POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Un approccio automatizzato al Threat Modeling

Relatore
Prof. Fulvio Valenza
Dr. Daniele Bringhenti

Candidato
Antonio Centola

Tutor Aziendali Liquid Reply Srl
Dr. Ivan Aimale
Dr. Marco Oria
Dr.ssa Luisa Gatto

Anno Accademico 2022-2023

Sommario

Il Cloud Computing ha rivoluzionato il panorama dell'industria IT, ridefinendo le modalità di progettazione, implementazione e gestione delle infrastrutture. La promessa di scalabilità e flessibilità ha spinto le organizzazioni verso il cloud, creando architetture complesse e distribuite. Questa trasformazione, sebbene vantaggiosa, ha portato nuove sfide, soprattutto in termini di sicurezza, poichè ha reso la protezione dei dati e la gestione delle minacce più complesse. Il threat modeling è una risposta a questa sfida. Esso è emerso come approccio fondamentale per valutare e mitigare i rischi legati alla sicurezza nelle architetture cloud. Questo processo permette di identificare, valutare e gestire le potenziali minacce che potrebbero mettere a repentaglio l'integrità, la disponibilità e la riservatezza dei dati. Tuttavia, il threat modeling tradizionalmente richiede un'analisi dettagliata di ogni componente dell'architettura, un processo laborioso, intensivo in termini di tempo e potenzialmente soggetto a errori umani. La sua automazione si presenta come una soluzione efficace per semplificarlo e migliorarlo. Ad esempio, una schedulazione del processo di threat modeling permetterebbe di effettuare analisi più frequenti e di ottenere i risultati in maniera tempestiva rispetto ai metodi tradizionali. In questo modo, le organizzazioni possono valutare rapidamente le minacce o vulnerabilità emerse. Una delle soluzioni per automatizzare il threat modeling è l'utilizzo di Jenkins, una piattaforma ampiamente adottata nel campo dello sviluppo software e dell'implementazione continua (CI/CD). La tesi offre un esempio concreto di come sia possibile utilizzare un approccio automatizzato al threat modeling, utilizzando una pipeline Jenkins. Il lavoro si articola in diverse fasi. La prima riguarda la progettazione di un'architettura di riferimento, basata su infrastruttura cloud, microservizi containerizzati, orchestrazione dei container e distribuzione automatizzata tramite Jenkins. La seconda fase si concentra sull'analisi delle minacce, il cui obiettivo è individuarne le potenziali per ciascun componente dell'ambiente di riferimento. La terza fase si occupa dell'automazione vera e propria, andando ad identificare i controlli automatizzabili e configurando una pipeline per l'esecuzione

regolare del flusso di analisi. Quest'ultima è composta da vari stage, ognuno dei quali corrisponde a un controllo specifico che è stato automatizzato, impiegando strumenti d'analisi esistenti o script creati ad hoc. I risultati di ciascun controllo vengono raccolti in un report dettagliato che evidenzia le vulnerabilità o le potenziali minacce. Infine, l'ultima parte del lavoro di tesi si concentra sull'esecuzione della pipeline d'automazione e sulla validazione dei risultati. La validazione dei risultati va a verificare che quanto emerso dal report della pipeline sia coerente con i risultati ottenuti dall'analisi manuale. In conclusione, questo lavoro rappresenta un primo passo verso l'automazione del threat modeling nelle architetture cloud, evidenziando l'utilizzo di Jenkins come strumento chiave in questo processo. È importante sottolineare che ciò non costituisce una soluzione definitiva, bensì un punto di partenza per migliorare il processo di sicurezza in un ambiente cloud.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Contesto

Il Cloud Computing ha rivoluzionato il panorama tecnologico, guadagnandosi un'ampia adozione in organizzazioni di diverse dimensioni e settori. Questo modello, noto per la sua agilità, scalabilità ed efficienza economica, è stato abbracciato in tutto il mondo. La sua diffusione su larga scala può essere attribuita principalmente a benefici economici significativi, eliminando gli onerosi investimenti in infrastrutture hardware e consentendo un controllo più flessibile dei costi operativi. Un altro dei fattori che stanno contribuendo in modo significativo a questa diffusione è la possibilità di accedere e collaborare in modo flessibile da qualsiasi luogo, rendendo il tutto molto più agevole e flessibile. La capacità di adattare rapidamente le risorse alle mutevoli esigenze aziendali è un altro vantaggio chiave, soprattutto in un'era in cui i picchi di traffico e di elaborazione possono variare notevolmente.
Tuttavia, insieme a questi benefici, il Cloud Computing ha anche introdotto sfide significative, con una crescente attenzione sulla sicurezza a causa dei dati e delle applicazioni ospitati in ambienti cloud. La complessità delle infrastrutture cloud e la comunicazione tra risorse che potrebbero avere vulnerabilità intrinseche hanno sollevato preoccupazioni. Per affrontare queste sfide, è imperativo condurre un'analisi dettagliata delle minacce e delle vulnerabilità e sviluppare strategie di mitigazione per proteggere in modo efficace i dati e le risorse aziendali.

1.2 Struttura del documento

La seguente tesi è strutturata come segue:

- **Capitolo 2**: Introduce il lettore alle fondamenta teoriche necessarie per comprendere gli argomenti trattati nel documento;
- **Capitolo 3**: Fornisce quelli che sono gli obiettivi pratici del lavoro di tesi;
- **Capitolo 4**: Contiene una descrizione delle tecnologie impiegate e della configurazione iniziale dei componenti utilizzati per la messa a punto dell'architettura;
- **Capitolo 5**: Descrive il processo di Threat Modeling sull'architettura di riferimento e la sua automatizzazione;
- **Capitolo 6**: Presenta i risultati ottenuti dall'analisi automatizzata e la loro validazione;
- **Capitolo 7**: Conclude la tesi con una sintesi delle scoperte e suggerimenti per potenziali ricerche future;

Capitolo 2

Fondamenti Teorici

In questo capitolo vengono esplorati i fondamenti teorici che costituiscono la base della tesi, offrendo al lettore le conoscenze necessarie per una comprensione approfondita del lavoro svolto e delle sue motivazioni.
Si inizia con un'analisi del cloud computing, un paradigma che ha trasformato radicalmente il settore dell'IT. Vengono esaminati i fattori chiave del suo trionfo e, in parallelo, si indagano i rischi associati alla sua rapida evoluzione. Questo approccio conduce a riconoscere l'importanza fondamentale della sicurezza in questo ambito in evoluzione, dove le minacce sono in costante cambiamento e la protezione dei dati diventa una sfida sempre più complessa.
In risposta a queste sfide, il capitolo si concentra su uno strumento essenziale per la gestione della sicurezza: il threat modeling. Questo processo, vitale per identificare e attenuare i rischi di sicurezza in ambienti cloud complessi, è esplorato attraverso varie metodologie e applicazioni pratiche.
L'obiettivo del capitolo è fornire una comprensione solida di questi concetti chiave, preparando il terreno per le discussioni più tecniche e dettagliate nei capitoli successivi.

2.1 Cloud computing

2.1.1 Introduzione al Cloud Computing

Il National Institute of Standards and Technology (NIST) definisce il cloud computing come *"un modello che consente un accesso di rete comodo e su richiesta a un pool condiviso di risorse informatiche configurabili (ad esempio, rete, server, storage, applicazioni e servizi), che possono essere rapidamente messe a disposizione e rilasciate con un minimo sforzo di gestione, o di interazione con il fornitore di servizi"* [1].
Questo modello di calcolo, che ha rivoluzionato il settore IT, si è diffuso rapidamente grazie alla sua capacità di fornire risorse scalabili e flessibili. Nel corso degli ultimi anni, la sua popolarità è cresciuta in modo esponenziale, e questa tendenza sembra destinata a continuare. Secondo Statista, il fatturato nel mercato del Cloud pubblico dovrebbe raggiungere 525,601 miliardi di dollari nel 2023, espandendosi a 881,80 miliardi di dollari entro il 2027, con un tasso di crescita annuo composto del 13,81% dal 2023 al 2027 [2].

Le cinque caratteristiche essenziali del cloud computing, come identificate dal NIST [1], includono:

- **Servizio self-service su richiesta**: Le capacità di calcolo, come potenza di elaborazione e capacità di storage, possono essere automaticamente allocate ai consumatori, permettendo una gestione agile e flessibile delle risorse.
- **Accesso in rete diffuso**: Utilizzando standard di rete comuni, le risorse cloud sono accessibili attraverso varie piattaforme client, come desktop, laptop e dispositivi mobili.
- **Condivisione di risorse**:In un modello multi-tenant, le risorse sono condivise tra più utenti, con assegnazione e ri-assegnazione dinamica in base alla domanda. Questo solleva importanti considerazioni sulla sicurezza dei dati condivisi e sulla privacy.
- **Elasticità rapida**: Le risorse possono essere rapidamente scalate su o giù, offrendo alle aziende una flessibilità senza precedenti per rispondere a mutevoli esigenze operative.

- **Servizio misurato**: Il monitoraggio, il controllo e la segnalazione dell'utilizzo delle risorse assicurano trasparenza sia per il fornitore che per il consumatore del servizio.

2.1.2 Modelli di Servizio

Il NIST [1] identifica anche tre modelli di servizio fondamentali:

- **Software as a Service**: In questo modello, le applicazioni software sono ospitate su infrastrutture cloud e accessibili via Internet. Gli utenti possono utilizzare software come servizi di posta elettronica basati su cloud senza doverli installare sui propri dispositivi, eliminando la necessità di gestione e manutenzione locale.
- **Platform as a Service**: Qui, i consumatori possono distribuire applicazioni personalizzate su infrastrutture cloud. Questo modello offre una significativa flessibilità e controllo sull'ambiente di hosting delle applicazioni, liberando gli sviluppatori dalla gestione dell'infrastruttura sottostante.
- **Infrastructure as a Service**: In questo modello, i consumatori hanno accesso a risorse informatiche fondamentali come potenza di calcolo, archiviazione e reti. Il controllo esteso su sistemi operativi, storage e applicazioni distribuite rende l'IaaS ideale per costruire e gestire infrastrutture virtuali personalizzabili.

Ogni modello di servizio presenta specifiche implicazioni e sfide di sicurezza, come la gestione di identità e accessi in SaaS, la protezione dell'ambiente di runtime in PaaS e la sicurezza delle infrastrutture virtuali in IaaS.
Nella figura sottostante è possibile vedere un'illustrazione grafica dei modelli di servizio cloud:

Si noti che in ciascun modello di servizio, il consumatore non gestisce o controlla l'infrastruttura cloud sottostante. È il provider di cloud l'entità responsabile di gestire l'infrastruttura, garantendo l'affidabilità e l'efficienza delle risorse offerte. Questa divisione di responsabilità è un pilastro fondamentale del cloud computing e consente ai consumatori di concentrarsi sullo sviluppo e l'utilizzo delle proprie applicazioni, senza doversi preoccupare della complessità sottostante dell'infrastruttura.

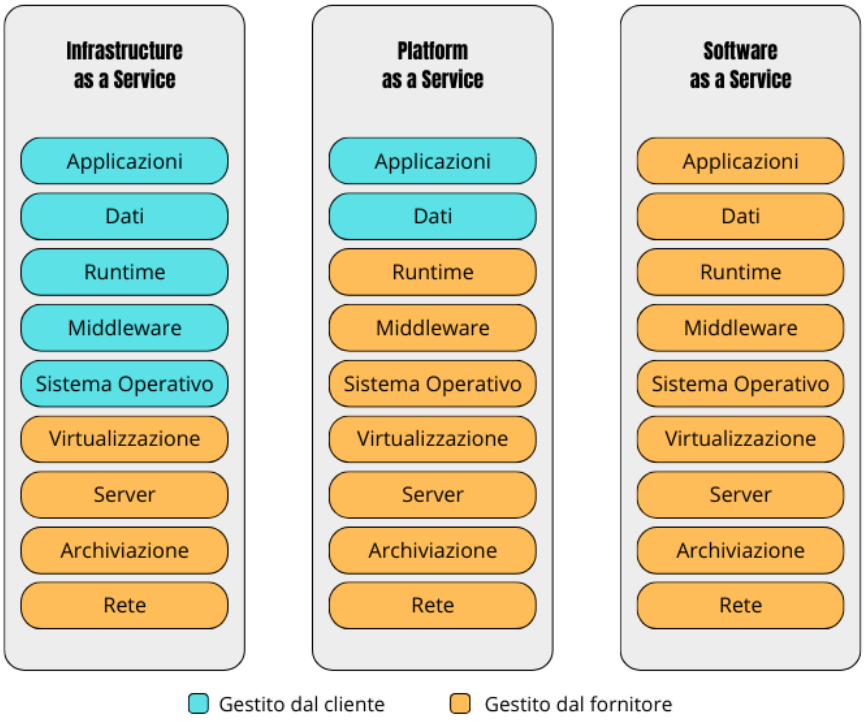


Figura 2.1. Modelli di Cloud

2.1.3 Modelli di Distribuzione

Secondo il NIST [1], esistono quattro modelli di distribuzione principali, ognuno con caratteristiche e applicazioni distintive:

- **Cloud privato**: Un'infrastruttura cloud di tipo privato è progettata per l'uso esclusivo di una singola organizzazione. Questo modello può essere implementato all'interno dell'azienda o esternamente e può essere gestito direttamente dall'organizzazione stessa, da un fornitore terzo o da una combinazione di entrambi. Questa esclusività offre un alto grado di controllo sulla sicurezza e sulla gestione dei dati, rendendolo ideale per aziende che richiedono standard elevati di sicurezza e privacy.

- **Cloud comunitario**: Destinato a comunità con interessi comuni, questo modello consente la condivisione di risorse tra organizzazioni, mantenendo un certo grado di controllo.

- **Cloud pubblico**: Gestito da un fornitore esterno, è accessibile a chiunque su Internet, offrendo alta scalabilità e convenienza, ma con controllo limitato per l'utente.

- **Cloud ibrido**: Combina diversi modelli di cloud, mantenendo l'autonomia di ciascuno ma permettendo la portabilità dei dati e delle applicazioni.

La scelta del modello di distribuzione appropriato dipende dagli obiettivi specifici dell'organizzazione, bilanciando fattori come accessibilità, controllo, costi e scalabilità.

2.2 I vantaggi del Cloud

I vantaggi del cloud computing sono numerosi e possono avere un impatto significativo su aziende e individui:

1. **Accessibilità e Mobilità**: Il cloud consente l'accesso a dati e applicazioni da qualsiasi luogo con una connessione a Internet, supportando il lavoro a distanza e migliorando la collaborazione tra team distribuiti geograficamente.

2. **Scalabilità**: Le risorse nel cloud possono essere facilmente aumentate o ridotte, adattandosi rapidamente alle esigenze aziendali e eliminando la necessità di costosi investimenti in hardware.

3. **Riduzione dei Costi**: Adottare il cloud significa ridurre i costi di acquisto e manutenzione hardware. I modelli di pagamento basati sull'uso offrono ulteriori risparmi pagando solo per le risorse utilizzate.

4. **Backup e Recupero dei Dati**: Il cloud offre metodi affidabili per il backup e il recupero dei dati, proteggendo le aziende da perdite di dati dovute a guasti hardware o disastri.

5. **Aggiornamenti Automatici**: I servizi cloud vengono costantemente aggiornati, fornendo agli utenti le ultime innovazioni e miglioramenti di sicurezza senza interventi manuali.

6. **Collaborazione Migliorata**: Il cloud stimola la produttività e il flusso di lavoro migliorando la collaborazione in tempo reale tra i membri del team.

7. **Sicurezza**: I provider cloud offrono livelli elevati di sicurezza, inclusa la crittografia dei dati e la protezione contro le minacce informatiche, spesso superiori alle soluzioni on-premises.

8. **Rispetto dell'Ambiente**: L'uso di risorse condivise nel cloud aiuta a diminuire l'impronta di carbonio e promuove pratiche di lavoro sostenibili.

9. **Gestione Semplificata**: La gestione centralizzata delle risorse IT nel cloud riduce il carico di lavoro amministrativo e semplifica la supervisione dell'infrastruttura tecnologica.

10. **Innovazione**: Il cloud dà accesso a tecnologie avanzate come l'intelligenza artificiale e l'analisi dei dati, consentendo alle aziende di rimanere competitive e di innovare.

2.3 Minacce in ambiente Cloud

Come i tradizionali sistemi IT, anche il Cloud Computing non è esente da minacce, vulnerabilità e attacchi alla sicurezza. Le caratteristiche peculiari del cloud, come il pooling di risorse distribuite e l'ampio accesso alla rete, benché offrano vantaggi significativi in termini di scalabilità e efficienza, possono anche aprir la strada a rischi specifici. Questi rischi sono aggravati dal fatto che i dati e le applicazioni sono spesso gestiti e immagazzinati su server remoti, rendendo la sicurezza nel cloud un aspetto cruciale. Di conseguenza, nonostante la crescente popolarità e l'adozione del cloud in diversi settori, l'attenzione e la preoccupazione per la sua sicurezza sono aumentate significativamente negli ultimi anni. Ricerche recenti, come quelle condotte da A. Hendre e K. P. Joshi [3], hanno messo in luce una serie di minacce specifiche all'ambiente cloud:

- **Violazioni dei dati**: Questo fenomeno avviene quando informazioni sensibili, normalmente protette, vengono esposte a soggetti non autorizzati. Ciò può accadere a causa di aperture nella sicurezza informatica, attacchi di phishing, o errori umani. Le conseguenze includono la perdita di fiducia dei clienti, danni alla reputazione dell'organizzazione, e potenziali sanzioni legali.
- **Perdita di dati**: La perdita di dati può essere devastante per un'organizzazione. Può derivare da guasti hardware, come danni ai dischi rigidi, o da attacchi software, come ransomware che cripta i dati rendendoli inaccessibili. Questo può causare interruzioni operative significative e perdita di informazioni cruciali.
- **Compromissione di Account e Traffico**: Gli attaccanti possono rubare le credenziali di accesso degli utenti (come nomi utente e password) e utilizzarle per accedere a sistemi sensibili. Questo può portare al furto di identità, transazioni finanziarie fraudolente, e intercettazione di dati sensibili durante la trasmissione.
- **Interfacce e API insicure**: Le API (Application Programming Interfaces) e le interfacce utente che non sono adeguatamente protette possono essere sfruttate dagli hacker. Questo può portare a vari attacchi, inclusi quelli che compromettono i dati, modificano le operazioni del servizio, o permettono l'accesso non autorizzato.
- **Attacchi di Denial-of-Service (DoS)**: In questi attacchi, i servizi di un'organizzazione vengono sovraccaricati con richieste fittizie, rendendo impossibile l'accesso legittimo. Ciò può causare un'interruzione significativa delle operazioni e dei servizi online.
- **Rischi Interni all'Organizzazione**: Si riferisce al pericolo posto da dipendenti, collaboratori o partner che hanno accesso interno ai sistemi. Questi individui potrebbero abusare del loro accesso per danneggiare l'organizzazione o rubare dati sensibili.
- **Abuso dei servizi Cloud**: In un ambiente cloud condiviso (multitenancy), un attaccante potrebbe sfruttare vulnerabilità nella configurazione o nella separazione delle risorse per accedere o compromettere dati di altre organizzazioni ospitate sulla stessa piattaforma.

- **Negligenza nella Gestione del Cloud**: Questo problema emerge quando le organizzazioni trascurano di valutare i rischi e le minacce associate all'adozione del cloud, spesso attratte dal risparmio sui costi. Questa negligenza può portare a cattive configurazioni, vulnerabilità di sicurezza non gestite e mancanza di conformità.
- **Vulnerabilità della tecnologia condivisa**: In un ambiente cloud, diverse organizzazioni condividono la stessa infrastruttura fisica. Se una tecnologia multitenant non è adeguatamente progettata o gestita, può presentare vulnerabilità che gli hacker possono sfruttare per accedere a risorse e dati condivisi tra diversi clienti.

Nonostante alcune di queste minacce siano ben note nel contesto della sicurezza informatica generale, ci sono aspetti particolari come l'Abuso dei servizi Cloud, la Negligenza nella Gestione del Cloud e le Vulnerabilità della tecnologia condivisa che sono intrinsecamente legati all'ambiente cloud. Queste problematiche specifiche richiedono soluzioni e strategie di sicurezza mirate, che considerino la natura distribuita, scalabile e condivisa del cloud computing. È essenziale che le organizzazioni siano consapevoli di questi rischi unici e adottino misure proattive per la loro mitigazione, assicurando così che i benefici del cloud computing non siano oscurati dalle sue potenziali vulnerabilità. La sicurezza nel cloud è un campo in continua evoluzione, richiedendo un impegno costante per l'aggiornamento delle pratiche di sicurezza e per l'adattamento alle nuove sfide che emergono con l'evoluzione della tecnologia.

2.4 Strategie per la Gestione della Sicurezza nel Cloud

Dopo aver analizzato le principali sfide di sicurezza associate al cloud computing, è fondamentale comprendere come gestire questi situazioni efficacemente. La gestione dei rischi nel cloud non si limita a salvaguardare l'organizzazione da potenziali pericoli, ma permette anche di sfruttare le opportunità offerte dal cloud in modo sicuro.
La gestione dei rischi nel cloud richiede una comprensione approfondita dei rischi potenziali e delle misure di mitigazione efficaci. Le strategie di gestione

devono essere allineate con gli obiettivi aziendali e devono includere una valutazione completa dei rischi, la valutazione dei fornitori e un monitoraggio continuo.
Di seguito sono elencate alcune strategie chiave:

- **Valutazione e Analisi dei Rischi**: Prima di attuare misure di mitigazione, è necessario effettuare una valutazione approfondita dei rischi, identificando i potenziali pericoli, valutandone l'impatto e determinandone la probabilità di occorrenza.

- **Selezione del CSP (Cloud Service Provider)**: È cruciale scegliere il fornitore di servizi cloud giusto, valutando le sue capacità di sicurezza e l'affidabilità complessiva. Gli SLA devono includere disposizioni per la sicurezza, la privacy dei dati, il recupero dei disastri e la risposta agli incidenti.

- **Gestione delle Configurazioni e Sicurezza di Rete**: Prestare attenzione alla configurazione delle risorse cloud e alla sicurezza di rete, comprese le VPN, i firewall e le tecniche di segmentazione per ridurre la superficie di attacco.

- **Controllo dell'Accesso e Gestione delle Identità**: Implementare soluzioni di Identity and Access Management (IAM) che includano autenticazione multifattoriale, privilegi minimi e gestione degli accessi basata sui ruoli.

- **Sicurezza nella Condivisione dei Dati**: Adottare politiche e tecnologie per la sicurezza dei dati in transito e in riposo, utilizzando crittografia, tokenizzazione e altre tecniche per proteggere i dati condivisi.

- **Protezione delle API**: Assicurare che le API del cloud siano protette contro vulnerabilità comuni, come l'injection SQL e i problemi di sicurezza nelle configurazioni.

- **Sicurezza dei Servizi Web**: Proteggere i servizi web e le applicazioni cloud con meccanismi come Web Application Firewalls (WAF), scanning di vulnerabilità e patching regolare.

- **Gestione Proattiva dei Sistemi Software**: Mantenere aggiornati tutti i sistemi software con le ultime patch di sicurezza e monitorare attivamente per nuove vulnerabilità.

- **Formazione e Consapevolezza dei Dipendenti**: Sensibilizzare i dipendenti sui rischi di sicurezza nel cloud e formarli su pratiche di sicurezza informatica, come la gestione delle password e il riconoscimento di attacchi di phishing.

- **Piani di Continuità e Recupero Dati**: Sviluppare e testare regolarmente piani di continuità aziendale e di recupero dati per garantire la resilienza in caso di interruzioni o disastri.

- **Monitoraggio Continuo e Threat Intelligence**: Utilizzare strumenti di monitoraggio in tempo reale e servizi di threat intelligence per rilevare e rispondere rapidamente a eventuali minacce alla sicurezza.

Come è possibile intuire da questa panoramica sulla gestione dei rischi, una gestione efficace nel cloud prevede un processo dinamico, che richiede un approccio olistico e adattativo.
In questo contesto, il threat modeling diventa uno strumento fondamentale per identificare e mitigare i rischi in modo proattivo, assicurando che le strategie di sicurezza siano allineate con l'architettura cloud e le esigenze aziendali specifiche. Un threat modeling ben eseguito permette di comprendere meglio il panorama dei rischi e di implementare misure di sicurezza mirate, riducendo così la probabilità e l'impatto delle minacce nel cloud.

2.5 Threat Modeling

2.5.1 Introduzione al Threat Modeling

Il "threat modeling" è un processo fondamentale e complesso nel campo della sicurezza informatica, una disciplina in costante evoluzione che abbraccia una vasta gamma di metodologie, spesso sovrapposte. Per comprendere meglio il threat modeling, è essenziale definire chiaramente il concetto di "minaccia" e di "modello".

Una "minaccia" rappresenta ogni potenziale pericolo che può mettere a rischio un sistema, come la perdita di risorse, la divulgazione di informazioni riservate o interruzioni dei servizi. Queste minacce possono derivare sia da vulnerabilità intrinseche del sistema sia dalla capacità di un aggressore di sfruttare tali debolezze.
Dall'altra parte, il "modello" nel contesto del threat modeling si riferisce a un approccio strutturato e sistematico per rappresentare e analizzare queste minacce. Il modello offre un quadro per capire come le minacce interagiscano con il sistema e tra loro. Questo include la creazione di rappresentazioni, che possono variare da semplici diagrammi a modelli complessi con flussi di dati, relazioni tra componenti e potenziali punti di ingresso per gli attacchi. I modelli permettono agli analisti di simulare vari scenari di attacco e valutare l'efficacia delle contromisure esistenti, identificando anche aree che richiedono maggior protezione.
Il threat modeling mira a identificare, categorizzare e mitigare queste minacce. Guida gli analisti nella valutazione delle relazioni e interazioni tra le varie componenti di un sistema, andando oltre la semplice elencazione di vulnerabilità e vettori di attacco. Questo implica un'analisi approfondita dei possibili percorsi d'azione e delle strategie che un attaccante potrebbe utilizzare.
Il processo di threat modeling si articola in quattro fasi principali:

1. **Modellazione e Rappresentazione del Sistema**: Questo iniziale passaggio consiste nel delineare in dettaglio l'ambiente oggetto di analisi. Si identificano componenti chiave come applicazioni, reti, dati critici e altri elementi sensibili. L'obiettivo è di creare un quadro esauriente che illustri come le minacce possano insorgere, attraverso diagrammi di flusso dei dati e mappature delle interazioni sistemiche.

2. **Rilevamento delle Minacce**: Dopo aver modellato il sistema, l'attenzione si sposta sull'identificazione di potenziali minacce. Questa fase implica l'analisi dei tipi di attaccanti e delle loro strategie, esplorando vulnerabilità e possibili scenari di attacco. Si possono utilizzare checklist standardizzate o framework come STRIDE per una categorizzazione completa delle minacce.

3. **Messa a Punto delle Strategie di Mitigazione**: Qui si elaborano approcci per mitigare o eliminare i rischi rilevati. Le azioni possono variare dal rafforzamento delle difese sistemiche fino alla correzione di specifiche vulnerabilità e alla formazione degli utenti. Si sviluppa un piano d'azione chiaro per l'attuazione di queste misure.

4. **Controllo dell'Efficacia e Aggiornamento Continuo**: L'ultima fase prevede la verifica dell'efficacia delle contromisure adottate e la loro revisione periodica. Attraverso test di sicurezza e valutazioni, si assicura che le strategie di sicurezza siano adeguatamente efficaci e si aggiornano continuamente per fronteggiare nuove minacce e cambiamenti nel sistema.

Figura 2.2. Fasi del processo di Threat Modeling

L'approccio del threat modeling è cruciale sia nella fase iniziale di sviluppo di un prodotto, per identificare e mitigare proattivamente le vulnerabilità, sia nella fase operativa, per offrire diverse strategie di risposta in caso di attacchi. La sua efficacia dipende dalla dimensione e complessità del sistema,

dal contesto applicativo e dall'esperienza degli analisti coinvolti.
In definitiva, il threat modeling, profondamente legato al concetto di "risk assessment", si rivela una componente essenziale nella gestione della sicurezza informatica, aiutando le organizzazioni a comprendere e mitigare i rischi in un ambiente tecnologico in costante evoluzione.

2.5.2 Metodologie principali per il Threat Modeling

Ci sono diverse metodologie per applicare il processo di threat modeling, ciascuna con i suoi approcci e strumenti distintivi. L'articolo pubblicato dalla Carnegie Mellon University [4] offre un'analisi dettagliata di queste metodologie, le quali vengono descritte approfonditamente nei paragrafi successivi di questo capitolo.

STRIDE

La metodologia STRIDE è un framework strutturato per la modellizzazione delle minacce nei sistemi informatici e cibernetico-fisici. La sua applicazione inizia esaminando in dettaglio il design del sistema. Questo processo include la creazione di *Diagrammi di Flusso Dati* (DFD), fondamentali per identificare entità, eventi e confini del sistema.
Successivamente si utilizza un set di minacce basate sull'acronimo: *Spoofing*, *Tampering*, *Repudiation*, *Information disclosure*, *Denial of service*, e *Elevation of privilege*. Il modello del sistema, sviluppato in precedenza, è utilizzato per identificare queste minacce. Supporti come checklist e tabelle possono aiutare nella descrizione delle minacce, violazioni, vittime tipiche e azioni degli aggressori.
Pur essendo semplice da adottare, può risultare dispendioso in termini di tempo, specialmente in sistemi complessi dove il numero delle minacce può aumentare rapidamente. E' staato dimostrato che con questo approccio si ha un basso tasso di falsi positivi e un moderato tasso di falsi negativi.

PASTA

La metodologia PASTA è un framework avanzato per la modellazione delle minacce, che incorpora una serie di passaggi metodici per identificare e mitigare i rischi di sicurezza. Questo processo si articola in sette fasi distinte.

1. **Definizione degli Obiettivi**: Identificazione degli obiettivi aziendali, dei requisiti di sicurezza e valutazione dell'impatto aziendale delle minacce.

2. **Definizione dell'Ambito Tecnico**: Delineazione dell'ambiente tecnico, inclusione delle infrastrutture, delle applicazioni e delle dipendenze software.

3. **Decomposizione dell'Applicazione**: Identificazione di casi d'uso, punti di ingresso, livelli di fiducia, attori, asset, servizi, ruoli e fonti di dati. Creazione di diagrammi di flusso dei dati e definizione dei confini di fiducia.

4. **Analisi delle Minacce**: Identificazione e analisi delle minacce potenziali (dettagli non forniti nel documento).

5. **Analisi delle Vulnerabilità e delle Debolezze**: Scoperta e analisi delle vulnerabilità e debolezze. Utilizzo di punteggi e enumerazioni come CVSS, CWSS, CWE e CVE.

6. **Modellazione degli Attacchi**: Analisi probabilistica degli scenari di attacco, analisi di regressione sugli eventi di sicurezza, correlazione e analisi dell'intelligence sulle minacce.

7. **Analisi del Rischio e dell'Impatto**: Valutazione dell'impatto aziendale delle minacce identificate, identificazione di contromisure, analisi del rischio residuo e definizione delle strategie di mitigazione del rischio.

PASTA mira a elevare il processo di threat modeling a un livello strategico, coinvolgendo i principali decisori. Questo approccio si concentra principalmente sul valutare i rischi, adottando una visione che parte dal punto di vista dell'attaccante. L'obiettivo è di fornire un'analisi dettagliata e quantificata delle minacce, concentrandosi sugli asset aziendali e fornendo una valutazione precisa delle minacce attraverso sistemi di punteggio e classificazione

LINDDUN

La metodologia LINDDUN si focalizza sulla valutazione delle minacce in termini di privacy e sicurezza dei dati. Questo metodo è un acronimo che

rappresenta diverse categorie di minacce: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, e Non-compliance.
La metodologia si articola in sei fasi:

1. **DFD (Data Flow Diagram)**: Si inizia con un diagramma dei flussi di dati del sistema, che definisce flussi di dati, archivi di dati, processi e entità esterne. Questo passo è fondamentale per comprendere come i dati si muovono attraverso il sistema e dove possono essere esposti a rischi.

2. **Mappatura**: Durante questa fase, si mappano le categorie di minacce (LINDDUN) alle parti del sistema dove potrebbero manifestarsi. Questo passo è guidato da questionari che aiutano a determinare in quali parti del sistema ciascuna tipologia di minaccia potrebbe concretizzarsi.

3. **Identificazione degli scenari di minaccia**: In questa fase, vengono identificati scenari specifici in cui le minacce identificate possono verificarsi. Serve a comprendere meglio il contesto e le condizioni sotto le quali le minacce possono diventare realtà.

4. **Elaborazione degli alberi delle minacce**: Questa fase comporta la costruzione di alberi delle minacce, che aiutano a visualizzare come una minaccia possa essere realizzata, fornendo una rappresentazione strutturata delle possibili vie di attacco.

5. **Selezione dei controlli**: In questa fase, vengono identificate e selezionate le contromisure appropriate per mitigare le minacce individuate. Questo passo è cruciale per la pianificazione delle azioni di sicurezza.

6. **Documentazione e validazione**: Infine, tutti i risultati delle fasi precedenti vengono documentati e validati. Questo assicura che le valutazioni delle minacce siano state eseguite correttamente e che le contromisure selezionate siano adeguate.

Persona non Grata

La metodologia "Persona non Grata" (PnG) nel contesto del threat modeling si concentra sulle motivazioni e le competenze degli attaccanti umani. Questo metodo caratterizza gli utenti come archetipi che potrebbero abusare del sistema, obbligando gli analisti a considerare il sistema dal punto di vista di un uso imprevisto.
L'idea chiave è quella di "introdurre" un esperto tecnico (come un analista della sicurezza) a un personaggio immaginario che rappresenta un potenziale aggressore del sistema. Questo personaggio, o "persona", è costruito in modo dettagliato, con particolare attenzione alle sue competenze, motivazioni e obiettivi. Questo significa considerare quali aspetti del sistema potrebbero attirare l'attenzione dell'attaccante, quali vulnerabilità potrebbero essere sfruttate e come l'attaccante potrebbe procedere per compromettere il sistema.
Il metodo PnG è facile da adottare, ma è raramente utilizzato o oggetto di ricerca. Produce pochi falsi positivi e ha un'elevata coerenza, tuttavia tende a rilevare solo un certo sottoinsieme di tipi di minacce.

Security Cards

La metodologia delle "Security Cards" è una tecnica di brainstorming per identificare attacchi insoliti e complessi a sistemi informatici. Non si tratta di un metodo formale, ma piuttosto di un approccio creativo per esplorare possibili minacce.

La metodologia utilizza un mazzo di 42 carte divise in quattro dimensioni:

1. **Impatto Umano** (9 carte): Esplora come gli attacchi influenzano le persone e l'ambiente.

2. **Motivazioni dell'Avversario** (13 carte): Indaga le possibili motivazioni dietro un attacco.

3. **Risorse dell'Avversario** (11 carte): Considera le risorse che un avversario potrebbe utilizzare.

4. **Metodi dell'Avversario** (9 carte): Analizza i vari metodi che un avversario potrebbe impiegare per attaccare.

Gli analisti utilizzano le carte per generare domande e scenari di attacco, focalizzandosi su aspetti come:

- Chi potrebbe attaccare il sistema?
- Perché il sistema potrebbe essere attaccato?
- Quali risorse sono di interesse?
- Come possono essere implementati questi attacchi?

Questo metodo aiuta a identificare una vasta gamma di minacce, ma può produrre molti falsi positivi. È particolarmente efficace in situazioni non standard, pur essendo raramente utilizzato nell'industria.

Quantitative Threat Modeling Method

La metodologia "Quantitative Threat Modeling Method" (Quantitative TMM) è un metodo ibrido che integra gli approcci degli Attack Trees, STRIDE e CVSS. Ecco una descrizione dettagliata delle sue fasi:

1. **Costruzione degli Alberi di Attacco per i vari componenti**: Inizialmente, si costruiscono alberi di attacco per ogni componente del sistema, focalizzandosi sulle cinque categorie di minacce di STRIDE. Questa fase serve a mostrare le dipendenze tra le categorie di attacchi e gli attributi di basso livello dei componenti.
2. **Applicazione della metodologia CVSS**: Successivamente, si applica il metodo CVSS per calcolare i punteggi dei componenti all'interno degli alberi di attacco. Questi punteggi aiutano a valutare il rischio associato a ciascun componente.
3. **Generazione di Porte di Attacco per i vari componenti**: Un ulteriore obiettivo del metodo è generare "porte di attacco" per ogni componente. Queste porte di attacco, che funzionano come nodi radice degli alberi di attacco dei componenti, illustrano le attività che possono trasferire rischio ai componenti connessi. Il punteggio di rischio di un componente influisce sulla valutazione del rischio della porta di attacco associata. Se un nodo radice di un componente ha un punteggio di rischio elevato, anche la porta di attacco collegata avrà un punteggio di rischio elevato e sarà più probabile che venga eseguita, e viceversa.

Trike

La metodologia "Trike" è un framework di audit di sicurezza basato sulla modellazione delle minacce e si articola nelle seguenti fasi:

1. **Definizione del Sistema**: Creazione di una matrice attore-risorsa-azione, definendo attori, risorse, azioni e regole del sistema.
2. **Assegnazione di Valori nella Matrice**: Ogni cella della matrice è valutata per le azioni CRUD (Creazione, Lettura, Aggiornamento, Cancellazione) con possibili valori di azione consentita, azione non consentita o azione con regole.
3. **Costruzione del Data Flow Diagram (DFD)**: Mappatura di elementi del DFD a selezioni di attori e asset, identificazione di minacce categorizzate come elevazioni di privilegio o negazioni del servizio.
4. **Valutazione del Rischio di Attacchi**: Valutazione del rischio per ogni attacco potenziale, utilizzando una scala a cinque punti per la probabilità di ciascuna azione CRUD e una valutazione tridimensionale degli attori per le azioni su ogni asset.

Questa metodologia, tuttavia, presenta delle criticità: il sistema di scale di Trike è considerato troppo vago per rappresentare un metodo formale. Sfortunatamente, la versione 2.0 di Trike non è ben mantenuta e manca di documentazione adeguata, nonostante il suo sito web sia ancora attivo.

VAST Modeling

La metodologia "VAST Modeling" (Visual Agile and Simple Threat Modeling) si basa sulla piattaforma automatizzata di modellazione delle minacce chiamata ThreatModeler. Questa metodologia è stata ideata per offrire una soluzione scalabile e facilmente utilizzabile, adatta per l'adozione in grandi organizzazioni all'interno della loro intera infrastruttura.
VAST si articola in due tipi di modelli:

1. **Modelli di Minaccia per Applicazioni:** Utilizzano diagrammi di flusso di processo per rappresentare l'architettura delle applicazioni. Sono focalizzati sulla costruzione e interazione delle applicazioni all'interno dell'infrastruttura.

2. **Modelli di Minaccia Operativi:** Basati su un punto di vista attaccante e realizzati mediante Data Flow Diagrams (DFD). Concentrati sull'identificazione delle minacce dal punto di vista di potenziali aggressori.

Questa metodologia può essere integrata efficacemente nei cicli di sviluppo e operativi delle organizzazioni, fornendo una visione comprensiva sia dal punto di vista architetturale delle applicazioni che da quello operativo dell'infrastruttura.

OCTAVE

Il metodo OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) è una metodologia per valutare i rischi legati alla sicurezza informatica nelle organizzazioni. Sviluppato dal CERT Division del Software Engineering Institute, si concentra sui rischi che riguardano le operazioni aziendali e le pratiche di sicurezza, più che sui problemi tecnici specifici.

1. **Costruzione di Profili di Minaccia Basati sugli Asset**: Si identificano gli elementi importanti dell'organizzazione (asset) e si analizzano le minacce a cui sono esposti, considerando come proteggerli.

2. **Identificazione della Vulnerabilità dell'Infrastruttura**: Si esamina la tecnologia usata dall'organizzazione per trovare punti deboli che potrebbero essere attaccati.

3. **Sviluppo di Strategie di Sicurezza e Piani di Azione**: Si individuano i principali rischi per l'organizzazione e si elaborano strategie e azioni per ridurre questi rischi e aumentare la sicurezza.

OCTAVE è ideato soprattutto per grandi organizzazioni, ma esiste anche una versione per realtà più piccole (20-80 persone) chiamata OCTAVE-S. Questo metodo richiede un impegno notevole in termini di tempo, ma è molto flessibile e approfondito.

La tabella 2.1 elenca le varie metodologie di Threat Modeling e le caratteristiche principali di ognuna.

Metodologia di Threat Modeling	**Caratteristiche**
STRIDE	- Aiuta a identificare tecniche di mitigazione rilevanti - È il più maturo - È facile da usare ma richiede tempo
PASTA	- Aiuta a identificare le tecniche di mitigazione rilevanti - Contribuisce direttamente alla gestione del rischio - Favorisce la collaborazione tra le parti interessate - Contiene una prioritizzazione integrata della mitigazione delle minacce - È laborioso ma offre una documentazione dettagliata
LINDDUN	- Aiuta a identificare tecniche di mitigazione rilevanti - Contiene una prioritizzazione integrata per la mitigazione delle minacce - Può essere intensivo in termini di lavoro e richiedere tempo
CVSS	- Contiene una prioritizzazione integrata della mitigazione delle minacce - Fornisce risultati consistenti quando ripetuto - Componenti automatizzati - Comprende calcoli di punteggio che non sono trasparenti
Attack Trees	- Aiuta a identificare tecniche di mitigazione rilevanti - Produce risultati consistenti quando ripetuto
Continua nella pagina successiva	

Tabella 2.1 – continua dalla pagina precedente

Metodologia di Threat Modeling	**Caratteristiche**
	- È di facile utilizzo se si ha già una conoscenza approfondita del sistema
Persona non Grata	- Aiuta a identificare tecniche di mitigazione rilevanti - Contribuisce direttamente alla gestione del rischio - Fornisce risultati consistenti quando ripetuto - Tende a rilevare solo alcuni sottoinsiemi di minacce
Security Cards	- Favorisce la collaborazione tra gli stakeholder - Si concentra sulle minacce insolite - Può portare a numerosi falsi positivi
Quantitative TMM	- Contiene una prioritizzazione integrata delle misure di mitigazione delle minacce - Dispone di componenti automatizzati - Produce risultati consistenti quando ripetuto
Trike	- Aiuta a identificare tecniche di mitigazione rilevanti - Contribuisce direttamente alla gestione del rischio - Contiene una prioritizzazione integrata per la mitigazione delle minacce - Favorisce la collaborazione tra le parti interessate - Ha componenti automatizzati - Ha documentazione vaga e insufficiente
VAST Modeling	- Aiuta a identificare tecniche di mitigazione rilevanti

Continua nella pagina successiva

Tabella 2.1 – continua dalla pagina precedente

Metodologia di Threat Modeling	**Caratteristiche**
	- Contribuisce direttamente alla gestione del rischio - Contiene una prioritizzazione integrata della mitigazione delle minacce - Favorisce la collaborazione tra le parti interessate - Produce risultati consistenti quando ripetuto - Dispone di componenti automatizzati - È esplicitamente progettato per essere scalabile - Ha documentazione pubblicamente disponibile limitata
OCTAVE	- Aiuta a identificare tecniche di mitigazione rilevanti - Contribuisce direttamente alla gestione del rischio - Contiene una prioritizzazione integrata della mitigazione delle minacce - Favorisce la collaborazione tra le parti interessate - Fornisce risultati coerenti quando ripetuto - È esplicitamente progettato per essere scalabile - Richiede tempo ed ha documentazione vaga

Tabella 2.1: Caratteristiche delle Metodologie di Threat Modeling

Capitolo 3

Obiettivi Pratici della Tesi

Dopo gli studi preparatori effettuati sulla tematica del Cloud Computing e sul processo di Threat Modeling, questo capitolo intende offrire una panoramica chiara delle motivazioni che hanno ispirato il presente lavoro di tesi, nonché degli obiettivi specifici che si è cercato di raggiungere.

3.1 Introduzione al problema

Questa tesi è stata realizzata in un'azienda di consulenza informatica specializzata in sicurezza. All'interno di questa organizzazione le pratiche di buona programmazione e di rilascio di software sicuro erano giò ampiamente integrate nelle fasi dello sviluppo e distribuzione delle applicazioni. Tuttavia, un aspetto che necessitava di un'analisi più approfondita era quello relativo non tanto allo sviluppo sicuro del software, quanto al monitoraggio della sicurezza all'interno delle piattaforme coinvolte nel ciclo di vita di quest'ultimo.
Da questa constatazione nasce l'idea che guida il presente lavoro di tesi: dimostrare come l'automazione dei controlli di sicurezza possa rivelarsi una scelta chiave, permettendo una gestione più efficente della sicurezza e riducendo i rischi legati agli errori umani, responsabili di una vasta gamma di problemi di sicurezza, dalla perdita di dati fino alla compromissione dell'intero sistema.

3.2 Scopo del lavoro di Tesi

In questo lavoro di tesi, ci si è concentrati su specifiche fasi dello sviluppo software, ovvero le fasi di operation e monitoring, anziché approfondire l'integrazione della sicurezza durante l'intero ciclo di vita del software. Questa scelta è stata motivata da diverse ragioni. Innanzitutto, spesso i controlli di sicurezza vengono eseguiti solo fino alla distribuzione del software, trascurando la necessità di effettuare controlli continui e regolari nel tempo. Inoltre, essendo la DevSecOps una pratica già assai diffusa, la tesi si propone di approfondire la sicurezza da un altro punto di vista rispetto a quello dello sviluppo di codice sicuro, cioè dal punto di vista della sicurezza delle componenti architetturali e infrastrutturali coinvolte nel ciclo di sviluppo e distribuzione del software. Nell'immagine seguente sono rappresentate le diverse fasi che compongono il ciclo di vita del software, con particolare enfasi su quelle su cui si è focalizzato il lavoro di automazione sviluppato durante la tesi.

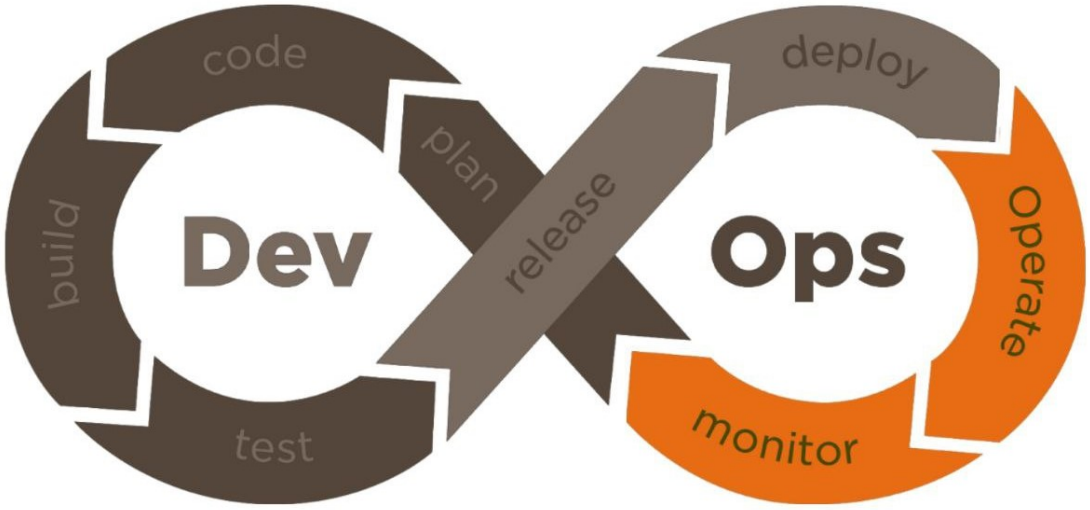


Figura 3.1. Fasi del processo di DevOps

3.3 Fasi del lavoro di Tesi

Questo lavoro di tesi è stato articolato in quattro fasi:

- **Studio Iniziale sul Cloud**: Dopo un'analisi approfondita del mondo del cloud computing, è stata definita e configurata un'architettura utilizzata come Proof of Concept (PoC), impiegando per lo sviluppo e la distribuzione del software varie piattaforme, la cui selezione è dipesa da quelle già utilizzate all'interno dell'azienda.
- **Analisi delle Metodologie di Threat Modeling**: Sono state esaminate varie metodologie di threat modeling presenti in letteratura per identificare l'approccio più adatto agli obiettivi del lavoro.
- **Modellazione dell'Architettura e delle Sue Minacce**: A seguito dello studio sulle diverse metodologie di threat modeling, è stata adottata una strategia personalizzata, che integra elementi delle metodologie esaminate e che meglio si adatta agli scopi della tesi. Questo approccio ha permesso una modellazione dell'architettura e delle sue minacce, concentrandosi sulle vulnerabilità e i rischi alla sicurezza legati alle configurazioni delle piattaforme e delle tecnologie utilizzate.
- **Fase Sperimentale e Sviluppo del Framework**: Nella parte sperimentale del lavoro, è stato sviluppato un framework per automatizzare una serie di controlli di sicurezza. Grazie a questo, tramite la schedulazione di una pipeline automatizzata, è possibile monitorare costantemente il livello di sicurezza del sistema, permettendo una reazione più reattiva e tempestiva alle potenziali problematiche.

Capitolo 4

Configurazione dell'Architettura Cloud

In questo capitolo, si procederà a fornire una dettagliata panoramica sulle tecnologie e gli strumenti impiegati nel processo di sviluppo e messa a punto dell'architettura cloud che costituisce il Proof-of-Concept della tesi. Successivamente, sarà condotta un'approfondita disamina degli elementi costituenti l'architettura stessa, insieme alle loro specifiche configurazioni, al fine di offrire una comprensione completa e dettagliata del contesto tecnologico e dei dettagli implementativi associati.

4.1 Tecnologie e strumenti utilizzati

Nel contesto della realizzazione del Proof-of-Concept, vi sono tre tecnologie di fondamentale importanza: Jenkins, Kubernetes e Docker. Ognuna di queste piattaforme riveste un ruolo chiave in un particolare aspetto dell'implementazione e gestione dell'architettura cloud.
Jenkins è la spina dorsale del processo di automazione e integrazione continua, garantendo un flusso di sviluppo senza interruzioni. Kubernetes si pone come l'orchestratore principale, assicurando una gestione scalabile e affidabile dei container applicativi. Infine, Docker fornisce l'ambiente di contenimento per le applicazioni, consentendo un'implementazione isolata e portabile. Ecco ora una breve introduzione su ciascuna di queste tecnologie.

4.1.1 Jenkins

Jenkins è un server di automazione open-source che può essere impiegato come server di integrazione continua (CI) semplice o trasformato in centro di distribuzione continua (CD).
Questo strumento può essere installato in vari modi, tra cui utilizzando pacchetti di sistema nativi, Docker o eseguendolo in modalità "standalone" su qualsiasi macchina equipaggiata con Java.
Molte attività legate alla creazione, al testing e alla consegna o distribuzione del software possono essere automatizzate grazie a Jenkins [5]. Inoltre, Jenkins è altamente personalizzabile grazie all'uso di plugin, i quali semplificano l'integrazione con altri strumenti, e può essere configurato facilmente attraverso la sua interfaccia web. Queste caratteristiche lo rendono un eccellente candidato per l'implementazione di pipeline di integrazione continua e distribuzione continua (CI/CD). Jenkins è infatti lo strumento preferito tra gli sviluppatori DevOps, detenendo il 47,45% del mercato del software di integrazione continua [6].
L'implementazione e l'integrazione di pipeline di consegna continua in Jenkins sono supportate da una serie di plugin noti come Jenkins Pipeline. La definizione di queste pipeline avviene attraverso l'utilizzo di un linguaggio di dominio specifico (DSL) basato su Groovy [7].
Nell'immagine seguente vediamo una rappresentazione grafica di una pipeline Jenkins, in cui ogni passaggio viene eseguito senza errori e la pipeline si conclude con successo.

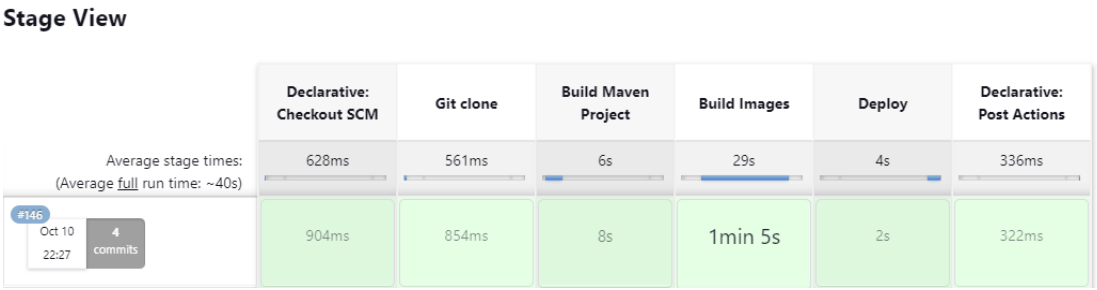


Figura 4.1. Esempio di pipeline Jenkins

Il codice della pipeline Jenkins viene utilizzato per definire il processo di

creazione, che di solito include fasi per la costruzione, il testing e la distribuzione di un'applicazione.
Le pipeline possono essere definite tramite l'interfaccia web o all'interno di un file chiamato Jenkinsfile. Entrambi i metodi utilizzano la stessa sintassi, ma la definizione tramite il Jenkinsfile è quella preferita. Questo perché può essere inserita nel repository che contiene il codice sorgente e sfruttare questi vantaggi:

- Creazione automatica di un processo di compilazione della pipeline per tutti i rami e le richieste di pull.
- Revisione/iterazione del codice sulla Pipeline (insieme al restante codice sorgente).
- Traccia di audit per la pipeline.
- Esiste una singola fonte di verità per la pipeline, che può essere visualizzata e modificata da più membri del progetto.

Le pipeline sono scrivibili con due diversi tipi di sintassi: Dichiarativa e Scriptata (imperativa).

- **Sintassi della Pipeline Dichiarativa**. Tutto il lavoro svolto durante l'intera pipeline è definito dal blocco *pipeline*, e il codice contenuto in quel blocco definisce l'intero processo di creazione.
- **Sintassi della Pipeline Scriptata**. Il lavoro principale nell'intera pipeline è svolto da uno o più blocchi *nodo*.

È importante notare che, anche se le pipeline dichiarative e scriptate sono costruite in modo diverso, condividono molte delle componenti sintattiche scritte in un Jenkinsfile. Un possibile esempio è il blocco *stage*, che viene utilizzato per descrivere una fase della pipeline. Ogni fase è composta da un sottoinsieme di passi eseguiti da Jenkins lungo l'intera pipeline.

4.1.2 Docker

Docker è una piattaforma open-source che consente di sviluppare, distribuire ed eseguire applicazioni.

Docker offre la possibilità di separare le applicazioni dalle infrastrutture, accelerando così la messa in produzione del software.
Docker si basa su immagini: un'immagine è un modello di sola lettura contenente le istruzioni per creare un container Docker. Spesso un'immagine si basa su un'altra immagine, con qualche personalizzazione aggiuntiva.
Per creare un'immagine personalizzata, è necessario un Dockerfile. Il Dockerfile è composto da una sequenza di passi utilizzati per creare l'immagine ed eseguirla. Ogni istruzione in un Dockerfile crea un livello nell'immagine. Quando si modifica il Dockerfile e si ricostruisce l'immagine, vengono ricostruiti solo i livelli che sono stati modificati ed è questo che rende le immagini così leggere.
Con la tecnologia Docker, il software può essere impacchettato e quindi eseguito in un ambiente relativamente isolato, chiamato **container**. Un container è un'istanza leggera di un'immagine e contiene tutto ciò che è necessario per l'esecuzione dell'applicazione senza dipendere da ciò che è installato sull'host, consentendo di condividere il contenitore stesso fra macchine diverse. Grazie alle proprietà di isolamento e di sicurezza, è possibile eseguire contemporaneamente molti container su un determinato host. Il livello di isolamento di un container rispetto agli altri container e alla macchina host può essere controllato manipolando la rete, lo storage e altri sottosistemi sottostanti.
Un container può essere avviato, fermato, spostato o cancellato.
Inoltre, un container è caratterizzato dalla sua immagine e dalle opzioni di configurazione che possono essere fornite quando il contenitore stesso viene creato o avviato.
I container Docker hanno alcune proprietà specifiche, quali:

- **Standardizzazione**: un container è rappresentato in un modo standard che ne consente l'esecuzione ovunque sia disponibile Docker.
- **Leggerezza**: i container condividono il kernel della macchina host e per questo motivo non è necessario avere un sistema operativo per ogni applicazione. Questa caratteristica permette di avere server più efficienti e di ridurre i costi delle licenze.
- **Sicurezza**: per impostazione predefinita, Docker fornisce un elevato livello di isolamento che protegge le applicazioni che girano all'interno dei container.

- **Isolamento delle risorse**: prestazioni prevedibili delle applicazioni.

Nell'immagine sottostante viene proposta una semplice descrizione dell'architettura di Docker:

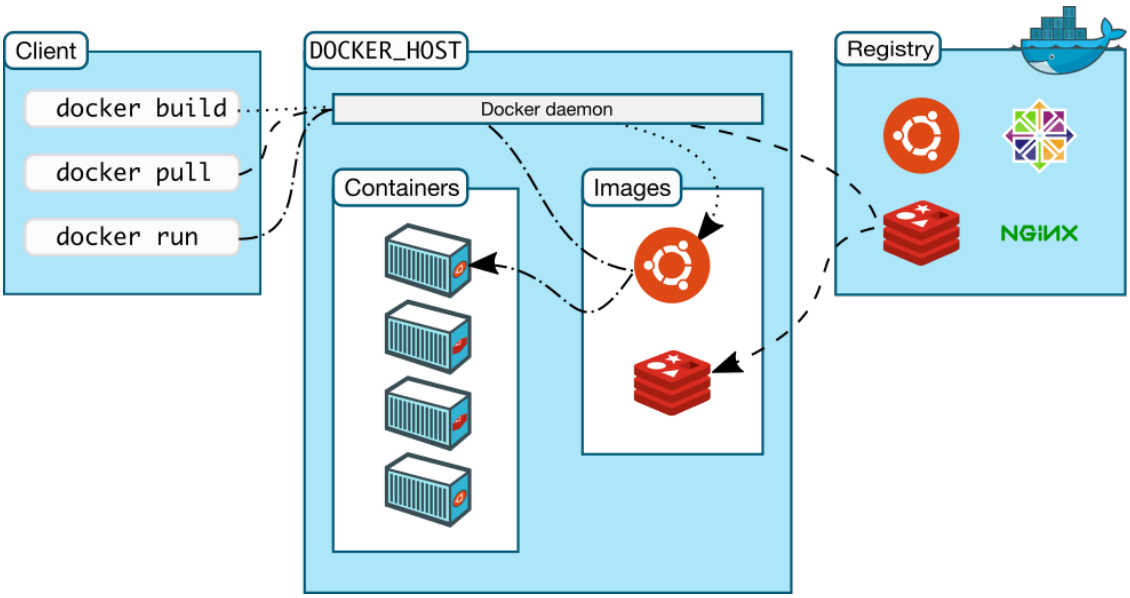


Figura 4.2. Architettura Docker

Il client di Docker interagisce con il Docker daemon, che esegue i comandi: il daemon fa la build, esegue e distribuisce i container Docker. Il client Docker può comunicare con molti Docker daemon.
Il Docker daemon e il client possono essere eseguiti sullo stesso sistema o possono essere collegati in remoto. Comunicano utilizzando un'API REST, tramite socket UNIX o un'interfaccia di rete.

4.1.3 Kubernetes

Kubernetes è una piattaforma open-source, estensibile e portabile per la gestione di carichi di lavoro e servizi containerizzati, in grado di semplificare sia la configurazione che l'automazione.
Kubernetes si basa sui container, che rappresentano una buona soluzione per distribuire ed eseguire applicazioni.
In un ambiente di produzione è necessario gestire i container e garantire che

non ci siano interruzioni; ad esempio, se un container smette di funzionare, un altro deve sostituirlo.
Kubernetes fornisce un framework per l'esecuzione di sistemi distribuiti in modo resiliente, occupandosi dello scaling e del failover delle applicazioni e fornisce anche modelli di distribuzione.
Tra tutte le funzionalità, Kubernetes offre:

- **Bilanciamento del carico**: un container Kubernetes può essere raggiungibile dall'esterno del cluster utilizzando un nome DNS o il suo indirizzo IP. Inoltre, Kubernetes permette di reindirizzare il traffico a diversi container se è troppo elevato per un singolo container.
- **Self-healing**: Per mantenere i servizi disponibili, Kubernetes manipola i container, riavviando quelli che non funzionano, sostituendoli e rimuovendo quelli che non corrispondono ai controlli di salute definiti dall'utente.
- **Orchestrazione dello storage**: Kubernetes consente di montare automaticamente un sistema di archiviazione, che può essere, ad esempio, un archivio locale o uno storage su cloud.
- **Rollback automatico**: In Kubernetes è possibile descrivere lo stato desiderato per i container distribuiti. Kubernetes è in grado di cambiare lo stato attuale cercando di raggiungere quello desiderato. Ad esempio, Kubernetes può essere automatizzato per creare nuovi container per la distribuzione, rimuovere i container esistenti e agganciare tutte le loro risorse al nuovo container.

Una volta descritte alcune delle caratteristiche fondamentali, è possibile andare un po' più a fondo e vedere qual è l'infrastruttura di Kubernetes.

Un cluster Kubernetes è composto da un insieme di nodi in cui vengono eseguite applicazioni containerizzate. Ogni cluster ha almeno un nodo worker che ospita alcuni Pod. Il ruolo del control plane è quello di gestire i nodi e i Pod nel cluster, prendendo decisioni globali (come lo scheduling) e rispondendo a eventi specifici (come l'avvio di un nuovo pod per un ReplicaSet). Per fornire un'elevata disponibilità e tolleranza ai guasti, il control plane dovrebbe essere eseguito su computer diversi, mentre il cluster dovrebbe essere eseguito su nodi diversi.

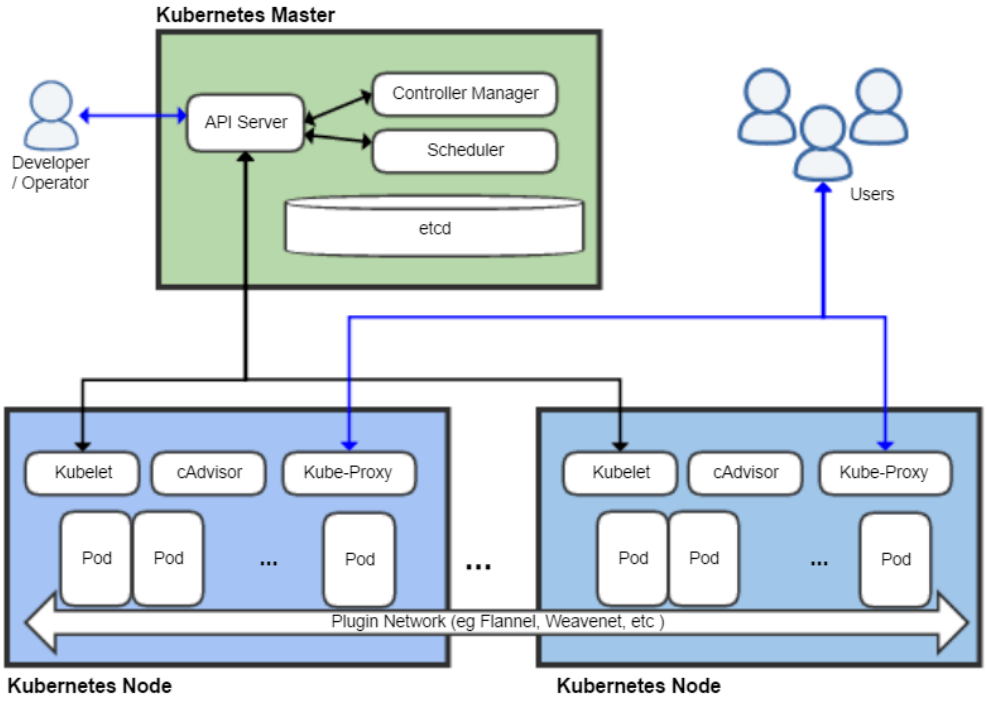


Figura 4.3. Componenti di Kubernetes

Kubernetes fornisce diversi tipi di risorse utili per personalizzare il cluster e per distribuire le applicazioni; le risorse Kubernetes più comuni sono:

- **Deployment**: Un deployment fornisce aggiornamenti dichiarativi per i Pod e i ReplicaSet. Il controllore di deployment cambia lo stato attuale in quello desiderato. I deployment sono usati per creare nuovi ReplicaSet, o per rimuovere i deployment esistenti e adottare tutte le loro risorse con nuovi deployment.

- **Pod**: I Pod sono le più piccole unità di calcolo distribuibili che possono essere create e gestite in Kubernetes. Un Pod racchiude uno o più container, con risorse di rete e di storage condivise. Ogni Pod ha una configurazione specifica per l'esecuzione dei container.

- **ReplicaSet**: Lo scopo del ReplicaSet è quello di mantenere sempre

stabile il numero di Pod di replica in esecuzione: di solito viene utilizzato per garantire la proprietà di "disponibilità".

4.2 Architettura Cloud

Dopo una l'introduzione sulle tecnologie utilizzate per la messa a punto dell'architettura cloud, vediamo ora i dettagli relativi alla sua realizzazione.
Va subito detto che questo sistema è stato pensato e progettato seguendo il modello di un ambiente di lavoro completo destinato a coprire il ciclo di sviluppo, test e distribuzione di applicazioni.
Va sottolineato però, che nonostante siano stati inclusi gli elementi fondamentali per un ambiente di produzione, le configurazioni sono state semplificate o trascurtate per concentrare l'attenzione sugli aspetti critici dell'analisi.
Inoltre, nella fase iniziale di configurazione dell'architettura, l'obiettivo primario è stato garantire l'operatività del sistema, senza porre un'attenzione prioritaria alle misure di sicurezza consigliate. L'architettura di riferimento è composta da:

- **Istanza EC2**: Ospitata su AWS, funge da server principale, su cui sono installati Jenkins, Docker e Minikube;
- **Server Jenkins**: Utilizzato per creare una pipeline di Continuous Integration/Continuous Deployment (CI/CD) che automatizza il processo di deploy dei microservizi;
- **Repository GitHub**: Contiene il codice sorgente dei microservizi, fornendo uno spazio centrale per la collaborazione e la gestione delle modifiche al codice;
- **Repository Docker**: Utilizzato per archiviare e distribuire i container dei microservizi, semplificando la gestione delle immagini containerizzate;
- **Cluster Minikube**: Consente l'esecuzione e la gestione delle applicazioni, creando un ambiente locale per lo sviluppo e l'esecuzione all'interno di Kubernetes.

Ora vediamo nel dettaglio la configurazione dei varie componenti dell'architettura.

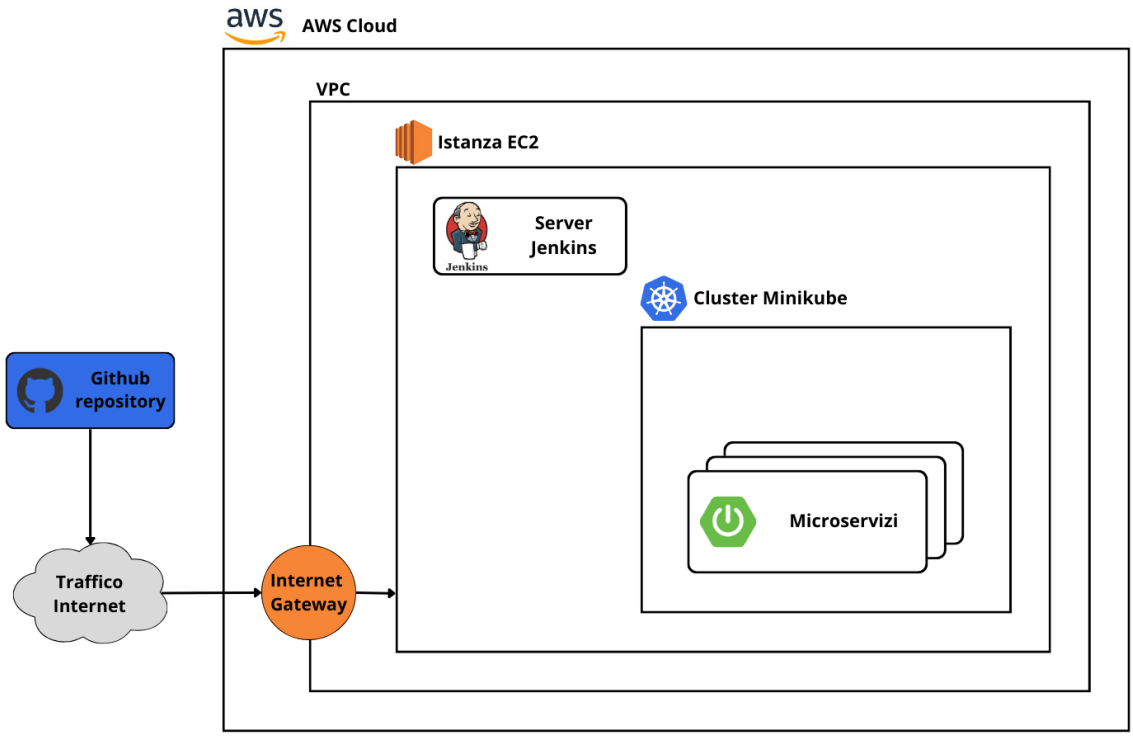


Figura 4.4. Architettura Cloud

4.2.1 Istanza EC2 AWS

L'infrastruttura cloud è stata distribuita all'interno di un ambiente AWS [8]. La scelta del service provider è ricaduta su AWS essendo attualmente la piattaforma cloud con una più lunga storia alle spalle [10], sinonimo di esperienza, ed anche la più utilizzata dagli sviluppatori [9].
Come già detto precedentemente, si è scelto di emulare un ambiente simile a quello di produzione. A tal proposito, sono stati impiegati due servizi offerti da AWS [11]: Amazon Elastic Compute Cloud (Amazon EC2), Amazon Virtual Private Cloud (Amazon VPC).

Amazon EC2 è un servizio di Cloud Computing di AWS che consente di creare e gestire macchine virtuali (istanze) su infrastrutture cloud. E' possibile scegliere e personalizzare il tipo di istanza in base alle proprie necessità e

pagare solo per le risorse effettivamente utilizzate. Quando si crea un'istanza, questa viene avviata da un'Amazon Machine Image (AMI). Un'AMI è un modello preconfigurato che include un sistema operativo e le configurazioni software associate. Da un singolo AMI è possibile avviare più istanze EC2 con le stesse configurazioni software di base.
Nel lavoro di tesi, durante il processo di selezione del tipo di istanza, la scelta è ricaduta su un'istanza EC2 avente specifiche tecniche minime necessarie a garantire prestazioni adeguate per i servizi richiesti, mantenendo così un bilanciamento tra risorse computazionali e costi di utilizzo.
Nella fase iniziale di selezione e configurazione di un'istanza EC2, ci sono diversi aspetti da tenere in considerazione, tra cui i principali risultano essere: caratteristiche architetturali, opzioni di sistema operativo e gestione della sicurezza.
Le scelte architetturali in fase di configurazione di un'istanza EC2 si riferiscono agli aspetti legati alla struttura di base dell'istanza stessa e alle specifiche tecniche.
Nel caso specifico, l'istanza EC2 è istanziata da un'AMI dpotata di un sistema operativo Linux Ubuntu 22.04 LTS, con architettura x86. Questa istanza offre 4 vCPU, 16 GB di memoria RAM e uno spazio su disco SSD di 50 GB. Come già precedentemente discusso, queste caratteristiche sono state definite sulla base di quelle che sono le risorse richieste dagli strumenti impiegati all'interno della macchina.
Per quanto riguarda le configurazioni inerenti alla sicurezza, ne troviamo diverse che direttamente o indirettamente incidono su di essa.
Innanzitutto, è fondamentale garantire un accesso sicuro all'istanza. A tal proposito è stata generata una coppia di chiavi crittografiche, associata all'istanza, utilizzate per la connessione mediante il protocollo SSH.
Successivamente, è stata configurata la sicurezza a livello di gruppi di sicurezza (Security Groups) e Virtual Private Cloud (VPC).
I gruppi di sicurezza permettono di definire regole di rete in ingresso e in uscita per le istanze EC2. Attraverso la configurazione accurata di tali regole, è possibile controllare quali indirizzi IP o range di IP sono autorizzati a comunicare con l'istanza e su quali porte di rete [12]. Ciò aiuta a isolare l'istanza e a ridurre i possibili vettori di attacco.

Invece, la VPC offre un ambiente di rete virtualizzato che permette di definire subnet, tabelle di routing e regole di controllo del traffico. Configurando una VPC in modo appropriato, è possibile garantire la separazione tra le risorse all'interno della rete, controllare il traffico tra le subnet e connettere la VPC in modo sicuro a una rete on-premises o ad altre VPC attraverso VPN o peering [13].
Per quanto riguarda le regole di sicurezza, inizialmente sono state configurate solamente due regole, una per il traffico in ingresso e una per il traffico in uscita, che sostanzialmente non pongono restrizioni, consentendo all'istanza di comunicare liberamente con qualunque destinazione.
Mentre, la VPC a cui l'istanza EC2 è stata associata, presentava anch'essa una configurazione basilare, avente una route table di default. Questa route table contiene due regole di routing: la prima per il traffico in locale, mentre l'altra consente il traffico verso Internet tramite un collegamento all'Internet Gateway, senza alcuna restrizione riguardo a destinazioni o subnet. In pratica, ciò significa che la VPC permette all'istanza di accedere a Internet e di comunicare con qualsiasi destinazione su Internet senza restrizioni aggiuntive.

4.2.2 Cluster Kubernetes

Dopo aver discusso della configurazione dell'infrastruttura cloud, questa sottosezione è dedicata alla scelta del cluster Kubernetes, dove verrà distribuita l'applicazione.
Il cluster in questione dovrà avere caratteristiche di base, facilità di utilizzo e configurazione ed essere semplicemente in grado di distribuire i microservizi al suo interno e permetterne l'esecuzione. Sulla base di questo, la scelta è stata indirizzata verso un cluster Kubernetes locale.
Questa tipologia di cluster risulta essere particolarmente adatta per lo scopo, poiché consente di creare e testare le applicazioni in un ambiente locale, controllato e isolato, prima di trasferire qualsiasi cosa su un cloud pubblico, consentendo così la separazione tra l'ambiente di sviluppo e l'ambiente di produzione.
Un cluster Kubernetes locale è generalmente costituito da un singolo nodo o da un numero limitato di nodi. Questi nodi possono essere macchine virtuali o server fisici. La loro gestione è semplificata, il che facilita il processo di

distribuzione e l'esecuzione dei microservizi.
Inoltre, questi tipi di cluster possono essere avviati rapidamente e senza costi aggiuntivi significativi, a differenza delle soluzioni basate su cloud che comportano costi operativi [14].
Esistono diversi strumenti che permettono la creazione di cluster Kubernetes locali, come KinD, Minikube, K3s [15].
Alcune delle loro caratteristiche fondamentali sono:

- Supporto multipiattaforma (Linux, macOS, Windows);
- Disponibilità della dashboard;
- Architetture supportate (AMD64, ARMv7...);
- Runtime di container supportati (Docker, CRI-O, gvisor...);
- Endpoint dell'API di Docker per un caricamento delle immagini veloce;
- Accesso root: se sono richiesti i privilegi di root.

È importante notare che questi cluster, come tutti gli altri esistenti, sono soggetti a evoluzioni e aggiornamenti frequenti, che comportano miglioramenti delle prestazioni e l'aggiunta di nuove funzionalità.
Dopo ricerche e confronti, il cluster scelto è stato Minikube. Minikube fornisce un'utile dashboard per interagire con il cluster, un'installazione semplice, tutte le funzionalità di base di K8S e richiede solo 2 GB di RAM.
L'installazione e la configurazione di Minikube sono state condotte seguendo le istruzioni dettagliate fornite da una guida online [16].
Il primo componente installato è stato "kubectl". Kubectl è uno strumento di amministrazione di Kubernetes che consente agli utenti di interagire con i cluster Kubernetes, compreso il cluster locale creato da Minikube. È la principale interfaccia a riga di comando per controllare, gestire e monitorare i cluster Kubernetes, indipendentemente dalla loro posizione. In particolare, all'interno di Minikube, kubectl svolge un ruolo cruciale poiché permette agli sviluppatori di eseguire una serie di operazioni chiave, come creare, aggiornare e gestire risorse Kubernetes (pod, servizi e deployment), visualizzare i log dei container all'interno dei pod e eseguire comandi all'interno dei container.
Successivamente, è stato installato Docker, uno strumento di virtualizzazione dei container, di cui si è già parato in precedenza. Docker è un componente

essenziale per il funzionamento di Minikube, poiché consente di creare, gestire e orchestrare i container che vengono eseguiti all'interno del cluster.
Una volta completata l'installazione di tutti i componenti necessari e di Minikube stesso, il cluster è stato avviato con le configurazioni predefinite, senza quindi aver stabilito restrizioni d'accesso alle risorse o politiche di rete, né aver configurato un sistema di monitoraggio, procedure di aggiornamento o backup.

4.2.3 Applicativo a microservizi

L'applicativo distribuito all'interno dell'architettura funge da esempio di prodotto sviluppato e distribuito in un contesto aziendale. È importante sottolineare che lo sviluppo di questo applicativo non è stato parte integrante del lavoro di tesi, al contrario, l'applicativo è stato estratto da una repository GitHub pubblica[1]. Questa decisione è stata motivata dalla volontà di garantire un'esperienza di analisi della sicurezza il più possibile realistica. Spesso, infatti, l'analista della sicurezza non è coinvolto nella fase iniziale di configurazione del sistema, ma piuttosto nell'analisi dei rischi in un contesto in cui il sistema è già operativo.
Il linguaggio utilizzato per lo sviluppo dell'applicazione è Java, facendo uso dei framework Spring e Spring Cloud. Inoltre, l'applicativo è stato containerizzato utilizzando Docker, un'azione che favorisce l'isolamento delle componenti del sistema. La presenza dei file Dockerfile garantisce un'efficace creazione delle immagini dei container, consentendo un ambiente di esecuzione coeso e separato per ciascun microservizio.
Il software è composto da tre microservizi, ciascuno con uno scopo ben definito:

- Catalog: Questo microservizio è responsabile della gestione degli articoli presenti nel catalogo. La sua funzione principale è consentire agli utenti di selezionare prodotti disponibili nel catalogo all'interno del sistema.

- Customer: Questo microservizio gestisce i dati dei clienti, consentendo la registrazione e la gestione dei profili utente.

[1] `https://github.com/ewolff/microservice-kubernetes`

- Order: Questo microservizio si occupa dell'elaborazione degli ordini, e interagisce con i microservizi Catalog e Customer per fornire una soluzione completa per la gestione degli ordini.

L'implementazione della comunicazione tra microservizi tramite il protocollo REST (Representational State Transfer) favorisce una connessione leggera ed efficiente. Questo avviene perché REST sfrutta la struttura stateless delle richieste HTTP, riducendo al minimo il mantenimento di uno stato di sessione tra le richieste. Questa caratteristica semplifica le comunicazioni tra i servizi e garantisce un'efficace scalabilità.
Un componente aggiuntivo di rilievo è il server Apache HTTP, configurato come reverse proxy. Questo server fornisce un punto di ingresso unificato per l'intero sistema sulla porta 8080, semplificando l'accesso agli utenti. Inoltre, svolge un ruolo di inoltro delle richieste HTTP ai microservizi.

Per quanto riguarda la repository GitHub contenente il codice sorgente, è stata inizialmente configurata con visibilità pubblica, consentendo a chiunque di accedere, visualizzare e apportare modifiche al codice, nonché richiedere lo status di collaboratore. Questa scelta è stata effettuata con l'obiettivo di stabilire una base neutra da cui condurre l'analisi di sicurezza.
La struttura della repository è rimasta semplice, con tutto il codice ospitato nel branch principale e senza l'adozione di convenzioni avanzate di versionamento. La collaborazione e la comunicazione sono state gestite in modo informale, senza l'implementazione di politiche o protocolli specifici.

4.2.4 Pipeline CI/CD

Per automatizzare il processo di distribuzione dell'applicativo all'interno del cluster, è stata configurata una pipeline di Continuous Integration e Continuous Deployment (CI/CD). La prima fase di questa configurazione ha coinvolto l'installazione e la configurazione di Jenkins all'interno dell'architettura.
Lo strumento è stato installato sull'istanza EC2 mediante il sistema di gestione dei pacchetti "apt-get". Dopo l'installazione, sono stati aggiunti i plugin necessari per supportare le funzionalità richieste, tra cui il plugin Docker che consente di gestire e orchestrare container Docker direttamente da Jenkins.

Per consentire la comunicazione tra l'agent di Jenkins e i repository GitHub e DockerHub, sono state create credenziali specifiche. In entrambi i casi è stato generato un token d'accesso sulle rispettive piattaforme, utilizzato successivamente all'interno di Jenkins per l'autenticazione delle richieste verso queste.
A questo punto è stato possibile definire ed eseguire la pipeline. Questa è stata scritta in Groovy, un linguaggio di scripting ampiamente utilizzato per la definizione di pipeline Jenkins.
Lo script si compone delle seguenti fasi:

- **Git Clone**: nella prima fase, il codice sorgente è stato clonato dal repository GitHub. È stato specificato il ramo master per ottenere la versione più aggiornata del codice.
- **Build Maven Project**: questa fase si occupa della compilazione del progetto Maven. Il comando Maven è stato eseguito all'interno della directory del progetto, compilando il codice sorgente.
- **Build Images**: in questa fase, le immagini Docker dei microservizi vengono create eseguendo uno script contenuto nella directory del progetto.
- **Deploy**: nell'ultima fase, le immagini containerizzate vengono distribuite nell'ambiente di destinazione. La pipeline utilizza un file di configurazione Kubernetes, contenuto anch'esso nella repository del progetto, e le credenziali Kubernetes per applicare le configurazioni ai microservizi.
- **Post-action**: infine, all'interno del blocco post, è stato effettuato il logout da Docker. Questo passaggio garantisce la sicurezza impedendo accessi non autorizzati.

Capitolo 5

Threat Modeling e Automazione del processo

In seguito alla definizione dell'architettura presentata nel capitolo precedente, qui si procederà, in un primo momento, ad illustrare in dettaglio il processo di Threat Modeling eseguito su di essa.
Successivamente, il capitolo si concentrerà sulla fase di automazione del processo, evidenziando come l'integrazione di strumenti e tecniche abbiano contribuito ad ottimizzare e velocizzare il monitoraggio della sicurezza.

5.1 Processo di Threat Modeling

In questa sezione verrà descritto il processo di modellazione delle minacce effettuato come fase preliminare del lavoro di automazione. Questo si è rivelato fondamentale, poichè è servito a dare una chiara visione di quelle che fossero le risorse del sistema e le varie criticità da non trascurare.

5.1.1 Approccio utilizzato

Il processo di Threat Modeling adottato in questo lavoro di tesi è stato concepito per ottenere una profonda comprensione del sistema, con un focus particolare sulle minacce che potrebbero influenzarne l'architettura, specialmente dal punto di vista delle piattaforme che la compongono. Considerando le limitazioni, come la mancanza di esperienza tecnica e la complessità del

campo, l'approccio adottato si è distinto per il suo esame diretto e mirato delle componenti del sistema. Invece di seguire un percorso standardizzato di modellazione delle minacce, come la creazione di diagrammi di flusso dei dati (DFDs) tipici delle metodologie come STRIDE, si è optato per un'analisi diretta di ciascuna componente. Questo approccio ha favorito un'indagine dettagliata focalizzata sulle caratteristiche tecniche e funzionali delle piattaforme, assicurando così un'analisi accurata e mirata delle minacce maggiormente rilevanti e impattanti per il contesto specifico del sistema.

5.1.2 Confronto con le metodologie note in letteratura

Nel contesto di questo studio, l'approccio al Threat Modeling ha mostrato somiglianze con metodologie note come STRIDE e PASTA, pur mantenendo alcune distinzioni chiave.
Come STRIDE, è mirato a identificare specifiche categorie di minacce (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service e Elevation of Privilege) per ogni elemento dell'ambiente, un approccio che STRIDE persegue attraverso l'analisi dettagliata del design del sistema. Tuttavia, a differenza di questo, che utilizza principalmente diagrammi di flusso dei dati (DFDs), il metodo qui impiegato si è concentrato sull'analisi diretta delle componenti, senza un'enfasi specifica sulla creazione di DFDs.
L'adozione del framework PASTA permette un'integrazione più stretta tra obiettivi aziendali e requisiti tecnici, simile all'approccio utilizzato in questo lavoro di tesi, volto a valutare le minacce in relazione alle best practice e ai requisiti specifici delle varie componenti del sistema. Tuttavia, l'approccio impiegato si distingue per la sua focalizzazione più ristretta sulle specifiche tecnologie e piattaforme utilizzate, piuttosto che su un'analisi globale.
In conclusione, benché incorpori elementi di metodologie consolidate, l'approccio adottato per il processo di Threat Modeling si caratterizza per la sua attenzione specifica alle componenti e alle tecnologie impiegate, offrendo un'analisi mirata e contestualmente rilevante delle minacce e delle strategie di mitigazione.

5.1.3 Fasi del processo

Modellazione dell'architettura

Nella fase di modellazione dell'architettura, si è proceduto con un'indagine dettagliata di ciascuna componente del sistema, concentrandosi in particolare sulle loro configurazioni e sulle modalità di interazione. L'intento era quello di capire non solo come ogni elemento fosse strutturato individualmente, ma anche come si integrasse all'interno dell'ecosistema complessivo.
Di seguito sono riportate le risorse modellate e su cosa il modello di ognuna si è concentrato.

- **Istanza AWS EC2**: L'analisi dell'istanza EC2 si è concentrata sulle sue configurazioni di rete, capacità di storage e opzioni di sicurezza. Questo ha incluso la revisione delle impostazioni di configurazione a livello di intero account, delle politiche di accesso e autorizzazione e delle procedure di backup e cifratura dei dati, al fine di comprendere come queste configurazioni influenzassero l'interoperabilità con le altre componenti del sistema.

- **Cluster Minikube**: Per il cluster Minikube, l'attenzione si è rivolta alle sue impostazioni di containerizzazione e gestione del traffico di rete. Si è valutato il modo in cui i container erano configurati per comunicare tra loro e con l'esterno, analizzando le configurazioni di sicurezza e le potenziali interazioni con l'istanza EC2 e Jenkins.

- **Pipeline Jenkins**: L'analisi di Jenkins si è concentrata sulle sue funzioni di CI/CD, esaminando come le fasi di integrazione e distribuzione fossero configurate e come interagissero con il repository GitHub e l'istanza EC2. È stata data particolare attenzione alla gestione delle credenziali e all'utilizzo dei plugin.

- **Repository GitHub**: Per il repository GitHub, l'analisi ha riguardato le procedure di gestione del codice, il controllo degli accessi e la sincronizzazione con Jenkins. Si è cercato di capire come le modifiche al codice fossero gestite e integrate nel flusso di lavoro complessivo, e come le politiche di accesso influenzassero la sicurezza dei dati.

Modellazione delle minacce

Nel processo di modellazione delle minacce, si è prestata particolare attenzione alle vulnerabilità specifiche di ciascuna componente dell'architettura, esaminando attentamente i possibili vettori d'attacco e valutando in che modo queste vulnerabilità potessero essere sfruttate, nonché quali impatti avrebbero potuto avere sul sistema.

- **Istanza AWS EC2**:
    1. *Accessi Non Autorizzati*: Una delle minacce più gravi identificate per l'EC2 era il rischio di accessi non autorizzati. Questo problema, derivante da credenziali deboli, mal gestite o rubate, poteva permettere ad attori malevoli di accedere all'istanza, sottraendo dati sensibili o assumendo il controllo dell'infrastruttura. Ad esempio, un attaccante potrebbe utilizzare tecniche di brute force o sfruttare password esposte per accedere all'istanza.
    2. *Perdita o Compromissione dei Dati*: Un'altra minaccia critica riguardava la perdita o la compromissione dei dati sul cloud. Scenari come attacchi ransomware, che criptano i dati rendendoli inaccessibili, o violazioni di dati dovute a configurazioni di sicurezza inadeguate, erano considerati rischi significativi. Questi attacchi non solo potevano portare a perdite dirette di informazioni ma avrebbero anche potuto avere ripercussioni sulle operazioni aziendali e sulla reputazione.
- **Cluster Minikube**:
    1. *Privilegi Eccessivi dei Container*: Una configurazione impropria dei container, in particolare l'assegnazione di privilegi eccessivi, emergeva come una minaccia significativa. Questo scenario di rischio si manifestava quando i container erano configurati per operare con un livello di accesso più elevato del necessario. In tale contesto, un attore malintenzionato che riuscisse a compromettere un container poteva sfruttare questi privilegi per ottenere l'accesso root al sistema sottostante. Questo accesso elevato offriva agli attaccanti la possibilità di manipolare il cluster Minikube da un livello profondo, potenzialmente compromettendo l'intero ambiente.

2. *Uso di Immagini di Container Obsolete o Non Certificate*: L'uso di immagini vecchie o non certificate rappresenta un rischio significativo, poiché queste potrebbero contenere vulnerabilità non corrette, esponendo il cluster a potenziali attacchi. Per esempio, nel caso in cui un container operasse con un'immagine contenente una vulnerabilità nnota, sarebbe possibile per un attaccante sfruttare tale debolezza per infiltrarsi nel sistema. Un tale exploit non si limiterebbe a compromettere il container in questione, ma potrebbe anche permettere all'attaccante di spostarsi lateralmente all'interno del cluster. Ciò significherebbe che l'attaccante potrebbe estendere la propria presenza e accedere ad altre aree del sistema che potrebbero essere vulnerabili, amplificando così la portata e l'impatto dell'attacco.

- **Repository GitHub**:

1. *Accesso Non Autorizzato*: Minaccia derivante da credenziali compromesse, portando a modifiche illecite o divulgazione di dati.
2. *Modifiche Illecite del Codice*: Una volta ottenuto l'accesso, gli attori malevoli potrebbero manipolare il codice sorgente. Questo tipo di attacco non solo comprometterebbe l'integrità del codice, ma potrebbe anche avere ripercussioni a catena su tutto il sistema, specialmente se il codice modificato venisse poi utilizzato in produzione.
3. *Vulnerabilità nel Codice Sorgente*: Queste possono essere legate a errori di programmazione, all'uso di librerie non aggiornate o alla mancata attenzione verso le pratiche di buona programmazione. Se l'attaccante dovesse identificare queste debolezze, potrebbe sfruttarle per eseguire una serie di attacchi, dall'esecuzione di codice remoto al furto di dati, con impatti potenzialmente devastanti.
4. *Collegamento con Altre Componenti del Sistema*: essendo GitHub spesso integrato con altre componenti, come Jenkins, qualsiasi compromissione del repository potrebbe propagarsi rapidamente attraverso il sistema. Un esempio potrebbe essere un attacco in cui codice malevolo, introdotto nel repository, viene automaticamente distribuito attraverso la pipeline Jenkins, compromettendo così l'intero processo di distribuzione del software.

- **Pipeline Jenkins**:

  1. *Sicurezza dei Plugin*:l'utilizzo di plugin non aggiornati o contenente vulnerabilità potrebbe essere causa di attacchi esterni. Ad esempio, un plugin vulnerabile potrebbe essere sfruttato per eseguire codice dannoso, andando ad alterare il comportamento della pipeline o compromettendo i dati gestiti dal server. Questo potrebbe portare a una serie di conseguenze, come la distribuzione di applicazioni con componenti dannose o la violazione della sicurezza dell'intero processo di sviluppo.
  2. *Gestione delle Credenziali*: La gestione sicura delle credenziali è vitale in Jenkins, in quanto queste consentono l'accesso a risorse critiche durante il processo di CI/CD. Una gestione inadeguata, come l'uso di credenziali deboli, la loro esposizione nel codice o l'assegnazione impropria di permessi, può esporre la pipeline a rischi significativi. Per esempio, credenziali compromesse potrebbero permettere ad un attaccante di accedere ai processi di build e deploy, modificando il codice sorgente o accedendo a dati sensibili. Questo può avere implicazioni dirette sulla sicurezza dell'applicazione e sulla riservatezza delle informazioni trattate.

Mitigazione

La fase di mitigazione nel processo di Threat Modeling ha comportato lo sviluppo di strategie specifiche per affrontare le minacce identificate nelle varie componenti dell'architettura. Di seguito sono riportate le azioni di mitigazione per ciascuna componente:

- **Istanza AWS EC2**:

  1. Implementazione del 2FA (Autenticazione a due fattori) per rafforzare la sicurezza degli accessi.
  2. Uso del protocollo SSH per l'accesso remoto e conservazione sicura delle chiavi di accesso.
  3. Configurazione delle Security Rules per limitare l'accesso solo alle entità autorizzate.

4. Disabilitazione delle porte non utilizzate per ridurre i potenziali punti di ingresso per gli attaccanti.
5. Attivazione del logging delle attività per monitorare eventi di sicurezza e rilevare accessi sospetti.
6. Cifratura dei dati mantenuti su cloud per proteggerli da accessi non autorizzati.
7. Implementazione di un piano di backup dei dati per garantire la resilienza in caso di perdite o compromissioni.

- **Cluster Minikube**:

1. Mantenimento dell'aggiornamento alla versione più recente di Minikube per evitare lo sfruttamento di vulnerabilità note.
2. Adozione del principio di LEAST-PRIVILEGE, evitando l'esecuzione di container con privilegi di root.
3. Scanning regolare delle immagini per individuare e mitigare vulnerabilità prima della loro esecuzione.
4. Utilizzo di immagini di container affidabili e certificate per ridurre il rischio di vulnerabilità.
5. Configurazione dei permessi e delle autenticazioni all'interno del cluster per limitare l'accesso solo alle entità autorizzate.

- **Repository GitHub**:

1. Verifica che tutti i collaboratori utilizzino il 2FA per un controllo degli accessi più sicuro.
2. Limitazione dell'accesso al repository solo agli utenti autorizzati.
3. Utilizzo degli strumenti di GitHub per una gestione sicura delle informazioni riservate.
4. Impiego di strumenti di analisi statica e dinamica del codice per rilevare e mitigare le vulnerabilità.
5. Aggiornamento regolare delle dipendenze e delle librerie per prevenire l'uso di componenti vulnerabili.

6. Crittografia delle informazioni riservate per proteggerle in caso di compromissione del repository.

- **Pipeline Jenkins**:

1. Aggiornamento regolare dei plugin per assicurare che siano privi di vulnerabilità di sicurezza.
2. Gestione sicura delle credenziali e dei secret, evitando di inserirli direttamente nella pipeline.

5.2 Automazione del processo di Threat Modeling

Dopo aver completato il processo di Threat Modeling manuale su ciascun componente dell'architettura, è stata intrapresa la selezione dei controlli che era possibile automatizzare. È importante sottolineare che, nonostante la fattibilità di definire e automatizzare un numero maggiore di controlli, si è deciso di focalizzare l'automazione prevalentemente sulle configurazioni delle piattaforme di sviluppo e distribuzione, quali Jenkins, GitHub, AWS e Kubernetes.
In seguito, sono state integrate anche fasi di analisi del codice sorgente dell'applicativo distribuito, al fine di fornire un'analisi più completa tramite l'automazione, che tenesse in considerazione tutti gli aspetti rilevanti. Inoltre, una volta identificato il metodo per automatizzare un insieme di controlli relativi a una specifica piattaforma, ad esempio mediante l'uso di richieste API, si è optato per non aggiungere ulteriori controlli simili. Questo approccio è stato adottato per dimostrare la fattibilità dell'automazione del processo da diverse prospettive, piuttosto che mirare a un'analisi esaustiva.

5.2.1 Tool utilizzati per l'automazione

Kube-Bench

Kube-Bench è uno strumento open-source sviluppato da Aquasecurity. Viene utilizzato per valutare la sicurezza di implementazioni Kubernetes. Questo strumento è stato progettato per assistere gli amministratori di sistemi e

gli operatori Kubernetes nell'identificare e mitigare potenziali vulnerabilità e configurazioni non sicure all'interno dei propri cluster Kubernetes.

Le principali caratteristiche di kube-Bench includono:

- **Conformità al CIS Kubernetes Benchmark**: Kube-Bench esegue controlli e verifiche conformi alle raccomandazioni stabilite nel CIS (Center for Internet Security) Kubernetes Benchmark [17]. Questo benchmark rappresenta una guida di riferimento per garantire la sicurezza delle implementazioni Kubernetes.
- **Test Configurabili**: Gli utenti possono configurare i test utilizzando file YAML, consentendo personalizzazioni in base alle specifiche esigenze dell'implementazione Kubernetes. Questa flessibilità rende possibile aggiornare il tool seguendo le linee guida di sicurezza in evoluzione.
- **Segnalazione dei Risultati**: Una volta completati i test, Kube-Bench genera report che mostrano i risultati delle verifiche eseguite. Questi report possono essere utilizzati per identificare aree in cui il cluster Kubernetes potrebbe non essere configurato in modo sicuro.
- **Supporto per Diverse Versioni di Kubernetes**: Kube-Bench offre supporto per diverse versioni di Kubernetes, adattandosi a cluster in diverse fasi di sviluppo.
- **Open Source e Comunità Attiva**: Kube-Bench è un progetto open source con una comunità attiva di sviluppatori e utenti, offrendo accesso a supporto, aggiornamenti e la possibilità di contribuire al progetto in modo collaborativo.

Questo strumento può essere eseguito in vari modi, per esempio direttamente sul nodo di controllo o su tutti i nodi del cluster, fornendo un'ampia panoramica della sicurezza. C'è anche la possibilità di lanciare i test da remoto, collegandosi ai nodi tramite SSH.

Trivy

Trivy è uno strumento open-source sviluppato da Aquasecurity, che incorpora numerosi scanner di sicurezza progettati per individuare una vasta gamma

di problemi legati alla sicurezza. Trivy è progettato per individuare questi problemi in vari contesti e con obiettivi differenti.

Gli elementi analizzati da Trivy includono:

- Immagini dei container;
- File system;
- Repository Git;
- Cluster o risorse Kubernetes;

I vari scanner di Trivy analizzano:

- Pacchetti del sistema operativo e dipendenze software in uso (SBOM, Software Bill Of Materials);
- Vulnerabilità conosciute (CVE, Common Vulnerabilities and Exposures);
- Configurazioni errate o vulnerabili di file utilizzati per Iac (Infrastructure as Code) (per Kubernetes, Docker, Terraform e altri...);
- Informazioni sensibili e secret.

Le analisi effettuate da Trivy si basano sul database delle vulnerabilità di Aqua e altre numerose fonti. Questo database supporta aggiornamenti automatici, eliminando la necessità di dipendenze e middleware del database. Inoltre, Trivy può essere facilmente integrato nel processo di integrazione continua e distribuzione continua (CI/CD) mediante l'installazione dell'eseguibile nella macchina host della pipeline.

Kube-Hunter

Kube-Hunter è uno strumento open-source progettato per valutare la sicurezza delle infrastrutture basate su Kubernetes. Kubernetes è un sistema di orchestrazione dei container ampiamente utilizzato per distribuire, gestire e scalare applicazioni containerizzate. Kube-Hunter è stato sviluppato per

identificare e rilevare potenziali vulnerabilità e debolezze nella configurazione e nella sicurezza di un cluster Kubernetes.

Alcune delle principali caratteristiche di Kube-Hunter sono:

- **Scansione automatizzata**: Kube-hunter esegue scansioni automatiche e test di penetrazione contro un cluster Kubernetes per rilevare potenziali vulnerabilità e debolezze di sicurezza.
- **Supporto multi-strumento**: Utilizza una varietà di strumenti di scansione e test di penetrazione, tra cui nmap, nse, nikto, e altri, per eseguire ricerche su possibili punti deboli nel cluster.
- **Report dettagliati**: Fornisce report dettagliati che includono informazioni sulle vulnerabilità rilevate, le loro criticità e le raccomandazioni per affrontare i problemi rilevati.
- **Estensibilità**: Kube-Hunter è estendibile e consente agli utenti di personalizzare i test e aggiungere nuovi moduli per eseguire scansioni personalizzate.
- **Compatibilità con ambienti cloud**: Può essere utilizzato per scansionare cluster Kubernetes su diversi provider di servizi cloud, tra cui Amazon Web Services (AWS), Google Cloud Platform (GCP) e Microsoft Azure.

Ecco come può essere utilizzato Kube-Hunter:

- **Valutazione della sicurezza iniziale**: Kube-Hunter può essere utilizzato per eseguire una scansione iniziale di un cluster Kubernetes appena implementato per identificare rapidamente le potenziali vulnerabilità e le aree critiche da affrontare.
- **Auditing regolare**: È possibile utilizzare Kube-Hunter in modo regolare per effettuare audit di sicurezza periodici sui cluster Kubernetes, garantendo che i cambiamenti nella configurazione o nell'applicazione non abbiano introdotto nuove vulnerabilità.
- **Preparazione per test di penetrazione**: Prima di eseguire test di penetrazione più avanzati, Kube-Hunter può essere utilizzato per

individuare i punti deboli in modo da concentrare gli sforzi su aree specifiche.

- **Formazione e sensibilizzazione**: Kube-Hunter può essere utilizzato come strumento di formazione per educare gli amministratori di cluster Kubernetes e gli sviluppatori sulla sicurezza delle applicazioni containerizzate.

5.2.2 Configurazione della pipeline Jenkins

La pipeline Jenkins è organizzata in una serie di stage, ognuno dedicato a una specifica verifica. Come risultato di queste verifiche, gli stage producono due tipi di report:

- **Report di Dettaglio**: Ciascun stage genera un report dettagliato che elenca le vulnerabilità individuate o le configurazioni che necessitano di verifica. Questi report sono salvati in file distinti, nominati in base allo stage corrispondente, per facilitare l'analisi post-esecuzione da parte dell'operatore responsabile.

- **Report Riepilogativo**: In aggiunta, ogni stage produce un sommario che fornisce un'indicazione generale delle aree che potrebbero necessitare di attenzione, senza entrare nei dettagli specifici. Questo report funge da avviso immediato per l'operatore, che può poi consultare i report dettagliati per un'analisi più approfondita. Alla fine dell'esecuzione della pipeline, un report riepilogativo complessivo viene inviato via email all'operatore incaricato.

Si sottolinea l'importanza di non includere dettagli sensibili nel report riepilogativo per motivi di sicurezza, in modo da proteggere le informazioni in caso di accesso non autorizzato al contenuto dell'email. Questa misura precauzionale mira a ridurre il rischio di esposizione di dati sensibili a potenziali minacce.
Passiamo ora ad esaminare più dettagliatamente i singoli stage della pipeline.

1. **Preparazione del Report della Pipeline** Nella fase iniziale della pipeline Jenkins, vengono eseguiti due stage che contribuiscono alla preparazione del report finale. Questi due stage preliminari non sono focalizzati sui controlli di sicurezza, ma offrono una panoramica utile delle impostazioni e del contesto di ogni esecuzione della pipeline, contribuendo così alla documentazione completa del processo.

   - Nello stage *'Add Timestamp'*, viene registrato il momento esatto dell'esecuzione della pipeline. Si crea un timestamp che indica data e ora, e lo si inserisce nel file di report. Questo timestamp, accompagnato dal numero della build, serve a fornire un riferimento temporale per ogni esecuzione della pipeline.
   - Successivamente, lo stage *'Write stages log'* registra quali parametri della pipeline sono stati selezionati durante l'esecuzione. Questo include un'annotazione per ogni parametro, indicando se è stato attivato o meno. Queste informazioni vengono poi aggiunte al file di report.

2. **Controlli sulla Repository GitHub** Nella pipeline Jenkins, diversi stage si concentrano sul controllo della repository GitHub, ciascuno focalizzato su un aspetto specifico della sicurezza e della configurazione del repository.

   - Il primo stage, *'Repository: List the Github repository collaborators'*, elenca i collaboratori della repository GitHub. Questo controllo verifica i permessi di accesso di ciascun collaboratore, assicurando che solo gli utenti autorizzati abbiano l'accesso necessario.
   - Successivamente, lo stage *'Repository: Check the WebHooks configuration of the GitHub repository'* esamina la configurazione dei WebHooks della repository. Lo scopo è identificare eventuali WebHooks configurati con connessioni non sicure, un aspetto cruciale per prevenire potenziali vulnerabilità legate alla trasmissione dei dati.
   - Lo stage *'Repository: Check the repository visibility'* verifica se la repository è impostata come pubblica o privata. Tale controllo è fondamentale per garantire che le informazioni sensibili non siano esposte involontariamente.

- Infine, lo stage *'Repository: Git version check'* valuta la versione corrente di Git utilizzata nella repository, controllando la presenza di vulnerabilità note. Questo controllo aiuta a identificare le versioni di Git che potrebbero esporre il repository a rischi di sicurezza e suggerisce la necessità di aggiornamenti per mitigare tali rischi.

3. **Controlli su AWS** Nella pipeline Jenkins, vengono eseguiti diversi controlli specifici per AWS, ognuno mirato a garantire configurazioni sicure all'interno dell'infrastruttura cloud.

   - Lo stage *'AWS: Inspection of EC2 security rules'* verifica le regole di sicurezza associate alle istanze EC2. Lo script esegue un'analisi dettagliata delle regole di sicurezza configurate per ogni gruppo di sicurezza EC2, identificando quelle che non corrispondono a un elenco predefinito di CIDR consentiti. Le regole rilevate vengono poi elencate in un report, fornendo una panoramica delle possibili configurazioni di sicurezza che potrebbero richiedere un'attenzione particolare.
   - Il controllo *'AWS: Inspection of EC2 user MFA status'* si focalizza sulla verifica dello stato dell'autenticazione a più fattori (MFA) per gli utenti EC2. Lo script elenca tutti gli utenti di EC2 e controlla se hanno configurato l'MFA. Gli utenti senza MFA attivo vengono segnalati nel report, sottolineando potenziali lacune nella sicurezza dell'accesso.

   Questi stage rappresentano un approccio sistematico per garantire che le configurazioni di sicurezza all'interno di AWS siano conformi a standard stabiliti, contribuendo così a mantenere la sicurezza complessiva dell'ambiente cloud.

4. **Controlli di Sicurezza del Server Jenkins** Gli stage Jenkins qui descritti si focalizzano sulla verifica della sicurezza dei plugin Jenkins e sulla protezione del server Jenkins.

   - Lo stage *'Jenkins: Check Jenkins plugin updates'* esamina i plugin installati nel server Jenkins alla ricerca di aggiornamenti disponibili. Utilizzando un approccio automatizzato, lo script controlla ogni

plugin per identificare eventuali nuove versioni rispetto a quelle attualmente installate. Le informazioni sugli aggiornamenti disponibili vengono poi riportate in un report dettagliato.

- Il controllo successivo, *'Jenkins: Check vulnerabilities in Jenkins plugins'*, valuta i plugin Jenkins sotto l'aspetto della sicurezza. Questo stage identifica i plugin che presentano vulnerabilità note, verificando l'esistenza di versioni aggiornate che risolvono queste vulnerabilità. Anche in questo caso, le informazioni rilevanti vengono compilate in un report, fornendo un quadro chiaro dello stato di sicurezza dei plugin Jenkins.

Questi controlli sono cruciali per mantenere l'integrità e la sicurezza del server Jenkins, assicurando che sia protetto da vulnerabilità conosciute e sempre aggiornato con le ultime versioni dei plugin.

5. **Controlli sui Microservizi** Gli stage Jenkins dedicati ai microservizi si focalizzano principalmente sulla valutazione della sicurezza delle immagini Docker impiegate, utilizzando il tool Trivy per effettuare le scansioni.

   - Nello stage *'Microservices: Check Minikube image with Trivy'*, viene eseguita una scansione dell'immagine Docker di Minikube per identificare vulnerabilità di sicurezza. Trivy analizza l'immagine e produce un report in formato JSON, che viene poi elaborato per estrarre informazioni rilevanti sulla sicurezza.
   - Successivamente, lo stage *'Microservices: Check the Dockerfile of images with Trivy'* si concentra sui Dockerfile delle immagini Docker utilizzate dai microservizi. Anche in questo caso, Trivy è utilizzato per scandagliare i Dockerfile alla ricerca di potenziali vulnerabilità, generando dei report che forniscono una panoramica dettagliata dei rischi di sicurezza.
   - Infine, lo stage *'Microservices: Check Docker Images with Trivy'* estende la verifica di sicurezza a tutte le immagini Docker nel cluster, eseguendo scansioni su ciascuna per identificare vulnerabilità. Questo stage assicura che tutte le immagini Docker utilizzate nei microservizi siano state analizzate sotto il profilo della sicurezza, contribuendo a mantenere un ambiente di deploy sicuro.

In sintesi, questi stage rappresentano un approccio metodico per garantire che le immagini Docker utilizzate nei microservizi siano libere da vulnerabilità conosciute, contribuendo significativamente alla sicurezza dell'infrastruttura del cluster Minikube.

6. **Controlli del Cluster Kubernetes Minikube** Gli stage Jenkins si concentrano sulla verifica della configurazione e sulla rilevazione delle vulnerabilità del cluster Kubernetes, utilizzando gli strumenti Kube-Bench e Kube-Hunter.

   - Nello stage *'Microservices: Check Minikube cluster configuration with Kube-Bench'*, viene eseguita un'analisi della configurazione del cluster utilizzando Kube-Bench. Il tool viene lanciato attraverso un job Kubernetes, che viene applicato nel cluster e monitorato fino al suo completamento. Una volta terminato il job, i log vengono raccolti e analizzati per identificare eventuali mancanze nella configurazione del cluster rispetto alle best practice di sicurezza.
   - Lo stage *'Microservices: Check Minikube cluster vulnerabilities with Kube-Hunter'* utilizza Kube-Hunter per identificare vulnerabilità nel cluster. Kube-Hunter viene eseguito in modo simile a Kube-Bench, lanciando un job Kubernetes e attendendo il suo completamento. I risultati vengono poi raccolti dai log, con un focus particolare sulla rilevazione di vulnerabilità identificate dal tool. Questo stage fornisce una panoramica delle possibili debolezze di sicurezza presenti nel cluster.

   Questi controlli rappresentano un aspetto fondamentale nella gestione della sicurezza dei microservizi deployati, assicurando che la configurazione e l'ambiente del cluster Minikube siano conformi agli standard di sicurezza attuali.

7. **Invio del Report via Email** L'ultimo stage della pipeline Jenkins, *'Send report via email'*, è incaricato di inviare il report di sicurezza finale all'operatore responsabile. Questo stage rappresenta il passaggio conclusivo del processo di verifica della sicurezza.

   - Per l'invio del report via email, è stato necessario installare un plugin Jenkins specifico e configurare un server SMTP per consentire

a Jenkins di autenticarsi al server di posta elettronica. Lo stage utilizza il comando *emailext* per definire i dettagli dell'email da inviare, inclusi soggetto, corpo del messaggio, destinatario e mittente. Il report viene allegato all'email e inviato all'indirizzo specificato.

Capitolo 6

Risultati Ottenuti e Validazione

Questo capitolo è suddiviso in due sezioni. La prima sezione si concentra sulla presentazione dettagliata dei risultati ottenuti tramite l'automazione del processo di Threat Modeling. Qui, l'obiettivo è illustrare i risultati dei controlli realizzati sull'architettura, evidenziando come i dati siano stati rappresentati e organizzati nei diversi report generati.
La seconda parte del capitolo, invece, si dedica alla validazione di questi risultati automatizzati, andando a definire l'approccio adottato per la sua esecuzione e quanto emerso. In questo modo si vuole dimostrare la coerenza tra i risultati dell'automazione e quelli dell'analisi manuale, attestando così l'efficacia e la validità del processo automatizzato.

6.1 Risultati ottenuti dalla pipeline

Una volta terminata la configurazione, la pipeline di sicurezza è stata eseguita sull'architettura "funzionante", avente quindi l'istanza EC2 in esecuzione e i microservizi distribuiti all'interno del cluster Minikube attraverso il server Jenkins.
Le figure 6.1 e 6.2 riportano un'esecuzione della pipeline andata a buon fine.

Nel proseguo di questa sezione vengono mostrati i risultati ottenuti dalle analisi svolte da ogni stage della pipeline.

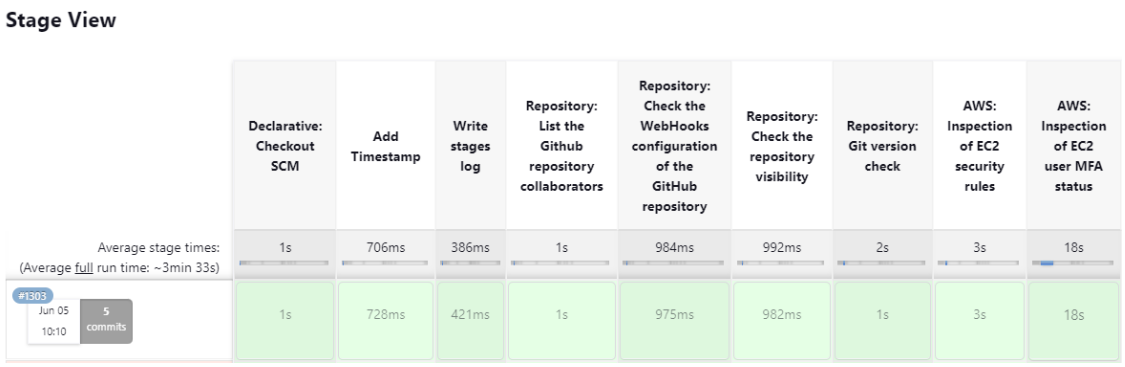


Figura 6.1. Pipeline per controlli di sicurezza completata con successo (Inizio)

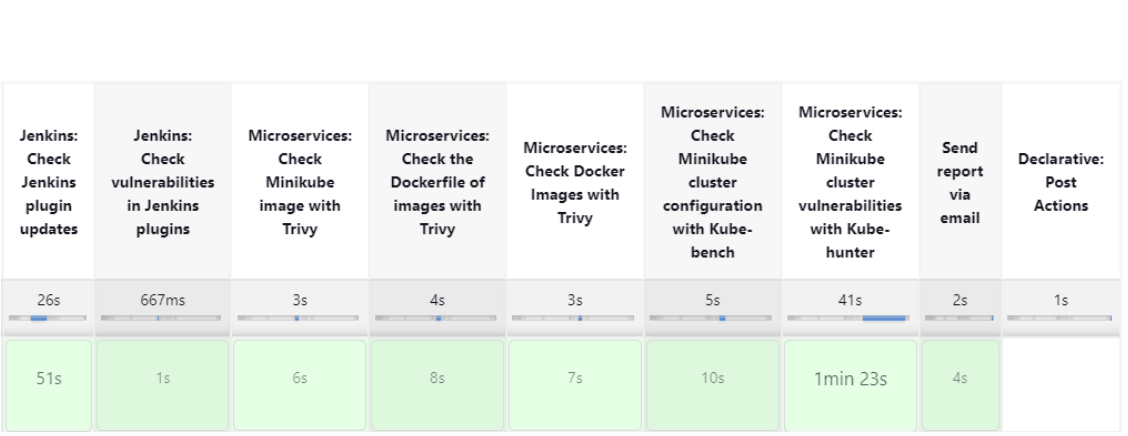


Figura 6.2. Pipeline per controlli di sicurezza completata con successo (Fine)

6.1.1 Risultati analisi repository GitHub

Repository: List the Github repository collaborators

L'analisi della sezione del report sui collaboratori del repository ha identificato due collaboratori e i loro specifici livelli di accesso. Sebbene queste informazioni non indichino direttamente la presenza di vulnerabilità, sono fondamentali per attività di monitoraggio, in quanto facilitano il rilevamento di accessi non autorizzati o potenzialmente illeciti.

```
||   Repository: List the Github repository collaborators   ||
============================================================
- xxxx has push, triage, pull access to the repository
- yyyy has admin, maintain, push, triage, pull access to the repository
```

Figura 6.3. Elenco dei collaboratori della repository

Repository: Check the WebHooks configuration of the GitHub repository

Il report generato dal rispettivo stage evidenzia la presenza di un WebHook configurato in maniera errata o insicura. Come per lo stage precedente, anche qui le informazioni sono riportate solamente nel report riepilogativo, poichè successivamente sarà l'operatore incaricato delle correzioni ad ispezionare il repository gitHub e verificare/correggere la configurazione dei WebHooks.

```
||   Repository: Check the WebHooks configuration of the GitHub repository   ||
================================================================================
- (insecure) WebHook URL: https://requestbin.com/       Updated at: 2023-05-16T07:46:50Z
```

Figura 6.4. Risultati analisi sui WebHooks del repository

Repository: Check the repository visibility

L'informazione riportata dallo stage sulla verifica della visibilità del repository mostra che la repository contente il codice sorgente risulta essere pubblica. Questo genere di informazione è importante, poichè nel caso di un ambiente di lavoro reale è assolutamente sconsigliato che il codice sorgente sia visibile all'esterno dell'organizzazione.

Repository: Git version check

La sezione del report riepilogativo relativa a questo stage, figura 6.6, evidenzia la presenza di vulnerabilità nella versione di git installata sull'istanza EC2. Invece, nella figura 6.7 è riportato quanto presente nel report di dettaglio prodotto dal controllo, che fornisce le informazioni sulla versione di Git

```
||   Repository: Check the repository visibility   ||
=======================================================
- The GitHub repository is public.
```

Figura 6.5. Risultati sulla verifica della visibilità del repository

installata,sulle vulnerabilità rilevate e sulle fonti da cui sono state ottenute queste informazioni.

```
||   Repository: Git version check   ||
=====================================
- Git has 12 vulnerabilities.
```

Figura 6.6. Risultato analisi versione di Git installata

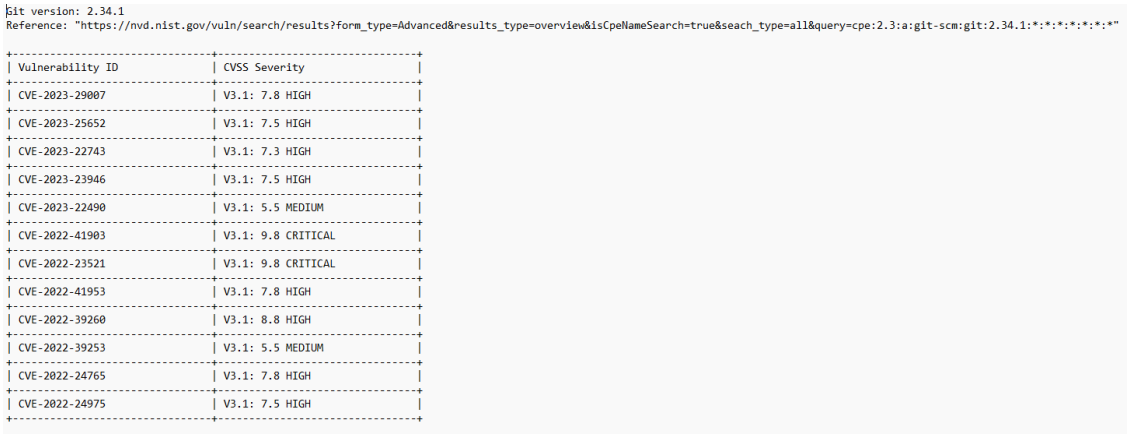
6.1.2 Risultati analisi AWS

AWS: Inspection of EC2 security rules

L'analisi delle regole di sicurezza configurate per l'istanza EC2 ha rivelato un totale di 6 regole che richiedono attenzione. Questo numero deriva dal fatto che per ognuna di queste 6 regole, l'indirizzo IP autorizzato a interagire con l'istanza EC2 non appartiene agli indirizzi IP interni dell'organizzazione. In altre parole, come dimostrato nel report di dettaglio 6.8, queste regole sono state configurate per consentire l'accesso all'istanza EC2 da qualsiasi indirizzo IP esterno, come evidenziato dall'uso dell'indirizzo IP '0.0.0.0'.

AWS: Inspection of EC2 user MFA status

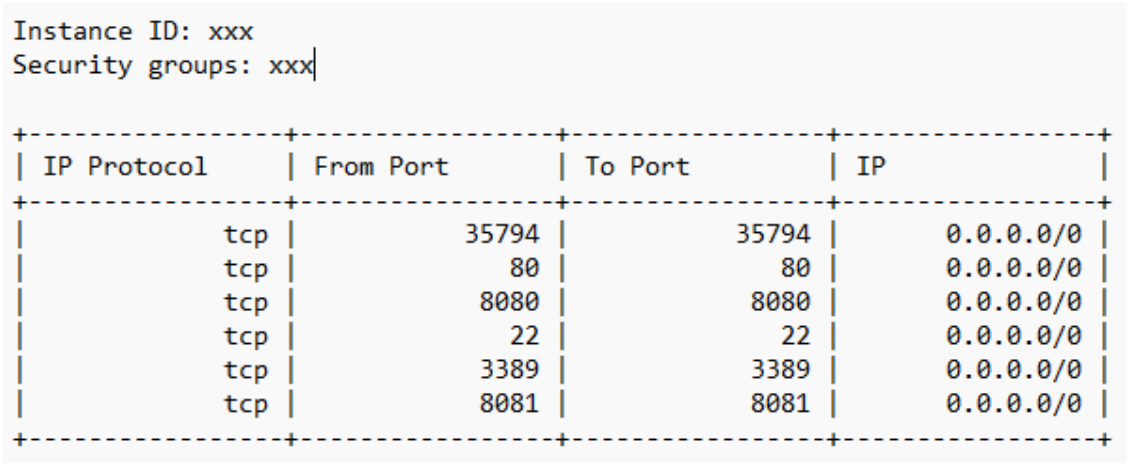
La sezione del report riepilogativo sul controllo dell'autenticazione degli utenti per l'accesso all'istanza EC2 ha rivelato che tre utenti non hanno attivato l'autenticazione a più fattori (MFA). Questo dettaglio è cruciale in quanto la mancanza di MFA aumenta il rischio di frodi e compromissione dell'account,



```
Git version: 2.34.1
Reference: "https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:a:git-scm:git:2.34.1:*:*:*:*:*:*:*"

+---------------------------+-----------------------------+
| Vulnerability ID          | CVSS Severity               |
+---------------------------+-----------------------------+
| CVE-2023-29007            | V3.1: 7.8 HIGH              |
+---------------------------+-----------------------------+
| CVE-2023-25652            | V3.1: 7.5 HIGH              |
+---------------------------+-----------------------------+
| CVE-2023-22743            | V3.1: 7.3 HIGH              |
+---------------------------+-----------------------------+
| CVE-2023-23946            | V3.1: 7.5 HIGH              |
+---------------------------+-----------------------------+
| CVE-2023-22490            | V3.1: 5.5 MEDIUM            |
+---------------------------+-----------------------------+
| CVE-2022-41903            | V3.1: 9.8 CRITICAL          |
+---------------------------+-----------------------------+
| CVE-2022-23521            | V3.1: 9.8 CRITICAL          |
+---------------------------+-----------------------------+
| CVE-2022-41953            | V3.1: 7.8 HIGH              |
+---------------------------+-----------------------------+
| CVE-2022-39260            | V3.1: 8.8 HIGH              |
+---------------------------+-----------------------------+
| CVE-2022-39253            | V3.1: 5.5 MEDIUM            |
+---------------------------+-----------------------------+
| CVE-2022-24765            | V3.1: 7.8 HIGH              |
+---------------------------+-----------------------------+
| CVE-2022-24975            | V3.1: 7.5 HIGH              |
+---------------------------+-----------------------------+
```

Figura 6.7. Report di dettaglio dell'analisi sulla versione di Git installata



```
Instance ID: xxx
Security groups: xxx

+------------------+------------------+------------------+------------------+
| IP Protocol      | From Port        | To Port          | IP               |
+------------------+------------------+------------------+------------------+
|              tcp |            35794 |            35794 |        0.0.0.0/0 |
|              tcp |               80 |               80 |        0.0.0.0/0 |
|              tcp |             8080 |             8080 |        0.0.0.0/0 |
|              tcp |               22 |               22 |        0.0.0.0/0 |
|              tcp |             3389 |             3389 |        0.0.0.0/0 |
|              tcp |             8081 |             8081 |        0.0.0.0/0 |
+------------------+------------------+------------------+------------------+
```

Figura 6.8. Risultati dell'analisi sulle Security Rules dell'istanza EC2

esponendo potenzialmente a rischi l'intera istanza EC2 e le risorse accessibili dall'utente.

```
||   AWS: Inspection of EC2 user MFA status   ||
================================================
- 3 user(s) do not have MFA enabled.
```

Figura 6.9. Risulati dell'analisi sul livello d'autenticazione degli utenti AWS

6.1.3 Risultati analisi Jenkins

Jenkins: Check Jenkins plugin updates

L'analisi sulle versioni dei plugin installati sul server Jenkins ha rivelato che 83 di essi dispongono di aggiornamenti. Sebbene questa scoperta non implichi direttamente la presenza di vulnerabilità, resta un dato rilevante: mantenere i plugin aggiornati è essenziale per ridurre il rischio di exploit basati su vulnerabilità non ancora identificate o rese pubbliche.
Nella figura 6.10 viene mostrata una sezione del report dettagliato, dove sono elencati i nomi dei plugin, le versioni correntemente installate, le versioni più recenti disponibili e le informazioni sul sito web da cui è possibile effettuare l'aggiornamento.

Plugin Name	Installed Version	Latest Version	URL
blueocean-pipeline-scm-api	1.27.2	1.27.4	https://plugins.jenkins.io/blueocean-pipeline-scm-api/
workflow-job	1282.ve6d865025906	1301.v054d9cea_9593	https://plugins.jenkins.io/workflow-job/
github	1.37.0	1.37.1	https://plugins.jenkins.io/github/
authentication-tokens	1.4	1.53.v1c90fd9191a_b_	https://plugins.jenkins.io/authentication-tokens/
blueocean-pipeline-api-impl	1.27.2	1.27.4	https://plugins.jenkins.io/blueocean-pipeline-api-impl/
command-launcher	90.v669d7ccb_7c31	100.v2f6722292ee8	https://plugins.jenkins.io/command-launcher/

Figura 6.10. Report di dettaglio dei plugin Jenkins da aggiornare

Jenkins: Check vulnerabilities in Jenkins plugins

L'analisi delle vulnerabilità nei plugin di Jenkins ha evidenziato che 7 di essi sono vulnerabili, e per 5 di questi è disponibile un aggiornamento che ne corregge la vulnerabilità.

La figura 6.11, rappresentante il report di dettaglio, fornisce una chiara indicazione sui plugin interessati, descrivendo le specifiche vulnerabilità identificate, le versioni implicate e, per quelli applicabili, la disponibilità di un aggiornamento correttivo. Inoltre, viene fornito l'URL del sito da cui sono state ottenute queste informazioni.

Plugin Name	Installed Version	Vulnerability	Affected Version	URL
workflow-job	1282.ve6d865025906	Stored XSS vulnerability	1292.v27d8cc3e2602 and earlier	https://plugins.jenkins.io/workflow-job/
email-ext	2.94	Missing permission check	2.96 and earlier	https://plugins.jenkins.io/email-ext/
email-ext	2.94	CSRF vulnerability	2.96 and earlier	https://plugins.jenkins.io/email-ext/
ldap	659.v8ca_b_a_fe79fa_d	CSRF vulnerability	673.v034ec70ec2b_b_ and earlier	https://plugins.jenkins.io/ldap/
recipe	1.2	CSRF vulnerability and missing permission checks allow XXE	latest	https://plugins.jenkins.io/recipe/
kubernetes	3893.v73d36f3b_9103	Improper masking of credentials	3909.v1f2c633e8590 and earlier	https://plugins.jenkins.io/kubernetes/
pipeline-utility-steps	2.15.1	Arbitrary file write vulnerability on agents	2.15.2 and earlier	https://plugins.jenkins.io/pipeline-utility-steps/
ec2-deployment-dashboard	1.0.10	Password stored in plain text	latest	https://plugins.jenkins.io/ec2-deployment-dashboard/
ec2-deployment-dashboard	1.0.10	Missing permission checks allow enumerating credentials IDs	latest	https://plugins.jenkins.io/ec2-deployment-dashboard/
ec2-deployment-dashboard	1.0.10	CSRF vulnerability and missing permission checks	latest	https://plugins.jenkins.io/ec2-deployment-dashboard/
ec2-deployment-dashboard	1.0.10	Stored XSS vulnerability	latest	https://plugins.jenkins.io/ec2-deployment-dashboard/

Figura 6.11. Report di dettaglio dei plugin Jenkins con vulnerabilità

6.1.4 Risultati analisi Microservizi

I risultati dell'analisi con Trivy generano due tipi di report: uno in formato JSON prodotto dallo strumento stesso, e un altro in formato testuale incluso nel report riepilogativo. Quest'ultimo report dettaglia le vulnerabilità per livello di criticità e per layer dell'immagine Docker analizzata. Un esempio di questo può essere visto in figura 6.12, dove viene mostrata l'analisi di un specifico layer dell'immagine Docker di Minikube.

```
||   Microservices: Check Minikube image with Trivy   ||
=======================================================
- VULNERABILITY of the 67a4b1138d2d (ubuntu 20.04) layer of the Docker image gcr.io/k8s-minikube/kicbase:v0.0.39
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| TOTAL           | CRITICAL        | HIGH            | MEDIUM          | LOW             | UNKNOWN         |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| 329             | 0               | 0               | 201             | 126             | 0               |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
```

Figura 6.12. Esempio report riepilogativo dell'analisi di Trivy

Microservices: Check Minikube image with Trivy

Dall'analisi condotta con Trivy sull'immagine Docker di Minikube, è emerso che essa presenta un totale di 400 vulnerabilità. Tra queste, 1 è classificata come critica (Critical), 35 come di alto rischio (High), mentre le restanti si distribuiscono tra livelli medio (Medium) e basso (Low) di pericolosità.

Microservices: Check the Dockerfile of images with Trivy

L'esame dei Dockerfile relativi ai diversi microservizi ha rivelato due problematiche ricorrenti in ciascuno di essi: una di gravità alta (High) e una di gravità bassa (Low). La prima è connessa all'utilizzo della modalità root per l'esecuzione dell'immagine, mentre la seconda riguarda la mancanza di un comando per monitorare lo stato attivo del container.

Microservices: Check Docker Images with Trivy

L'analisi delle immagini Docker dei diversi microservizi ha evidenziato un numero significativo di vulnerabilità. In dettaglio, ciascun microservizio evidenzia 265 vulnerabilità, categorizzate in 26 di livello critico (Critical), 99 di livello alto (High), 83 di livello medio (Medium), 47 di livello basso (Low) e 10 di tipo non identificato (Unknown). La causa principale di queste vulnerabilità è l'utilizzo di codice sorgente da un repository pubblico non aggiornato da tempo, il quale utilizza una versione obsoleta di Java e librerie non aggiornate.

6.1.5 Risultati analisi Cluster Minikube

Microservices: Check Minikube cluster configuration with Kube-Bench

L'analisi effettuata con il tool Kube-Bench sul cluster minikube ha evidenziato risultati misti. Dall'insieme di 124 test effettuati sui diversi componenti del cluster per valutarne la configurazione, 59 sono stati superati, 13 sono falliti e 52 hanno generato un avviso (warning), indicando la necessità di un'ulteriore verifica manuale delle impostazioni. I test superati confermano la correttezza della configurazione del cluster. Tuttavia, i test falliti o con

avvisi sono attribuibili al fatto che Kube-Bench è progettato per cluster Kubernetes con configurazioni infrastrutturali differenti da quelle di minikube. Di conseguenza, l'applicazione di questo strumento su minikube ha prodotto risultati che non riflettono pienamente la configurazione reale del sistema.

Microservices: Check Minikube cluster vulnerabilities with Kube-Hunter

Dal report prodotto da Kube-Hunter è possibile constatare la presenza di 6 vulnerabilità presenti nel cluster Minikube. Come è possibile vedere dalla figura 6.13, in cui viene mostrato un pezzo d'esempio del report generato dal tool, per ogni vulnerabilità sono riportate diverse informazioni sulla vulnerabilità, tra cui una evidenza che ne dimostra l'exploit e le informazioni ricavate da questo e l'id con cui poter vedere altri dettagli sulla vulnerabilità accedendo al repository di aquasec.
E' importante sottolineare che anche in questo caso le differenze tra Minikube e Kubernetes hanno fatto sì che le vulnerabilità rilevate siano relative alla cungifurazione di test di Minikube, e che non avendo una versione di Kubernetes questa analisi non risultà verificabile nella pratica.

6.2 Validazione risultati

Il processo di validazione dei risultati si è articolato in due distinte fasi.
Inizialmente, è stato effettuato un confronto tra i risultati emersi dalla pipeline Jenkins e quelli dell'analisi manuale, allo scopo di verificare la coerenza e la consistenza dei risultati prodotti dall'automazione.
Successivamente, è stata intrapresa una seconda fase di validazione, durante la quale sono state implementate remediation sulle configurazioni e sui componenti dell'architettura. Dopo ogni intervento di correzione, è stata rieseguita la pipeline per assicurarsi che i risultati riflettessero fedelmente le modifiche apportate.

Vulnerabilities
For further information about a vulnerability, search its ID in:
https://avd.aquasec.com/

ID	LOCATION	MITRE CATEGORY	VULNERABILITY	DESCRIPTION	EVIDENCE
None	Local to Pod (kube-hunter-686df)	Lateral Movement // ARP poisoning and IP spoofing	CAP_NET_RAW Enabled	CAP_NET_RAW is enabled by default for pods. If an attacker manages to compromise a pod, they could potentially take advantage of this capability to perform network attacks on other pods running on the same node	
KHV002	10.96.0.1:443	Initial Access // Exposed sensitive interfaces	K8s Version Disclosure	The kubernetes version could be obtained from the /version endpoint	v1.26.3
KHV053	Local to Pod (kube-hunter-686df)	Discovery // Instance Metadata API	AWS Metadata Exposure	Access to the AWS Metadata API exposes information about the machines associated with the cluster	cidr: 172.31.32.0/20

Figura 6.13. Struttura del report di Kube-Hunter

6.2.1 Fase 1: Confronto tra Analisi Manuale e Automatica

Validazione dei Controlli AWS: Per la validazione dei controlli AWS relativi all'MFA e alle security rules, è stato sufficiente accedere alla console AWS. È stato verificato se gli utenti con accesso all'account avessero attivato l'MFA. Successivamente, all'interno del security group collegato all'istanza EC2, sono state esaminate le regole configurate, focalizzandosi sui protocolli, le porte e gli indirizzi IP autorizzati. In entrambe le situazioni, i risultati ottenuti hanno mostrato coerenza con le informazioni riportate nel report, confermando l'accuratezza dei controlli.

Validazione delle informazioni sulla Repository GitHub: Nel corso di questa fase di validazione è stato fatto un controllo sul repository, così da verificare le vare impostazioni. Quindi, sono state esaminate le impostazioni di visibilità, l'elenco dei collaboratori, compresi i relativi diritti d'accesso,

e i webhooks configurati, al fine di individuare eventuali irregolarità. Infine, dalla console dell'istanza ec2 ho verificato la versione di git installata e dal sito del NIST verificato la presenza di vulnerabilità per quella specifica versione. Ciascuna di queste verifiche ha prodotto risultati positivi, confermando l'aderenza dei dati ottenuti manualmente ai risultati riportati nel report automatizzato.

Validazione Jenkins: Per la validazione dei plugin Jenkins, è stato effettuato un controllo su tutti i plugin da aggiornare, verificando la nuova versione disponibile. Queste informazioni sono risultate in linea con quanto riportato nel report dettagliato sugli aggiornamenti disponibili. Per quanto riguarda i plugin vulnerabili, è stata consultata la documentazione ufficiale di ciascun plugin per accertare la presenza di tutte le vulnerabilità identificate sulla versione corrente e per verificare la disponibilità di aggiornamenti che potessero risolvere tali problemi. Anche in questo contesto, le informazioni riportate nel report si sono confermate complete e veritiere.

Validazione dei controlli sui Microservizi: Nell'analisi dei risultati relativi ai microservizi, è stata verificata l'accuratezza dei report prodotti da Trivy, sia per le vulnerabilità che per le configurazioni errate identificate. Per i Dockerfile, è stato esaminato direttamente il loro contenuto, confermando la presenza delle configurazioni rischiose segnalate. Invece, nella validazione dell'analisi statica del codice dei microservizi, vista l'alta quantità di vulnerabilità riportate, si è concentrata l'attenzione sulle vulnerabilità più facilmente identificabili, cioè quelle legate all'utilizzo di librerie non aggiornate o obsolete. Tale verifica ha confermato l'esistenza delle vulnerabilità analizzate.

Validazione dei controlli sulle Configurazioni del Cluster Minikube: Per la validazione dei controlli sulle configurazioni del cluster Minikube, è stato adottato un approccio differente. Si è proceduto raccogliendo e verificando manualmente tutti i test eseguiti da Kube-Bench, per assicurare che i risultati corrispondessero alle effettive configurazioni del cluster. Questo processo ha incluso l'accesso al filesystem dell'immagine Minikube, al fine di esaminare i percorsi, i permessi, le ownership dei file e i vari parametri

impostati che erano oggetto dei test. È emerso che i risultati dei test superati hanno accuratamente riflettuto le configurazioni del cluster. Tuttavia, alcuni test falliti non hanno fornito indicazioni affidabili, in gran parte a causa di percorsi e impostazioni predefiniti per Kubernetes che non sono direttamente applicabili in un contesto Minikube

6.2.2 Fase 2: Validazione Post-Remediation

In questa fase del lavoro, è stato eseguito l'applicazione di remediation esemplificative al fine di verificare eventuali cambiamenti nei risultati del report, assicurando che fossero in linea con le modifiche apportate.

Validazione Post-Remediation per AWS: Per la rivalidazione relativa ad AWS, sono state apportate modifiche alle security rules dell'istanza. Inizialmente, sono state esaminate le regole per identificare i protocolli effettivamente utilizzati, con l'obiettivo di disattivare le porte non utilizzate. Successivamente, per ciascuna regola, sono stati rivisti gli indirizzi IP autorizzati per l'accesso, limitandoli esclusivamente agli indirizzi IP dell'organizzazione. Dopo aver apportato queste modifiche, è stata eseguita nuovamente la pipeline di sicurezza.
Il risultato ottenuto dalla pipeline è stato conforme alle aspettative, come evidenziato nella figura 6.14.

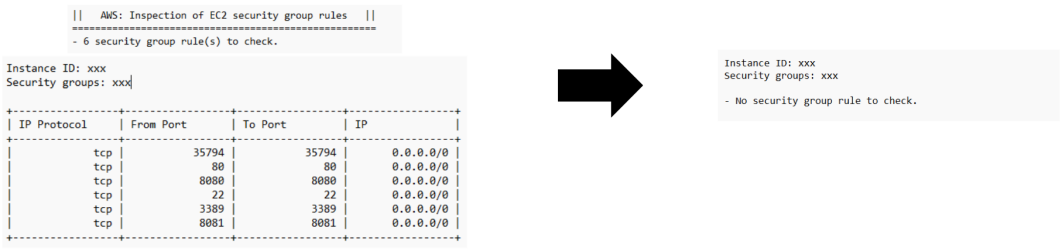


Figura 6.14. Risultati post-remediation su AWS

Validazione Post-Remediation per GitHub Repository: Nella rivalidazione dei controlli sul repository, sono state apportate alcune modifiche.

Inizialmente, sono stati rimossi tutti i collaboratori tranne uno e sono state apportate modifiche all'impostazione del webhook, introducendo una fase di autenticazione per le richieste inviate ad esso. In seguito, è stata aggiornata la versione di Git all'ultima disponibile e, a questo punto, l'intera pipeline è stata nuovamente eseguita, dando risultati in linea con le modifiche apportate.

Figura 6.15. Risultati post-remediation su Repository GitHub

Validazione Post-Remediation per Jenkins: Per quanto riguarda la rivalidazione dei controlli sui plugin Jenkins, è stata effettuata un'operazione di aggiornamento di tutti i plugin esistenti. Successivamente, è stata condotta una verifica per identificare la presenza di plugin vulnerabili. Nel caso in cui fossero stati rilevati plugin vulnerabili, sono stati presi provvedimenti adeguati, tra cui la loro disinstallazione o la loro conservazione in base alle necessità. Anche in questo caso, come visibile dalla figura in basso 6.16, i risultati hanno rispecchiato le modifiche.

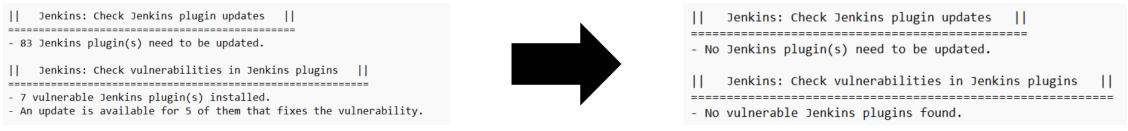


Figura 6.16. Risultati post-remediation su Jenkins

Validazione Post-Remediation per Microservizi e immagine Minikube: Nella rivalidazione dei controlli effettuati tra Trivy sui microservizi e sull'immagine di Minikube, sono state apportate le seguenti modifiche:

1. È stato modificato il dockerfile di ognuno, introducendo un utente e un controllo sull'operatività del container. I dettagli di questa modifica sono illustrati nella figura 6.17, in cui è presente anche un confronto tra il prima e il dopo.

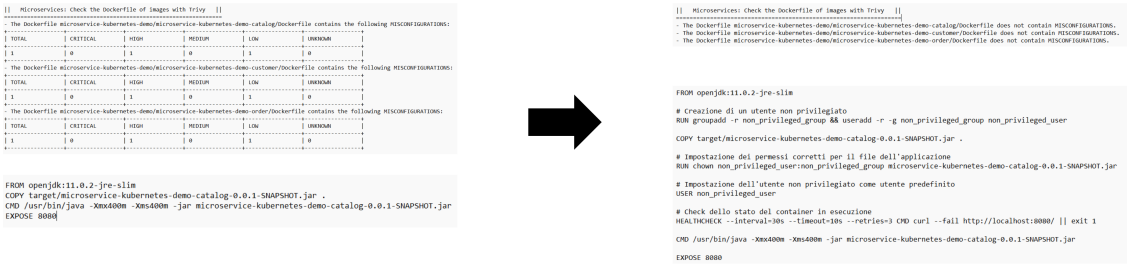


Figura 6.17. Risultati post-remediation Dockerfile dei microservizi

2. È stata effettuata l'aggiornamento dell'immagine utilizzata per Minikube dalla versione 1.29.0 alla versione 1.30.1, che era l'ultima disponibile al momento della modifica. Successivamente, il cluster è stato riavviato e l'applicativo è stato ridistribuito al suo interno.

3. È stata apportata una correzione alle dipendenze, aggiornandole in base alle compatibilità con il resto del codice.

Apportando queste modifiche e rieseguendo la pipeline si è constatato che i problemi riguardanti i Dockerfile sono stati risolti, mentre le vulnerabilità rilevate nel resto del sistema sono diminuite, non essendo appunto stata fatta su questa parte una remediation completa.

Validazione Post-Remediation per Cluster Minikube In questa fase, sono state apportate modifiche alle configurazioni e ai permessi identificati come potenzialmente problematici, che precedentemente avevano causato il fallimento dei test. Successivamente, è stato rieseguito il tool Kube-Bench. È importante notare che, nonostante questa fase non abbia coperto tutte

le possibili correzioni necessarie, poiché alcuni risultati di test in stato di allerta o falliti erano dovuti all'uso di Minikube anziché di Kubernetes, le modifiche apportate sono state rilevate correttamente dal tool e riportato coerentemente nel report.

Capitolo 7

Conclusioni e Lavori Futuri

Nei capitoli precedenti è stato esplorato in maniera dettagliata il percorso che ha guidato il lavoro di tesi, iniziando dalla definizione del problema, ovvero l'ottimizzazione del processo di monitoraggio delle minacce in un'architettura cloud, per giungere alla progettazione e implementazione di una soluzione concreta: una pipeline Jenkins che automatizzasse una serie di controlli di sicurezza. Da questo percorso è emerso chiaramente come l'automazione dei processi rappresenti una scelta efficace e strategica nel contesto del monitoraggio della sicurezza, spostando l'enfasi da attività ripetitive e manuali a meccanismi automatizzati. Questo cambio di paradigma libera gli operatori da compiti onerosi, permettendo loro di concentrarsi su aspetti più strategici e reattivi della sicurezza informatica.
In questo capitolo conclusivo si intende riflettere su quanto emerso dal lavoro svolto. Verranno messe in luce le principali forze del modello proposto, esaminando i vantaggi offerti dalla pipeline Jenkins. Si discuteranno inoltre le limitazioni intrinseche dell'approccio adottato, delineando le sfide ancora presenti.

7.1 Punti di Forza

In questa sezione si esplorano i vantaggi derivanti dall'uso di una pipeline Jenkins per automatizzare i controlli di sicurezza, evidenziando come l'approccio proposto ottimizzi il monitoraggio e la gestione della sicurezza nelle varie piattaforme:

1. **Riduzione dei Tempi Operativi**: Con la pipeline Jenkins i controlli di sicurezza vengono configurati una sola volta e applicati automaticamente a tutte le piattaforme. Questo sistema centralizzato consente di evitare configurazioni ripetitive e controlli manuali su ogni singola piattaforma, riducendo drasticamente i tempi di esecuzione. Inoltre, avendo un unico punto di configurazione, il monitoraggio risulta essere più facilmente gestibile, contribuendo ulteriormente all'efficienza e all'efficacia complessiva del processo.

2. **Riduzione degli Errori Umani**: L'automazione fornita dalla pipeline Jenkins porta a una significativa riduzione degli errori umani. Separando gli operatori dai compiti ripetitivi e soggetti a errore, permette loro di concentrarsi su incarichi più strategici e di alto livello. Questo spostamento non solo aumenta l'efficienza operativa, ma migliora anche la qualità e l'affidabilità dei controlli di sicurezza.

3. **Flessibilità e Personalizzazione**: Grazie alla schedulazione e ai parametri personalizzabili di ogni fase, l'analisi automatica diventa non solo ottimizzata ma anche adattabile. È possibile variare la frequenza di controllo in base alle necessità e focalizzarsi su componenti specifici, garantendo così un monitoraggio mirato e efficiente.

4. **Uniformità e Coerenza dei Risultati**: L'introduzione dell'automazione nel processo non solo garantisce la precisione dei risultati, ma assicura anche che questi siano presentati in modo uniforme e coerente nel corso del tempo. Inoltre, questa standardizzazione facilita i confronti tra diversi set di dati raccolti in momenti differenti, agevolando così la produzione di statistiche utili a comprendere le evoluzioni e le tendenze nel tempo.

5. **Versatilità**: La pipeline Jenkins, pur essendo strutturata in modo statico, offre una grande versatilità grazie alla possibilità di modificare parametri, come il numero dell'istanza e le credenziali di accesso delle piattaforme. Questa flessibilità la rende adatta non solo al contesto specifico della tesi, ma anche applicabile a una vasta gamma di contesti e architetture diverse. La capacità di adattarsi a vari ambienti e requisiti tecnici sottolinea la robustezza e l'efficacia del metodo proposto.

7.2 Limiti e Considerazioni

Nonostante i vantaggi, ci sono alcune limitazioni riguardanti il lavoro, da considerare:

- **Architettura All-in-one**: La pipeline di automazione è stata realizzata sulle caratteristiche di un'architettura di prova in cui i vari componenti risiedevano sulla stessa infrastruttura cloud e all'interno della stessa rete. Questo ha notevolmente semplificato le interazioni tra le piattaforme: Jenkins, ad esempio, non solo esegue i controlli di sicurezza all'interno del suo stesso account AWS, ma è anche alloggiato sull'istanza EC2 che ospita il cluster Minikube monitorato. Inoltre, lo stesso account Jenkins viene utilizzato sia per la pipeline di sicurezza che per la distribuzione dei microservizi nel cluster, permettendo a Jenkins di effettuare controlli di sicurezza sui propri plugin. Questa configurazione ha reso più agevole la gestione della comunicazione e dell'accesso alle informazioni necessarie per i controlli di sicurezza. Tuttavia, bisogna considerare che un'architettura così integrata e compatta, sebbene efficiente per scopi dimostrativi, potrebbe non riflettere pienamente la complessità e le sfide di ambienti di produzione più estesi e distribuiti.

- **Accesso a Informazioni Sensibili**: Un aspetto cruciale della mia pipeline di automazione riguarda la necessità di accedere a informazioni sensibili per eseguire i vali controlli di sicurezza. Queste informazioni, che possono includere dati di configurazione, credenziali di accesso e dettagli riguardanti l'infrastruttura interna, sono spesso considerate riservate dalle organizzazioni. Molte aziende, per ragioni di sicurezza e privacy, sarebbero quindi riluttanti a condividere tali dati, specialmente

con entità esterne come fornitori di servizi o consulenti di sicurezza. Di conseguenza, l'implementazione di questo tipo di monitoraggio si presenta più fattibile all'interno della stessa organizzazione, dove l'accesso a questo tipo di informazioni è più facilmente gestibile e controllato. Tuttavia, estendere questo modello di monitoraggio a clienti esterni presenta sfide significative. Non solo sarebbe necessario instaurare un elevato livello di fiducia, ma anche implementare rigide misure di sicurezza e protocolli di condivisione dati per garantire che le informazioni sensibili siano protette adeguatamente durante tutto il processo di monitoraggio.

- **Uso di Minikube**: Nel lavoro di tesi è stato utilizzato Minikube per emulare un ambiente Kubernetes. Tuttavia, gli strumenti di sicurezza impiegati, Kube-Bench e Kube-Hunter, sono concepiti per scenari tipici di Kubernetes, che normalmente opera su configurazioni multi-nodo, mentre Minikube è ideato per esecuzioni locali su un unico nodo virtuale. Questa differenza infrastrutturale non è trascurabile: ci sono aspetti chiave, come la distribuzione di carichi di lavoro e la gestione della rete, centrali in Kubernetes, che non trovano riscontro in Minikube. Di conseguenza, l'affidabilità dei test condotti con questi strumenti sul cluster Minikube risulta essere limitata, poiché questi presuppongono una complessità che il simulatore non possiede. In sintesi, pur fornendo indicazioni valide sulla conformità e sulle vulnerabilità, l'efficacia di Kube-Bench e Kube-Hunter risulta ridotta nell'ambito semplificato offerto da Minikube.

- **Necessità di Maggiori Controlli**: La pipeline corrente, nonostante integri controlli per varie piattaforme, presenta una limitazione sostanziale nella mancanza di un'analisi dettagliata per ciascuna di esse. Attualmente, mentre vengono effettuati controlli generali, si riscontra una carenza nell'esplorazione approfondita delle singole piattaforme. Questa situazione porta a una copertura parziale e superficiale nella sorveglianza della sicurezza, lasciando potenzialmente scoperte aree critiche e vulnerabilità specifiche. La mancanza di un monitoraggio minuzioso per ogni piattaforma implica che importanti aspetti di sicurezza potrebbero non essere identificati o valutati adeguatamente. Di conseguenza, emerge la

necessità urgente di arricchire la pipeline con un approccio più focalizzato e dettagliato, che possa garantire una valutazione completa e precisa delle minacce in ogni ambiente della struttura dell'architettura.

- **Mancanza di Statistiche sul livello generale di Sicurezza**:Questo studio si focalizza prevalentemente sull'affinamento e l'ottimizzazione del processo di identificazione delle minacce. Tuttavia, non si estende a fornire una valutazione complessiva del grado di sicurezza dell'architettura. Tale limitazione implica che, sebbene si possano identificare efficacemente le minacce specifiche, manca una visione d'insieme che quantifichi o valuti la sicurezza generale del sistema. Questo aspetto sottolinea l'importanza di integrare il threat modeling con un'analisi più ampia, che possa offrire una stima del livello di sicurezza dell'intera architettura.

7.3 Sviluppi Futuri

Partendo dalle limitazioni di questo studio, qui di seguito vengono elencate alcune delle tematiche che potrebbero essere affrontate in lavori futuri:

- **Automazione delle Contromisure**: Esplorare l'implementazione di un approccio automatizzato anche per le azioni di remediation. Ciò consentirebbe di ridurre la dipendenza dall'intervento umano e di reagire in modo più rapido ed efficace alle minacce identificate.
- **Dal Threat Modeling al Risk Assessment**: Trasformare il processo di threat modeling in un'analisi di risk assessment più ampia. L'obiettivo sarebbe generare metriche che possano fornire un indice di rischio complessivo del sistema, permettendo così un'analisi più approfondita e strategica della sicurezza.

Bibliografia

[1] Mell, P. and Grance, T. (2011), "The NIST Definition of Cloud Computing, Special Publication (NIST SP)," National Institute of Standards and Technology, Gaithersburg, MD, 2011, DOI: 10.6028/NIST.SP.800-145

[2] "Public Cloud - Worldwide", Statista. [Online]. Available: https://www.statista.com/outlook/tmo/public-cloud/worldwide

[3] A. Hendre and K. P. Joshi, "2015 IEEE 8th International Conference on Cloud Computing," New York, NY, USA, 2015, pp. 1081-1084, DOI: 10.1109/CLOUD.2015.157

[4] Nataliya Shevchenko, Timothy A. Chick, Paige O'Riordan, Thomas Patrick Scanlon,Carol Woody, "Threat Modeling: A Summary of Available Methods," Software Engineering Institute 4500 Fifth Avenue, Pittsburgh, PA 15213-2612, 2018 DOI: https://insights.sei.cmu.edu/documents/569/2018_019_001_524597.pdf

[5] "Jenkins User Documentation," Jenkins [Online]. Available: https://www.jenkins.io/doc/

[6] "Continuous Integration Software Market Share," Datanyze [Online]. Available: https://www.datanyze.com/market-share/ci--319

[7] "Pipeline," Jenkins [Online]. Available: https://www.jenkins.io/doc/book/pipeline/

[8] "Amazon Web Services," Amazon Web Services [Online]. Available: https://aws.amazon.com/

[9] "Most wanted cloud platform among developers worldwide as of 2022," Statista [Online]. Available: https://www.statista.com/statistics/793884/worldwide-developer-survey-most-wanted-platform/

[10] Larry Dignan, "Top cloud providers" [Online]. Available: https://www.zdnet.com/article/the-top-cloud-providers-of-2021-aws-microsoft-azure-google-cloud-hybrid-saas/

[11] "AWS Documentation," Amazon Web Services [Online]. Available: https://docs.aws.amazon.com/

[12] "Control traffic to resources using security groups," Amazon Web Services [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

[13] "What is Amazon VPC?" Amazon Web Services [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html

[14] "Local Cluster vs. Remote Cluster for Kubernetes-Based Development" [Online]. Available: https://medium.com/swlh/local-cluster-vs-remote-cluster-for-kubernetes-based-development-6efe2d9be202

[15] "4 ways to run Kubernates locally" [Online]. Available: https://opensource.com/article/20/11/run-kubernetes-locally

[16] "How to install Minikube on Ubuntu 22.04 LTS," [Online]. Available: https://www.fosstechnix.com/how-to-install-minikube-on-ubuntu-22-04-lts/

[17] "CIS Kubernetes Benchmarks ," Center for Internet Security [Online]. Available: https://www.cisecurity.org/benchmark/kubernetes