurezza è a carico del utente. Nel caso **PaaS** l'utente ha come zona di responsabilità la sicurezza dell'applicazione, la sicurezza dei dati e la gestione di GRC[15]. Nel caso **Saas** la superficie relativa alla gestione della sicurezza per un utente è meno ampia, tuttavia sicurezza dei dati e GRC sono ancora sotto la responsabilità dell'utilizzatore.

[15]Governance, Risk and Compliance

Capitolo 3

I quattro pilastri della sicurezza in cloud

.

Il capitolo che segue è il capitolo relativo agli studi effettuati sulla sicurezza in cloud. Gran parte dello studio è basata sul libro CCSP che organizza il lavoro in compartimenti. Da qui la volontà di individuare 4 pilastri fondamentali della sicurezza in cloud. Più specificamente sono stati individuati 4 settori, a noi maggiormente congeniali per lo sviluppo del lavoro di tesi. Questi 4 settori sono:

- Sicurezza dell'applicazione
- Sicurezza dei container
- Sicurezza dei dati
- Gestione dell'ambiente cloud

Il libro individua anche un ulteriore settore, riguardante la sicurezza dell'infrastruttura fisica. Il motivo per cui si è scelto di non inserirlo nella trattazione è che quest'ultimo esula leggermente dal nostro scopo. Parlare di sicurezza dell'infrastruttura significa affrontare argomenti di sicurezza riscontrabili nel caso in cui si decida di adottare un modello cloud di tipo IaaS. In questo progetto di tesi tuttavia, si è scelto di utilizzare come cloud provider Amazon Web Services, che risulta essere più un PaaS, che pertanto si assume la responsabilità della sicurezza dell'infrastruttura fisica.

3.1 Sicurezza dell'applicazione

In questa sezione verrà presentata in maniera approfondita e dettagliata la tematica della sicurezza di un'applicazione in cloud. La tematica verrà trattata esplorando quello che è il ciclo di vita di sviluppo del software, andando in fine a cercare di spiegare quali sono i controlli necessari per lo sviluppo sicuro di applicazioni in cloud. Infine verrà trattata

la cosidetta "application AIC[1]" attraverso un processo di validazione e assicurazione del software in cloud.

3.1.1 Principali trappole nel rilascio e sviluppo di applicazioni in cloud

Lo sviluppo di applicazioni in cloud può risultare particolarmente impegnativo e la mancata abilità nel realizzare un progetto simile può comportare all'azienda costi aggiuntivi, progetti falliti, perdita di credibilità. È bene individuare quali possano essere i principali pericoli e quali possano essere i punti importanti da considerare prima di approcciarsi allo sviluppo in cloud.

- Non sempre è possibile migrare applicazioni on-premises in cloud

  Le funzionalità e le prestazioni di un'applicazione non pensata per il cloud possono non essere trasferibili. Potrebbe essere difficile replicare una configurazione on-premises sui servizi cloud. Le motivazioni dietro ciò sono essenzialmente due:

  1. Queste applicazioni non sono state pensate avendo in mente i servizi in cloud. Sono basate su tecnologie magari non inerenti al cloud.
  2. Non tutte le applicazioni possono essere prese per intero e migrate in cloud con minori cambi del codice. Nei sistemi le applicazioni possono essere con poche dipendenze e autosufficienti. Trasportare per intero queste applicazioni può introdurre a cambiamenti significativi con necessità di aggiungere ulteriori dipendenze e requisiti.

- Mancanza di formazione e consapevolezza

  Nuove tecnologie richiedono nuove tecniche di sviluppo e nuove tipologie di approccio. Questo richiede formazione e volontà e necessità di imparare ad utilizzare nuovi tool e tecniche. Questo può introdurre a nuove sfide, con l'introduzione del rischio di mancanza di formazione e mancanza di esperienza.

- Mancanza di documentazione e linee guida

  Nella sfera delle migliori norme da seguire rientrano pratiche come seguire la documentazione, le linee guida, le metodologie. Questo serve a ridurre al minimo l'introduzione di errori e rischi aggiuntivi. La rapida diffusione del cloud ha portato in alcuni casi ad avere scarsa documentazione e mancanza di linee guida da seguire. È bene pertanto avere a mente l'intero ciclo di sviluppo software e le relative fasi, in modo da poter avere una struttura di base su cui appoggiarsi per lo sviluppo il quanto più possibile efficiente e di alta qualità.

- Complessità nell'integrazione

  L'integrazione di nuove applicazioni con applicazioni preesistenti può essere una parte chiave dello sviluppo software. In situazioni in cui il CSP gestisce infrastruttura, applicazioni e piattaforme di integrazione questo può essere particolarmente complesso.

[1] Availability, Integrity, Confidentiality

Inoltre può essere difficile tracciare e raccogliere eventi e transazioni su componenti interdipendenti allo scopo di risolvere problemi venutisi a creare.

- Sfide troppo grandi

  Ci sono due elementi chiave da considerare negli applicativi in cloud

  – Multitenancy
  – Amministrazione da parte di terzi

  È bene inoltre che gli sviluppatori abbiano piena coscienza i requisiti di sicurezza sulla base di:

  – Modello di distribuzione dei servizi in cloud (pubblico, privato ibrido)
  – Categoria di servizio in cloud (IaaS, PaaS, SaaS)

  e comprendere in base a queste cose quale sia il loro grado di responsabilità in materia di sicurezza (Figura 2.5).

3.1.2 Comprendere il ciclo di vita dello sviluppo software in un ambiente cloud

Il cloud computing aumenta ulteriormente la necessità per un software di attraversare tutte le fasi di quello che viene indentificato come il ciclo di vita dello sviluppo software. Aderire a queste fasi è un ottimo modo per fare in modo che la scrittura di un software sia quanto più possibile di alta qualità.

1. Pianificazione e analisi dei requisiti

   È la fase preliminare. Qui devono essere determinati i requisiti di business e di sicurezza della futura applicazione. È necessario incontrarsi con project manager e stakeholder per poter definire con precisione questi requisiti, prima che la fase di design abbia inizio. In questa fase bisogna pianificare i requisiti per la quality-assurance e identificare i rischi associati al progetto. Bisogna poi analizzare i requisiti per verificarne la validità e la possibilità di inserimento nel sistema che deve essere sviluppato.

2. Definizione

   Questa è la fase di definizione e stesura della documentazione dei requisiti del prodotto. Bisogna a questo punto ottenere l'approvazione dei clienti. Il tutto avviene tramite un documento scritto che contiene tutti i requisiti del prodotto in esame.

3. Design

   La fase di design aiuta a specificare i requisiti hardware e software del sistema e più in generale serve ad avere un'immagine generale dell'architettura che dovrà essere realizzata. Tutto ciò servirà come input per la fase successiva. In questa fase è bene eseguire una modellazione delle possibili minacce ed è bene iniziare ad individuare quali siano gli elementi fondamentali da avere per ottenere uno sviluppo sicuro.

4. Sviluppo

   Una volta realizzato il documento di design del progetto, può iniziare la fase vera e propria di scrittura del codice. Questa può essere senz'altro la fase più lunga del progetto. Il lavoro viene suddiviso in moduli o unità. Le attività in questa fase possono includere **code review**, **unit test** o **analisi statica**.

5. Test

   Una volta che il codice è stato scritto, si passa alle fasi finali del ciclo di vita di sviluppo. Una tra queste è la fase di test. Nella fase di test ciò che viene fatto è una serie di test per verificare che il prodotto soddisfi effettivamente tutti i requisiti e necessità richieste nelle fasi iniziali. I test eseguiti in questa fase sono **unit test, integration test, system test, acceptance test**.

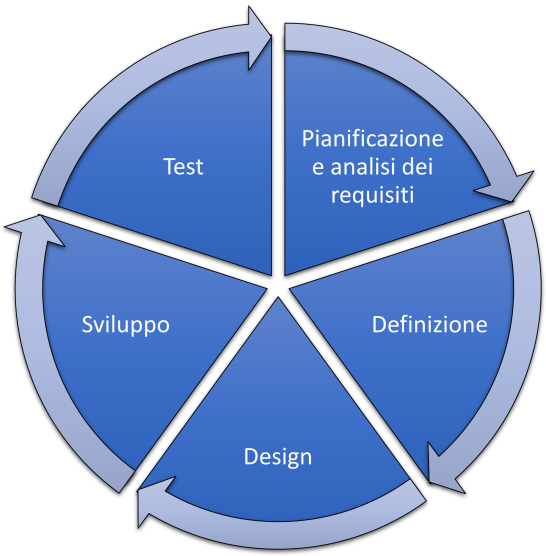

Figura 3.1: Ciclo di vita del software

Fase di operazioni di sicurezza

In questa sezione introdurrò in maniera breve la fase di operazioni di sicurezza perchè facente parte delle fasi di sviluppo software. Una trattazione più approfondita verra eseguita in seguito nella sezione relativa alla gestione dell'ambiente cloud.

In una prospettiva incentrata sulla sicurezza, una volta che l'applicazione è stata implementata seguendo i principi del ciclo di vita di sviluppo software, entra in una fase di operazioni di sicurezza. Una gestione corretta delle configurazioni e delle versioni del software è fondamentale alla sicurezza dell'applicazione. Possono essere usati alcuni tool per assicurarsi che il software venga configurato secondo determinati requisiti. Tra questi software si possono citare **Puppet** e **Chef**.

- Puppet: è un sistema per la gestione del sistema. Permette la definizione dello stato desiderato dell'infrastruttura e permette il raggiungimento automatica di questo stato.
- Chef: serve ad automatizzare la costruzione, il rilascio e la gestione dell'infrastruttura.

Indipendentemente dal tool utilizzato, l'obiettivo in questa fase è assicurarsi che le configurazioni sono aggiornate ogni volta che ce ne sia necessità e che ci sia consistenza e coerenza nelle versioni di questi update. Questa fase inoltre è la fase in cui altre attività fondamentali per la sicurezza vengono eseguite; tra queste possiamo citare **analisi dinamica**, **penetration testing**, **monitoraggio delle attività**.

Fase di smaltimento

Alla fine del ciclo di vita di un'applicazione è necessario smaltirla ed eliminarla dal cloud. Non avendo accesso fisico ai drive non si può eliminare fisicamente l'applicazione. È qui che entra in gioco la cosidetta "crypto-shredding" ovvero la cifratura dei dischi con conseguente eliminazione della chiave. In questo modo ci si assicura che i dati vengano persi del tutto anche se non eliminati fisicamente.

3.1.3 Eseguire un processo di analisi in cerca delle vulnerabilità comuni

Le applicazioni che girano in cloud devono essere conformi alle migliori pratiche e linee guida per l'accertamento di assenza di vulnerabilità e per la risoluzione di problemi in caso di presenza di queste. La OWASP[2] Foundation, stila con regolarità una lista delle 10 vulnerabilità più comuni con una descrizione per ognuna di queste[3]. Ci si può appoggiare a questa lista per eseguire almeno i primi step per un adeguato vulnerability assessment. L'ultima top10 era stata stilata nel 2017. Questa del 2021 presenta alcune differenze con quella del 2017, tra cui tre nuove vulnerabilità inserite, alcune scese di posizione e altre salite. La top ten del 2021 presenta queste vulnerabiltà:

1. **A01:2021-Broken Access Control**
2. **A02:2021-Cryptographic Failures**
3. **A03:2021-Injection**
4. **A04:2021-Insecure Design**
5. **A05:2021-Security Misconfiguration**
6. **A06:2021-Vulnerable and Outdated Components**
7. **A07:2021-Identification and Authentication Failures**

[2]Open Web Application Security Project

[3]https://owasp.org/www-project-top-ten/

8. **A08:2021-Software and Data Integrity Failures**
9. **A09:2021-Security Logging and Monitoring Failures**
10. **A10:2021-Server-Side Request Forgery**

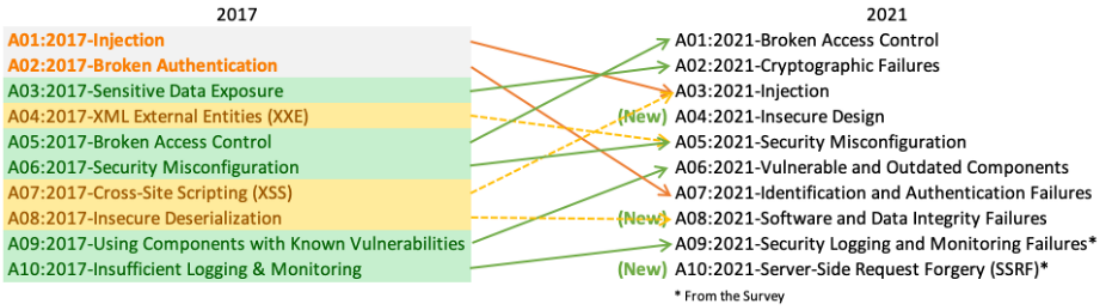

Figura 3.2: Differenza tra Top10 2017 e Top10 2021

3.1.4 Stimare le possibili minacce

Una volta completata la fase di design dell'applicazione, è buona pratica fare una stima delle possibili minacce. Questa pratica nota come threat modeling ha come obiettivo quello di trovare possibili debolezze dell'applicazione e quali possano essere gli attori in gioco, così come gli ingressi e le uscite. L'obiettivo è evitare che queste debolezze vengano esportate in ambienti di produzione. Nel cloud inoltre, la superficie d'attacco essendo molto più ampia, aumenta significativamente la probabilità di avere possibile minacce. Siccome l'insieme delle possibili minacce ha natura variegata ed è difficile determinazione si può pensare di appoggiarsi a dei modelli di supporto per fare una stima. Tra questi figura come modello il cosidetto **STRIDE model**.

- **Spoofing** l'attaccante assume il ruolo di qualcun altro. Un esempio che può essere fornito è il caso di un attaccante che entra in possesso delle credenziali di qualcun altro e compie azioni malevole "a suo nome".
- **Tampering** l'attaccante modifica in maniera maliziosa dati. Sono azioni che vanno a intaccare il principio di integrità.
- **Repudiation** attacchi che violano il principio di non ripudio. Per essere più precisi, sono attacchi che permettono all'attaccante di ripudiare una determinata azione, senza che sia possibile risalire a lui come esecutore.
- **Information Disclosure** Informazioni ottenute senza averne l'autorizzazione. Sono attacchi che violano il principio di confidenzialità delle informazioni.
- **Denial of Service** attacchi volti a sovraccaricare le risorse per impossibilitare il legittimo utilizzo. In cloud è bene fare attenzione a questo tipo di attacchi perchè vanno a violare il principio di disponibilità di servizi e dati.

- **Elevation of Privilege** attacchi che mirano ad acquisire maggiori privilegi rispetto a quelli che si hanno.

3.1.5 Test di sicurezza per l'applicazione

I test di sicurezza per un'applicazione possono comprendere una serie di tecniche e tool automatizzati volti a cercare e trovare possibili vulnerabilità. Prima di addentrarci nelle varie tecniche per eseguire i test di sicurezza, è bene anche notare come esistano due metodologie di analisi per un'applicazione.

- **White Box** quando si ha piena conoscenza dell'applicazione che si va a testare, compreso ad esempio il codice sorgente.
- **Black Box** quando si ha conoscenza nulla o parziale dell'applicazione che si va a testare, in poche parole risulta essere il punto di vista di un possibile attaccante.

Static Application Security Testing

Con l'analisi statica, si intende essenzialmente un tipo di analisi white-box. Ciò che si va a compiere sono dei test sul codice sorgente dell'applicazione. Ciò a cui può portare questa metodologia è l'individuazione di vulnerabilità o errori come **XSS**[4], **SQL Injection**, **buffer overflow**, **condizioni di errore non gestite**, **potenziali backdoor**. Essendo un'analisi del codice sorgente può essere più esaustiva di un'analisi dinamica, ad esempio può trovare più vulnerabilità, può fornire informazioni sulla causa della vulnerabilità (essendo analisi sul codice sorgente può fornire anche la riga esatta in cui questa è presente), può essere usata in tutti gli stage di sviluppo. Tuttavia ha una serie di contro che vanno tenuti in considerazione e che devono far riflettere sull'utilizzo simultaneo di tecniche di analisi statica e dinamica. Tra questi contro possiamo individuare la presenza di un maggior numero di falsi positivi[5], la copertura delle librerie e dipendenze può essere un problema perchè magari non si ha a disposizione il codice sorgente di queste, gli strumenti utilizzati sono caratterizzati da un'alta specificità rispetto al linguaggio o framework utilizzato, può consumare molto tempo se si vuole un'analisi quanto più esaustiva possibile.

Dynamic Application Security Testing

L'analisi dinamica invece viene considerata di tipo black-box. Viene eseguita su un'instanza in corso di esecuzione dell'applicazione. Ne consegue che uno dei compiti di questo tipo di analisi è ricostruire i vari flussi di esecuzione dell'applicazione. Viene considerata più efficacie se si analizzano casi in cui vengono esposte interfacce HTTP e HTML di un'applicazione web. Come nell'analisi statica presenta i suoi vantaggi e i suoi svantaggi. Essendo meno esaustiva il numero di vulnerabilità che si riesce a trovare è sicuramente minore rispetto all'analisi statica, non può essere usata negli stage iniziali dato che richiede che l'applicazione sia in esecuzione e non può fornire informazioni sulla causa della vulnerabilità, riservandosi

[4]Cross Site Scripting

[5]Elementi individuati come vulnerabilità ma che in realtà non lo sono

di mostrarne la presenza. È bene prestare attenzione su un argomento importante, ovvero il non aver trovato una vulnerabilità non ne garantisce l'assenza. Per quanto riguarda i vantaggi si può sottolineare come i falsi positivi individuati siano in numero molto inferiore rispetto all'analisi statica, se si è trovata una vulnerabilità è molto probabile che questa sia sfruttabile, e inoltre non dipende dal linguaggio o framework utilizzato. In figura 3.3 un confronto fra le due metodologie di test per un'applicazione analizzati finora.

SAST	DAST
Individua più vulnerabilità	Individua meno vulnerabilità
Più falsi positivi	Meno falsi positivi
Può essere usata in tutti gli stage di sviluppo	Può essere usata solo negli stage finali di sviluppo
Può fornire informazioni sulla causa della vulnerabilità	Non può fornire info sulla causa della vulnerabilità
La copertura delle librerie può essere un problema	
Ogni tool si applica a un set specifico di linguaggi	È indipendente dal linguaggio o framework utilizzato
Può consumare molto tempo	

Figura 3.3: SAST vs DAST

Interactive Application Security Testing

Un tipo di analisi che sta prendendo piede ultimamente è la cosidetta analisi interattiva. Questo tipo di analisi cerca di combinare analisi dinamica e statica per poter trarre da queste il meglio. Il motivo della nascita di questo tipo di analisi risiede nel fatto che le applicazioni negli ultimi anni stiano diventando sempre più complesse. Il funzionamento è il seguente: presenta un agent all'interno di un'applicazione in esecuzione. Questo agent ha accesso al codice e ai dettagli di esecuzione, ma ciò che viene analizzato staticamente è solo la porzione di codice che viene eseguito in quel momento. I vantaggi di questo tipo di analisi sono notevoli. Può trovare un alto numero di vulnerabilità, paragonabile all'analisi statica, ma individuando per contro meno falsi positivi. Può essere molto veloce e scalabile perchè eseguita su un'istanza in esecuzione dell'applicazione. Quest'ultima considerazione la rende un'ottima soluzione per l'approccio DevSecOps. Tuttavia essendo una metodologia recente, risulta essere ancora un po' acerba, non disponibile per tutti i linguaggi o framework e ancora poco conosciuta (e utilizzata).

Penetration Testing

Il penetration testing è il processo usato per raccogliere informazioni sulle vulnerabilità e i possibili modi di sfruttarle del sistema. È tipicamente un'analisi di tipo black-box in cui il

tester si posiziona nei panni di un possibile attaccante, non ha conoscenza dell'applicazione e deve scoprire da se quali possano essere i problemi di sicurezza del sistema in esame. L'obiettivo inoltre è quello di cercare di capire cosa si può ottenere dallo sfruttamento di queste vulnerabilità. In un ambiente cloud la possibilità di eseguire penetration testing dipende essenzialmente dal tipo di modello utilizzato. Nel caso SaaS difficilmente i fornitori permettono al cliente di eseguire penetration testing, di solito questa è un'attività riservata ai fornitori stessi invece che ai clienti. Nel caso PaaS e Iaas, tipicamente, la responsabilità delle applicazioni è sulle spalle dell'utente ed è pertanto permesso che questo esegua questo tipo di analisi. I problemi del penetration testing risiedono nel fatto che richiede personale altamente qualificato, può essere costoso (per via dell'ingaggio di questo tipo di personale) e non può essere eseguito spesso, ma viene eseguito solo qualora fosse necessaria un'analisi più approfondita. In figura 3.4 un confronto tra vulnerability assessment e penetration testing.

VA	PT
Report di tutte le vulnerabilità individuate	Report delle vulnerabilità sfruttabili e cosa se ne può ottenere
Essenzialmente basato su strumenti automatici	Essenzialmente basato su attività manuali
Facile da eseguire	Richiede un elevato grado di esperienza
Eseguito da personale interno all'azienda	Eseguito da personale esterno all'azienda
Economico	Costoso
Eseguito ogni volta che un cambiamento lo richiede	Eseguito solo nei casi in cui sia necessaria un'analisi più accurata

Figura 3.4: Vulnerability Assesment vs Penetration Testing

Raccomandazioni da parte di OWASP

OWASP stila costantemente una lista di raccomandazioni da seguire per poter eseguire un testing accurato dell'applicazione[6].

- Configuration and Deployment Management Testing
- Identity Management Testing
- Authentication Testing
- Authorization Testing

[6] https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing

- Session Management Testing
- Input Validation Testing
- Testing for Error Handling
- Testing for Weak Cryptography
- Business Logic Testing
- Client-side Testing

3.2 Sicurezza dei container

In questa sezione verrà affronato il secondo pilastro individuato sulla sicurezza in cloud. Tale pilastro riguarda i cosiddetti container, una tecnologia fondamentale per lo sviluppo di applicativi in cloud. Di seguito viene riportata un'analisi dettagliata di cosa sono i conteiner, e quali sono le relative considerazioni e problematiche circa la loro sicurezza.

3.2.1 I container

Come già affrontato nella sezione 2.2 con il cloud si è passati da uno sviluppo monolitico ad uno sviluppo a microservizi. La tecnologia alla base dello sviluppo a microservizi è la containerizzazione. I container si basano sul concetto di "lightweight virtualization". La differenza però rispetto alla virtualizzazione classica è che in questo caso si parla di virtualizzazione a livello di sistema operativo (container linux) o a livello applicativo (Docker), a differenza della virtualizzazione classica che riguarda anche l'hardware. In questo modo più container possono sfruttare uno stesso hardware di base andando ad alleggerire di gran lunga il carico sul singolo elemento. Questa soluzione ha una serie di vantaggi intrinsechi, oltre al sopra citato alleggerimento del carico su singolo elemento.

- **Separazione delle responsabilità**: La containerizzazione consente una chiara separazione delle responsabilità, in quanto gli sviluppatori si concentrano sulla logica e sulle dipendenze dell'applicazione, mentre i team delle operazioni IT possono concentrarsi sul deployment e sulla gestione anziché sui dettagli relativi alle applicazioni, come le versioni e le configurazioni specifiche del software.
- **Portabilità del carico di lavoro**: I container sono in grado di funzionare praticamente ovunque, facilitando notevolmente lo sviluppo e il deployment: su sistemi operativi Linux, Windows e Mac, su macchine virtuali o server fisici, sulla macchina di uno sviluppatore o in data center on-premise e, ovviamente, nel cloud pubblico.
- **Isolamento delle applicazioni**: I container virtualizzano le risorse di CPU, memoria, archiviazione e rete a livello di sistema operativo, offrendo agli sviluppatori una visualizzazione del sistema operativo logicamente isolata da altre applicazioni.

Analizziamo ora le due tecnologie per la realizzazione di container.

3.2.2 Linux container

I container linux sono un tipo di containerizzazione basata su virtualizzazione a livello di sistema operativo. In questo modo è possibile far girare più isolati sistemi linux (di fatto container) su un singolo host di controllo. Il kernel linux in questo caso è condiviso tra tutti i container.

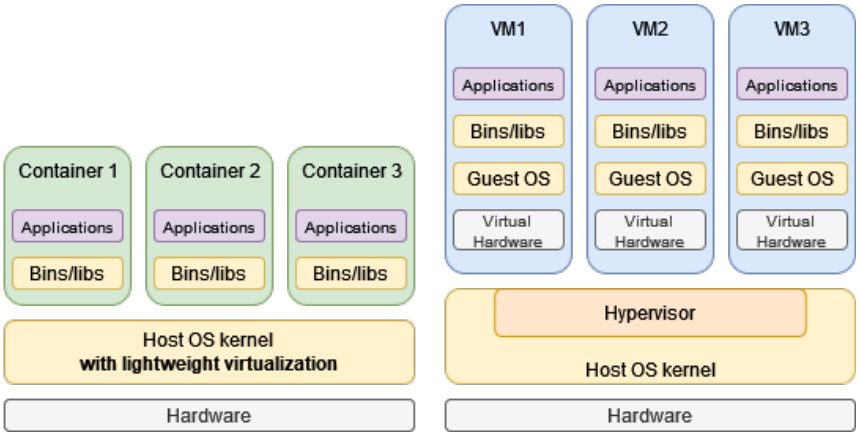

Figura 3.5: Differenza fra virtualizzazione classica e lightweight virtualization

Come si ottiene questo tipo di virtualizzazione? Linux offre una serie di meccanismi quali cgroups e namespaces che permettono di isolare a livello di sistema operativo rispettivamente risorse e processi. I container a questo punto sembrano apportare un gran numero di vantaggi rispetto alla virtualizzazione classica. È bene ricordare però che nonostante effettivamente siano vantaggiosi portano con se una serie di limitazioni alle quali prestare attenzione.

- L'isolamento dei container non è così profondo come lo è nella virtualizzazione classica. La virtualizzazione in questo senso risulta essere una tecnologia intrensicamente più sicura. La sicurezza dei container per tanto è un argomento che va trattato con serietà e precisione. I container non sono sicuri come si può pensare.
- I container linux sono limitati a girare su kernel linux. Non possono girare su kernel di sistemi operativi differenti. Questo è il motivo per cui nel tempo si è passati all'utilizzo di container Docker che permette virtualizzazione a livello applicazione. Si affronterà questo argomento, con maggior precisione,nella sezione successiva.
- L'isolamento riguardante l'utilizzo delle risorse non è sempre garantito. Bisogna configurare quanto ogni singolo container può consumare, e nonostante questo il consumo delle risorse di un container può influenzare il consumo di un altro.

Sicurezza dei container linux

Come anticipato in precedenza, la sicurezza dei container linux non va data per scontata, anzi c'è il rischio di considerare la virtualizzazione intrinsecamente sicura, con la conseguenza di considerare sicura una tecnologia, basata su virtualizzazione ma meno sicura di

quest'ultima. In uno degli studi[50] consultati, la tematica della sicurezza dei container è stata suddivisa in quattro macrocategorie:

- **Protezione di un container da una applicazione al suo interno:** in questa casistica gli autori considerano che l'applicazione all'interno di un container può essere onesta, semi-onesta o malevola. L'applicazione viene considerata non abile di rompere policy di controllo degli accessi se impostate. L'applicazione può richiedere privilegi di root. In questo scenario un'applicazione può essere abile di prendere il controllo sul container manager e attaccare altri container nel sistema.

- **Protezione inter-container:** in questo caso si assume che uno o più container siano semi-onesti o malevoli. Questi container possono essere su uno stesso host o su host differenti. Assume molto significato questo caso d'uso in relazione al campo delle service mesh. La situazione in questo caso è quella di un container semi onesto o malevolo che può entrare in possesso di informazioni confidenziali di un altro container. Oppure un container malevolo che può consumare molte più risorse di un altro sullo stesso host.

- **Protezione dell'host e le applicazioni dai container:** in questo caso il target di un possibile attacco sarebbe l'host. Si suppone quindi che il container sia malevolo o semi-onesto. Il modo in cui si può attaccare l'host è prendendo informazioni sensibili sulla componentistica oppure andando ad utilizzare più risorse del dovuto rendendone impossibile l'utilizzo. La soluzione in questi casi è cercare di non dare modo al container di poter agire sull'host, isolandolo in maniera più rigida.

- **Protezione dei container dall'host:** in questo caso è l'host ad essere semi-onesto o malevolo. Un host malevolo può entrare in possesso di una serie di informazioni sensibili riguardanti il container avendo modo di controllarne il traffico di rete, il disco, i processori e la memoria.

3.2.3 Docker

Docker è il sistema di containerizzazione attualmente utilizzato. Il motivo dipende dal suo focus sulle applicazioni. Per essere più precisi, Docker, permette di containerizzare codice e relative dipendenze. Di fatto quello che si va a pacchettizzare è l'applicazione. All'interno del container Docker quello che sarà presente sarà tutto e solo il necessario per poter far girare un'applicazione. Il vantaggio di questa soluzione è che può girare, grazie al Docker Engine, su tutte le infrastrutture. Docker quindi può interfacciarsi con i vari sistemi operativi (Linux e Windows). Docker come tecnologia porta con se una serie di vantaggi. Il vantaggio più importante è sicuramente la portatilità di questo tipo di container. Essendo già di per sè completo di tutto il necessario per essere eseguito, che sia nella macchina locale, o in un datacenter fa poca differenza. I container di questo tipo sono in grado di essere eseguiti ovunque facendo affidamento sullo strato di docker engine sottostante. Docker si basa su due concetti base:

- **Docker Image** è un template immutabile per il container. È l'elemento base da cui viene generato un container.

- **Docker Container** è l'istanza in esecuzione di una immagine docker. Può essere eseguita, stoppata, riavviata. Mantiene i cambiamenti eseguti al suo interno grazie all'interazione con il filesystem.

La realizzazione di un'immagine docker avviene attraverso la scrittura di un cosiddetto Dockerfile. Non è altro che una sorta di "ricetta" con i comandi in forma dichiarativa per la costruzione dell'immagine.

```
1 # syntax=docker/dockerfile:1
2 FROM ubuntu:18.04
3 COPY . /app
4 RUN make /app
5 CMD python /app/app.py
```

Listato 3.1: esempio di Dockerfile

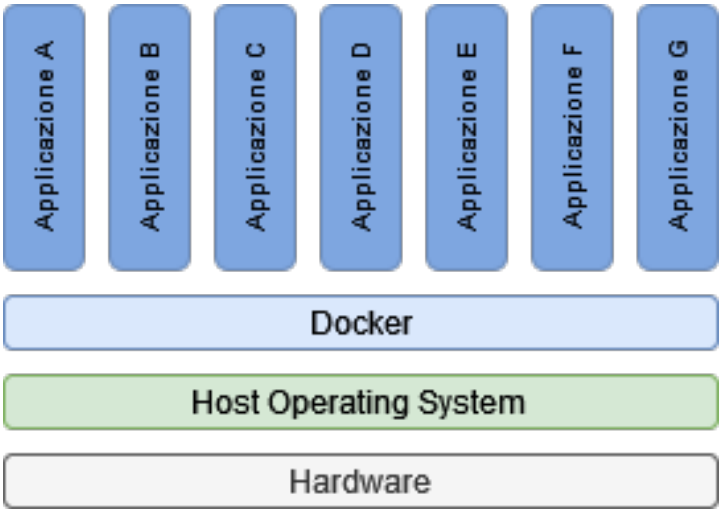

Figura 3.6: Rappresentazione dell'architettura di Docker

Sicurezza di Docker

Quali risultano essere i problemi di sicurezza relativi a Docker? A differenza della virtualizzazione classica, dove la virtual machine può comunicare solo con il suo kernel e non con il kernel dell'host, in un sistema containerizzato c'è possibilità di comunicazione con il kernel host. Questo solleva una serie di preoccupazioni riguardanti la sicurezza di Docker, essendo una tecnologia di containerizzazione. Le considerazioni base di sicurezza di un sistema containerizzato sono state già affrontate nella sezione precedente (quali problemi relativi all'utilizzo di risorse, protezione tra container ecc.). Quello che si vuole affrontare in questa sezione è più nel dettaglio i possibili problemi specifici di Docker, essenzialmente derivanti da delle mal configurazioni, e da una scrittura del Dockerfile che non rispetta le buone norme di condotta.

Essenzialmente le possibili misconfiguration individuate sono le seguenti:

- **Utilizzo di account root nei container:** di default, se non specificato nel Dockerfile, viene creato un account con privilegi di root nel container. Anche se questo utente non ha tutti i privilegi come l'utente root di linux è comunque una buona idea non permetterne la creazione specificando l'utente con i privilegi minimi.
- **Utilizzo di immagini vecchie, vulnerabili e con la presenza di backdoor:** Le immagini docker hanno la possibilità di essere scaricate da repository pubbliche come quella standard di docker, DockerHub. Chiunque può caricare un'immagine creata da sè. Il problema di questo è che si rischia di fare il download di immagini non sicure e non ufficiali. La buona norma è quella di scaricare solo immagini da editori verificati e che presentano il tag riguardante l'assenza di vulnerabilità senza patch.
- **Esporre il Daemon di Docker su HTTP:** Esporre le API remote di docker è una pratica utile agli amministratori per interagire con il daemon di docker. Tuttavia, va ricordato che docker ha necessità di privilegi root per essere eseguito, e pertanto l'esposizione di queste API deve essere fatta con criterio e sicurezza. Di fatto la buona norma è non esporre queste API sulla rete ma usare appropriata autenticazione con SSL/TLS.
- **Segreti non propriamente protetti:** La gestione dei segreti è una questione particolarmente delicata. È buona norma posizionare questi segreti in modo che questi non vengano accoppiati all'immagine docker creata. Il modo migliore per gestire i segreti con docker è l'utilizzo dei cosiddetti Secret Files. In questo modo la creazione dell'immagine avviene senza che i segreti restino persistenti al suo interno.

3.3 Sicurezza dei dati

In questa sezione verrà affrontato l'argomento sicurezza dei dati. Verrà affrontato dal punto di vista di data discovery e data classification; gestione dei diritti digitali; privacy dei dati; retention[7] dei dati, cancellazione, logging di eventi, catena di custodia, e principio di non ripudio.

3.3.1 Il ciclo di vita dei dati in ambiente cloud

Il ciclo di vita dei dati è un elemento importante da tenere in considerazione per poter operare una gestione sicura dei dati, dalla creazione alla finale distruzione. Sebbene nell'immaginario comune il ciclo di vita è visto come un processo lineare, nel particolare caso dei dati, possono esserci balzi in avanti o dietro tra le varie fasi.

- **Creazione**: Consiste nella creazione o acquisizione di nuovo materiale digitale oppure nella modifica di materiale già esistente. Questa fase può avvenire in cloud oppure esternamente. La fase di creazione è anche la fase più adatta alla classificazione del dato, prestando attenzione al tipo di dato (se sensibile e quanto) e al valore che questo può avere per l'azienda. Una accurata classificazione può fare la differenza fra adeguati controlli di sicurezza o meno.

[7] possesso, conservazione, mantenimento

- **Memorizzazione**: È l'atto di memorizzazione del dato. Avviene quasi simultaneamente alla creazione. Nella memorizzazione bisogna tenere conto del grado di sicurezza che si vuole adottare, in accordo con la classificazione prima eseguita. Bisognerebbe adottare controlli come cifratura, controllo degli accessi, monitoraggio, acquisizione di log, e soprattutto meccanismi di backup per far fronte a eventuali corruzioni e perdite del dato. I dati sono uno dei principali obiettivi degli attaccanti.
- **Uso**: Qualsiasi atto che comporti la visualizzazione, il processamento, ma non la modifica dei dati. I dati in uso sono senz'altro nella fase più vulnerabile del ciclo di vita. In questa fase possono essere trasportati in posti insicuri, e inoltre, per essere usati non possono essere cifrati. I controlli di sicurezza più adeguati in questa fase sono controlli del tipo DLP[8], IRM[9], e controllo degli accessi in file e database, in modo da prevenire accessi e utilizzi non autorizzati.
- **Condivisione**: Atto di rendere le informazioni accessibili ad altri (ad esempio fra diversi utenti). Situazione analoga alla fase precedente per la questione sui controlli di sicurezza da adottare. In questo caso ciò che si vuole prevenire è la condivisione non autorizzata.
- **Archiviazione**: Situazione in cui il dato passa dall'essere attivamente utilizzato all'essere archiviato e mantenuto per lungo tempo. L'archiviazione a lungo termine è necessaria, ma rischia di scontrarsi con problemi tecnologici. Basti pensare alla necessità di dover recuperare dati su nastri magnetici vecchi di 15 anni; la tecnologia attuale sarà compatibile? Inoltre è bene archiviare questi dati e proteggerli sempre in accordo con il grado di classificazione che li riguarda.
- **Distruzione**: La distruzione dei dati è una fase delicata. Può essere operata in svariati modi, sia logica, cancellando ad esempio i puntatori o attraverso la crypto-shredding (trattata in 3.1.2), sia fisica, distruggendo fisicamente i dischi.

Posizione e accesso ai dati

Nonostante il ciclo di vita dei dati non ponga specifiche sulla posizione del dato e chi e da dove può accedervi, è bene avere una piena comprensione di quanto segue.

Posizione Il dato è una risorsa portatile, può (e deve) essere acceduto da diverse punti. Può essere spostato e processato in posizioni diverse rispetto a quella della creazione, e può essere spostato anche su provider diversi per questioni di backup o semplice archivio.

- Chi sono gli attori che possono potenzialmente avere accesso a quel dato?
- Quali sono le possibili posizioni di questo dato?
- Quali sono i controlli di sicurezza in ognuna di queste posizioni?

[8] Data Loss Prevention

[9] Information Rights Management

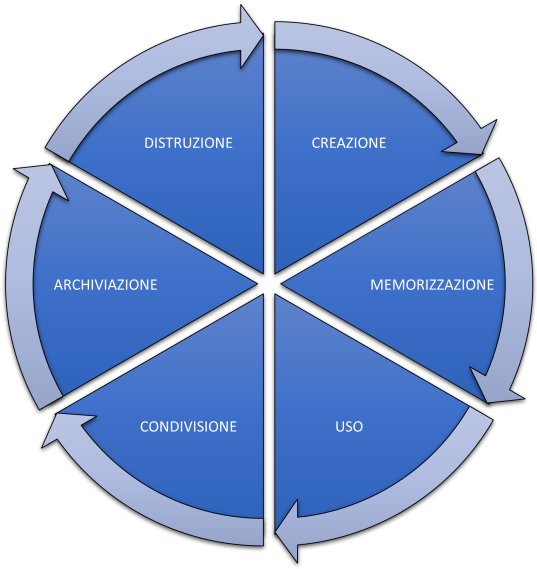

Figura 3.7: Ciclo di vita del dato

- In quali fasi del ciclo di vita il dato può essere spostato da una posizione all'altra?
- In che modo il dato viene spostato da una posizione all'altra (attraverso quali canali)?
- Da dove vi accedono i potenziali attori? Sono sorgenti fidate o no?

Accesso ai dati Il modello del ciclo di vita del dato non specifica neanche i requisiti per chi può accedere, e in che modo, al dato. Le nuove tecnologie hanno creato ovviamente una moltitudine di possibilità di accesso ai dati, in più i vari meccanismi di archiviazione, processamento e trasmissione dei dati hanno di fatto amplificato l'impatto di questa mancanza di requisiti.

3.3.2 Archiviazione dei dati

A seconda del modello cloud in considerazione il modo con cui vengono archiviati i dati risulta essere differente.

IaaS

Come già trattato in precedenza, in questo tipo di modello, la gran parte della gestione è nelle mani dell'utente, e di conseguenza lo sono anche i dati. Il provider tuttavia in questo caso gestisce ancora la virtualizzazione, gli hard drive, la rete. Per la memorizzazione le tipologie sono le seguenti:

- **Volume Storage** In questo caso viene utilizzato un hard drive virtualizzato, collegato a una macchina virtuale, che di fatto si comporta come un hard disk fisico. Un esempio di questo tipo di storage è Elastic Block Store di Amazon.

- **Object Storage** Simile alla condivisione file tramite API o interfaccia web. Un esempio di questo tipo di storage sono i Bucket S3 di Amazon.

PaaS

Come già trattato in sezioni precedenti, in questo caso l'utente agisce su una piattaforma gestita dal cloud provider. Essenzialmente gli utenti gestiscono le applicazioni che si appoggiano su questa piattaforma. Lato archiviazione il modello PaaS utilizza due tipologie:

- **Strutturato** dove le informazioni vengono memorizzate con un alto grado di organizzazione come ad esempio tabelle in un database relazionale. In questo caso la ricerca e il ritrovamento di queste informazioni risulta essere semplice e veloce.

- **Non strutturato** dove le informazioni non vengono memorizzate in maniera organizzata (come ad esempio su righe e colonne). Questo tipo di archiviazione spesso include file multimediali di difficile organizzazione come messaggi email, foto, video, file audio ecc.

SaaS

In questo caso il cloud provider gestisce l'intero servizio offerto sottoforma di applicazione. I tipi di archiviazione usati da questo modello sono:

- **Memorizzazione e gestione dell'informazione:** I dati entrano nel sistema attraverso interfacce web e vengono memorizzati nell'applicazione (di solito attraverso un database nel back-end). I database utilizzati di solito sono installati su sistemi di memorizzazione di tipo volume storage oppure object storage.

- **Memorizzazione di file:** Semplice memorizzazione all'interno dell'applicazione di contenuto in formato di file.

- **Memorizzazione effimera:** Questo tipo di memorizzazione è relativo a istanze IaaS ed esiste fino a quando l'istanza è "up & running". È tipicamente usata per scambiare file o altri elementi temporanei.

- **Content delivery network (CDN):** I dati sono memorizzati in sistemi di tipo object storage che poi vengono distribuiti in molte località geografiche per migliorare la velocità di accesso a questi dati.

- **Archiviazione a lungo termine:** Alcuni vendor offrono servizi di cloud storage allo scopo di dare possibilità di archiviazione dati. Le necessità lato cliente in questo caso sono ad esempio la ricerca, la garanzia di immutabilità del dato, e la gestione del ciclo di vita.

Minacce ai tipi di archiviazione

L'archiviazione dei dati è soggetta ai seguenti tipi di minacce:

- **Uso non autorizzato:** In cloud, c'è il rischio di uso non autorizzato di dati a seguito ad esempio di account hijacking[10] o nel caso in cui viene caricato in cloud materiale non autorizzato. La natura multitenancy del cloud rende però difficile tracciare l'utilizzo non autorizzato.

- **Accesso non autorizzato:** Simile al caso precedente, con aggiunta del caso in cui siano stati forniti permessi non adeguati ad utenti.

- **Attacchi DoS o DDoS:** Rendere inutilizzabili i dati può essere senza dubbio uno dei sistemi di attacco in cloud, essendo la disponibilità del dato uno dei punti di forza del cloud.

- **Corruzione, modifica, distruzione:** Può essere causato da una serie di fattori, non tutti di natura malevola, ma è bene tenere in considerazione tutti gli scenari. Ad esempio può verificarsi a seguito di cause naturali (incendio o allagamento) oppure per fallimenti hardware o software.

- **Perdita o violazione dei dati:** Essendo spesso i dati replicati e distribuiti in cloud, la probabilità che si verifichi una perdita o un furto dei dati viene aumentata. Gli utilizzatori del cloud devono essere a conoscenza di questa situazione.

- **Attacchi o introduzione di malware:** L'obiettivo target nella maggior parte dei casi sono i dati. L'ottenimento di dati può portare a un attaccante ingenti guadagni. Ne consegue che spesso i malware hanno come obiettivo proprio i dati.

- **Trattamento o sanificazione impropria alla fine del ciclo di vita:** Il fine vita dei dati in cloud può essere una sfida, questo perchè tendenzialmente non si può operare distruzione fisica. Oltre alle già citate tecniche per l'eliminazione dei dati, si può comunque tenere conto del fatto che la dispersione caratterizzante del cloud può rendere difficile localizzare le varie porzioni, andando difatto a mitigare in parte il problema derivante da una cancellazione impropria.

3.3.3 Tecnologie rilevanti per la sicurezza dei dati

Tra le varie soluzioni che possiamo identificare per la sicurezza dei dati in cloud, queste sono le principali:

- **DLP:**[11] per evitare perdite di dati

- **Cifratura:** per prevenire visualizzazione non autorizzata dei dati

- **Offuscamento, anonimizzazione, tokenizzazione e mascheramento:** sono alternative alla cifratura nella protezione dei dati

[10] Caso in cui un attaccante prende il controllo di un altro account

[11] Data Loss Protection:

Data Loss Protection

DLP sta per Data Loss Protection oppure anche per Data Leak Prevention; essenzialmente consiste nella serie di meccanismi messi in atto da un'organizzazione per assicurarsi che i dati restino sotto il suo controllo, in linea con politiche regolatorie e standard. Si possono individuare tre elementi fondamentali di un DLP:

- **Scoperta e classificazione** La maggior parte delle tecnologie in cloud fanno grande affidamento su questo punto. Una buona fase di individuazione e classificazione dei dati può fare la differenza. Molti DLP in cloud classificano i dati in base a una serie di categorie; queste categorie possono identificare dati più o meno sensibili quali possono essere dati di carte di credito, dati pubblici ecc.
- **Monitoraggio** Il monitoraggio dell'utilizzo dei dati viene eseguito sia sul traffico in entrata che su quello in uscita. Le migliori strategie di DLP di fatto monitorano l'uso di dati su tutte le possibili località e piattaforme che ne fanno uso, e in più abilitano una serie di policy di utilizzo.
- **Enforcement** Molti tool per la DLP hanno la possibilità di verificare che i dati stiano rispettando una serie di policy sulla posizione, l'uso, la destinazione di un trasferimento. In questi casi, qualora venisse individuata la violazione di una policy possono essere eseguite una serie di azioni di rinforzo di policy, quali ad esempio allertare, bloccare un trasferimento, riposizionarli per eseguire ulteriori validazioni oppure eseguire cifratura prima che i dati lascino i bordi di una organizzazione.

Gli strumenti utilizzati per una DLP fanno di solito fede ai seguenti scenari in cui possano trovarsi i dati:

- **Data In Motion (DIM)** Riguarda i dati in uscita dal perimetro di un'organizzazione. Spesso i sistemi di DLP di questo tipo sono anche detti "network-based" oppure "gateway DLP". Servono essenzialmente a monitorare i protocolli di uscita quali HTTPS[12] o SMTP[13] o FTP[14]. Per fare analisi di traffico HTTPS è necessario avere dei meccanismi per intercettare il protocollo SSL.
- **Data At Rest (DAR)** Gli strumenti per la DLP in questo caso sono posizionati in prossimità di sistemi per l'archiviazione. Questo tipo di DLP è utile per la data discovery e classificazione. Può essere necessario integrare questa soluzione con soluzioni di DLP indirizzate ai Data In Motion.
- **Data In Use (DIU)** Sono i dati nel momento in cui vengono usati dall'utente o dall'applicazione. Può offrire delle informazioni su come gli utenti utilizzano i vari dati. La difficoltà in questa situazione sta più che altro nella necessità di dover installare gli strumenti su una area estremamente vasta e su un numero di utenti potenzialmente altissimo.

[12]Hypertext Transfer Protocol Secure

[13]Simple Mail Transfer Protocol

[14]File Transfer Protocol

Considerazioni per gli strumenti per la DLP in cloud

L'implementazione di una buona strategia di DLP in cloud può essere molto complessa. I motivi sono molteplici; i **dati in cloud tendono a muoversi molto e ad essere replicati**, tra i vari data center, per motivi di backup, per motivi di accessibilità. Per operare poi una buona strategia di DLP è necessario che i dati vengano ben individuati e classificati. Infine **le tecnologie per una buona DLP possono degradare le prestazioni generali**; bisogna tenere a mente che sia che i dati siano in movimento o in uso, eseguire costanti scan può portare ad avere un impatto non più trascurabile.

Cifratura

La cifratura è uno degli elementi più importanti da considerare quando si va a trattare la sicurezza dei dati. La cifratura può essere implementata in diverse fasi del ciclo di vita del dato:

- **DIM** In questo caso le tecnologie per la cifratura dei dati in movimento sono ben note, mature e ben implementate. Si può parlare di protocolli quali IPSec, o TLS/SSL, o meccanismi quali le virtual private network (VPN).
- **DAR** In questo caso la situazione dipende dal tipo di memorizzazione del dato e soprattutto anche dal tempo che questo dato dovrà rimanere archiviato. Se un dato ha necessità di restare archiviato, e segreto, per molto tempo devono essere usati meccanismi più sicuri e che abbiano una certezza di sicurezza duratura (ad esempio lunghezza di chiave adeguata, che renda non fattibile in tempi consoni un attacco di tipo brute force).
- **DIU** La cifratura a questa fase del ciclo di vita del dato risulta essere difficile. Se un dato deve essere visto, usato e processato difficilmente può essere anche cifrato.

Ci sono una miriade di fattori da considerare per quanto riguarda la cifratura in cloud. A seconda del tipo di situazione (DIM/DIU/DAR), a seconda di considerazioni sulle regolamentazioni internazionali e a seconda di una serie di eventuali vincoli aggiuntivi che un'azienda può introdurre per casi specifici (ad esempio informazioni personali), possono essere adottate differenti strategie.

Ne consegue che la cifratura porta con sè una serie di sfide da tenere bene in considerazione:

- L'integrità della cifratura è strettamente dipendente dalla gestione delle chiavi. A seconda se è il provider o il cliente a gestirle possono esserci dei rischi differenti riguardanti la compromissione di queste.
- I dati in cloud sono altamente portabili. Vengono replicati, copiati e spostati da una posizione all'altra, rendendo la gestione delle chiavi particolarmente complessa.
- In situazioni di multitenancy dove c'è condivisione di hardware fisico, può comportare rischi per la sicurezza delle chiavi.
- Hardware sicuro per la cifratura delle chiavi può non essere presente in cloud, con conseguente necessità di utilizzare sistemi di archiviazione chiavi software intrensicamente più vulnerabili.

- La cifratura può degradare le prestazioni soprattutto in quegli ambienti dove le performance sono un requisito fondamentale (esempio data warehouse dove si performano processamenti di grandi quantità di dati).
- La natura del cloud richiede di gestire un numero di chiavi nettamente maggiore rispetto al numero di chiavi utilizzabili in un ambiente tradizionale.
- La cifratura può influenzare la disponibilità dei dati. Questo perchè i controlli sui dati cifrati a livello di backup, e piani di disaster recovery non possono essere accurati come sul dato in chiaro. Inoltre se la cifratura venisse applicata in maniera errata potrebbe addirittura non rendere possibile recuperare i dati.
- La cifratura non risolve i problemi relativi all'integrità del dato. Il dato può restare privato e nonostante questo alterato e/o corrotto. In questo caso dovrebbero essere applicati controlli crittografici aggiuntivi come la firma digitale.

Offuscamento, anonimizzazione, tokenizzazione e mascheramento

Non sempre è possibile eseguire cifratura, sia per motivi di performance sia per motivi tecnici. Ci sono una serie di alternative che possono essere utilizzate a seconda dei casi.

- **Offuscamento**: anche noto come mascheramento, è il processo mediante il quale si vanno a "nascondere", sostituire o omettere informazioni sensibili. Tendenzialmente questo tipo di tecnica è usata per proteggere dati quali informazioni personali (esempio parti di codice fiscale) o porzioni di dati commerciali (esempio parte del numero di telefono). Gli approcci per questo tipo di tecnica sono i seguenti:
  - Sostituzione casuale: il valore è sostituito con un valore casuale.
  - Sostituzione algoritmica: il valore è sostituito con un valore generato algoritmicamente.
  - Mescolazione: i valori (tipicamente di una stessa colonna) vengono mescolati tra loro.
  - Mascheramento: vengono usati caratteri speciali per nascondere alcune parti di un dato, ad esempio gli asterischi nei numeri di carte di credito (esempio 45** **** **** 0876).
  - Eliminazione: vengono usati valori nulli o semplicemente vengono eliminati dei dati.

  Essenzialmente le metodologie di approccio per il mascheramento dati sono due:

  - Statico: usato in ambienti non produttivi; viene creata una copia del dato con dei valori mascherati.
  - Dinamico: usato in ambienti produttivi; spesso conosciuto anche "mascheramento on-the-fly". Viene inserito uno strato adibito al mascheramento tra l'applicazione e il database. Essenzialmente lato visualizzazione non vengono mostrati dati sensibili mentre il dato resta comunque disponibile per essere processato.

- **Anonimizzazione** Spesso i dati sono strettamente collegati all'individuo. in relazione all'individuo possono esere individuati degli identificatori primari e secondari. Quelli primari sono quelli che univocamente identificano un individuo (esempio nome, cognome ecc), e in questo caso vengono utilizzati meccanismi di mascheramento per proteggere queste informazioni. Gli identificatori secondari sono quegli identificatori che non identificano direttamente un individuo, ma in combinazione con altri possono ricondurre ad esso. Il processo di anonimizzazione consiste proprio nel proteggere questi identificatori secondari in modo da evitare che strumenti di analisi possano utilizzarne molteplici per poter essere ricondotti ad un individuo specifico.
- **Tokenizzazione** La tokenizzazione consiste nel sostituire un dato sensibile con uno dallo stesso formato ma casuale e non sensibile. È differente dalla cifratura e presenta sfide e benefici diversi. La cifratura usa una chiave per cifrare il dato, la tokenizzazione invece sostituisce direttamente il dato sensibile con un dato non sensibile all'interno del database (Figura 3.8).

  Di seguito il funzionamento della tokenizzazione:

  1. L'applicazione genera o raccoglie dei dati sensibili
  2. Il dato sensibile viene inviato al server per la tokenizzazione
  3. Il server per la tokenizzazione genera il token. Dato sensibile e token sono entrambi memorizzati nel token database.
  4. Il server per la tokenizzazione ritorna il token all'applicazione
  5. L'applicazione memorizza il token invece che il dato originale
  6. Quando un'applicazione autorizzata richiede il dato, questo viene richiesto e recuperato dal token database

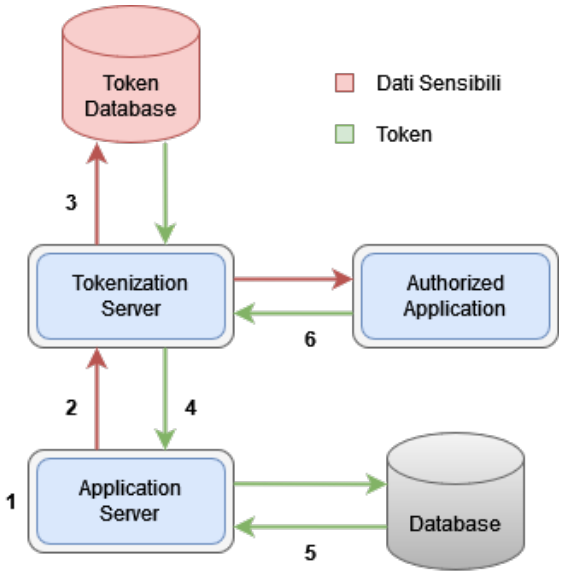

Figura 3.8: Come funziona la tokenizzazione

3.4 Gestione dell'ambiente cloud

In questo capitolo verranno illustrate le operazioni prese in considerazione in questo lavoro di tesi per una gestione sicura dell'ambiente cloud. Quali sono queste operazioni post deploy dell'infrastruttura? Quali sono le operazioni di mantenimento dell'infrastruttura cloud? Verranno presentate alcune delle operazioni individuate, mentre alcune trattate in maniera più superficiale sono state inserite nel paragrafo relativo agli sviluppi futuri (6.1). Gli elementi affrontati nello studio per quanto concerne le operazioni di sicurezza in un ambiente cloud riguardano la gestione delle identità, il controllo degli accessi, un accurato logging e monitoraggio dell'infrastruttura, più una serie di considerazioni riguardanti alcune operazioni di routine che necessitano un certo grado di attenzione e relativa pianificazione.

3.4.1 IAM e controllo degli accessi

IAM sta per Identity & Access Management. Consiste in quell'insieme di tool e pratiche volte a gestire le identità di processi e persone e garantire a questi il giusto livello di accesso alle risorse del sistema cloud in uso. In che modo si ottiene una sicura gestione dell'identità e un giusto livello di accesso?

- La creazione e la distruzione delle identità è un aspetto critico dello IAM. Più precisamente si parla di provisioning e deprovisioning:
  - **Provisioning** l'atto di creare un account per permettere all'utente di accedere alle dovute risorse e sistemi dell'ambiente cloud. È un processo delicato per via del fatto che si vuole dare ad ogni utente accesso al minimo set indispensabile di risorse e non di più, per ovvi motivi di sicurezza, seguendo il principio del need-to-know.
  - **Deprovisioning** è l'atto opposto al provisioning. Con più precisione è l'atto di eliminazione dell'utente nel momento in cui questo non abbia più necessità e autorizzazione di accedere alle risorse cloud dell'azienda. È il caso in cui ad esempio un utente non è più dipendente di quella azienda.
- La gestione dell'accesso invece è l'atto di verificare l'identità di un utente e le rispettive autorizzazioni. In parole semplici è l'atto di ricevere una risposta alle domande "chi sei?" e "a cosa hai accesso?"

  I meccanismi che sono alla base di queste domande sono i meccanismi di autenitica-zione e autorizzazione.

Autenticazione

In maniera molto semplificativa l'autenticazione stabilisce l'identità di un utente sulla base di domande del tipo "Chi sei?" o "Come so che posso fidarmi di te?". In maniera più puntuale consiste nel verificare che un utente sia chi chi dichiara di essere attraverso una serie di meccanismi di verifica dell'identità.

L'autenticazione avviene attraverso meccanismi software, in cui attraverso delle credenziali si dimostra la propria identità. La pratica migliore è utilizzare formati di credenziali differenti e indipendenti tra loro. Questa è la cosidetta autenticazione multifattore. Un

esempio di fattori diversi e indipendenti tra loro può essere l'uso di una password più una password temporanea inviata su un canale diverso (la cosidetta OTP[15], oppure una password e un fattore biometrico. I principali sistemi di autenticazione in ambito aziendale sono LDAP[16] e AD[17].

Segue un analisi di alcuni meccanismi di autenticazione basati su SSO[18]

Con Single Sign-On si intende quel meccanismo che grazie ad un unico set di credenziali (esempio password + OTP) permette l'autenticazione su tutti i sistemi di un'organizzazione, evitando l'utilizzo di numerose credenziali e sgravando il carico di credenziali da dover memorizzare lato azienda.

- **OpenID** è uno standard di autenticazione che permette l'autenticazione di un utente sui relying party[19] attraverso il supporto di una terza parte che fa da identity provider e gestisce pertanto la lista di utenti da autenticare. Un utente così può fare accesso sui sistemi che supportano questo meccanismo, e i sistemi non dovranno mantenere la lista degli utenti che hanno accesso. L'ultima versione di openID è il protocollo OpenID Connect (OIDC) la quale è creata sulla base del protocollo OAuth.
- **OAuth** Questo sistema di autenticazione permette l'accesso su sistemi senza necessità di condividere password. Il meccanismo alla base di questo è basato sul fornire un OAuth token che i relying party possono verificare per consentire l'accesso. Molto di più sull'argomento verrà trattato nel Capitolo 5 essendo OAuth2.0 uno dei sistemi utilizzati in questo lavoro di tesi.
- **SAML** sta per Security Assertion Markup Language. Funziona sulla base di richieste e risposte di token. Si basa su tre elementi/ruoli: l'utente, il service provider e l'identity provider. Il token scambiato non contiene alcuna credenziale dell'utente e inoltre questo standard permette la cifratura e la codifica delle comunicazioni fra identity provider e service provider.

Autorizzazione

Sempre in maniera molto semplificativa, l'autorizzazione stabilisce, dopo l'autenticazione, a cosa si ha accesso, dando una risposta alla domanda "A cosa hai accesso?". In maniera più puntuale consiste nel ricavare, sulla base dell'identità fornita e dimostrata quali siano le risorse accessibili da quel tipo di utenza, attraverso una serie di meccanismi.

In cloud visto il grande numero di utenze, e vista la necessità di dover gestire un gran numero di diversi permessi si tende a preferire meccanismi quali RBAC[20] oppure ABAC[21]

[15] One Time Password)

[16] Lightweight Directory Access Protocol

[17] Active Directory

[18] Single Sign-On

[19] sistemi che richiedono all'utente di essere autenticato

[20] Role Based Access Control

[21] Attribute Based Access Control

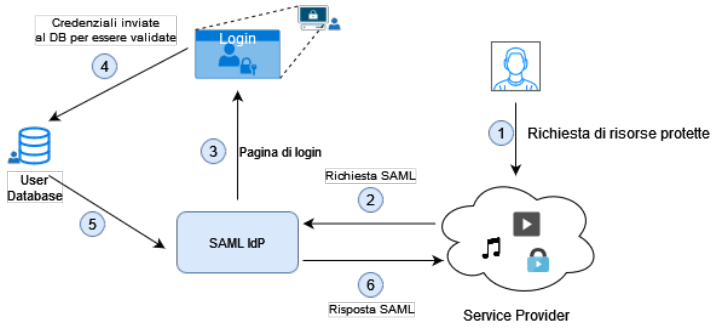

Figura 3.9: Funzionamento SAML

- **Entitlment/task based access control** Sicuramente uno dei più sofisticati, permette di avere una grana di permessi estremamente fine. Si basa sul consentire all'utente una lista di permessi basati su task o processi. Tuttavia il risvolto negativo è che si rischia di dover avere liste troppo grandi e di difficile gestione.

- **RBAC** uno dei meccanismi più utilizzati. Fornisce diritti di accesso sulla base di ruoli. Ogni ruolo ha un set specifico di permessi. Il vantaggio in questo caso è che si creano i ruoli e li si assegna ai rispettivi utenti, scaricando di molto la necessità di avere enormi liste di permessi.

- **ABAC** Vengono generate delle policy che determinano il giusto set di attributi basati su risorse, ambienti, oggetti. Molti ruoli e privilegi sono predefiniti in questo sistema, risolvendo gran parte della complessità.

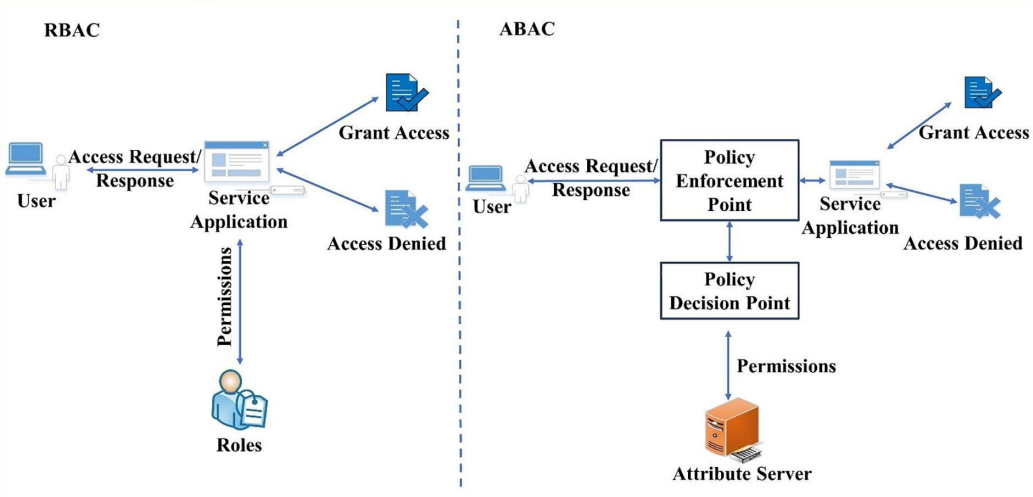

Figura 3.10: RBAC vs ABAC

3.4.2 Logging e Monitoring

In accordo con l'articolo del NIST[22] un log è la registrazione di un evento avvenuto all'interno dei sistemi o rete di un'azienda. I log sono composti da una serie di righe ognuna relativa a un singolo evento. I log possono e devono essere relativi anche a eventi di sicurezza. Vengono generati da diverse sorgenti, tra cui software come antivirus, firewall, e sistemi di prevenzione o rilevamento di intrusione[23]. I log dovrebbero essere protetti a seconda del tipo di log che si sta producendo o anche in base a dove verranno memorizzati. In base al tipo di cloud utilizzato, che sia IaaS, PaaS o SaaS ci sarà un diverso tipo di livello di log producibili lato fornitore e lato cliente. Per fare un esempio, nel caso IaaS, il CSP non può fare collect di log delle varie virtual machine. Nel caso PaaS e SaaS il CSP può fare collect di log rispettivamente a livello sistema operativo o applicativo. Sempre l'articolo del NIST SP 800-92 offre una serie di raccomandazioni che possono aiutare l'azienda ad effettuare una gestione efficiente ed efficacie dei log.

- Le organizzazioni dovrebbero stabilire delle politiche e procedure per la gestione dei log, come mantenerli, dove memorizzarli, per quanto a lungo.
- Le organizzazioni dovrebbero prioritizzare la gestione dei log in base alla riduzione del rischio a cui la gestione di questi può portare.
- Le organizzazioni dovrebbero creare delle infrastrutture per la gestione dei log. Queste infrastrutture consistono in hardware, software, rete, e sistemi utilizzati per creare, memorizzare e analizzare i log.

Security Information and Event Management

Con Security Information and Event Management[24] si intende una raccolta centralizzata di log ed eventi da sistemi differenti. Questo sistema permette la possibile correlazione di eventi differenti da differenti sorgenti per la possibile individuazione anticipata di attacchi. Con SIEM perciò si intende un prodotto software che unisce Security Event Information (SIM) e Security Event Management (SEM) (Figura 3.11).

Per essere più specifici la sezione denominata SEM permette di monitorare in tempo reale eventi, correlare eventi e visualizzarne i risultati. La sezione denominata SIM è relativa più che altro alla memorizzazione, analisi e report di dati di log. Insieme questi due sistemi possono fornire:

- **Aggregazione di dati:** La gestione di log permette aggregazione di dati ed evitare di mancare eventi cruciali.
- **Correlazione:** Questo permette di collegare eventi differenti e apparentemente scorrelati in eventi effettivamente legati da una logica di fondo.

[22] NIST SP 800-92

[23] Intrusion detection systems o Intrusion prevention system

[24] SIEM

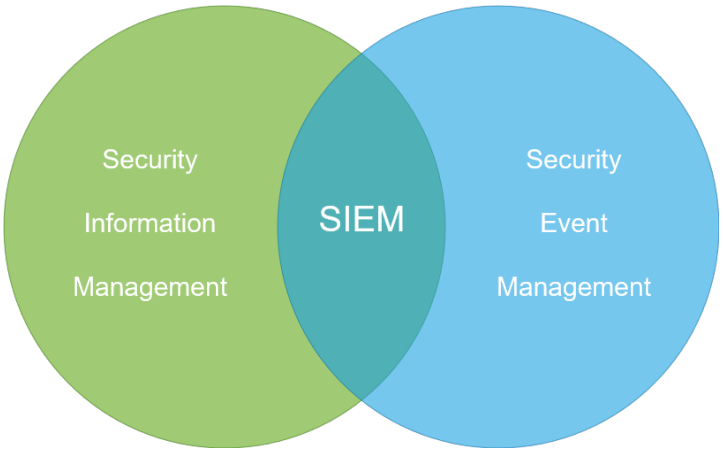

Figura 3.11: SIEM

- **Avvisi:** L'analisi automatizzata può sfociare in avvisi automatizzati in caso di rilevamento di anomalie.
- **Dashboard:** Si può organizzare i dati in grafici o elementi di più facile visualizzazione.
- **Conformità:** Si può fare analisi di log per verificare la conformità con standard e regolamentazioni.
- **Conservazione:** Conservazione a lungo termine di eventi e dati storici per facilitare la correlazione di questi lungo periodi di tempi prolungati. Elemento poi utile nell'analisi forense.

Performance monitoring

Il monitoraggio delle performance di un sistema è un elemento fondamentale per la sicura gestione di un ambiente cloud. Quello che bisognerebbe andare a guardare in questi casi è:

- **Rete:** ad esempio un eccessivo numero di pacchetti scartati, oppure, un traffico eccessivo.
- **Disco:** ad esempio lettura e scrittura lenta.
- **Memoria:** ad esempio eccessivo uso di memoria o completo utilizzo della memoria allocabile disponibile.
- **CPU:** ad esempio eccessivo utilizzo della cpu.

L'analisi di questi elementi può portare al rilevamento di comportamenti malevoli. Bisognerebbe impostare delle soglie oltre le quali scatta un avviso. In questo modo si può intervenire per capire se si tratta di un vero attacco o semplici malfunzionamenti della piattaforma.

3.4.3 Gestione delle operazioni

Le operazioni da gestire in un ambiente cloud sono svariate, e possono spesso interlacciarsi tra di loro. Di seguito una breve trattazione di quali alcune di queste possono essere e come gestirle.

- **Gestione della configurazione:** la configurazione dell'ambiente è un elemento importante da tenere in considerazione. Bisognerebbe sviluppare un chiaro processo per la gestione della configurazione che miri a mantenerla stabile e conforme alle normative e alle buone norme di condotta di sicurezza.
- **Gestione dei cambiamenti:** gli ambienti cloud tendono a subire molti cambiamenti nel corso del loro operato. Avere un dettagliato processo di come gestire questi cambiamenti e adattare le aggiunte all'infrastruttura presente è considerata buona pratica.
- **Gestione degli incidenti:** in questo caso si parla di situazioni che possono interrompere l'erogazione del servizio. Possono essere classificati in base alla gravità e al tipo di impatto che possono avere. Più precisamente le metriche utilizzate sono *impatto* e *urgenza*. La moltiplicazione di questi due fattori da come risultato la *priorità* che deve essere associata a questi incidenti.

  PRIORITÀ = URGENZA X IMPATTO

  Inoltre si può realizzare una tabella di urgenza e impatto su cui basare le proprie stime.

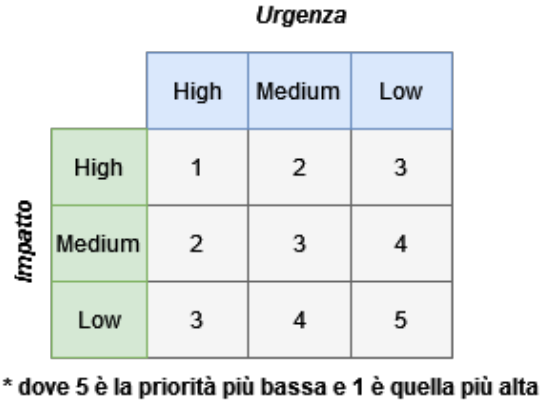

Figura 3.12: Matrice di impatto/urgenza/priorità

- **Gestione dei rilasci:** È importante avere una pianificazione adeguata di rilasci sull'ambiente. È buona norma avere diversi ambienti, ad esempio un ambiente di test. L'obiettivo di pianificare i rilasci è quello di mantenere l'ambiente di produzione stabile durante un nuovo rilascio. I rilasci di nuovo software devono in questo caso essere coerenti con quanto definito nel piano di gestione dei cambiamenti, e un piano di gestione degli incidenti può fare da supporto qualora un rilascio non vada a buon fine.

Ci sono senz'altro altri elementi da considerare. Questi sono quelli individuati e ritenuti di maggior peso.

Capitolo 4

Obiettivi pratici della tesi

Dopo gli studi preparatori effettuati sulla tematica relativa alla sicurezza in cloud, si vuole andare a produrre un framework che funga da modello di base per la realizzazione di applicativi sicuri in cloud. Come già anticipato nei precedenti capitoli, avere delle linee guida di base, come se ci fosse una sorta di ricetta, permette agli sviluppatori di seguire un processo ben strutturato che limita al minimo la possibilità di errore. Quello che si vuole realizzare è un progetto d'esempio che possa essere proposto alle aziende come modello da cui partire per realizzare i propri applicativi.

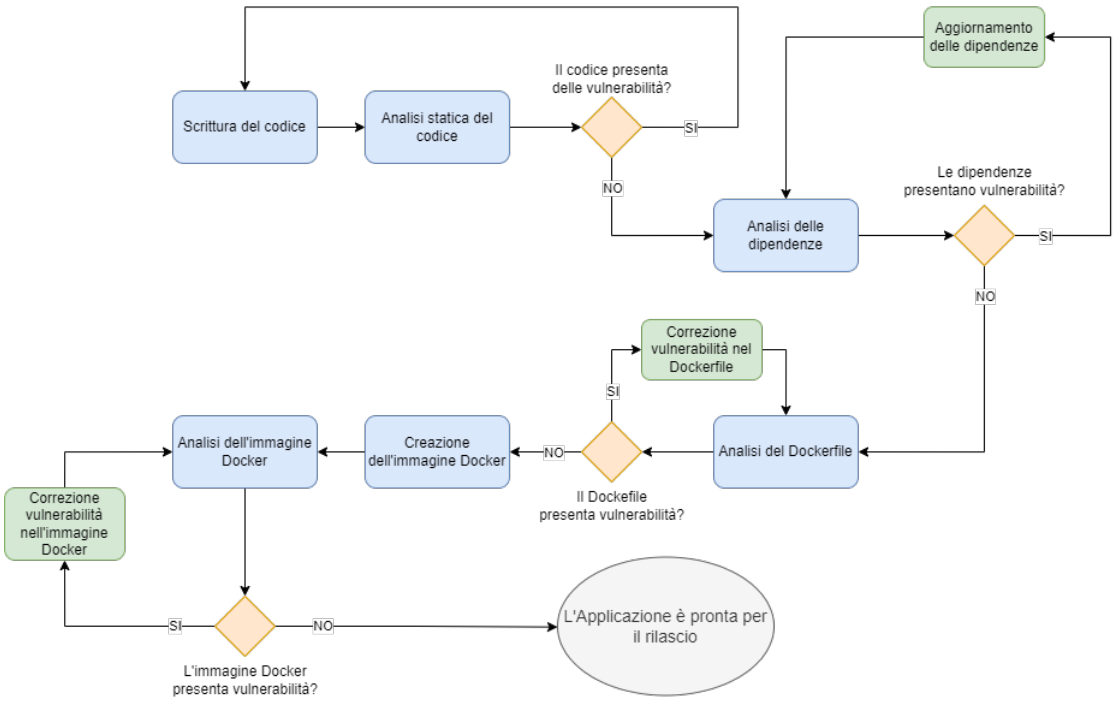

Figura 4.1: Rappresentazione grafica dell'approccio procedurale

In figura 4.1 viene mostrata graficamente l'idea dietro all'approccio procedurale. Ciò su cui si vuole porre particolare attenzione è l'integrazione della sicurezza in ogni step riguardante la progettazione software. Introdurre la sicurezza in ogni step, anche attraverso l'ausilio di prodotti software permette il raggiungimento di un prodotto finale privo almeno delle vulnerabilità note. Gli step che vogliono essere essere tenuti in considerazione sono i seguenti:

- **Scrittura sicura del codice**
- **Analisi statica del codice**
- **Analisi delle dipendenze**
- **Analisi del Dockerfile**
- **Analisi dell'immagine Docker**

Se si presta attenzione a questi passaggi, e si applicano le dovute misure di sicurezza, si può avere un certo grado di certezza riguardante la sicurezza dell'applicazione prodotta. Una volta ultimato il processo di produzione dell'applicazione si può passare alla pubblicazione dell'immagine Docker relativa. In questo caso si può scegliere in base alle necessità dove e se pubblicare l'immagine Docker relativa. Si può scegliere il registry di un cloud provider in particolare (esempio Amazon ECR) oppure Dockerhub oppure soluzioni alternative dettate dalle policy aziendali.

Pubblicata l'immagine dell'applicazione si può realizzare l'infrastruttura cloud. Si sceglie il provider di riferimento (Nel caso d'uso presentato in questa tesi verrà utilizzato Amazon Web Services), e si pubblica l'applicazione. La realizzazione dell'infrastruttura cloud, ovvero la creazione di una rete virtuale, la connessione ad altri servizi, l'introduzione di servizi aggiuntivi può essere fatta in svariati modi. AWS offre una console per la creazione manuale dell'infrastruttura. È possibile anche attraverso la AWS CLI[1]. Una terza soluzione, affrontata nel capitolo 7 riguardante gli sviluppi futuri, è l'utilizzo di file di configurazione con il paradigma noto come Infrastructure as a Service. Questa terza soluzione è la soluzione verso cui si sta tendendo sempre di più ultimamente. Permette, in linea con i principi su cui si fonda questo lavoro di tesi, di creare in maniera standardizzata e attraverso codice dichiarativo le infrastrutture cloud, permettendo inoltre di collegare tool di analisi dei file di configurazione al fine di scongiurare la creazione di infrastrutture mal configurate e vulnerabili.

I capitoli che seguono saranno di realizzazione del framework e presentazione del caso d'uso di esempio. Verranno presentati in dettaglio tutti gli strumenti utilizzati per la realizzazione del framework, verranno trattati nel dettaglio i singoli step di progettazione, verrà specificata nel dettaglio l'architettura cloud realizzata sul cloud provider.

[1]Command Line Interface

Capitolo 5

Tool e tecnologie utilizzate

In questo capitolo verranno presentati in dettaglio gli strumenti e i software usati nella realizzazione di questo lavoro. Non verrà affrontata la soluzione nel dettaglio; questa trattazione verrà rimandata al capitolo 6 in cui si parlerà nello specifico degli strumenti utilizzati in relazione agli studi effettuati nel capitolo 3.

Le tre sezioni principali su cui questo capitolo si incentrerà sono il **Framework Spring**, **Pipeline Jenkins** e **Amazon Web Services**. All'interno di ognuna di queste sezioni verranno trattati gli strumenti relativi ad esse.

5.1 Il framework Spring

Il framework Spring è, per la programmazione in Java il punto di riferimento per la programmazione di microservizi. Il motivo di ciò è dovuto al paradigma alla base di questo framework. Spring ha come focus principale le applicazioni senza essere in un qualche modo legata a un modello di deploy di queste. Spring si basa su alcuni concetti fondamentali quali:

- **Inversion Of Control (IoC):** con inversion of control si intende quella pratica volta a disaccoppiare la dipendenza diretta fra classi nel codice Java. I motivi dietro a questa scelta risiedono nel fatto che una dipendenza troppo stretta tende a rendere poco riutilizzabile il codice. Andando ad intervenire, interponendo delle interfacce tra le classi, si va a rendere possibile il riutilizzo del codice. Più precisamente queste interfacce separano due tipi di responsabilità:
  - *Cosa* deve essere fatto (se ne occupano le classi che la implementano)
  - *Quando* deve essere fatto (se ne occupano le classi che usano l'interfaccia)

  In Figura 5.1 una rappresentazione grafica. Con la classe FooClass che utilizza l'interfaccia BarInterface. L'interfaccia astrae il comportamento delle tre possibili implementazioni (BarImplement1, BarImplement2, BarImplement3) Questo tipo di soluzione può essere molto utile in fase di test, andando a sostituire ad esempio la vera implementazione con un'implementazione di test. L'accoppiamento fra oggetti è reso possibile dal concetto successivo ovvero quello di dependency injection.

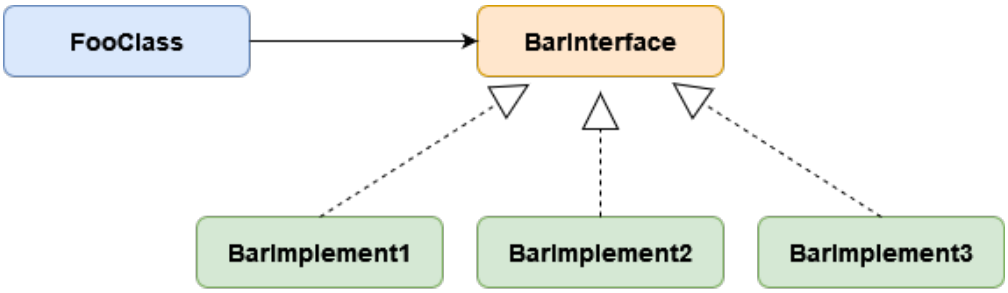

Figura 5.1: Esempio di Inversion of Control

- **Dependency Injection (DI):** Questo è un pattern di programmazione che permetto l'accoppiamento fra oggetti in fase di esecuzione e non in fase di compilazione. Questo tipo di comportamento è reso possibile dall'uso di una serie di elementi quali annotazioni o framework esterni.

- **Aspect Oriented Programming (AOP):** Questo è un paradigma di programmazione che permette di definire un comportamento (aspetto) comune a più elementi dell'applicazione e definirlo in un unico punto. Questo comportamento (in gergo cross-cutting concern) può essere ad esempio legato alla sicurezza oppure alla scrittura di log per ogni API invocata. Può essere definito come classe e il suo comportamento (detto **advice**) può essere applicato tramite annotazione a più API di una classe.

Il framework Spring è costituito da una serie di moduli utilizzabili. Quelli utilizzati per questo lavoro di tesi sono il modulo Spring Boot e il modulo Spring Security.

5.1.1 Spring Boot

Spring Boot è il modulo essenzialmente usato per la programmazione in Java di microservizi. Il modo di approccio con Spring Boot è la scrittura di API REST[1], grazie alla serie di vantaggi che questo modulo offre. Questi vantaggi consistono in gestione pressochè automatica delle dipendenze, autoconfigurazione (include già server HTTP come Tomcat o Jetty) SpringBoot basa il suo paradigma di programmazione su una serie di componenti, seguendo la logica di componentizzare l'applicazione. Si può parlare di approccio noto come MVC[2], ma nello specifico di servizi REST (come nel nostro caso) al concetto di View se ne sostituisce un altro definito ViewModel, il quale si occupa di inviare, tipicamente in formato JSON, i dati ad un altro microservizio il cui scopo è quello di implementare l'interfaccia grafica.

Per quanto riguarda lo sviluppo di servizi REST la metodologia di approccio utilizzata è quella di avere una serie di componenti all'interno dell'applicativo Java, ognuno adibito ad un compito specifico.

[1] Representational State Transfer

[2] Model View Controller

- **Controller:** è il componente più esterno, ha il compito di mappare le url a cui viene contattata l'applicazione.
- **Interfaccia del service:** come anticipato prima, in base al concetto di Inversion of Control si tende a disaccoppiare implementazione con definizione di una classe. È il caso dello strato di servizio che viene diviso in interfaccia e implementazione vera e propria.
- **Implementazione del service:** qui è presente la vera logica dell'applicazione. Lo strato di servizio si relaziona con lo strato di persistenza attraverso degli oggetti chiamati entità che mappano 1:1 gli elementi presenti nella base dati.
- **Repository:** strato di persistenza, usa gli oggetti entità per interfacciarsi con il database andando ad implementare i metodi di creazione, lettura, modifica e cancellazione (CRUD[3]). In questo caso si va ad utilizzare un altro modulo del framework Spring chiamato Spring Data.

5.1.2 Spring Security

Spring Security è un altro modulo del framework Spring. In questo caso questo modulo è utilizzato per applicare concetti di sicurezza all'interno di un applicativo. Il concetto alla base di questo modulo è che la sicurezza è un cross-cutting concern. Pertanto viene sfruttata la programmazione orientata agli aspetti per applicare vincoli di sicurezza in maniera orizzontale sull'applicazione. Un esempio di utilizzo di questo tipo di modulo è il controllo degli accessi e la protezione delle API. Un esempio di ciò che può essere fatto in questo caso è annotare dei metodi con degli advice (5.1), per poter applicare dei controlli di sicurezza. Più nello specifico, per fornire un esempio: si può pensare a una richiesta GET che per essere eseguita ha necessità di avere un utente autenticato, al metodo relativo a questa richiesta verrà apposta l'annotazione @PreAuthorize("hasRole('ADMIN')") per poter abilitare il controllo dei permessi e autorizzare o meno l'utente. Spring Security in questo fornisce una serie di meccanismi aderenti agli standard attuali come il supporto a OAuth2.0 e token di autenticazione come JWT.

```
1
2 @PreAuthorize("hasRole('ADMIN')")
3 @GetMapping({"/users"})
4   public List<UserDTO> getUsers() {
5
6     return userService.getAllUsers();
7   }
```

Listato 5.1: esempio di utilizzo di Spring Security

Ci sono svariati metodi per eseguire autorizzazione nel framework spring. Questo è uno di questi. Va specificato che è necessaria una configurazione delle utenze prima di poter applicare questi aspetti. Bisognerebbe avere una tabella nel database con gli utenti e rispettivi ruoli.

[3]Create, Read, Update, Delete

5.1.3 OAuth2.0

OAuth2.0[4], è un protocollo standard per l'autorizzazione. Come anticipato nel capitolo 3.4.1 l'obiettivo di OAuth2.0 è quello di sganciare le applicazioni dai meccanismi di autorizzazione utilizzando una terza parte per la gestione delle identità e il rilascio di token. OAuth definisce quattro ruoli:

- **Resource Owner:** un'entità che ha la possibilità di offrire accesso a una risorsa protetta.
- **Resource Server:** il server che ospita la risorsa protetta; è in grado di accettare e rispondere a richieste per la risorsa protetta verificando la validità del token.
- **Client:** l'applicazione che richiede la risorsa protetta. Deve fornire l'autorizzazione di cui è in possesso.
- **Authorization Server:** il server che rilascia il token al client.

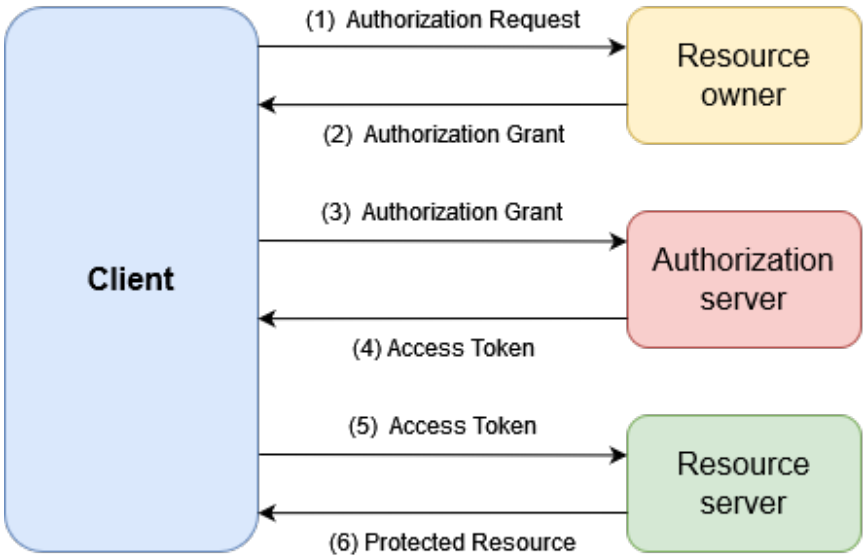

Figura 5.2: Flusso di autenticazione basato su OAuth2.0

Di seguito una trattazione del funzionamento del protocollo.

1. Il client richiede l'autorizzazione al resource owner.
2. Il client riceve un authorization grant, che non è altro che una credenziale che rappresenta l'autorizzazione del resource owner. Il tipo di authorization grant può essere uno dei quattro supportati da questo standard e dipende dal metodo di richiesta da parte del client e da quelli supportati dall'autorization server.

[4]OAuth sta per Open Authentication

3. Il client richiede un access token autenticandosi con l'authorization server presentando anche l'autorization grant.

4. L'authorization server autentica il client e valida l'authorization grant. Se valido, rilascia un access token.

5. Il client richiede la risorsa protetta al resource server autenticandosi con l'access token.

6. Il resource server valida l'access token e se valido fornisce la risorsa.

JSON Web Token

Il JSON Web Token (JWT) è uno standard (RFC 7519 [26]) che definisce un modo compatto e autosufficiente di trasmettere informazioni tra parti in formato JSON. L'informazione trasmessa può essere ritenuta fidata e verificata perchè viene firmata digitalmente. I modi in cui viene firmato il JWT possono essere attraverso HMAC oppure attraverso meccanismi a chiave pubblica/privata come RSA o ECDSA.

Il JWT può essere usato essenzialmente in questi casi:

- **Autorizzazione** questo è anche il caso più comune. Una volta che l'utente ha effettuato il login può associare ad ogni richiesta il JWT per consentirgli l'accesso a risorse protette. È ampiamente usato in scenari con Single Sign On per via del piccolo overhead che apporta il suo utilizzo e la possibilità di essere usato su diversi domini.

- **Scambio di informazioni** caso meno comune ma comunque risulta essere un possibile utilizzo dei JWT. Il motivo è che in questo caso si possono scambiare in maniera sicura delle informazioni. Questo è reso possibile dalla firma digitale che certifica chi sta inviando le informazioni ed essendo la firma calcolata su tutti gli elementi del JWT si può essere certi della sua integrità.

Il JWT è diviso in tre parti; viene codificato in Base64 e le parti sono separate da un punto. Le tre parti sono header, payload e signature.

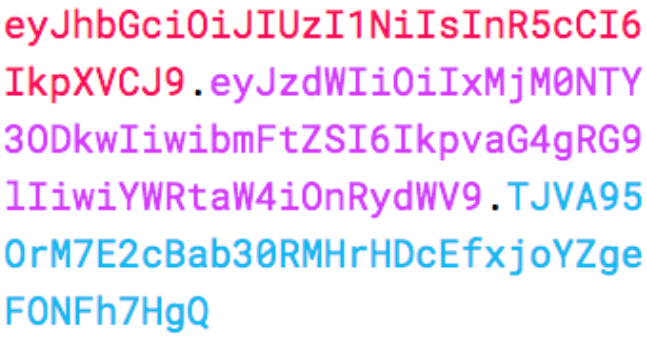

Figura 5.3: Esempio di JWT in Base64

Figura 5.4: Struttura di un JWT

5.2 La pipeline Jenkins

La Pipeline Jenkins è una serie di plugin che supportano e integrano le pipeline di Continuous Delivery all'interno di Jenkins. Jenkins è un server open source che offre una miriade di strumenti e plugin a supporto della costruzione, della distribuzione e dell'automatizzazione di progetti. Una pipeline di continuous delivery è un modo per automatizzare il processo che va dal software di version control (esempio Git) alla distribuzione dell'applicazione. Ogni modifica sul codice può potenzialmente innescare attraversa tutti gli step della pipeline prima di poter essere rilasciato. In questo modo si ha la possibilità di scrivere il software in maniera affidabile e ripetibile facendolo passare per tutti i dovuti stage compresi quelli di test.

La definizione della Pipeline Jenkins avviene attraverso la scrittura di un file, chiamato Jenkinsfile, seguendo la pratica chiamata Pipeline-as-a-code. Questo file può essere incluso anche nel repository del software di version control dove è anche presente il codice del

nostro progetto, rendendo di fatto la pipeline parte integrante del progetto.

Il file supporta due metodi/linguaggi di scrittura: dichiarativo o di scripting. Per questo progetto si è usata la forma dichiarativa.

```
    pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```

Listato 5.2: esempio di pipeline dichiarativa

Si possono individuare alcuni concetti chiave della pipeline tra cui:

- **stage** definisce un subset di compiti (task) e riguarda uno stadio dello sviluppo o un particolare utilizzo di uno strumento associato alla pipeline. Si può fare l'esempio dello stadio di build dell'applicazione o dello stadio di test.
- **step** un singolo compito all'interno di uno stage. Ad esempio l'esecuzione di un comando (esempio: sh 'make') è considerato uno step.

Jenkins offre la possibilità di "collegare" alla pipeline una grande quantità di plugin e strumenti. I seguenti sono quelli scelti per la realizzazione di questo progetto.

5.2.1 SonarQube

Sonarqube è uno strumento universale per l'analisi statica del codice. È di fatto lo standard utilizzato dalla maggior parte delle aziende.Sonarqube è opensource e offre la possibilità di effettuare analisi statica continua del codice offrendo un dettagliato report di bug, code smells (essenzialmente cattive pratiche di programmazione), vulnerabilità, codice duplicato. Sonarqube offre una serie di benefici:

- **Sostenibilità:** Riduce complessità, possibili vulnerabilità, duplicazioni di codice andando ad ottimizzare la durata della vita di un'applicazione.
- **Aumenta la produttività:** Andando ad individuare errori, bug, codice duplicato ma a rimuovere la necessità di dover passare molto tempo a cambiare il codice.

- **Qualità del codice:** Aumenta la qualità del codice.
- **Individua errori:** Riesce ad individuare errori nel codice permettendo agli sviluppatori di correggerli prima di distribuire l'applicazione.
- **Scalabilità:** Non ci sono restrizioni sul numero di progetti che possono essere valutati.
- **Incrementa le capacità degli sviluppatori:** Avere un regolare feedback su problemi di qualità permette agli sviluppatori di migliorare la loro abilità di programmazione.

Sonarqube è uno strumento integrabile nella Pipeline Jenkins in diversi modi. Dettagli aggiuntivi sull'implementazione verranno forniti nel capitolo successivo, in relazione al caso d'uso realizzato.

5.2.2 Trivy

Trivy è uno strumento open source sviluppato da AquaSecurity. È uno strumento pensato per fare l'analisi di file IaC[5] (di più sull'argomento nel capitolo 7 nella sezione riguardante gli sviluppi futuri), ma grazie alla sua grande compatibilità può essere usato anche per l'analisi di Dockerfile e immagini Docker derivanti da questi derivanti. È perfettamente integrabile nelle pipeline CI/CD, basta averlo installato nella macchina host della pipeline. Permette un'ottima copertura sulle possibili vulnerabilità grazie all'utilizzo del database di Aqua Security e al supporto a numerosi linguaggi. Il risultato in output di trivy offre una dettagliata spiegazione del tipo di vulnerabilità, della severità, e offre delle spiegazioni sulle possibili risoluzioni. Trivy è uno strumento stateless e non richiede l'installazione di un database o delle librerie. Di fatto è organizzato in due moduli, client e server, dove il server contiene il database per fare il controllo delle vulnerabilità. Questa è una differenza rispetto a SonarQube che richiede lato client una configurazione più complessa dovendo andare a installare database e un'istanza del server.

5.2.3 OWASP Dependency Check

OWASP Dependency-Check è uno strumento di Software Composition Analysis (SCA) il cui obiettivo è quello di individuare vulnerabilità note all'interno delle dipendenze di un progetto. Questo obiettivo lo raggiunge andando a verificare se esiste un identificativo CPE[6] per una data dipendenza. Se individuato genera un report con le rispettive CVE individuate. Grazie a questo tool è possibile andare a controllare se l'applicazione che si sta sviluppando ha delle vulnerabilità note presenti nelle proprie dipendenze. L'analisi delle dipendenze risulta essere una buona norma di condotta in ambito sviluppo software sicuro. La ragione è che si fa sempre più utilizzo di librerie di terze parti andando ad aumentare la probabilità che queste contengano delle vulnerabilità note. Dependency-check rende questa analisi possibile. È dotato di un'interfaccia su riga di comando e un plugin per Jenkins,

[5] Infrastructure as Code

[6] Common Platform Enumeration

rendendolo integrabile con la pipeline di CI/CD. Il motore centrale contiene una serie di analizzatori che vanno ad ispezionare le dipendenze del progetto. Uno dei vantaggi di questo strumento è che è in grado di aggiornarsi da se ad ogni esecuzione. Per questo aggiornamento si serve del NVD[7] Data Feeds offerto dal NIST[8]. Il NVD è il database standard del governo americano per la gestione delle vulnerabilità.

5.3 Amazon Web Services

Come anticipato nel capitolo 2 nel mondo del cloud computing ci sono tre/quattro attori principali nel campo di fornitura di servizi in cloud. Amazon con il suo AWS risulta essere il leader del mercano occupando la maggior parte del mercato. Per questo progetto infatti ci si è serviti di questa piattaforma. AWS è un provider di servizi in cloud, offre un elevato numero di strumenti. Permette di avere una moltitudine di servizi ed inoltre ha una serie di offerte che spaziano su tutte le categorie cloud (IaaS, PaaS, SaaS). In questo progetto l'approccio è stato più orientato all'utilizzo di servizi e strumenti che fossero aderenti alla categoria PaaS. Di fatto gli strumenti AWS che sono stati utilizzati sono stati le istanze EC2, il sistema di realizzazione di un cluster ECS (Elastic Container Service), lo strumento di memorizzazione delle immagini Docker ECR (Elastic Container Registry) e uno strumento di monitoraggio chiamato Amazon CloudWatch. Di seguito dei dettagli aggiuntivi su questi strumenti.

5.3.1 Istanze EC2

Amazon Elastic Compute Cloud (Amazon EC2) è un servizio offerto da Amazon che fornisce capacità di elaborazione in cloud. In poche parole una istanza EC2 non è altro che un sistema per offrire all'utente la possibilità di installare i propri applicativi su delle piattaforme in cloud. Di fatto con EC2 c'è possibilità di definire un sistema operativo di base su cui verranno poi installate le nostre applicazioni, in altre parole un server virtuale in cloud. Amazon offre un grande quantitativo di tipologie di EC2. La tipologia dipende dalle esigenze del cliente. Le tipologie si differenziano per risorse e capacità offerte. Ad esempio ci sono le istanze ad uso generale utilizzabili oppure possiamo trovare istanze ottimizzate per il calcolo, oppure ottimizzate per la memoria e così via. Una istanza all'avvio sarà la copia di un AMI[9]. Un AMI non è altro che un modello con una configurazione software un po' come lo è l'immagine Docker per un container3.2.3. Da un'AMI puoi infatti avviare più istanze.

5.3.2 Elastic Container Service

Amazon ECS è un servizio di orchestrazione di container completamente gestito che aiuta a implementare, gestire e dimensionare facilmente applicazioni containerizzate. Si integra

[7]National Vulnerability Database

[8]National Institute of Standards and Technology

[9]Amazon Machine Image

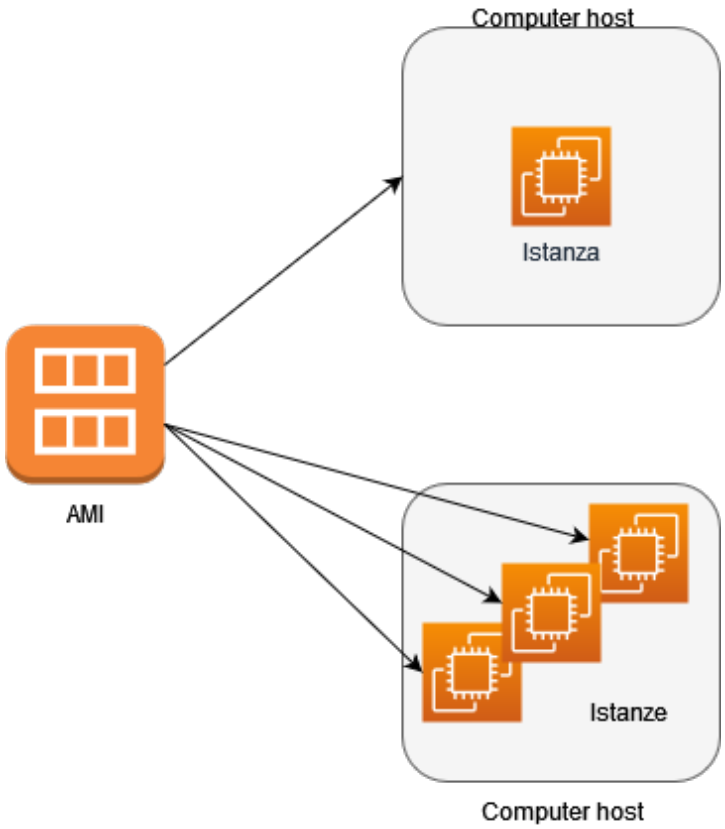

Figura 5.5: Caption

perfettamente con la piattaforma Amazon. Permette l'orchestrazione di un numero potenzialmente molto elevato di container in maniera automatica. Ha essenzialmente due modi di utilizzo in relazione ai container:

- **Fargate:** per poter lanciare i container in maniera automatica e serverless.
- **EC2:** per poter configurare e lanciare istanze EC2.

I due modelli di utilizzo riescono a soddisfare quelle che possono essere le esigenze di un cliente. Possono essere usati anche in simultanea.

5.3.3 Elastic Container Registry

Amazon ECR è un archivio per immagini di container. È più o meno l'equivalente di DockerHub in Amazon. Offre la possibilità di memorizzare immagini di container in forma sia pubblica che privata. Nel caso di memorizzazione privata si possono gestire gli accessi a queste immagini con il sistema di Identity & Access Management (IAM) di AWS.

Amazon ECR è formato dai seguenti componenti:

- **Registry:** Ad ogni account AWS è fornito un registry[10] privato dove poter creare uno o più repository per memorizzarci le immagini.
- **Authorization token:** il client si deve autenticare per poter fare pull o push delle immagini.
- **Repository:** Un repository è un "contenitore" di immagini di container come ad esempio immagini Docker. Possono essercene più di uno dentro ogni registry privato di un utente AWS.
- **Repository policy:** si possono gestire le politiche di accesso ai vari repository.
- **Image:** le immagini presenti all'interno dei vari repository. Dal registry di Amazon possono essere usate sia per creare container in Amazon ECS sia per creare container in locale sulle macchine degli sviluppatori.

5.3.4 Amazon CloudWatch

Amazon CloudWatch è uno strumento offerto da AWS per monitorare le risorse e le applicazioni che vengono eseguite in un ambiente AWS. Permette di raccogliere e tenere traccia di parametri per misurare le risorse delle applicazioni. La dashboard offerta permette di visualizzare direttamente i parametri e le metriche relative ad ogni applicazione monitorata. In più consente di creare dei pannelli di controllo personalizzati per poter visualizzare i parametri scelti di alcune applicazioni. I parametri monitorabili sono CPU, memoria in uso, letture e scritture su disco. Si possono impostare delle soglie di controllo che in caso di superamento possono reagire andando ad apportare modifiche direttamente alle risorse, interrompere l'esecuzione oppure inviare delle notifiche di avviso su canali secondari. Permette di integrarsi con la raccolta log delle applicazioni in esecuzione potendo avere una visualizzazione in un unico punto (la dashboard) di tutte le metriche di interesse.

[10] Archivio

Capitolo 6

Cloud Security: un caso d'uso

In questo capitolo verrà presentato tutto il lavoro svolto per la realizzazione del framework che fungerà da elemento di base, quasi come un ricettario, per poter costruire in maniera sicura applicazioni in cloud. L'obiettivo, come già anticipato in precedenza, era quello di fornire ai lettori, una metodologia di approccio standardizzata, riproducibile per quanto più possibile a seconda delle proprie necessità. Ne consegue che il caso d'uso che qui verrà presentato non deve essere visto come un esempio da riprodurre in toto, bensì, deve essere visto come una spiegazione dell'idea che c'è alla base della realizzazione di questo lavoro. L'idea che c'è alla base risulta pertanto essere, riallacciandosi in parte alla sezione sul ciclo di vita del software (3.1.2), quella di avere una metodologia di approccio collaudata e procedurale. Nel caso d'uso qui presentato, ci si è serviti di una serie di strumenti, trattati nel capitolo precedente, perfettamente sostituibili con altri equivalenti, o integrabili con altri diversi che possano svolgere un lavoro complementare. Un esempio di possibili integrazioni o espansioni verrà trattato nel Capitolo 7 nella sezione sviluppi futuri. Il motivo è che queste integrazioni/espansioni sono state trattate nell'azienda presso la quale è stata affrontata l'esperienza di tirocinio ma al di fuori del contesto di questo; pertanto non sono propriamente argomento di tesi, ma essendo soluzioni complementari si è deciso di inserirle nella trattazione come possibili sviluppi di questo lavoro.

6.1 Analisi del problema

Prima di partire con la descrizione della soluzione realizzata si vuole provare a descrivere quali siano state le premesse che hanno portato all'implementazione della stessa. Come anticipato in precedenza, uno dei grandi problemi del cloud computing è che essendo una tecnologia relativamente giovane, presenta ancora degli approcci acerbi e non ben consolidati negli ambienti aziendali. Unito tutto ciò a una mancanza di esperienza pratica sull'argomento si rischia di creare applicazioni/servizi in cloud con rischi per la sicurezza non trascurabili. L'obiettivo di questo lavoro è porsi come un esempio di approccio, ed essendo stato un progetto realizzato sotto la supervisione di un'azienda di consulenza, si vuole porre come modello da poter presentare a possibili clienti che abbiano la necessità o la semplice volontà di iniziare a realizzare i propri servizi sfruttando il cloud computing.

Scendendo nel pratico per fornire dettagli aggiuntivi questo progetto è partito con l'obiettivo di realizzare una semplice applicazione di test e inserirla in un contesto di approccio DevSecOps. DevSecOps sta per Developing, Security and Operations. È un approccio che ha come filosofia l'integrazione della sicurezza in tutte le fasi del ciclo di vita di un sistema IT. Avendo ben presente il concetto di Security By Design si va ad inserire in un contesto di CI/CD[1] la sicurezza in ogni fase del ciclo di vita. Fino a non molto tempo fa l'approccio utilizzato era quello di DevOps, andando ad integrare la sicurezza solo nella fasi finali dello sviluppo. Con l'avvento di nuove tecnologie e metodologie di sviluppo più agili si è riusciti ad inserire la sicurezza in ogni fase, dedicandogli la dovuta attenzione. Secondo RedHat per garantire una sicurezza completa e costante occorre: mantenere cicli di sviluppo brevi e frequenti, integrare misure di sicurezza con un'interruzione minima delle operazioni, restare al passo con le tecnologie innovative come i container e i microservizi, il tutto promuovendo una maggiore collaborazione tra i team. Tutte queste misure sono inizialmente intraprese dai team che collaborano all'interno dell'azienda, ma in un framework DevSecOps l'automazione ha l'obiettivo di agevolare tali interventi manuali.

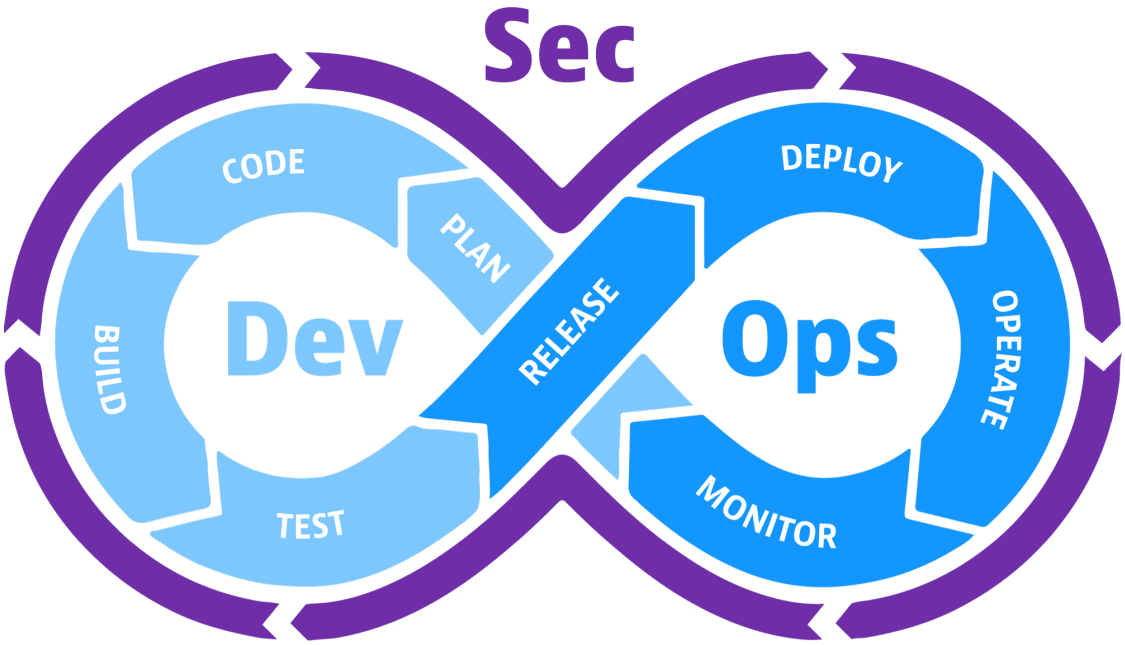

Figura 6.1: Rappresentazione grafica dell'approccio DevSecOps

Sempre secondo RedHat l'obiettivo finale è automatizzare il processo di sviluppo. Questo sta diventando via via più possibile grazie allo sviluppo agile e all'integrazione di nuovi strumenti di automazione.

Questo lavoro di tesi si è svolto con in mente l'obiettivo di rendere tutto quanto più automatico possibile, cercando di limitare l'intervento umano al minimo indispensabile, limitando di fatto anche gli errori.

[1]Continuos Integration Continuous Delivery

6.1.1 Un approccio procedurale

Con ben in mente l'obiettivo di rendere il processo di questo lavoro quanto più automatizzato possibile, si è sviluppato il progetto sulla Pipeline Jenkins. Il motivo per cui è stato definito procedurale è strettamente correlato al concetto di ciclo di vita del software e al tipo di approccio che offre uno sviluppo basato su pipeline. Partendo da un'analisi orizzontale sulle tematiche di sicurezza applicabili in cloud si è deciso di individuare quattro pilastri fondamentali. Per ognuno di questi pilastri si è fatta un'analisi su quali potessero essere gli strumenti e le tecnologie utilizzabili per integrarvi la sicurezza. In questo caso si può notare come la tematica di sicurezza entra già in gioco nella fase di pianificazione e analisi dei requisiti del ciclo di vita dello sviluppo software. Dopodichè si è passati allo sviluppo vero e proprio e grazie al supporto della pipeline si è potuto realizzare un processo di sviluppo che per ognuno di questi pilastri integrasse i rispettivi strumenti di sicurezza. La procedura, la "ricetta", risulta pertanto essere così organizzata:

1. Scrittura dell'applicazione di test, con ben a mente la volontà di integrare meccanismi di autorizzazione sulle API esposte.
2. Analisi statica del codice scritto.
3. Analisi delle dipendenze dell'applicazione.
4. Realizzazione dell'immagine Docker tramite Dockerfile e analisi sia dell'immagine che del Dockerfile.
5. Rilascio dell'applicazione con relativo monitoraggio dell'infrastruttura e dell'applicazione stessa.

Come si può notare la sicurezza risulta essere ben integrata in ogni fase di sviluppo. Questo non vuole significare che questa metodologia debba essere l'approccio unico, bensì vuole evidenziare come per ogni piccola sezione dello sviluppo sia possibile integrare la sicurezza.

6.1.2 I quattro pilastri della sicurezza in cloud

In questa sezione verrà presentata l'analisi relativa ai quattro pilastri della sicurezza in cloud individuati e trattati nel Capitolo 3. Più precisamente verrà analizzato, per ogni pilastro, in che modo è stato questo inserito nel lavoro svolto.

- **Sicurezza dell'applicazione**

  Per la parte relativa alla sicurezza dell'applicazione si è scelto di utilizzare i seguenti strumenti:

  - **Spring Security** trattato nella sottosezione 5.1.2, utilizzato per eseguire API security. È stato utilizzato in concomitanza con OAuth2.0 e JWT per fornire autorizzazione su alcune delle API implementate nell'applicazione.
  - **Sonarqube** trattato nella sottosezione 5.2.1, utilizzato per eseguire analisi statica del codice dell'applicazione di test realizzata.

  - **OWASP Dependency Check:** trattato nella sottosezione 5.2.3 utilizzato per eseguire analisi delle dipendenze.

- **Sicurezza dei dati**

  In questo caso essendo la categoria cloud la categoria PaaS, la protezione dei dati risultava essere un po' fuori dal nostro campo di operazione. Ci si è limitati ad implementare una soluzione di mascheramento di dati sensibili all'interno dei log raccolti. Per la raccolta di log è stata utilizzata la libreria LogBack.

- **Sicurezza dei container**

  Per la sicurezza dei container è stato usato lo strumento Trivy trattato nella sezione 5.2.2. È stato utilizzato in due modalità.

  1. Per l'analisi del Dockerfile, essendo uno strumento adibito alla sicurezza di file IaC è anche in grado di scansionare un Dockerfile in cerca di eventuali misconfigurations.
  2. Per l'analisi dell'immagine Docker derivante dalla build del Dockerfile, in cerca di vulnerabilità.

- **Gestione dell'ambiente cloud**

  Come già anticipato il cloud provider utilizzato per il deploy della soluzione è stato Amazon Web Services. Più precisamente è stato usato il sistema di orchestrazione di container Elastic Container Service (ECS) (5.3.2) per il deploy di un container sottoforma di istanza EC2 (5.3.1) con all'interno la nostra applicazione. L'immagine della applicazione era contenuta nel registry offerto da Amazon ovvero Elastic Container Registry (ECR) (5.3.3). Il tutto è stato successivamente tenuto sotto monitoraggio grazie allo strumento CloudWatch (5.3.4).

6.2 Implementazione della soluzione

In questo paragrafo vedremo nel dettaglio come è stato implementato questo framework, dalla scrittura dell'applicazione all'inserimento della stessa in un contesto di DevSecOps con la Pipeline Jenkins.

6.2.1 Scrittura dell'applicazione

L'applicazione di test implementata è un semplice banco di prova per testare il nostro framework. L'obiettivo primario non era quello di scrivere un'applicazione complessa, ma mostrare come, anche in casi molto semplici, l'attenzione alla sicurezza è fondamentale.

La logica di base dell'applicazione è quella di avere un database di utenti, i quali possono essere letti, aggiunti, modificati o rimossi.

L'applicazione è stata scritta in Java usando il framework Spring. L'esempio che si voleva fornire era il core (il back end) di un applicazione web che esponeva una serie di API tramite meccanismi REST. Le API esposte sono cinque. Due richieste GET, una

richiesta POST, una richiesta PUT e una richiesta DELETE. Si è scelto di utilizzare tutte le richieste HTTP che facessero fede al modello CRUD[2]. Di seguito il codice di queste API.

```
1
2  @Logged
3  @GetMapping({"/users"})
4  public List<UserDTO> getUsers() {
5
6    return userService.getAllUsers();
7  }
8
9  @Logged
10 @GetMapping("/users/{username}")
11 public UserDTO getUser(@PathVariable String username) {
12
13   return userService.getOne(username);
14 }
15
16 @Logged
17 @PostMapping("users/addOne")
18 public UserDTO addUser(@RequestBody UserDTO userDTO) {
19
20   userService.addUser(userDTO);
21   return userDTO;
22 }
23
24 @DeleteMapping("users/{username}")
25 public void deleteUser(@PathVariable String username) {
26   userService.deleteUser(username);
27 }
28
29 @PutMapping("users/{username}")
30 public UserDTO updateName(@PathVariable String username,
     @RequestBody String newName) {
31   return userService.updateUsername(username, newName);
32 }
```

Listato 6.1: Le API esposte dall'applicazione

Attraverso le richieste su alcune url specifiche ad esempio "/users" si possono eseguire delle operazioni di lettura/scrittura/modifica/cancellazione.

L'applicazione è organizzata secondo le convenzioni di sviluppo di un microservizio in Java tramite Spring Boot. È organizzata in Controllore, Servizio, Repository.

- **Controller**

  Il controllore è lo strato che si occupa di interagire con "l'esterno". Più precisamente fornirà i dati (in questo caso in formato JSON essendo un controllore REST) allo

[2]Create, Read, Update, Delete

strato di visualizzazione. Lo strato di visualizzazione non è stato realizzato in questo progetto. In un ambiente reale sarebbe un altro microservizio che si occuperebbe di raccogliere i dati forniti dal controllore e realizzare la vista di questa applicazione web. Il controllore interagisce con lo strato di servizio attraverso oggetti chiamati Data Transfer Object (DTO), che sono oggetti Java mappati sui dati che dovranno essere letti o scritti nel database e su cui si possono eseguire delle operazioni.

- **Servizio**

  Lo strato di servizio è lo strato dove viene implementata la logica dell'applicazione. Fa da tramite tra strato di persistenza e strato di controllo. Comunica con lo strato di controllo attraverso i DTO, mentre con lo strato di persistenza attraverso oggetti chiamati Entità che modella i dati esattamente come vengono tenuti nel DB. Questa pratica serve a disaccoppiare lo strato di servizio allo strato di persistenza. Una Entità rappresenta di fatto una tabella nel database. Al progetto è stata poi aggiunta una funzione di utilità che mappasse DTO su Entità e viceversa. Il motivo per cui si sceglie di utilizzare questi due oggetti è per concentrare le eventuali modifiche in un unico punto.

```
1  @Service
2  @Transactional
3  public class UserServiceImpl implements UserService {
4
5    @Autowired
6    UserRepository userRepository;
7
8    @Autowired
9    ModelMapper modelMapper;
10
11   @Override
12   public List<UserDTO> getAllUsers(){
13
14     return userRepository.findAll().stream().map(u ->
     modelMapper.map(u, UserDTO.class)
     ).collect(Collectors.toList());
15
16   }
17
18   @Override
19   public UserDTO getOne(String username) {
20     return modelMapper.map(userRepository.
21                                 getById(username),
     UserDTO.class);
22   }
23
24   @Override
25   public UserDTO addUser(UserDTO userDTO) {
26     if(userRepository.existsById(userDTO.
27                                 getUsername()))
28       return null;
```

```
29      userRepository.save(modelMapper.map(userDTO, User.class));
30
31      return userDTO;
32
33
34    }
35
36    @Override
37    public void deleteUser(String username) {
38      if(userRepository.existsById(username)) {
39        User u = userRepository.getById(username);
40        userRepository.delete(u);
41      }
42    }
43
44    @Override
45    public UserDTO updateUsername(String username, String
       newName) {
46      if(userRepository.existsById(username)) {
47        User u = userRepository.getById(username);
48        u.setName(newName);
49        userRepository.save(u);
50
51        return modelMapper.map(u, UserDTO.class);
52      }
53      return null;
54    }
55
56 }
57
```

Listato 6.2: Il codice del servizio

- **Persistenza**

  Lo strato di persistenza invece è realizzato attraverso una classe di Repository che implementa l'interfaccia JPARepository. In questo modo è possibile sfruttare sempre secondo la logica dell'Inversion of Control la possibilità di scrivere la propria implementazione. L'implementazione standard però era sufficiente per il nostro progetto quindi lo strato di persistenza pur essendo personalizzato per il database con i nostri utenti utilizza i metodi standard forniti dall'interfaccia implementata.

Per quanto riguarda invece l'implementazione di servizi di sicurezza di ci si è serviti di un authorization service di terze parti fornito da Auth0 [33], che permetteva il rilascio di un token autorizzativo, un JWT. Il JWT veniva validato poi dall'applicazione e serviva a proteggere alcune delle API esposte. Nel nostro caso le API che richiedevano autorizzazione erano quelle che operavano scrittura nel database. La validazione del JWT avviene attraverso una classe Java, mentre la configurazione delle regole per la protezione delle API è delegata ad un'altra classe.

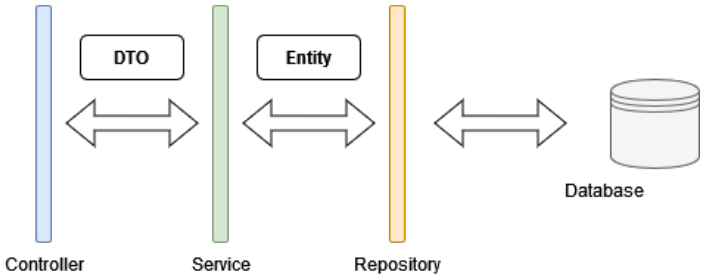

Figura 6.2: Rappresentazione della struttura dell'applicazione

```
@Override
  protected void configure(HttpSecurity http) throws Exception{

    http.cors().and().authorizeRequests()
        .mvcMatchers(HttpMethod.GET, "/users/**").permitAll()
        .anyRequest()
        .authenticated()
        .and()
        .oauth2ResourceServer()
        .jwt()
        .decoder(jwtDecoder());

  }
```

Listato 6.3: Configurazione delle regole di sicurezza per le API

6.2.2 Pipeline Jenkins

Nel contesto della scrittura del codice, questo è stato inserito in un repository github. In questo modo ogni qualvolta venisse eseguita una modifica al codice si può ripetere il processo realizzato nella Pipeline Jenkins per eseguire tutti i controlli di sicurezza implementati.

La Pipeline è realizzata attraverso un file di scripting. È suddivisa in stage, ognuno adibito ad un determinato controllo di sicurezza, oppure a operazioni classiche come la import del progetto o la pubblicazione dell'immagine sul registry di Amazon Web Services. Alcuni degli stadi richiedevano delle credenziali per poter contattare gli strumenti necessari. Jenkins mette a disposizione un sistema di gestione delle credenziali che restano conservate cifrate nella macchina ospite di Jenkins e vengono contrassegnate da un Id univoco. In questo modo è possibile inserire l'id nel file di scripting invece delle credenziali in chiaro.

Gli stage della pipeline sono i seguenti:

- **Git Clone**

  In questo stadio si fa la semplice clonazione del progetto dal repository di github. È stato necessario dover settare delle credenziali in quanto il repository è stato impostato a privato.

- **Build Application**

  Stadio in cui il codice dell'applicazione viene compilato attraverso l'utilizzo di Maven invocato da riga di comando. Jenkins supporta l'esecuzioni di istruzioni da riga di comando a patto che sulla macchina host sia installato il software che si sta invocando. In questo caso infatti è stato necessario avere Maven installato sulla macchina locale.

- **Sonarqube Analysis**

  Da questo stadio in poi inizia una serie di stadi di controlli di sicurezza. Il primo stadio è lo stadio di analisi statica del codice con SonarQube. Sonarqube ha un'architettura client-server. Per farlo funzionare è stato necessario installare sulla macchina locale il sonarqube server che restava in ascolto sulla porta 9000. È stato necessario creare un account e delle credenziali. Il report dell'analisi (i cui risultati verranno mostrati nella sezione successiva) può essere visionato direttamente dalla dashboard messa a disposizione da Sonarqube. Per poter essere integrato in Jenkins Sonarqube ha bisogno di un plug-in chiamato Sonar Scanner. Tutto il funzionamento su Jenkins avviene tramite questo plugin.

```
1     stage('SonarQube analysis'){
2       when{
3         expression{
4           return params.executeSonarQube;
5         }
6       }
7       steps{
8         dir('app'){
9           withSonarQubeEnv('sonarqube-9.4'){
10            sh 'mvn -f aws-lab/pom.xml sonar:sonar'
11            -Dsonar.host.url=http://localhost:9000
12
    -Dsonar.login=97477bf51b33e57583eea46d54c2bbb4db43c475'
13          }
14        }
15      }
16    }
```

Listato 6.4: Stadio di analisi di Sonarqube

- **Sonarqube quality gate**

  Il plugin di Sonarqube mette a disposizione dei sistemi per interrompere la pipeline qualora dei risultati ottenuti dall'analisi non fossero soddisfacenti. In pratica vengono impostate delle soglie di accettabilità (quality gates). Per sonarqube queste soglie richiedono che non siano presenti vulnerabilità di tipo High o Critical.

```
1     stage('SonarQube Quality Gates'){
2       when{
3         expression{
4           return params.enableQualityGates &&
    params.executeSonarQube;
5         }
```

```
6       }
7       steps{
8         dir('app'){
9           waitForQualityGate abortPipeline: true
10        }
11      }
12    }
```

Listato 6.5: SonarQube Quality Gate

- **OWASP Dependency Check**

  Anche per OWASP Dependency check è disponibile un plug-in per Jenkins. In questo caso non è stato necessario installare alcun software sulla macchina locale. Il plugin di OWASP è sufficiente. Il plugin è composto da tre elementi, un global tool configurator, un builder e un publisher. Il primo è in grado di installare tutto il necessario, il secondo è il modulo che esegue l'analisi mentre il publisher ha il compito di leggere il report in formato xml e generare le metriche che questo plugin offre. Di fatto nel comando vengono inseriti come argomenti i due formati di file che si vogliono generare ovvero XML e HTML. XML per generare le metriche e i grafici, HTML per poter permettere la lettura del report all'utente. In questo stadio vengono anche definite le regole per il quality gate. Come per sonarqube si parla di nessuna vulnerabilità critical o high.

```
1     stage('OWASP Dependency Check'){
2       when{
3         expression{
4           return params.executeDependencyCheck;
5         }
6       }
7       steps{
8         dir('app/aws-lab'){
9           script{
10            dependencyCheck additionalArguments: ' -o
    "./target" -s "./" -f "XML" -f "HTML" --disableRetireJS
    -n', odcInstallation: 'owasp-dependencycheck'
11            if (params.enableQualityGates){
12              dependencyCheckPublisher (
13              pattern: "target/${DEPENDENCY_CHECK_REPORT}",
14              failedTotalHigh: 1,
15              failedTotalCritical: 1
16              )
17            }else{
18              dependencyCheckPublisher (
19                pattern: "target/${DEPENDENCY_CHECK_REPORT}"
20              )
21            }
22            archiveArtifacts artifacts:
    'target/dependency-check-report.html'
23          }
24        }
```

```
25        }
26    }
```

Listato 6.6: OWASP Dependency Check

- **OWASP Dependency Check quality gate**

  Grazie alle regole definite nello stadio precedente il plugin è in grado di capire se l'analisi ha avuto successo o fallimento. Il risultato viene inserito in una variabile "currentBuild.result". Qualora il risultato fosse "FAILURE" la pipeline viene interrotta esattamente come succedeva per il quality gate di Sonarqube.

```
1     stage('Dependency Check quality gate'){
2       when{
3         expression{
4           return params.enableQualityGates &&
    params.executeSonarQube;
5         }
6       }
7       steps{
8         dir('app/aws-lab'){
9           script{
10            if(currentBuild.result == 'FAILURE'){
11              error('Dependency Check Quality Gate: found at
    least 1 Critical or High vulnerability in one dependency')
12            }
13          }
14        }
15      }
16    }
```

Listato 6.7: OWASP Dependency Check Quality Gate

- **Dockerfile check with Trivy**

  Trivy è uno strumento opensource per IaC scanning. A differenza dei due tool precedenti non ha necessità di un plug-in, ma è necessario che lo strumento sia installato sulla macchina locale. Da qui è poi solo necessario invocare il comando come un qualsiasi comando di shell. Trivy è stato utilizzato in due modalità differenti. In questa modalità l'uso che se ne è fatto è l'analisi del Dockerfile in cerca di eventuali misconfiguration.

```
1     stage('Dockerfile scan with Trivy'){
2       steps{
3         dir('app/aws-lab'){
4           script{
5             sh "trivy config Dockerfile >
    trivyDockerfileReport.txt"
6           }
7         }
8       }
```

```
9     }
```

Listato 6.8: Analisi del Dockerfile con Trivy

- **Trivy quality gate 1**

  Essendo Trivy stato utilizzato in due modi differenti, sono stati impostati anche due quality gate differenti. C'è da precisare tuttavia che essendo Trivy un tool molto semplice e da usare da riga di comando, non presentava un meccanismo di definizione automatica dei quality gate. In questo caso allora si è deciso di operare delle GREP[3] per controllare se nei file di report fossero presenti o meno delle vulnerabilità di tipo high o critical.

```
1  stage('Trivy quality gate 1'){
2        when{
3          expression{
4            return params.checkDockerImage &&
     params.enableQualityGates;
5          }
6        }
7        steps{
8          dir('app/aws-lab'){
9            script{
10
11             criticalVuln = sh(script: "grep
     --regexp='CRITICAL: [1-9]*'
     ./trivyDockerfileReport.txt",returnStatus: true)
12             highVuln = sh(script: "grep --regexp='HIGH:
     [1-9]*' ./trivyDockerfileReport.txt",returnStatus: true)
13
14             if (!criticalVuln || !highVuln) {
15               error('Trivy Dockerfile Quality Gate: found at
     least 1 Critical or High vulnerability in Dockerfile')
16             }
17           }
18         }
19       }
20     }
```

Listato 6.9: Primo quality gate di trivy

- **Docker Image Build**

  Semplice stadio di build dell'immagine Docker. Anche in questo caso non è necessario avere un plug-in, ma è necessario che lo strumento sia installato sulla macchina sulla quale è in esecuzione Jenkins. La build dell'immagine è stata posizionata subito dopo il quality gate di Trivy sul Dockerfile.

[3]filtro linux per individuazione delle occorenze

```
1     stage('Docker image build'){
2       steps{
3         dir('app/aws-lab'){
4           script{
5             img = docker.build("awslabimage")
6           }
7         }
8       }
9     }
```

Listato 6.10: Build dell'immagine Docker

- **Docker Image scan with Trivy**

  Questo è il secondo modo di utilizzo di Trivy. In questo caso si va ad eseguire un vulnerability scan sull'immagine docker appena creata.

```
1     stage('Check Docker image with Trivy'){
2       when{
3         expression{
4           return params.checkDockerImage;
5         }
6       }
7       steps{
8         dir('app/aws-lab'){
9           script{
10            sh "trivy image awslabimage > trivyReport.txt"
11          }
12        }
13      }
14    }
```

Listato 6.11: Vulnerability scan sull'immagine Docker con Trivy

- **Trivy quality gate 2**

  Come per il quality gate precedente è stato necessario andare ad analizzare attraverso delle GREP la presenza di vulnerabilità di tipo high o critical.

```
1     stage('Trivy quality gate 2'){
2       when{
3         expression{
4           return params.checkDockerImage &&
    params.enableQualityGates;
5         }
6       }
7       steps{
8         dir('app/aws-lab'){
9           script{
10
```

```
11              criticalVuln = sh(script: "grep
    --regexp='CRITICAL: [1-9]*'
    ./trivyReport.txt",returnStatus: true)
12              highVuln = sh(script: "grep --regexp='HIGH:
    [1-9]*' ./trivyReport.txt",returnStatus: true)
13
14              if (!criticalVuln || !highVuln) {
15                error('Trivy IaC Quality Gate: found at least 1
    Critical or High vulnerability in one dependency')
16              }
17            }
18          }
19        }
20     }
```

Listato 6.12: Trivy quality gate

- **Docker image push**

  Se la pipeline procede fino all'ultimo stadio con tutti i quality gate superati, si può procedere alla pubblicazione dell'immagine Docker dell'applicazione sul registry di Amazon, ECR. In questo caso è necessario avere un account AWS. Dopodichè è stato necessario creare delle credenziali per la pubblicazione dell'immagine. Jenkins mette a disposizione nel tool per la gestione delle credenziali anche il tipo di credenziali accettate da Amazon. Per la precisione si parla di Access Key ID e Secret Access Key. Nello strumento vengono definite proprio come "AWS Credentials". Il comando docker.Registry("...","...") permette di specificare l'indirizzo del registry e l'ID delle credenziali Amazon salvate in Jenkins.

```
1     stage('docker image push'){
2       when{
3         expression{
4           return params.pushDockerImage;
5         }
6       }
7       steps{
8         dir('app/aws-lab'){
9           script{
10
    docker.withRegistry('https://891288190130.dkr.ecr.eu-west-3
11              .amazonaws.com/awslabimage',
12              'ecr:eu-west-3:aws-credentials'){
13                img.push("latest")
14              }
15            }
16          }
17        }
18      }
19    }
```

Listato 6.13: Push dell'immagine docker su Amazon ECR

6.2.3 Deploy e infrastruttura AWS

Nella fase finale del nostro ciclo di sviluppo si posiziona la fase di setup dell'infrastruttura cloud e deploy dell'applicazione realizzata. Come spiegato nella sezione precedente nell'ultimo stadio della pipeline l'immagine docker dell'applicazione viene caricata su Amazon ECR. Il motivo per cui si è scelto il registry proprietario di Amazon deriva dal fatto che l'infrastruttura che si sta realizzando si servirà di Amazon come provider. Sarebbe stato ben fattibile anche avere l'immagine memorizzata su DockerHub, tuttavia si è provato ad essere il più possibili fedeli ad un caso d'uso reale immaginando come un eventuale azienda possa decidere di sfruttare quanto offerto da un unico provider senza delocalizzare troppo gli elementi che compongono il progetto.

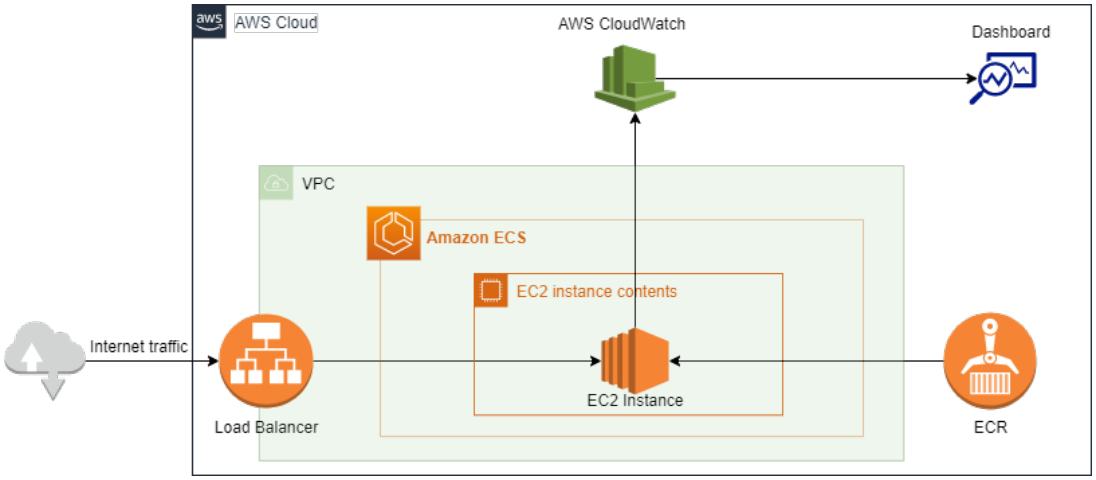

Figura 6.3: Rappresentazione grafica dell'infrastruttura realizzata

Come si può notare dalla Figura 6.3 all'interno della rete Amazon è stata creata una VPC[4] che rappresenta la rete entro la quale opera il nostro ipotetico datacenter. All'interno della VPC infatti è presente il cluster realizzato tramite ECS. Come spiegato nella sezione 5.3.2 Amazon ECS è un servizio di orchestrazione di container. Dalla console di Amazon è possibile, alla realizzazione del cluster, specificare tramite un sistema di task definition quali siano i container da voler deployare all'interno del cluster. Scelta l'immagine dal registry ECR viene deployata sottoforma di istanza EC2. È ovviamente possibile scegliere il tipo di istanza che si vuole utilizzare. In questo caso è stata scelta un'istanza di tipo t2.micro. Le istanze di tipo t2.micro sono dotate di poche risorse di calcolo e sono adatte a task dalle poche esigenze. Nella definizione del cluster era anche possibile scegliere se si volesse implementare un sistema di logging e monitoring quale Amazon CloudWatch. Amazon CloudWatch permette di fare la collect dei log prodotti dall'applicazione, più la collect di tutte le metriche su risorse e consumi, permettendo quanto più possibile una visione completa della "salute" del cluster realizzato.

[4]Virtual Private Cloud

6.3 Risultati ottenuti

In questa sezione verranno mostrati gli output di tutte le analisi svolte dai vari tool di sicurezza utilizzati. Si possono fare alcune considerazioni preliminari; come potremo vedere, nonostante l'applicazione sia estremamente semplice e a solo scopo di prova, le analisi mostrano come l'inclusione delle varie librerie esterne apportano un gran numero di vulnerabilità all'applicativo finale.

6.3.1 Sonarqube report

L'analisi con Sonarqube mostra come l'applicazione così come è stata scritta non presenta alcuna vulnerabilità nel codice. Gli unici elementi individuati sono i cosiddetti "code smells" ovvero delle caratteristiche all'interno del codice sorgente che possono indicare dei difetti di programmazione. Tra questi difetti si possono trovare codice duplicato, metodi troppo lunghi o porzioni di codice non utilizzato.

Figura 6.4: Risultati dell'analisi con Sonarqube

6.3.2 OWASP Dependency Check report

L'analisi di OWASP Dependency Check mostra una serie di vulnerabilità di tipo High o Critical all'interno di alcune dipendenze presenti nell'applicazione. Più precisamente sono state individuate 5 vulnerabilità di tipo Critical e 5 vulnerabilità di tipo High. Le vulnerabilità medium o low non impediscono il superamento del quality gate. Tuttavia quelle critical o high bloccano la pipeline fino a risoluzione del problema.

Il grafico in figura 6.5 mostra il quantitativo di vulnerabilità individuate.

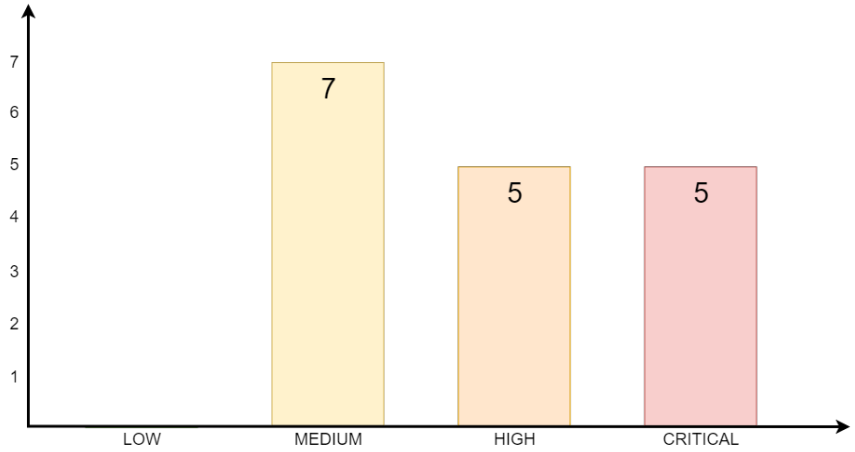

Figura 6.5: Risultati dell'analisi con OWASP Dependency Check

6.3.3 Trivy Report

Trivy presenta due report diversi, uno relativo all'analisi del Dockerfile. Nel Dockerfile era stata inserita volutamente una cattiva norma di programmazione, non specificando l'utente a livello user. Come specificato nella sezione 3.2.3 questo comportamemto porta alla definizione di un utente root di default comportando un rischio per la sicurezza. Di fatto la vulnerabilità individuata era di tipo High.

Nei due esempi di codice che seguono viene mostrata la differenza tra il Dockerfile con vulnerabilità, privo di comando per essere eseguito con privilegi utente, e il Dockerfile privo di vulnerabilità.

```
1 FROM openjdk:11
2 ARG JAR_FILE=target/*.jar
3 COPY ${JAR_FILE} app.jar
4 EXPOSE 8080
5 ENTRYPOINT ["java","-jar","/app.jar"]
```

Listato 6.14: Dockerfile con vulnerabilità

```
1 FROM openjdk:11
2 RUN useradd myuser
3 USER myuser
4 ARG JAR_FILE=target/*.jar
5 COPY ${JAR_FILE} app.jar
6 EXPOSE 8080
7 ENTRYPOINT ["java","-jar","/app.jar"]
```

Listato 6.15: Dockerfile senza vulnerabilità

Nel secondo report di Trivy (in figura 6.6) viene mostrato il numero di vulnerabilità individuate nell'immagine docker appena creata. Ovviamente anche in questo caso il quality gate non viene superato per via del numero di vulnerabilità Critical o High individuate.

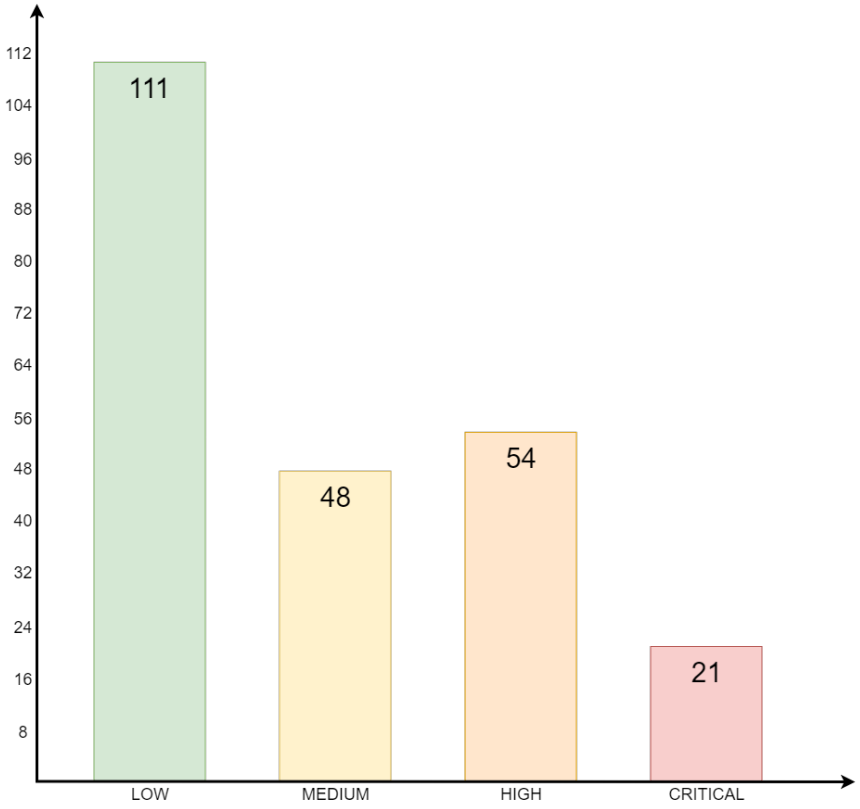

Figura 6.6: Risultati dell'analisi con Trivy

In figura 6.7 viene mostrata l'output della pipeline Jenkins nel caso particolare di non superamento del quality gate relativo alla scansione del Dockerfile da parte di Trivy. Nella figura 6.8 invece viene mostrato l'aspetto della Pipeline nel caso tutti gli stage avessero successo.

Git Clone	Build Application	SonarQube analysis	SonarQube Quality Gates	OWASP Dependency Check	Dependency Check quality gate	Docker image build	Check Dockerfile with Trivy	Trivy Dockerfile scan quality gate	Check Docker image with Trivy	Trivy quality gate	docker image push
2s	39s	28s	690ms	23s	355ms	3s	1s	618ms	24s	581ms	135ms
1s	37s	27s	650ms	28s	394ms	3s	775ms	930ms failed	75ms failed	74ms failed	101ms failed

Figura 6.7: Pipeline interrotta a causa di un quality gate non superato

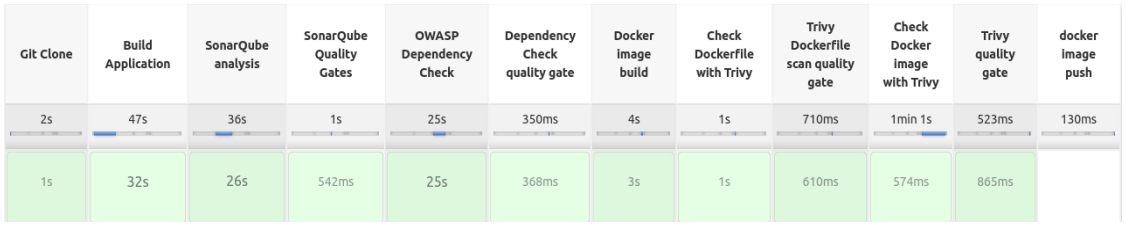

Figura 6.8: Pipeline con tutti gli stage superati

Capitolo 7

Possibili sviluppi futuri

Il lavoro svolto fino ad adesso, è adatto a tutta una serie di possibili integrazioni. Essendo questa una tesi di tipo sperimentale, queste possibili integrazioni non sono state inserite negli obiettivi predefiniti per questo lavoro di tesi. Tuttavia avendo avuto modo di entrare a conoscenza di determinate tematiche in ambito lavorativo, si è pensato di poter aggiungere una breve trattazione su di queste.

7.1 Runtime Security

Tra queste tematiche si può senza dubbio citare la sicurezza a runtime. Nel lavoro svolto finora, gli unici controlli che venivano eseguiti a runtime sono stati controlli su metriche di utilizzo dell'infrastruttura cloud, ad esempio utilizzo della cpu, utilizzo della rete, utilizzo della memoria ram e simili. Si è provato ad integrare l'utilizzo di un tool opensource chiamato Falco, distribuito da Sysdig[1]. Essendo un tool opensource e il progetto realizzato relativamente semplice e solo a scopo dimostrativo, è stato complesso riuscire a trovare una compatibilità tra Amazon e Falco. Le possibili soluzioni sarebbero state utilizzare la versione enterprise del tool, ovviamente non praticabile, oppure provare a realizzare un ambiente Kubernetes locale all'interno del cluster ECS. Tuttavia questa soluzione avrebbe rappresentato un cluster dentro un cluster e non sarebe stata aderente ad caso d'uso reale, pertanto si è scelto di non percorrerla. Sarebbe stata una semplice dimostrazione del funzionamento di Falco che non rientrava negli obiettivi di questa tesi.

7.2 Security As Code

In questa sezione invece verrà presentata la tematica nota come Security as Code[2]. Con Security as Code si intende quell'insieme di pratiche volte a formalizzare specifiche di sicurezza attraverso un linguaggio dichiarativo. La Security as Code viene vista come una

[1] https://sysdig.com/blog/intro-runtime-security-falco/

[2] SaC

delle soluzioni più efficienti ed efficaci per apportare sicurezza in un ambiente cloud. i principali benefit che la SaC apporta sono tre:

- **Velocità:** per poter essere perfettamente integrati in un contesto cloud, i team devono poter operare in agilità e rapidità. L'intervento umano nell'applicare la sicurezza riduce di molto l'agilità e la velocità andando a creare attrito. Poter codificare la sicurezza velocizza di molto i processi.

- **Riduzione del rischio** i classici controlli di sicurezza utilizzati in sistemi on-premises semplicemente non riescono ad adeguarsi perfettamente alle varie sfumature del cloud; il cloud richiede un carico di lavoro in ambito di sicurezza su tutto il ciclo di vita e sviluppo. L'unico modo per ottenere sicurezza in questo caso è inserendola in un contesto di programmazione della stessa.

- **Abilitazione di business** La SaC permette di dare una seria attenzione a policy e conformità che stanno via via diventando tematiche sempre più centrali, andando però a ridurre i tempi necessari ad applicare questi controlli di sicurezza con la possibilità di andare a concentrarsi anche su innovazione e creatività.

In ambito lavorativo, la tematica della SaC è stata suddivisa in due sottotematiche principali, Infrastructure as Code Security e Policy as Code. Queste due tematiche si applicano molto bene al lavoro svolto fino ad ora, andando ad integrarsi perfettamente all'approccio DevSecOps. Per fornire un esempio si può pensare di eseguire il deploy dell'infrastruttura cloud attraverso file di configurazione e inserire questo step all'interno di una pipeline.

7.2.1 Infrastructure as Code Security

Prima di affrontare la tematica dell'Infrastructure as Code Security è bene spiegare cosa si intende in primis con Infrastructure as Code. Con Infrastructure as code si intende quella pratica di definire in maniera descrittiva requisiti infrastrutturali quali reti, macchine virtuali, load balancer e connessioni, in modo da poter fare il rilascio dell'infrastruttura in maniera automatizzata e oltrettutto standard. Di fatto con questa pratica si può standardizzare il rilascio di infrastrutture cloud avendo dei file di configurazione. Questo permette oltretutto di abbassare il rischio di creare delle configurazioni non sicure. A supporto della sicurezza di questa pratica di fatto sono stati sviluppati strumenti in grado di fare analisi dei file di configurazione per poter individuare in maniera preventiva. In questo modo si può intervenire sul problema prima ancora di rilasciare l'infrastruttura e fare l'apply dei file IaC. Di fatto questa pratica non è per nulla dissimile a quella utilizzata nel progetto realizzato nello stadio di analisi del Dockerfile. La misconfiguration presente nel file non permetteva il superamento del quality gate permettendo di sistemare il file e generare l'immagine solo una volta privo di vulnerabilità.

7.2.2 Policy as Code

Nel deploy di una infrastruttura cloud, è necessario prestare attenzione a una serie di elementi quali la conformità e l'aderenza alle regolamentazioni internazionali. Quando si ha a che fare con dati sensibili, dati personali è fondamentale aderire e rispettare tutte

le regolamentazioni quali possono essere ad esempio il GDPR[3] per il trattamento dati in Europa, oppure il PCI DSS[4] per i sistemi di pagamento online. Dover prestare attenzione al rispetto dei numerosi sistemi di regolamentazione, in maniera manuale, può risultare un lavoro oneroso e senza dubbio incline all'errore. Con la Policy as Code l'obiettivo è quello di scrivere in un linguaggio dichiarativo regole di sicurezza, criteri e condizioni da rispettare. Questo permette di automatizzare e standardizzare le pratiche di sicurezza rendendole di fatto molto meno soggette all'errore umano. Con la Policy as Code e con i vari strumenti forniti dalle aziende possono essere utilizzati diversi tipi di approccio. Nei software per la PaC rilasciati spesso sono presenti regole predefinite e aderenti ai principali standard. Tuttavia è comunque, quasi sempre, possibile realizzare le proprie regole per poter adattare le infrastrutture realizzate ai propri standard e alle regole definite dalla propria azienda.

[3]General Data Protecion Regulation

[4]Payment Card Industry Data Security Standard

Capitolo 8

Conclusioni

Lo scopo di questo lavoro di tesi è stato fin da subito poter affrontare la tematica di sicurezza in cloud esplorando in maniera orizzontale quante più tematiche possibili per poter infine realizzare un framework che definisse un approccio quanto più standard possibile per lo sviluppo di un applicativo in cloud.

L'approccio si è quindi concentrato su una ricerca iniziale degli elementi ad oggi noti che caratterizzassero lo sviluppo in cloud. Le tematiche individuate sono state raggruppate in quattro filoni principali, che si è scelto di definire come quattro pilastri della sicurezza in cloud. Questi quattro pilastri sono la sicurezza dell'applicazione, la sicurezza dei dati, la sicurezza dei contaner, e la gestione dell'ambiente e infrastruttura in cloud. Si è scelto quindi di non focalizzarsi su uno di questi pilastri in particolare bensì provare a trattarli tutti, in maniera meno approfondita di quella che sarebbe stata una trattazione verticale, ma comunque sufficiente a poter realizzare il framework.

Affrontata la trattazione ci si è spostati sulla definizione del framework. Si è scelto di inserire lo sviluppo nella meccanica definita DevSecOps che fa della sicurezza una parte integrante dello sviluppo, in ogni sua fase. Si è pertanto voluto inserire almeno uno strumento di sicurezza per ogni pilastro individuato. Per alcuni è stato più semplice indivuduarne, per altri un po' meno. È il caso ad esempio della sicurezza dei dati dove essendo la gestione in mano al cloud service provider ci si è limitati a raccogliere dei log, ponendo attenzione al mascheramento di dati potenzialmente sensibili.

Essendo però l'idea di base quella di voler fornire una metodologia d'approccio e non trattazione approfondita su ciascun pilastro, anche se con alcune trattazioni più superficiali di altre, il lavoro svolto resta comunque una valida dimostrazione di come ci si possa concentrare sulla sicurezza in ogni fase dello sviluppo.

Il framework realizzato vuole porsi come una base d'appoggio per gli sviluppi di applicazioni in cloud. Può essere facilmente modificabile e adattabile ai propri obiettivi e requisiti. Ha dimostrato inoltre di essere un'ottima soluzione per poter individuare eventuali vulnerabilità anche in casi molto semplificati, ponendo attenzione sul fatto che una integrazione della sicurezza in ogni fase del ciclo di sviluppo non è solo consigliabile, ma deve essere considerata come una pratica obbligata.

Bibliografia

[1] Render 2022. *Using Secrets with Docker*. https://render.com/docs/docker-secrets#secret-files-in-docker-builds.

[2] Creative Commons Attribution-ShareAlike 4.0. *OWASP Dependency Check*. https://plugins.jenkins.io/dependency-check-jenkins-plugin/.

[3] *Amazon EC2*. https://aws.amazon.com/it/ec2/.

[4] AppsDeveloperBlog. https://www.appsdeveloperblog.com/spring-security-preauthorize-annotation-example/.

[5] Microsoft Azure. *What is infrastructure as code (IaC)?* https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.

[6] Vivek Balasubramaniam. *Defining JPA Entities*. https://www.baeldung.com/jpa-entities.

[7] Thanh Bui. *Analysis of Docker Security*. 2015. DOI: 10.48550/ARXIV.1501.02967. URL: https://arxiv.org/abs/1501.02967.

[8] Victor Jimenez Cerrada. *Getting started with runtime security and falco*. https://sysdig.com/blog/intro-runtime-security-falco/.

[9] Google Cloud. *Cosa sono i container?* https://cloud.google.com/learn/what-are-containers?hl=it.

[10] *Cloud Computing Stats - Security and Recovery*. https://www.slideshare.net/rapidscale/cloud-computing-stats-security-and-recovery.

[11] *Dell Study Reveals Companies Investing in Cloud, Mobility, Security and Big Data Are Growing More Than 50 Percent Faster Than Laggards*. https://www.businesswire.com/news/home/20151013005365/en.

[12] *Digital Transformation Market Size, Share & Trends Analysis Report By Solution (Analytics, Cloud Computing, Social Media, Mobility), By Service, By Deployment, By Enterprise, By End Use, By Region, And Segment Forecasts, 2022 - 2030*. https://www.grandviewresearch.com/industry-analysis/digital-transformation-market.

[13] Spring Framework. «Spring framework». In: *Available on:< https://spring. io/>. Access in* 3 (2018).

[14] Marco Rizzi Gaia Rizzato. *Garantire l'accesso in sicurezza ai dati aziendali: quali misure tecniche e organizzative implementare?* https://www.cybersecurity360.it/esperto-risponde/garantire-laccesso-in-sicurezza-ai-dati-aziendali-quali-misure-tecniche-e-organizzative-implementare/.

[15] *Global cloud services spend hits US$55.9 billion in Q1 2022*. https://www.canalys.com/newsroom/global-cloud-services-Q1-2022.

[16] Eric Goebelbecker. *A Guide to LogBack*. https://www.baeldung.com/logback.

[17] Adam Gordon. *The Official (ISC)2 Guide to the CCSP CBK*. Second Edition. Sybex, 2016.

[18] Umme Habiba et al. «Cloud identity management security issues & solutions: a taxonomy». In: *Complex Adaptive Systems Modeling* 2.1 (2014), pp. 1–37.

[19] Dick Hardt. *The OAuth 2.0 authorization framework*. Rapp. tecn. 2012.

[20] Red Hat. *Cosa significa DevSecOps?* https://www.redhat.com/it/topics/devops/what-is-devsecops.

[21] Cyral Inc. *What is Security as Code?* https://cyral.com/white-papers/what-is-security-as-code/.

[22] Docker Inc. *Use containers to build, share and run your applications*. https://www.docker.com/resources/what-container/.

[23] I. Indu, P.M. Rubesh Anand e Vidhyacharan Bhaskar. «Identity and access management in cloud environment: Mechanisms and challenges». In: *Engineering Science and Technology, an International Journal* 21.4 (2018), pp. 574–588. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2018.05.010. URL: https://www.sciencedirect.com/science/article/pii/S2215098617316750.

[24] Srinivas for InfoSec. *Common container misconfigurations and how to prevent them*. https://resources.infosecinstitute.com/topic/common-container-misconfigurations-and-how-to-prevent-them/.

[25] Cybersecurity Insider. *Cloud Security Report 2022*. https://pages.checkpoint.com/2022-cloud-security-report.html.

[26] Michael Jones, John Bradley e Nat Sakimura. *Json web token (jwt)*. Rapp. tecn. 2015.

[27] Karen Kent e Murugiah Souppaya. *Guide to computer security log management*. Createspace Independent Publishing Platform, mar. 2012.

[28] Kheenvraj Lomror. *Sonarqube: what is and why use it?* https://blog.loginradius.com/engineering/sonarqube/.

[29] Aqua Security Software Ltd. https://www.aquasec.com/products/trivy/#block_5.

[30] Peter Mell e Timothy Grance. *The NIST Definition of Cloud Computing*. en. 2011-09-28 2011. DOI: https://doi.org/10.6028/NIST.SP.800-145.

[31] *Nozioni di base su Amazon EC2*. https://aws.amazon.com/it/ec2/getting-started/.

[32] Auth0 by Okta. *Introduction to JSON Web Tokens*. https://jwt.io/introduction.

[33] Auth0 by Okta. *Spring Boot Authorization Tutorial: Secure an API (Java)*. https://auth0.com/blog/spring-boot-authorization-tutorial-secure-an-api-java/.

[34] *OWASP Dependency Check*. https://owasp.org/www-project-dependency-check/.

[35] *OWASP Top 10 - 2021*. https://owasp.org/www-project-top-ten/.

[36] *OWASP Web Security Testing Guide*. https://owasp.org/www-project-web-security-testing-guide/.

[37] Andres Pihlak. «CONTINUOUS DOCKER IMAGE ANALYSIS AND INTRUSION DETECTION BASED ON OPEN-SOURCE TOOLS». In: (2020).

[38] QOS.ch Sarl. https://logback.qos.ch/.

[39] *Security as code: The best (and maybe only) path to securing cloud applications and systems*. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/security-as-code-the-best-and-maybe-only-path-to-securing-cloud-applications-and-systems.

[40] Amazon Web Services. *Amazon Elastic Container Registry (Amazon ECR)*. https://aws.amazon.com/it/ecr/.

[41] Amazon Web Services. *Amazon Elastic Container Service (Amazon ECS)*. https://aws.amazon.com/it/ecs/.

[42] Amazon Web Services. *Istanze e AMI*. https://docs.aws.amazon.com/it_it/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html.

[43] Amazon Web Services. *tipi di istanze Amazon EC2*. https://aws.amazon.com/it/ec2/instance-types/.

[44] Amazon Web Services. *What is Amazon Elastic Container Registry?* https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html.

[45] Amazon Web Services. *What is Amazon Elastic Container Service?* https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html.

[46] Amazon Web Services. *What is Amazon VPC?* https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

[47] Amazon Web Services. *What Is CloudWatch*. https://docs.aws.amazon.com/it_it/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

[48] Switzerland SonarSource S.A. *Install the Server*. https://docs.sonarqube.org/latest/setup/install-server/.

[49] National Institute of Standard e Technology. https://nvd.nist.gov/general.

[50] Sari Sultan, Imtiaz Ahmad e Tassos Dimitriou. «Container Security: Issues, Challenges, and the Road Ahead». In: *IEEE Access* 7 (2019), pp. 52976–52996. DOI: 10.1109/ACCESS.2019.2911732.

[51] Dmytro Vedetskyi. *Jenkins integration with SonarQube*. https://medium.com/devoops-and-universe/jenkins-integration-with-sonarqube-2e9fb3d18919.

[52] Spring by VMware Tanzu. https://spring.io/projects/spring-boot.

[53] Spring by VMware Tanzu. https://spring.io/projects/spring-security.

[54] Stephen Watts. *What is Security as Code?* https://www.bmc.com/blogs/security-as-code/.

[55] *What is AWS.* https://aws.amazon.com/it/what-is-aws/.

[56] *What is Java Spring Boot.* https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-java-spring-boot/.

[57] *What is Jenkins Pipeline?* https://www.jenkins.io/doc/book/pipeline/.

[58] Wikipedia. *Code smell — Wikipedia, L'enciclopedia libera.* 2022. URL: %5Curl%7Bhttps://it.wikipedia.org/wiki/Code_smell%7D.

[59] Terence Wong. *Building a Docker image in Jenkinsfile and publishing to ECR.* https://octopus.com/blog/jenkins-docker-ecr#building-a-docker-image-in-jenkinsfile-and-publishing-to-ecr.