POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

ATTACKING CONTAINERIZED SYSTEMS AND INFRASTRUCTURES

Relatore :

Prof. Riccardo SISTO

Candidato :

Matteo CALÒ

Tutor aziendali:

Dr. Ivan AIMALE

Dr. Luigi CASCIARO

Ottobre 2022

Sommario

Attualmente, l'utilizzo di piattaforme cloud e l'impiego di containers ricoprono un ruolo sempre più di rilievo per la fornitura dei servizi di cui ci facciamo uso.
Questi ambienti, però, non sono immuni alle insidie che il mondo digitale nasconde e, perciò, sono spesso bersaglio di attacchi informatici.

È dunque necessario, per chi fornisce un servizio, la piena consapevolezza dei problemi di sicurezza che possono verificarsi nel prodotto finale e, per gli sviluppatori, conoscere le buone norme per lo sviluppo sicuro.

Per questo lavoro di tesi, si è deciso di definire un framework per lo sviluppo sicuro di un servizio containerizzato. Esso si compone di due soluzioni: la prima permetterà di rintracciare eventuali vulnerabilità o errori di configurazione, la seconda di attuare delle contromisure in caso di attacco.

La trattazione definirà, in prima analisi, i servizi utilizzati e delle possibili criticità, successivamente verranno illustrate le best practices da rispettare per lo sviluppo sicuro.

Successivamente, si procederà con la presentazione della prima parte del framework, ovvero la creazione di una pipeline di sicurezza composta da tre tools: Sonarqube che permetterà di effettuare l'analisi statica del codice, OWASP Dependency-check per l'analisi delle librerie e Trivy per quella dei file IaC e delle immagini.
Ognuno dei precedenti stages, sarà seguito da un Quality Gate, in grado di bloccare l'esecuzione della pipeline nel caso in cui non vengano rispettati determinati criteri.

Il passo seguente, prevede l'uso di Kubescape per la ricerca delle vulnerabilità e degli errori di configurazione all'interno dell'infrastruttura cloud.

Per la seconda parte del framework, è stata realizzata una possibile architettura di un Response Engine, che tramite l'applicazione dei concetti di run-time security e di serverless computing, propone di fornire una contromisura automatica in caso di attacco ad un pod.

Per la gestione della run-time security all'interno del Response Engine si è scelto di utilizzare Falco e il suo plug-in Falco Sidekick. Il primo ha il compito di generare degli alerts da eventi scatenati da systemcalls, Kubernetes audit logs e AWS CloudTrail, mentre il secondo, quello di raccogliere gli alerts in una dashboard e di inoltrarli sia ad un canale Slack che ad OpenFaas, un tool per la creazione e gestione di servizi Function as a Service.

OpenFaas a questo punto, si occuperà dell'ispezione dell'alert e, nel caso in cui esso abbia un alto grado di criticità, di invocare un servizio FaaS che identificherà il pod sotto attacco per poi sospenderlo.

Infine, si produrrà un'applicazione containerizzata, scritta in Java SpringBoot e deployata su un cluster Kubernetes, sulla quale verrà applicato il framework.

Per concludere, si può constatare che il framework è stato validato in entrambe le sue parti: nella prima, lo si verificherà attraverso il riconoscimento, da parte della pipeline, del 100% delle vulnerabilità volutamente inserite durante lo sviluppo; nella seconda, in seguito alla simulazione di un attacco che il response engine sarà in grado di bloccare.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi anni il cloud computing e la cybersecurity stanno svolgendo - nell'ambito informatico - un ruolo sempre più di rilievo.

Aziende in tutto il mondo cercano di ottimizzare i loro servizi e la ricerca di questa necessità trova la soluzione nel cloud. Vantaggi nei costi, nella manutenzione, nell'affidabilità e possibilità di scalare l'infrastruttura sono solo alcuni dei benefici che questo vasto campo può offrire.

Alla crescita continua del cloud computing possiamo affiancare quella della cybersecurity. Infatti, negli ultimi anni, gli attacchi informatici sono cresciuti in modo vertiginoso e ciò ha fatto sì che molte aziende sviluppassero una maggiore consapevolezza riguardo le tematiche di sicurezza informatica.

Tra cloud computing e cybersecurity è possibile identificare una grande area di sovrapposizione, le cui tematiche possono andare dalla privacy, alla gestione dei dati fino alla garanzia di isolamento tra gli utenti; lo scopo ultimo, però, può essere facilmente descritto dalla necessità di offrire un servizio che sia il più sicuro e affidabile possibile.

Sarà proprio questo il punto fondamentale della trattazione che si proporrà di fornire un insieme di buone norme e di azioni che si dovrebbero seguire per mettere in sicurezza sia un applicativo organizzato in containers, che l'infrastruttura che lo ospita.

L'intero lavoro di tesi è stato condotto per il Politecnico di Torino sotto la supervisione del professor Riccardo Sisto nell'ambito di un tirocino svolto all'interno di Blue Reply SRL, società di consulenza che si occupa di molti campi innovativi, tra cui la Cybersecurity e il Cloud Computing.

1.1 Il cloud computing

Il cloud assunse un posto di rilievo alla fine degli anni '90, con la diffusione dei cosiddetti servizi SaaS (Software as a Service), i quali offrivano semplici ma cruciali funzionalità come siti web aziendali oppure spazi di archiviazione.
La svolta epocale si è avuta quando Amazon Web Service iniziò a fornire una suite di servizi basati su cloud che permettevano la condivisione di risorse hardware e software.

Il successo del cloud computing è da attribuire al progresso della virtualizzazione, la quale permette di eseguire macchine virtuali all'interno di macchine fisiche. Questa nuova possibilità offriva innumerevoli vantaggi, per citarne alcuni: l'isolamento temporale e spaziale tra le istanze, la minimizzazione dei costi di hardware e manutenzione.

Il traguardo successivo è stato quello di astrarre delle primitive che permettevano di ottenere la virtualizzazione in ambienti più elementari. Questo è stato il punto di partenza per la definizione dei containers, che hanno portato prima con LinuX Container (LXC) e poi con Docker, ad una semplificazione di tutto il ciclo produttivo del software che ne ha agevolato lo sviluppo, il test e la produzione, continuando ad assicurare l'isolamento delle risorse.

Tuttavia, il paradigma di sviluppo a container non riusciva da solo a far fronte alla continua richiesta di scalabilità dei servizi. Tale limite è stato colmato dai sistemi di orchestrazione, i quali permettono di godere della libertà di sviluppo tipica dei containers automatizzandone le procedure di gestione ed esecuzione.
Al giorno d'oggi, possiamo identificare in Kubernetes la soluzione più utilizzata per l'orchestrazione dei containers. Esso ci permette di gestire le risorse in modo completamente indipendente dal cloud service provider in cui vengono eseguite.

Per quanto riguarda gli aspetti della sicurezza, bisogna considerare tutti i possibili punti di fallimento dell'infrastruttura, partendo dai livelli più alti, ovvero dai servizi offerti dal cloud provider, passando per la gestione del cluster da parte dell'orchestratore, fino alla creazione e allo sviluppo del container.
In questo panorama, sarà fondamentale definire delle soluzioni che garantiranno:

- il monitoraggio degli eventi relativi alle minacce con anche la capacità di adottare eventuali meccanismi di difesa;
- di automatizzare il processo di costruzione dei containers, assicurandosi che vengano rispettati tutti i requisiti di sicurezza.

1.2 Il mondo della cybersecurity

Negli ultimi anni, le statistiche degli attacchi informatici riportano un continuo aumento dei soggetti colpiti, questo perchè i crimini informatici stanno diventando molto più redditizi di quelli reali. Inoltre, molte aziende contribuiscono ad agevolare questo sviluppo in quanto non danno alla sicurezza informatica il giusto rilievo.

Infatti, l'adozione di strategie di difesa e di prevenzione atte a migliorare la qualità e la sicurezza dello sviluppo non sono ancora una pratica comune, e ciò porta all'insorgere di problemi causati da minacce e vulnerabilità.

Da recenti analisi di RedHat, raccolte nel report annuale di sicurezza [1], è emerso che sempre più aziende che stanno effettuando una transizione verso l'ambiente cloud stanno anche sviluppando un occhio più critico riguardo alla sicurezza. In particolare, nel caso di sviluppi di applicazioni containerizzate la sicurezza ricopre un posto prioritario rispetto alle strategie di containerizzazione, fino a generare, nel 55% dei casi presi in esame, dei ritardi nei deployments.
Considerando le infinite possibilità di personalizzazione che l'ambiente cloud può offrire, la maggiore preoccupazione rimane, però, negli errori di configurazione che sono molto più rilevanti rispetto agli attacchi informatici.

I cloud service providers ospitano tutti i servizi più importanti di cui usufruiamo, ma ciò non implica che la sicurezza sia solo ed esclusivamente di loro competenza. Nonostante i providers offrano un'infrastruttura con impostazioni "secure by design", è sempre dovere dell'amministratore essere consapevole delle configurazioni che andrà a generare, in quanto, se non vengono rispettate le best practices, potrebbero essere introdotte vulnerabilità potenzialmente sfruttabili da eventuali attaccanti. Al fine di evitare errori di configurazioni, è necessario che gli sviluppatori siano coscienti delle best practices di sicurezza. Il principio del "least-privilege", ovvero definire i ruoli degli utenti garantendone i minimi privilegi necessari a svolgere il proprio lavoro, ne è un esempio. Nel caso in cui non venga seguita tale best practice, eventuali malintenzionati potrebbero ottenere il controllo su quelle credenziali e avere così un accesso incontrollato su tutti i servizi.

Inoltre, nel caso di sviluppo di applicazioni, è molto importante considerare le vulnerabilità che possono essere introdotte, come:

- codice non sicuro;
- l'utilizzo di immagini corrotte;
- l'uso di librerie vulnerabili.

Consultando la OWASP Top10 [2], una classifica che raccoglie i più frequenti rischi legati alle applicazioni web, possiamo vedere che le posizioni dalla quarta alla sesta sono occupate rispettivamente da Insecure Design, Security Misconfiguration e Vulnerable and Outdated Component, questo fa capire quanto sia fondamentale rispettare le best practices e affidarsi a patterns, i quali offrono uno sviluppo sicuro e permettono di mitigare i rischi. Ciononostante, pur applicando le best practices di sicurezza, potrebbero verificarsi delle incidenze d'attacco. Dunque, risulta necessario mantenere sempre aggiornati i componenti che fanno parte di un applicativo o di un'infrastruttura.

Per verificare la corretta applicazione delle buone pratiche di sicurezza, occorre effettuare periodicamente dei security assessment, che costituiscono un'arma molto potente per i responsabili della sicurezza in quanto tramite dei tools specifici riescono a trovare vulnerabilità di codice o di configurazione.

1.3 Focus della tesi

Dopo aver delineato quelli che saranno i macro-temi, la tesi si incentrerà sulla creazione di un framework: ossia fornire un insieme di strumenti per permettere a qualsiasi programmatore di sviluppare in modo sicuro un applicativo. Tale elaborato, inoltre, si concentrerà sul rilevamento e sul monitoraggio delle minacce rivolte ai containers e alla sua infrastruttura cloud.

La trattazione offrirà i mezzi per identificare i rischi più frequenti che possono verificarsi durante tutta la supply chain, in particolare:

- l'eccesso di privilegi, ovvero l'assegnazione di più privilegi ad un utente rispetto a quelli che dovrebbe avere;
- le misconfigurations che possono minare la sicurezza dell'ambiente, ovvero cattivi comportamenti che uno sviluppatore può inconsciamente seguire;
- l'utilizzo di immagini corrotte, cioè creare l'applicazione partendo da delle immagini che contengono vulnerabilità, oppure che vengono inserite nei repository per effettuare degli attacchi;
- le vulnerabilità dei software, ovvero bugs, che possono essere trovati all'interno di librerie o nel codice sorgente dell'applicazione;
- le minacce in tempo reale, vale a dire attacchi che possono essere lanciati durante l'esecuzione dell'applicativo, ad esempio tramite un malware o un miner.

Tutte queste minacce possono peggiorare la qualità e il tempo di sviluppo del servizio offerto, perciò è molto importante dotarsi di elementi che permettano di identificare le vulnerabilità, di notificare comportamenti inusuali e di eseguire delle contromisure.

Come si vedrà nei prossimi capitoli, lo studio è partito dall'interesse maturato nei confronti delle aziende che vogliono, per i vantaggi sopraccitati, effettuare una migrazione al cloud. È importante, dunque, che essa venga eseguita tenendo in considerazione tutte le difficoltà a cui queste possono incorrere nel garantire un buon grado di sicurezza.
Per raggiungere questo risultato, verrà simulato un intero ciclo di sviluppo sicuro di un servizio.

Verranno descritte delle possibili vulnerabilità e dei possibili attacchi che si possono identificare nei servizi più utilizzati nel panorama dello sviluppo a containers. Successivamente, si illustreranno delle best practices che permettano in anticipo di mitigare alcune vulnerabilità.

Si procederà, dunque, con la creazione di una pipeline di sicurezza che aiuterà a rintracciare vulnerabilità o errori di configurazione all'interno di applicazioni web. In seguito, si prenderà in considerazione anche un possibile strumento per l'analisi dell'infrastruttura cloud al fine di ricercare delle best practices non rispettate.
A questo punto si passerà alla costruzione di un applicativo di test sul quale verranno eseguite le precedenti analisi.

Infine, si offrirà un metodo per irrobustire la sicurezza basato su un meccanismo di risposta che intercetterà gli eventi del cluster e, nel caso di criticità, attuerà delle misure definite grazie al concetto di serverless computing.

I tools informatici che verranno utilizzati nella ricerca delle vulnerabilità e delle misconfigurazioni saranno completamente open source. Tale scelta è dettata dal desiderio di fornire indicazioni anche ad aziende che non vogliono impiegare risorse per strumenti di sicurezza costosi. Inoltre, lo studio si servirà di un ambiente cloud composto dai più frequenti servizi, ovvero AWS come cloud provider, Kubernetes come orchestratore e Docker per la gestione dei containers, questo al fine di rendere la trattazione più generale possibile e applicare il framework su diversi possibili contesti.

1.4 Sommario dei contenuti

Nel capitolo 2 verranno introdotti i concetti cardine che attualmente vengono seguiti dalle aziende per lo sviluppo sicuro di un'applicazione, ad essi si affiancherà la novità introdotta dalla run-time security.
Il capitolo 3 verterà sulla definizione del contesto, si tratteranno i servizi utilizzati e alcune possibili vulnerabilità, il tutto corredato da report di attacchi noti.
Il capitolo 4 illustrerà delle best practices che possono essere applicate per prevenire alcuni attacchi.
Nel capitolo 5 verrà descritto l'uso degli strumenti che serviranno per effettuare un Security Assessment sull'applicazione e sull'infrastruttura, una volta terminata questa sezione si passerà, nel capitolo 6, alla descrizione di un applicativo di test e dei risultati delle analisi eseguite sullo stesso.
Il settimo ed ultimo capitolo di affronterà il problema di avere un ambiente capace di identificare eventuali situazioni critiche che possono essere indipendenti dallo sviluppo vero e proprio.

Infine, verranno stilate le conclusioni relative alla validità del framework.

Capitolo 2

Stato dell'arte

Nel corso di questo capitolo si farà un excursus riguardo al flusso di sviluppo software, in particolare di un applicativo containerizzato. Verranno illustrati inizialmente i concetti di "Development and IT Operations" (DevOps) e "Continous Integration, Continous Delivery and Deployment" (CI/CD), e successivamente come l'inserimento della sicurezza nel meccanismo abbia creato il paradigma di DevSecOps. Si passerà poi a delineare la novità della run-time security e come l'utilizzo di uno dei tool disponibili verrà utilizzato all'interno del framework.

2.1 DevOps e CI/CD

Attualmente il mondo dello sviluppo software è fortemente basato sulla metodologia Agile [3], che rispetto al precedente modello a cascata, propone un approccio meno strutturato e una possibilità di rilasci più frequenti e di qualità migliore.

Da queste premesse, si definisce il paradigma di DevOps [4], il quale è basato sui concetti di collaborazione e di automazione.

Rilevante sarà la collaborazione: questo modello risolve, infatti, la netta divisione tra il team di sviluppo e quello di produzione, e presuppone una stretta cooperazione tra gli stessi per garantire un miglioramento nei tempi di esecuzione e una riduzione delle cause di inefficienza.

L'automazione, invece, consiste nel ridurre il numero di azioni manuali che il team deve svolgere, aumentando così la qualità del servizio offerto e riducendo la possibilità di errori.

Tale concetto trova un esempio concreto nel CI/CD ovvero un metodo che permette di automatizzare e monitorare tutte le fasi del ciclo di vita del software, dall'integrazione e al test, fino al deployment.

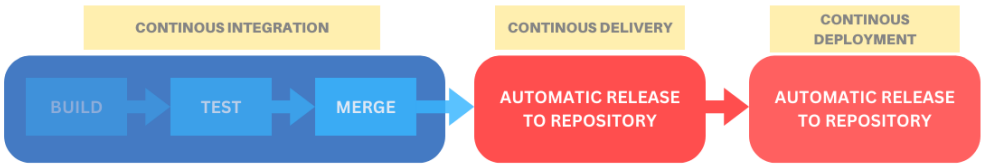


Figura 2.1: Flusso CI/CD

L'acronimo CI/CD indica "Continous Integration" e "Continous Delivery and Deployment"[5]:

- Il Continous Integration indica, appunto, l'integrazione continua, ovvero ogni volta che una modifica viene caricata all'interno del repository di codice, viene creata una build sulla quale verranno effettuati dei test che permetteranno di trovare eventuali bugs. Se i test falliscono la pipeline viene bloccata.
- Il Continous Delivery and Deployment può essere suddiviso in ulteriori due stadi: il Continous Delivery permette di inviare la nuova versione al testing e/o alla produzione per poi necessitare di un'approvazione per la distribuzione; invece, il Continuos Deployment automatizza il rilascio della nuova versione.

Questi concetti di DevOps e di CI/CD saranno fondamentali per delineare un nuovo modello di sviluppo nel quale la sicurezza avrà un posto di rilievo.

2.2 DevSecOps

In passato i controlli di sicurezza avvenivano solo alla fine del processo di sviluppo e attivati manualmente, ciò ritardava il processo di produzione vanificando i benefici che il modello DevOps riusciva ad ottenere.

Al giorno d'oggi, la sicurezza ricopre un ruolo sempre più di rilievo e, perciò, deve essere presa in considerazione in tutte le fasi della supply chain. In questo panorama si è voluto risolvere l'approccio del passato inserendo nel processo di DevOps i controlli di sicurezza, facendo nascere il concetto di DevSecOps [6].

DevSecOps sta per sviluppo (Dev), sicurezza (Sec) e operazioni (Ops) proprio per definire quanto l'integrazione della sicurezza non sia più un'azione marginale, ma che ogni team coinvolto nello sviluppo deve collaborare per raggiungere il miglior grado di sicurezza possibile; perciò, la responsabilità di offrire un servizio sicuro passa a tutto il team di sviluppo e non solo a quello della cybersecurity.

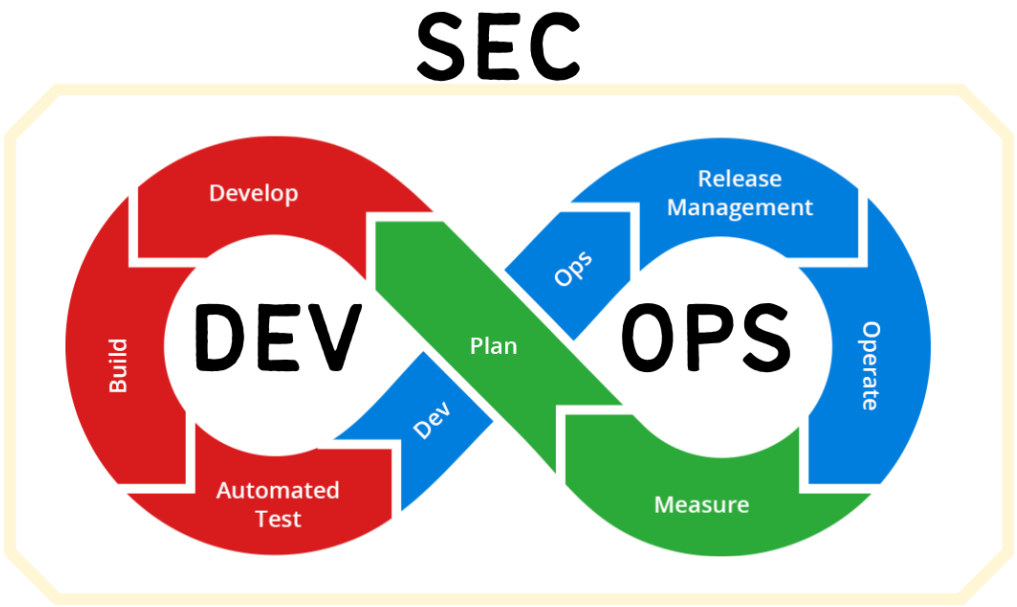


Figura 2.2: Fasi DevSecOps

La società di consulenza Kiratech [7], che opera nell'abito DevOps, Cloud e Security, definisce come per applicare il principio di DevSecOps ci sia bisogno di utilizzare le pratiche e i tools della Security Automation e di formare i lavoratori verso i temi di sicurezza attuali.

La Cybersecurity Awareness [8] è fondamentale per rendere il DevSecOps ancora più efficace aggiungendo i principi dello Shift-Left e della Security by Design, creando la declinazione del SecDevOps [9], nella quale la cybersecurity assume un posto principale già nelle fasi di pianificazione, analisi e progettazione. Questo riduce anche lo sforzo successivo che si potrebbe avere nella gestione delle vulnerabilità.

La Security Automation, invece, si basa sul concetto di CI/CD, ovvero quello di utilizzare una pipeline di sicurezza dove dei tools, in modo automatico, permettano di rintracciare vulnerabilità o errori di configurazione.

Questi due concetti, attualmente utilizzati nel mondo aziendale, verranno portati avanti e discussi più nel dettaglio nel prosieguo della trattazione, saranno inoltre impiegati nel caso d'uso dello sviluppo di un applicativo.

Capitolo 3

Descrizione del contesto

L'organizzazione di questo capitolo sarà sviluppata in sezioni, e, per ognuna di esse, verrà descritto un servizio comunemente utilizzato nel panorama aziendale. Tali servizi sono stati scelti perchè di grande interesse per l'azienda per cui è stata svolta la ricerca.
Si definiranno alcuni possibili errori di configurazione corredati da esempi di attacchi avvenuti negli anni.

C'è da segnalare che, tra tutte le possibili misconfiguration che possono essere inserite, sono state trattate quelle, secondo l'esperienza di Blue Reply, più frequentemente riscontrate e il cui impatto può generare più criticità.

Nella descrizione si partirà dal livello di astrazione più alto, ovvero quello che viene offerto dal cloud service provider, ovvero AWS, fino ad arrivare a quello più basso, cioè il codice sorgente dell'applicazione, Java Spring Boot.

3.1 Amazon Web Service

Amazon Web Service (AWS) [10] è attualmente la piattaforma leader mondiale in ambito cloud computing. Ha dato il via ai servizi Iaas, introducendo per la prima volta la condivisione di risorse computazionali grazie a delle istanze virtuali: le Amazon Elastic Compute Cloud, meglio note come EC2.

Il suo grande successo è da attribuire non solo al grande numero di servizi che può offrire, ma anche all'affidabilità e alla vastità di regions disponibili.
Tutte queste caratteristiche hanno fatto sì che AWS diventasse il maggiore cloud provider del mondo.
Si è deciso quindi di focalizzare l'attenzione su AWS in quanto consente di ottenere

un target più ampio possibile.

Uno dei vantaggi principali che AWS può offrire è l'uso dei servizi EC2 [11], in grado di mettere a disposizione server virtuali su cui far girare applicazioni e servizi web. Questo servizio, però, si differenzia dagli altri perchè ha la capacità di rimodulare immediatamente le risorse in base alle esigenze.
Per definire meglio questa peculiarità, si può prendere in considerazione un servizio web installato in un server locale, che registra un picco di utenti in una piccola finestra di tempo. In questo caso, il servizio subirà inevitabilmente dei rallentamenti perchè il server non riesce a soddisfare tutte le richieste; se invece il servizio fosse installato su una macchina EC2 sarebbe in grado di adeguarsi all'alto numero di richieste.

Entrando più nel dettaglio delle caratteristiche di questo servizio, troviamo il suo elemento base: le AWS AMI [12], dove AMI sta per Amazon Machine Image; questo componente contiene diversi template software necessari a generare un server virtuale. Un esempio di AMI può essere Amazon Linux 2: un'immagine Linux creata e gestita da AWS, che offre una grande possibilità di personalizzazione, con i vantaggi di non aggiungere ulteriori costi che altre AMI hanno e di garantire sempre le prestazioni delle EC2.

Copiando ed eseguendo un'AMI come server virtuale all'interno del cloud si genera un'istanza, ovvero il concetto principale di AWS EC2. Un'istanza è un ambiente virtuale composto dalle specifiche hardware che meglio si adattano ai requisiti degli utilizzatori; si può ad esempio modificare il numero di vCPU, la dimensione della RAM oppure lo spazio di archiviazione.
A seconda delle diverse caratteristiche hardware, le istanze possono essere divise in categorie [13]:

- uso generale: ovvero per creare microservizi o applicazioni web;
- memoria ottimizzata: ad esempio per database ad alte prestazioni oppure per data mining;
- storage ottimizzato: per ospitare database no-SQL o per il data warehousing;
- calcolo ottimizzato: per la creazione di modelli scientifici;
- calcolo accelerato: per gestire i processi di machine learning.

AWS inoltre, incentiva l'uso dei suoi servizi offrendo gratuitamente delle risorse, ad esempio, è possibile utilizzare delle istanze di T2.micro Linux o Windows per 750 ore senza costi aggiuntivi. Questo tipo di istanza appartiene alla categoria per uso generico ed è molto utile per la creazione di siti web, microservizio o ambienti di test.

3.1.1 AWS IAM misconfiguration

Uno dei servizi cardine di AWS consiste nello IAM, Identity and Access Management, il quale permette agli amministratori del cloud di gestire gli accessi e i privilegi. Tramite lo IAM è possibile creare e definire delle entità in AWS:

- gli utenti;
- i gruppi: corrispondono ad una raccolta di utenti e sono molto utili quando si vogliono assegnare gli stessi privilegi ad un insieme di utenti, perchè basterebbe definire un gruppo, assegnargli delle policies e di conseguenza tutti gli utenti che apparterranno al gruppo godranno di quei privilegi;
- i ruoli: ovvero entità che possono essere assunti dagli utenti nel caso necessitino di privilegi più specifici o più elevati. Ad esempio, si può definire un ruolo "EC2-manager" che gode dei privilegi di gestione delle istanze EC2 e, nel caso in cui un utente con privilegi base necessiti di crearne una, può assumere temporaneamente quel ruolo;
- le policies: cioè delle autorizzazioni per utilizzare le risorse in AWS e possono essere associate a utenti, a gruppi e ruoli. Un esempio di policy può essere, l'accesso in sola lettura alla console IAM [14]:

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:Get*",
            "iam:List*",
            "iam:Generate*"
        ],
        "Resource": "*"
    }
}
```

Gli utenti, dunque, sono delle entità tra di loro indipendenti che godono in modo diretto di privilegi definiti tramite policy e in modo indiretto da quelle del gruppo al quale appartengono oppure tramite l'assunzione temporanea di ruoli.

Ciò introduce molte possibilità di errore, soprattutto nel caso di amministratori non coscienti dei principi di sicurezza. Secondo uno studio prodotto da Fugue, è stato stimato che la principale fonte di misconfiguration in ambito cloud è da assegnare allo IAM [15].

Estremamente importante è rispettare il principio del *least privilege*, secondo cui, ad ogni utente devono essere garantiti solo i privilegi minimi per svolgere il proprio compito. In questo modo, è possibile evitare che un utente acceda ad informazioni confidenziali o che in caso di compromissione delle credenziali si permetta a malintenzionati di ottenere accesso incontrollato a vari servizi.

3.1.2 Report di attacchi alle credenziali AWS

Le vittime preferite degli hackers sono le credenziali del cloud service provider, che riflettono i cattivi comportamenti da parte dei gestori.
Per citarne alcuni:

- usare account con privilegi root. Questa tipologia di account è la più ricercata dagli attaccanti perchè una volta ottenuto il possesso, si ha il controllo completo su tutti i servizi del cloud;
- utilizzare credenziali statiche e vulnerabili ad attacchi di bruteforce;
- permettere agli utenti di assumere ruoli in modo incontrollato.

I malintenzionati sono alla continua ricerca di credenziali deboli e riescono addirittura a rendere l'investigazione più automatica ed efficace, tanto che, anche le aziende più note hanno potuto, loro malgrado, sperimentare questo nuovo trend.

Trovare in rete incidenze di attacchi che mirano a questa categoria non è per nulla difficile e le tipologie di attacco perpetrato sono molto varie. Per fini puramente informativi se ne vogliono riportare alcuni rivolti ad aziende avvenuti nel 2021:

- un dipendente di Glassdoor ha involontariamente condiviso la chiave di accesso AWS IAM di un utente su un repository pubblico; [16]
- dei ricercatori di BeVergil hanno identificato che in 40 popolari applicazioni Android, alcune delle quali anche con più di un milione di download, il file APK era stato generato utilizzando le chiavi di accesso statiche di AWS; quindi, tramite un attacco di reverse engineering si sarebbe potuto ottenerle ed utilizzarle per scopi illeciti;[17]

- un attaccante è riuscito ad ottenere delle credenziali temporanee di AWS di proprietà di Evernote tramite un attacco di tipo Server-Side Request Forgery nel quale il malintenzionato sfrutta una vulnerabilità di sicurezza per indurre il server ad aprire connessioni verso un sistema esterno ed ottenere le credenziali;[18]
- Il TeamTNT, il quale tramite dei crypto-mining worms ricerca le credenziali AWS contenute nei file di configurazioni locali.[19]

3.2 Kubernetes

Al livello successivo della trattazione troviamo l'orchestratore di microservizi: Kubernetes.
Kubernetes è un software creato da Google e poi donato come progetto open source alla CNCF [20]. Esso vanta una grandissima community di utilizzatori e può essere considerato il progetto, sviluppato in ambito cloud, che ha avuto più successo negli ultimi anni. Attualmente è il tool di orchestrazione di microservizi più usato al mondo grazie soprattutto alla scalabilità orizzontale e verticale, alla resistenza ai guasti e alla facilità e velocità di gestione dei containers.

Nonostante Kubernetes sia molto popolare, per utilizzarlo bisogna conoscerne bene il funzionamento ed essere in grado di gestirne il piano di controllo.
I cloud service provider, però, offrono delle soluzioni per gestire i cluster Kubernetes che permettono di non considerare le problematiche relative al control plane: AWS offre il servizio Elastic Kubernetes Service (EKS) mentre, per Google Cloud Platform (GCP) esiste il servizio Google Kubernetes Engine (GKE).

Queste soluzioni provider-managed possono introdurre dei vincoli per la configurazione del cluster. Si è deciso quindi di non considerare questi servizi e di focalizzare l'attenzione sulla creazione del cluster in maniera autonoma, per ottenere un livello di libertà maggiore che dia la possibilità di generare un ambiente dove non tutte le best practices siano state rispettate.

Per la stesura di questo sottocapitolo si prenderà in considerazione quindi, un cluster gestito in modalità self-managed tramite l'utilizzo di Kops (Kubernetes Operation) [21]. Quest'ultimo è un tool open source che permette di realizzare l'infrastruttura del cluster Kubernetes su piattaforme come AWS, Azure e GCP.

3.2.1 Possibili Misconfigurations

In questa sezione verranno identificate le misconfiguration più comuni [22], di seguito riportate in ordine crescente in base alla criticità.

- **BASSA:** usare il namespace di default.
  I namespaces in Kubernetes garantiscono l'isolamento logico tra i servizi. L'utilizzo di quello di default, nel caso di applicazioni multi-cluster, rischia di dare accesso a servizi ai quali non si dovrebbe;
- **MEDIA:** usare containers con "allowPrivilegeEscalation" attivo.
  Questa policy, se attiva, permette al container di operare in modalità privilegiata e quindi accedere ai file dell'host;

- **ALTA:** non configurare una soglia limite per CPU e memoria [23].
  Non definendo un limite delle risorse, un container potrebbe utilizzare tutte quelle messe a disposizione dal nodo, causando malfunzionamenti negli altri pods all'interno del nodo.

- **CRITICA:** non configurare correttamente la Kubernetes API oppure la kubelet API. Entrambe sono le componenti del cluster in grado di gestire le richieste dei containers e dei clients.
  Kubelet in particolare, è il principale agent all'interno di Kubernetes ed è presente in ogni nodo worker per permettere, tra le altre cose, una comunicazione con il master. Alla creazione, però, il flag "anonymous-auth" è impostato su true e questo tipo di configurazione di default accetta ogni connessione in ingresso senza il bisogno di autenticazione [24].
  Un comportamento di questo tipo lascia spazio a molte possibilità d'attacco, anche perchè Kubernetes stesso espone alcune API per accedere direttamente a kubelet, di seguito è possibile vederne alcune:

KUBELET API	DESCRIZIONE
/pods	Si ottiene la lista di tutti i pods
/run	Permette di lanciare un comando in un container
/exec	Lancia un comando usando uno stream
/configz	Permette di ottenere le configurazioni di Kubelet
/debug	Mostra le informazioni di debug

Tabella 3.1: Kubelet API esposte

3.2.2 Reports di attacchi

La trattazione di reports, anche in questo caso, può essere molto vasta, per questo motivo, si è deciso di considerare un attacco che negli ultimi anni è emerso in molte incidenze: l'utilizzo di tools automatici di cryptojacking [25], ovvero malware capaci di utilizzare le risorse di calcolo offerte da un dispositivo od un server per generare criptovalute.

Il comportamento di questi attacchi è stato spesso analizzato da molte aziende, perciò le risorse da cui attingere online sono varie. Per l'argomentazione di questa tesi è stata considerata l'analisi pubblicata nel Febbraio 2021 da "Unit 42" di PaloAlto [26]. Questa analisi può essere riassunta con il seguente insieme di comportamenti: l'attaccante inizialmente cerca di sfruttare delle misconfiguration o delle vulnerabilità nella libreria di kubelet per entrare nel sistema. Successivamente, esegue una Remote Code Execution (RCE) in modo da scaricare tutti i malware

necessari a effettuare lo scraping delle credenziali, installare il crypto-miner e cancellare tutti i file e la storia dei comandi.

Questo tipo d'attacco può essere spiegato meglio attraverso questo caso: si può immaginare che un'azienda abbia appena effettuato la creazione di un cluster Kubernetes e che, come definito nel sottocapitolo precedente, non abbia considerato la protezione delle API esposte di Kubelet, lasciando il flag "anonymous-auth" attivo e quindi permettendo a chiunque di aprire una connessione con l'agent.
Un attaccante si accorge di questo errore di configurazione e, tramite l'API "/run", invia una richiesta HTTP POST con all'interno un comando per creare un nuovo pod privilegiato, nel quale poi eseguirà il malware per il crypto-mining.

3.3 Docker

Come già descritto nel capitolo introduttivo, la piattaforma Docker ha segnato una svolta per lo sviluppo delle applicazioni fino a diventare un caposaldo della containerizzazione dei servizi.

Docker [27] è una tecnologia usata per creare containers che sfrutta il kernel di Linux e le sue funzionalità. Il componente base di Docker è l'immagine, un'entità immutabile che offre un insieme di funzionalità basilari. È possibile creare un servizio partendo da una base image ed aggiungere delle altre immagini per ampliarne le funzionalità.

Per la creazione di un'immagine si utilizza il DockerFile: un file testuale che riporta le istruzioni per creare una build. tra queste: l'immagine base da utilizzare, l'etichetta che le si assegnerà, la porta di esposizione, i comandi da eseguire e l'entrypoint. Una volta creata la build è possibile eseguire un container selezionando l'immagine appena prodotta.

Un'immagine, una volta generata, può essere caricata in un repository chiamato Docker Hub, che se resa pubblica, potrà essere scaricata da chiunque.

In questo contesto, si considererà un'immagine costruita tramite il Dockerfile, sviluppata in Java Spring Boot ed eseguita in un cluster Kubernetes.

3.3.1 Possibili Misconfigurations

L'immagine derivata dall'eseguibile dell'applicazione potrà presentare non solo gli errori mostrati di seguito ma anche le vulnerabilità generate dal codice sorgente, che per questioni discorsive, verranno descritte nella sezione 3.4.1.

Verranno ora presentate le misconfigurations più sfruttate dagli attaccanti nell'ultimo periodo [28]:

- errori di configurazione nel DockerFile: le istruzioni di cui è composto questo file non devono esporre il servizio a rischi, infatti, una non corretta configurazione di queste istruzioni potrebbe favorire l'utilizzo di container privilegiati o leakege di credenziali. Ad esempio: non usare l'istruzione USER lascia come unico utente attivo l'utente root oppure l'uso non ottimale dei comandi ADD e COPY può causare inserimento nell'immagine di dati confidenziali.

- uso di immagini corrotte: un'immagine si definisce corrotta se presenta delle vulnerabilità sfruttabili. La possibilità di scaricare immagini dal Docker Hub, oltre ad offrire un utile servizio, comporta anche dei rischi perchè si possono utilizzare immagini corrotte o caricate nel repository per essere sfruttate come vettore d'attacco.
- Docker API non sicure accessibili da remoto: in alcuni casi può essere utile esporre le API di Docker e per farlo bisogna abilitarne il socket TCP. Se però non si protegge questa comunicazione tramite protocolli di SSH o TLS, un attaccante può utilizzarla per lanciare comandi che possono compromettere il sistema.

3.3.2 Report di attacchi

Gli attacchi che si è deciso di riportare per questo servizio interessano l'esposizione di API insicure e l'uso di immagini corrotte.

È possibile riconoscere come il conflitto tra Russia e Ucraina si sia combattuto anche sul piano informatico: attivisti pro-Ucraina hanno pubblicato sul Docker Hub delle immagini che possono essere usate per effettuare attacchi DoS verso i più importanti siti russi e bielorussi.
Come riportato in un articolo nel blog di CrowdStrike [29], tra il 27 febbraio e l'1 Marzo 2022, molte honeypot contenenti Docker API volutamente insicure sono state sfruttate per scaricare immagini, come ad esempio *abagayev/stop-russia*, che avrebbero effettuato continue richieste HTTP in modo da stressare il più possibile un servizio.
Questa serie di attacchi è molto spesso lanciata in modo automatizzato, al fine di ricercare API che non utilizzano sistemi di protezione. Il modus operandi di questa tipologia d'attacco è inoltre utilizzato dai malware per il crypto-mining, che dopo aver ricercato servizi mal configurati, scarica da repository pubbliche le immagini lanciate poi per effettuare l'attacco.

Nel caso dell'uso di immagini corrotte è da menzionare anche un particolare tipo di attacco che gli analisti del Nautilus Team di AquaSecurity hanno definito "container image lookalike" [30].
In questo tipo di attacco, gruppi di malintenzionati creano profili con nome che assomiglia molto a quello di più noti team di sviluppo, come ad esempio Tesnorflow al posto di Tensorflow, e pubblicano sul Docker Hub immagini contenenti malware o backdoor con un tag simile alle più utilizzate, come nel caso di *alpine2*. Anche qui è possibile rintracciare delle incidenze d'attacco generate da gruppi di crypto-mining che utilizzano questa tecnica per diffondere il più possibile i malware per la generazione di criptovalute [31].

3.4 Codice sorgente

Per concludere la descrizione del contesto, si passerà col prendere in considerazione il codice sorgente di un servizio applicativo.

Lo sviluppo di un'applicazione web può avere diversi gradi di difficoltà e, considerando che il focus principale della tesi è quello di fornire delle linee guida per la sicurezza in ambiente containerizzato, si è deciso di focalizzarsi solo sulla parte backend delle applicazioni. È importante segnalare che, questa scelta può essere considerata come una limitazione del framework, in quanto escluderebbe dalla trattazione tutte le possibili vulnerabilità che possono essere introdotte dal front-end, ma offre ottimi spunti per una ricerca futura.

Si è scelto Java come linguaggio di programmazione, con l'ausilio del framework Spring Boot [32], molto utilizzato per la creazione di app e microservizi. Tale framework semplifica di molto lo sforzo dello sviluppatore in quanto offre un grande insieme di blocchi di codice già scritto.

3.4.1 Possibili Vulnerabilità

Le vulnerabilità del codice sorgente sono molto frequenti e, come accennato in precedenza, è molto facile che si propaghino verso i livelli superiori. È quindi fondamentale che uno sviluppatore sia consapevole delle falle di sicurezza che possono essere inserite durante la programmazione.

La prima misconfiguration che si andrà a trattare sta nell'utilizzo di librerie non aggiornate o che contengono vulnerabilità. L'uso inconsapevole di librerie di terze parti o di versioni ormai datate generano grandi incidenze di attacco e la sicurezza del prodotto finale diminuisce drasticamente.
Esistono vari gruppi di hacker che ogni giorno sono alla ricerca di falle nei codici, con un impatto diverso a seconda dei casi.

Una vulnerabilità di codice può essere identificata come pubblica quando viene inserita nei database mantenuti da esperti della sicurezza, come nel caso del CVE (Common Vulnerabilities and Exposures) del MITRE [33]. Questo tipo di vulnerabilità è spesso utilizzata per generare vari attacchi perchè facilmente introdotta da sviluppatori inconsapevoli.
In questo caso, lo sforzo per la correzione può essere basso in quanto, se presente un aggiornamento che contiene una patch, basterà solo portare la libreria ad una versione più recente o rimuovere la libreria corrotta.

A volte però, può verificarsi la scoperta di un nuovo tipo di vulnerabilità non ancora nota al pubblico, la cosiddetta *Zero Day Vulnerability* che, se sfruttata per

attacchi, può generare danni incalcolabili.

Un altro cattivo comportamento che uno sviluppatore può seguire è quello di non validare le richieste che vengono effettuate ai servers. Questo dà al client la possibilità di inviare qualsiasi dato che sarà poi direttamente eseguito dal server e nel caso di un malintenzionato si incorre in serie problematiche.
Vulnerabilità del genere hanno riscontrato grande importanza negli anni passati, basti pensare che nella classificazione OWASP del 2017 questo tipo di criticità era al primo posto.

Per quanto riguarda le vulnerabilità più frequenti nel linguaggio Java, sono state raccolte in una classifica nel blog di Spectral [34]:

1. Code Injection: permette di iniettare del codice malevolo;
2. Command Injection: come la precedente ma con la differenza che vengono usati comandi shell come vettore di attacco;
3. Connection String Injection: può presentarsi quando vengono inseriti parametri addizionali nelle stringhe di connessione alla sorgente di dato, separati dal punto e virgola. Questa vulnerabilità permette di bypassare il normale processo di autenticazione;
4. LDAP Injection: consiste nell'inserire degli input non sanificati all'interno degli statement del protocollo LDAP (Lightweight Directory Access Protocol);
5. Reflected Cross-site Scripting: lascia la possibilità di inserire nel sito vittima un link che porta ad una pagina contenente uno script malevolo;
6. Resource Injections: permette all'attaccante di modificare identificativi delle risorse, come ad esempio il numero della porta esposta;
7. SQL Injections: generata da una cattiva sanificazione degli input, implica l'esecuzione di codice malevolo SQL che potrebbe causare danni al sistema;
8. Second Order SQL Injections: simile alla precedente con la differenza che il comando SQL è prima salvato nel database vittima e poi verrà eseguito in un secondo momento;
9. Stored XSS: consiste nell'inserimento di script malevoli nel sito web vittima;
10. XPath Injections: simile alla SQL Injection ma sfrutta il codice XML.

Continuando nella descrizione delle possibili vulnerabilità, oltre a quelle riportate in precedenza, per completezza espositiva sono anche presentati i tipi di vulnerabilità più frequenti all'interno del framework Spring Boot, raccolti dal team di analisi di Snyk [35]:

1. Arbitrary Code Execution: ovvero dare la possibilità di eseguire codice che comprometterebbe il normale comportamento del sistema;
2. XML External Entity: queste vulnerabilità sono generate da una non corretta sanificazioni delle richieste inviate al parser XML;
3. Access Restriction Bypass: consistono nella possibilità di aggirare controlli di sicurezza durante gli accessi;
4. Denial of Service: vulnerabilità che possono far bloccare l'esecuzione del servizio;
5. Directory Traversal: permettono la modifica di file che non dovrebbero essere accessibili;
6. Cross-Site Request Forgery: generate dalla possibilità di inviare richieste fingendosi un altro utente;
7. Authentication Bypass: permettono di eludere i controlli di autenticazione;
8. Cross-site Scripting: consentono l'inserimento di script malevoli all'interno dell'applicativo.

3.4.2 Report di attacchi

Le vulnerabilità elencate in precedenza, se sfruttate offrono grandi possibilità di attacco e conseguenze dannose all'azienda. In questa sezione si vedranno alcune categorie attacchi che sfruttano le vulnerabilità sopra citate:

- **Log4j vulnerabilities** [36]

  Log4j è una popolare libreria open source sviluppata da Apache Foundation che permette di gestire i servizi di logging, ed è utilizzata da milioni di compagnie per servizi cloud e applicazioni.
  Il 9 Dicembre 2021 è stata trovata una zero-day vulnerability nella libreria, subito pubblicata nel CVE [37] con uno score di 10 su 10. Nei giorni a seguire, numerosi sono stati i tentativi di rilasciare delle patch ma, ad ogni nuovo aggiornamento, compariva anche una nuova vulnerabilità con diversi gradi di

criticità.
Le vulnerabilità riscontrate erano di tipo Arbitraty Code Execution e l'impatto è stato disastroso sia per il grande utilizzo della libreria, sia per la relativa facilità di sfruttamento. Bastava infatti inserire una stringa di codice malevolo che veniva poi eseguito da Log4j, per effettuare una remote code execution.

- **Injection Attacks** [38]

  Questa categoria di attacchi è considerata la più datata, ma ancora molto efficace perchè permette di causare gravi danni ad un'applicazione. Non validare gli input inviati al server rimane la principale sorgente delle injections. Le vulnerabilità in ballo in questo tipo di attacco coprono più di una categoria a seconda del tipo di linguaggio utilizzato per preparare il payload d'attacco. Ad esempio, tra le vulnerabilità citate in precedenza Cross-site Scripting e nuovamente Arbitrary Code Execution sono quelle che offrono la possibilità di questo tipo di attacco.

  Un attacco di injection consiste nell'invio di codice che permette all'attaccante di ottenere informazioni dal database e può avere diverse classificazioni a seconda del linguaggio utilizzato, le più frequenti sono: SQL Injection e Cross-Site scripting. La prima permette di utilizzare delle query SQL per ottenere o cancellare i dati del database, mentre tramite la seconda uno script viene inserito all'interno della pagina web ed eseguito dal client che la visita.

  Il sito Gab, un social network americano di estrema destra, ha subito questo tipo di attacco scatenato da un errore di programmazione che ha rimosso la protezione contro le SQL Injections. L'attacco ha avuto un grande successo e l'hacker è riuscito a scaricare 70 gigabytes di dati [39].

- **XML External Entity (XXE)** [40]

  Questo tipo di attacco sfrutta le vulnerabilità degli XML parser e permette di ottenere dati dal server come file contenenti le password o addirittura per eseguire attacchi di Server-Side Request Forgery.
  Il suo funzionamento si basa su un parser XML mal configurato in grado di processare entità esterne che, una volta ricevuto un payload XLM strutturato, risponde alla richiesta inviando dati sensibili.

  Per fare un esempio nel plug-in Recipe [41] di Jenkins, noto tool per fare CI/CD, è stata trovata una vulnerabilità di tipo XXE [42]. Grazie a questo plug-in si potevano condividere frammenti di codice per la configurazione di Jenkins, ma veniva anche offerta la possibilità di inviare payload XML in quanto il parser del plug-in accettava tutti i tipi di payload.

- **Denial of Service (DoS)** [43] Questa tipologia di attacco intende bloccare il funzionamento della vittima tramite lo spegnimento o il sovraccarico delle risorse, in modo da renderla inaccessibile agli utenti.
  Solitamente i bersagli principali di questi attacchi sono aziende o organizzazioni di alto livello, come ad esempio banche, social media, servizi governativi, etc.

  Il modo più utilizzato per performare questo attacco è stressare il servizio aprendo un alto numero di connessioni in una stretta finestra di tempo, uno dei più degni di nota fu quello dell'ottobre 2020 ai danni di Google. L'azienda subì questo tipo di attacco proveniente da IP cinesi, che durò addirittura sei mesi, toccando come picco la cifra di 2.5Tbps (Terabytes per seconds).

- **Directory traversal** [44] Un Directory traversal attack sfrutta una non corretta validazione di sicurezza dei parametri passati alle API e permette di ottenere l'accesso a file o cartelle ad accesso ristretto.
  Una vulnerabilità di questo tipo è stata trovata all'interno della libreria di Spring Cloud Config [45].

- **Buffer Overflow Attack** [46]

  Un buffer è una sezione di memoria che contiene dei dati, ed un buffer overflow è un tipo di attacco che consiste dell'inviare una quantità di dati che eccede la dimensione del buffer stesso, come conseguenza il programma cercherà di scrivere questi dati nelle celle di memoria adiacenti.
  Un attaccante potrà quindi cambiare il normale comportamento di un programma, eseguendo dei comandi oppure ottenendo delle informazioni sensibili.

Capitolo 4

Security by design

In questa sezione si illustreranno i comportamenti migliori da rispettare durante la creazione di un servizio.

Si inizierà trattando i concetti che sono alla base della prevenzione dei rischi informatici, riportando i concetti di "Shift Security Left" e "Security by Design" con i rispettivi vantaggi che questi portano all'azienda.

Si passerà poi ad enunciare alcune best practices che devono essere rispettate in tutto il processo produttivo. L'organizzazione del capitolo seguirà la struttura del precedente: si comincerà con le best practices per AWS IAM, per poi procedere con quelle di Kubernetes e infine, con quelle di Docker.

4.1 Security by design

Negli ultimi tempi, la rapida necessità di accelerare il passaggio globale alla digitalizzazione ha esposto una moltitudine di aziende a rischi di sicurezza; ciò ha aumentato l'attenzione aziendale ai concetti della cybersecurity.
Un eventuale incidente di sicurezza informatica implica direttamente un blocco delle attività produttive che causeranno un immediato danno economico. Per questo motivo il panorama commerciale sta evolvendo sempre di più verso l'impiego di risorse rivolte alla prevenzione.

Il concetto di prevenzione si delinea perfettamente nel principio di *"Shift Security Left"* [47], esso si propone di ribaltare il tradizionale comportamento di valutare la sicurezza solo a prodotto finito, sostituendolo con l'applicazione dei controlli all'atto di definizione dei requisiti.
L'applicazione di questo principio avvantaggerebbe la creazione del software perchè

si avrebbe uno sviluppo più veloce, si migliorerebbe la sicurezza e si ridurrebbero i costi.

L'obiettivo finale è, dunque, il raggiungimento della *"Security by Design"* [48] ovvero creare un software progettato esclusivamente per essere sicuro, in modo da anticipare e minimizzare gli impatti che eventuali vulnerabilità possono generare.

Si tratta principalmente di un approccio sistematico che, tramite l'applicazione di buone norme e l'inserimento di controlli durante le fasi di progettazione e di realizzazione di un servizio, riduce le possibilità di attacchi informatici.

La security by design, inoltre, introduce un vantaggio considerevole a livello economico e di immagine aziendale perchè riesce a: ottimizzare i costi di sviluppo, ridurre le spese dovute a sistemi difensivi retroattivi, limitare l'impatto negativo di eventuali attacchi, ottenere un codice pulito, funzionale e di qualità e, infine, semplificare la correzione di errori.

Uno dei comportamenti principali da seguire per lo sviluppo sicuro è il rispetto delle "Best Practices", che consistono in un insieme di attività, comportamenti e procedure univocamente definite da esperti per migliorare il livello di sicurezza di un servizio. L'applicazione di queste buone norme permette di correggere alcune vulnerabilità che possono presentarsi durante l'utilizzo di qualsiasi servizio. Di seguito, verranno mostrate delle best practices che permettono di prevenire il presentarsi delle minacce viste nel capitolo precedente.

4.2 Best practices per AWS IAM

Le credenziali AWS sono una componente molto delicata, perchè permettono di utilizzare gli strumenti del cloud service provider, perciò occorre che siano rispettate tutte le best practices del caso per non favorire l'insorgenza di errori di configurazione [49].

Innanzitutto, ogni possessore di credenziali AWS deve conservare in modo sicuro le sue password e cambiarle periodicamente. Inoltre, bisogna rinforzare la sicurezza dell'account abilitando la multi-factor authentication.

Di fondamentale importanza è la gestione dell'account root di AWS, che non deve mai essere usato per eseguire i compiti giornalieri, ma va usato solo per la creazione dei diversi utenti. Deve poi essere protetto nel modo più sicuro possibile, perchè tramite l'account root si può accedere a tutti i servizi di AWS, anche pagamenti o funzioni amministrative.

Durante la creazione di nuovi utenti bisogna rispettare il principio del *Least privilege*, per garantire che ognuno di loro abbia la possibilità di accedere solo alle

risorse per le quali è autorizzato.
Rispettando questa buona pratica si riesce a: ridurre la superficie esposta agli attacchi informatici, bloccare la propagazione dei malware e migliorare la produttività riducendo i rischi.

Tra le varie potenzialità che AWS offre ci sono le possibilità di creare gruppi, ruoli e policies, e per ognuna di queste si può definire una norma da seguire:

- Creazione dei gruppi: da usare quando si dispone di un numero considerevole di utenti che devono avere le stesse policies. Così facendo, verranno evitati possibili errori nella definizione delle policies stesse.
  Ad esempio, se si volessero dare dei privilegi a degli utenti basterebbe creare il gruppo "Administrator" con le corrispondenti policies e associare gli utenti a quel gruppo.

- I ruoli e la loro assunzione: i ruoli in AWS sono il concetto più potente e flessibile all'interno dello IAM.
  Sono molto simili agli utenti IAM ma con l'accezione che un ruolo può essere assunto da più entità, anche contemporaneamente. In questi casi è buona norma che gli utenti abbiano un account con autorizzazioni minime e che, per lavorare ad un determinato compito, assumano il ruolo corrispondente.

- Gestione delle policies: la gestione delle policies è molto delicata e, come già detto in precedenza, all'atto di creazione di un utente bisogna definire solo le policies che gli permettono di avere i minimi privilegi.

4.3 Best Practices per Kubernetes

Kubernetes è il servizio che permette di gestire tutte le risorse utilizzate all'interno dell'infrastruttura ed è perciò fondamentale che la sua configurazione sia effettuata con attenzione per evitare di lasciare terreno fertile agli attacchi.

Nel capitolo precedente sono stati introdotti quattro possibili errori di configurazione all'interno del cluster Kubernetes, di seguito saranno mostrate delle best practices [50] per la corretta gestione di questo servizio:

- Non usare il namespace di default: durante la creazione del file yaml per il deployment del pod è raccomandato specificare il namespace di destinazione. Un possibile esempio di file yaml con namespace correttamente configurato può essere questo:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: deployment
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

- Disabilitare la privilege escalation: l'abilitazione del flag "allowPrivilegeEscalation" consente ad un container di venire eseguito in modalità privilegiata, perciò, è consigliabile che questa policy sia disabilitata nel file yaml del pod. Per fare questo si può aggiungere il seguente frammento:

```
spec:
containers:
- name:
securityContext:
- allowPrivilegeEscalation: false
```

- Imporre dei limiti di utilizzo per la CPU e la memoria: così facendo il container non supererà mai la soglia di risorse prefissata e non utilizzerà tutte le risorse a disposizione.

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: image:latest
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

- Assicurarsi che il traffico tra la Kubernetes API e kubelet sia cifrato: esiste anche in questo caso un flag che permette a queste due entità di dialogare tramite protocolli non sicuri. È quindi utile specificare l'attivazione della policy *–kubelet-https*:

```
metadata:
creationTimestamp: null
labels:
component: kube-apiserver
tier: control-plane
name: kube-apiserver
namespace: kube-system
spec:
containers:
- command:
 - kube-apiserver
 - —-kubelet-https=true
```

4.4 Docker best practices

Le ultime best practices che si andranno a trattare sono quelle per la configurazione dei containers Docker [51].

- Eseguire i containers non in modalità root: questa best practices rientra tra quelle del DockerFile. Durante la creazione del file, bisogna evitare che il container venga eseguito con i privilegi di root. Per farlo, si deve aggiungere l'istruzione USER e passare ad un utente non-root. Nel frammento di Dockerfile seguente si può vedere una possibile configurazione corretta:

```
FROM alpine:3.15
RUN adduser -D myuser
USER myuser
```

- Proteggere le Docker APIs: nel caso in cui si volessero esporre delle API di Docker è importantissimo che le connessioni che si andranno a creare siano protette da protocolli sicuri come TLS o SSH. Per farlo si potrebbe cambiare la variabile d'ambiente DOCKER_HOST in modo da forzare la Docker CLI ad aprire una connessione usando SSH.

```
export DOCKER_HOST=ssh://docker-user@host1.example.com
```

Capitolo 5

Analisi di sicurezza

Questo capitolo mostrerà una possibile modalità di ricerca delle vulnerabilità di un'applicazione a container.

Si inizierà definendo il concetto di security assessment per poi restringere il campo alla creazione e all'uso di una pipeline di sicurezza spiegando il ruolo che avranno gli strumenti di cui è composta e come è stato possibile definirla.

5.1 Security Assessment

Attualmente, il punto di partenza per creare un sistema sicuro risiede nel concetto di Security Assessment [52] ovvero l'insieme di attività di identificazione dei componenti di un sistema informatico e la valutazione dei controlli di sicurezza, atti a determinare il grado di protezione e vulnerabilità dei componenti stessi.
In un mondo dove ormai i crimini informatici sono all'ordine del giorno, è conveniente per un'azienda essere consapevole che non fornire un prodotto sicuro implica, nel caso di un attacco, la perdita di reputazione e, conseguentemente, anche di mercato. Lo svolgimento di un security assessment va effettuato tramite l'uso di varie tecniche di sicurezza:

- Vulnerability Assessment;
- Penetration Testing;
- Code Analysis;
- Formal Verification;
- Auditing.

Per il prosieguo della trattazione si utilizzerà la pipeline Jenkins per coprire le tecniche di Vulnerability Assessment e di Code Analysis.

5.2 Pipeline Jenkins

Jenkins [53] è un automation server open source utilizzato per automatizzare qualsiasi processo di compilazione, testing, delivery e deploy. È uno degli strumenti più diffusi nel panorama aziendale e può essere utilizzato per applicare il concetto di DevSecOps. Jenkins, infatti, tramite una serie di stages che definiscono una pipeline permette di fare CI/CD.
Inoltre, la presenza di numerosi plugin permette di ottenere maggiore versatilità ed efficienza.

Per questi motivi, si è deciso di creare una pipeline Jenkins, che verrà lanciata manualmente, per verificare il livello di sicurezza dell'ambiente e dell'applicazione.

Una pipeline Jenkins è scritta in linguaggio Groovy [54] (usato per la programmazione di piattaforme Java) e può essere di due tipi: declarative o scripted [55].
Inizialmente l'unica possibile modalità di creare una pipeline Jenkins era tramite una scripted pipeline e come vantaggi si avevano un'ottima integrazione con tutti i task che si volevano svolgere e la possibilità di fare delle injection di codice Groovy.
Successivamente è nata la declarative pipeline allo scopo di renderne la creazione più facile e più leggibile, inoltre, sono state aggiunti il supporto di istruzioni condizionali, l'accesso alle variabili d'ambiente e possibilità di logging e segnalazioni di errori.
Per questi vantaggi e la possibilità di mantenere un codice snello e più semplice si è deciso di creare una declarative pipeline.

La sintassi che deve essere utilizzata in questo tipo di servizio è la seguente:

```
pipeline{

    agent any

    parameters{
        ...
    }

    environment{
        ...
    }
```

```
    stages{
        stage("Stage 1"){
            steps{
                ...
            }
        }

         stage("Stage 2"){
            steps{
                ...
            }
        }
        ...
    }
}
```

Inizialmente si dichiara l'intento di creare una pipeline con la corrispondente istruzione, che raccoglierà poi al suo interno i vari campi: "parameters" per la definizione di variabili che possono essere modificate prima dell'esecuzione, "environment" per le variabili ambientali e poi "stages" per la definizione dei vari compiti della pipeline.

Ognuno degli stage eseguirà al suo interno i comandi necessari per avviare un tool o per completare un'operazione.

5.2.1 Organizzazione della Pipeline

Gli stages della pipeline Jenkins sono organizzati nel seguente modo:

- **Variabili di controllo**: sono dei parametri, utili per la gestione degli stages. Infatti, nel caso in cui non si voglia utilizzare uno o più tools basterà smarcare la corrispondente variabile all'atto di avvio dell'analisi;

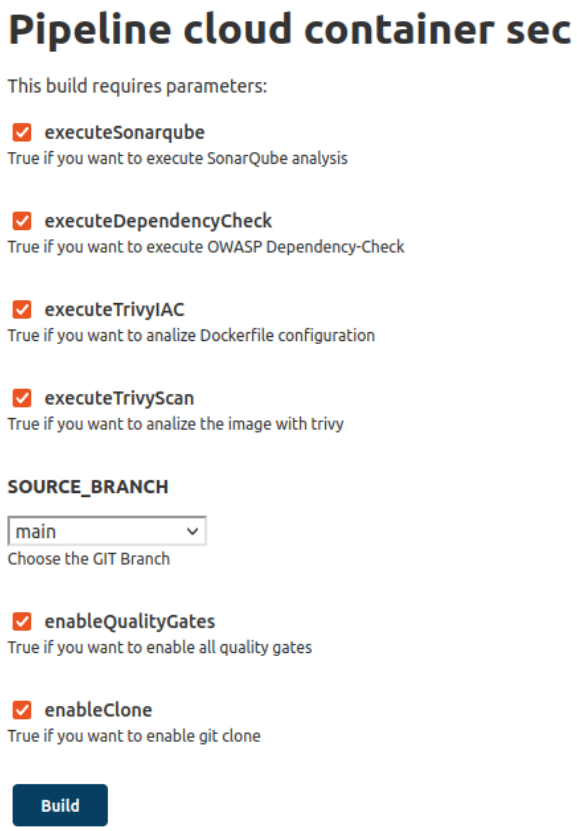


Figura 5.1: Variabili della pipeline

- **Git Clone**: in questo stage si ottiene il codice sorgente dell'applicazione dal repository GitHub tramite il comando di clone;

- **Build**: processo in cui dal file sorgente vengono generati i file eseguibili tramite Maven, strumento di compilazione e gestione dei progetti Java;

- **Tools di analisi**: si passerà poi per tre stages, in ognuno dei quali verrà eseguito un tool. Si partirà da un tool di analisi statica del codice, poi si passerà ad un tool per il controllo delle dipendenze di libreria e infine all'analisi del DockerFile.
  Alla fine di ognuno di questi passaggi ci sarà un Quality Gate che deciderà se la pipeline continuerà ad essere eseguita oppure, nel caso di vulnerabilità critiche, se deve essere fermata.

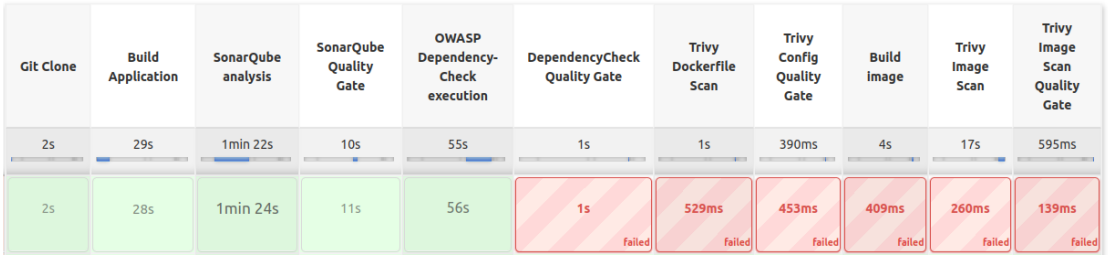


Figura 5.2: Pipeline bloccata dal quality gate

- **Build dell'immagine**: se il codice sorgente è privo di criticità si passerà al processo di creazione dell'immagine Docker;
- **Analisi dell'immagine**: l'immagine appena generata verrà poi nuovamente analizzata.

Una volta terminata l'esecuzione, se la pipeline non riporta fallimenti, si può procedere con il deploy sul cluster.

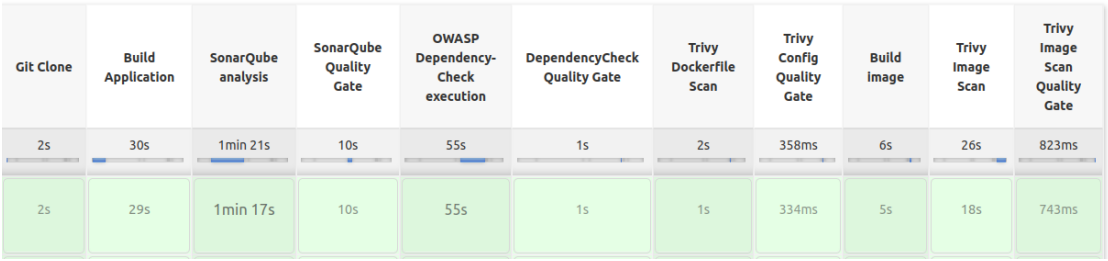


Figura 5.3: Pipeline terminata correttamente

5.3 Creazione della pipeline

Si passerà ora ad illustrare i passi seguiti per la definizione della pipeline, partendo dalla configurazione di Jenkins per poi passare alla descrizione di ogni stages.

5.3.1 Configurazioni di Jenkins

Il server di Jenkins può essere eseguito in diverse modalità: all'interno di un pod nel cluster oppure come servizio all'interno di una macchina host. In questo caso è stato deciso di utilizzare la seconda opzione per due motivi principali: il primo per questioni di sicurezza, in quanto non si voleva inserire altre sorgenti di vulnerabilità all'interno dell'infrastruttura, e il secondo per questioni di risorse, perchè eseguire come pod il server Jenkins avrebbe portato l'aumento del carico di lavoro all'interno del cluster.

Si è quindi passato all'istallazione e all'avvio del servizio che sarà raggiungibile sulla porta 8080, digitando all'interno del browser l'indirizzo *http://localhost:8080*. A questo punto si può procedere alla creazione un nuovo utente e all'inserimento di una nuova password, diversa da quella di default.

Dopo la corretta configurazione dell'utenza si prosegue con la generazione della pipeline, le si assegna un nome e tramite la corrispondente finestra di testo si può iniziare a scrivere il codice che la compone.

5.3.2 GitHub Stage

La prima azione necessaria da seguire è stata lo stage di clonazione del codice sorgente dal repository.
Vengono quindi definite tre variabili: la prima di tipo parametrico che permette di scegliere se saltare o meno questo stage, la seconda, ancora di tipo parametrico che consente alla variabile "SOURCE_BRANCH" di prendere il nome del branch che si vuole analizzare, mentre la terza di tipo ambientale che racchiude l'URL della reporitory.

Successivamente si prosegue con il passaggio operativo, ovvero quello composto dallo stage chiamato "Git Clone". Inizialmente la clausola *when* controlla che la variabile *enableClone* sia selezionata come true, se è così si passa ad eseguire il comando di clonazione del branch selezionato. Si fornisce, quindi, una directory dove scaricare i file indicati dall'URL e dal branch.
Dal codice successivo di può notare inoltre, la presenza della dicitura *"github_cred"*

che consiste in un token GitHub salvato all'interno di una variabile protetta da Jenkins e che garantisce l'accesso al repository privato.

```
parameters{
     booleanParam(name: 'enableClone', defaultValue: true,
description: 'True if you want to enable git clone')
     choice(name: 'SOURCE_BRANCH', choices: [...], description: '
Choose the GIT Branch');
}

environment {
     GITHUB_URL = 'https://github.com/repo_link'
}

stages {
     stage('Git Clone') {
        when {
          expression {
            return params.enableClone;
          }
        }
        steps {
          dir('app') {
            script {
                 git branch: params.SOURCE_BRANCH,
                 credentialsId: 'github_cred',
                 poll: false,
                 url: GITHUB_URL,
            }
         }

        }

     }
 }
```

5.3.3 Build Stage

Il passo successivo consiste nell'effettuare la build del codice sorgente, che genererà i corrispondenti file Jar dell'applicazione. Per fare ciò si è usato Maven con il suo comando:

```
mvn –B clean install
```

È stato scelto di lanciare il comando in modalità batch tramite l'istruzione "-B" perchè, al contrario della modalità interattiva, rimuove tutti i log aggiuntivi che mostrano le percentuali di completamento, in questo modo i log risulteranno più comprensibili.
Inoltre, l'aggiunta del comando clean permette di pulire eventuali precedenti build all'interno della cartella.

Passando ora al codice della pipeline, è stato nuovamente inserito uno stage nel quale si andrà ad utilizzare il comando groovy *sh* che consente di lanciare comandi nella directory.

```
stage('Build Application') {

    steps {
        script {
            dir('app') {
              script {
                     sh "mvn –B clean install"
                    }
            }
        }
    }
}
```

Dopo aver configurato gli step principali, si può passare alla creazione degli stages di analisi. Grazie ad essi, vengono eseguiti i comandi dei corrispondenti tools, che permetteranno di rintracciare le vulnerabilità.

Nel vasto panorama di tools presenti online, si è deciso di sceglierne alcuni che siano open source e il cui uso sia abbastanza frequente nel contesto aziendale.
Si riporteranno più nel dettaglio gli strumenti scelti e il codice utilizzato nella pipeline.

5.3.4 Sonarqube Stage

La prima parte operativa della pipeline è stata assegnata ai cosiddetti SAST (Static application security testing), ovvero strumenti che permettono di effettuare l'analisi statica del codice sorgente.
L'analisi statica è una tecnica che analizza e valuta struttura, forma e contenuto del codice senza che esso sia eseguito.

La scelta è stata dettata dal fatto che Sonarqube è ampliamente utilizzato in Blue Reply, l'azienda nella quale è stato svolto il lavoro di ricerca.

Sonarqube [56] è appunto uno strumento di analisi statica usato per identificare e migliorare la qualità del codice prodotto e, come anche Jenkins, viene eseguito come servizio all'interno dell'host e sarà in ascolto sulla porta 9001.
Tramite le analisi effettuate da Sonarqube si riescono ad indentificare dei problemi che, grazie a delle regole, verranno catalogati in quattro tipi: bugs, code smells, security hotspots e vulnerabilità. Ogni qualvolta che la ricerca trova un possibile issue nel codice, gli viene assegnato un livello di gravità tra *Blocker, Critical, Major o Minor*.

Per il codice operativo della pipeline, anche in questo caso è stato deciso di inserire una variabile parametrica per dare la possibilità di saltare questo passaggio. Invece, per la parte di comunicazione con il server del CI/CD è stato necessario l'ausilio di SonarQube Scanner, un plugin di Jenkins che come si vedrà nel successivo frammento di codice, permette una facile esecuzione dell'analisi.

```
stage('SonarQube analysis') {
      when {
      expression {
        return params.executeSonarqube;
      }
    }
    steps {
        dir('app') {
        withSonarQubeEnv('sonarqube') {
            sh 'mvn sonar:sonar'
        }
      }
    }
  }
```

In questo caso è stato anche utile sfruttare la possibilità di definire un Sonarqube Quality Gate che permette di bloccare l'esecuzione della pipeline nel caso in cui il risultato dell'analisi riporti valori che non rispettano delle condizioni.

La creazione di questa entità deve essere fatta dalla dashboard di SonarQube ed è possibile scegliere diverse condizioni da testare per adattarsi a tutte le applicazioni. Si possono fissare delle soglie basate sulle seguenti metriche [57]:

- Complessità: si intende il calcolo del numero dei path nel codice. Per esempio, quando il flusso di esecuzione di una funzione si divide, aumenta di uno la complessità. Istruzioni che portano all'aumento sono: if, for, while, case, catch, throw, &&, ||, ?;
- Duplicazioni: controlla che il codice o i file non siano duplicati;
- Issues: ovvero possibili problemi che possono verificarsi all'interno sia del nuovo codice da analizzato che su tutto il codice;
- Maintainability: basata sul numero dei *code smells* che non sono dei veri e propri bugs, ma indicazioni di possibili problematiche più profonde all'interno del codice. Un esempio di code smell può essere una funzione con più istruzioni di return, ma che danno come risultato lo stesso valore;
- Reliability: controllano il numero di bugs;
- Security: queste metriche identificano il numero di vulnerabilità;
- Dimensioni: permettono di definire una soglia sul numero delle classi, delle linee di commenti, delle directories, delle linee di codice.
- Test: verificano, per ogni linea di codice che contiene una condizione, che siano coperte tutte le possibilità;

Tra tutte le possibilità sopraccitate si è deciso di concentrarsi principalmente sul controllo delle metriche di Security e Issues e, perciò, come si può vedere dall'immagine seguente, il quality gate creato impone che il numero di vulnerabilità e di Critical issues non sia maggiore di zero, sia nell'intero codice, che ogni volta che vengono aggiunte nuove istruzioni.

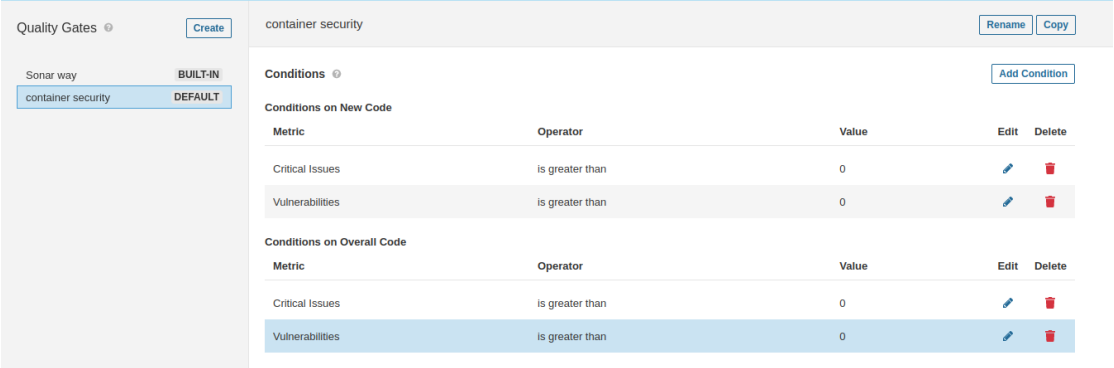


Figura 5.4: Sonarqube quality gate

Dopo la definizione delle condizioni, si può passare ad integrare nella pipeline il controllo da parte del quality gate ma, per farlo bisogna tenere in considerazione il flusso di comunicazione tra Jenkins e Sonarqube. A tal proposito, la seguente immagine sarà molto esplicativa:

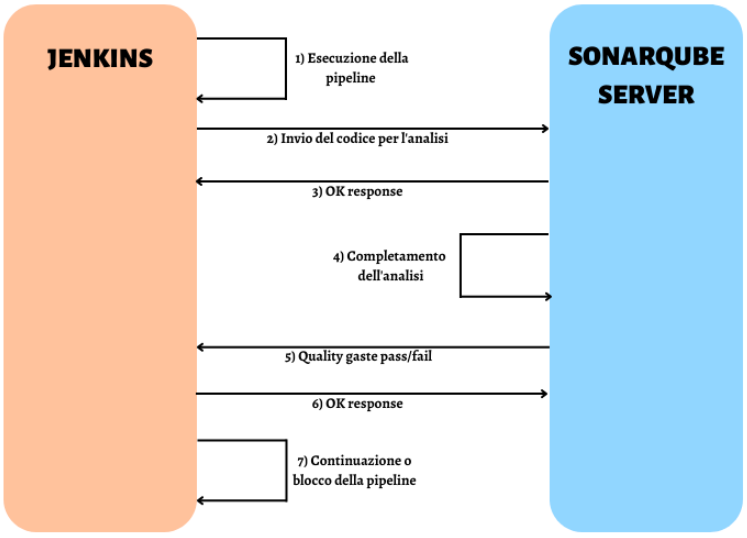


Figura 5.5: Flusso di comunicazione tra Jenkins e Sonarqube

Dopo che Sonarqube ha inviato la risposta di fine analisi a Jenkins (indicata nell'immagine con il numero 3), quest'ultimo entra nello stage "SonarQube Quality Gate" per verificare che i controlli siano corretti. Però, i test sul Quality gate vengono eseguiti da Sonarqube in asincrono e dopo aver inviato la notifica di fine analisi (punto 4 dell'immagine). Perciò, Jenkins leggerà un risultato che ancora non è stato caricato e lo interpreterà come positivo e, di conseguenza, la pipeline non viene bloccata anche se il Quality gate ha riportato il fallimento.

Per la risoluzione di questa piccola limitazione, si è deciso di ritardare l'esecuzione dello step corrispondente di dieci secondi, in modo da concedere un ampio margine di tempo a Sonarqube di rendere disponibili l'esito del confronto sulle condizioni. Inoltre, è stato inserito un timeout di dieci minuti per tutelarsi nel caso in cui dovessero verificarsi errori durante il controllo.

Come già visto negli stages precedenti, anche per questo è stata aggiunta una variabile parametrica per saltare il processo di controllo da parte di tutti i quality gate, che è messa in AND logico con quella che identifica l'esecuzione di Sonarqube.

```
    stage("SonarQube Quality Gate") {
     when {
       expression {
        return params.enableQualityGates && params.executeSonarqube;
       }
     }
     steps {
           dir('app') {
               sh 'sleep 10'
                   timeout(time: 10, unit: 'MINUTES') {
                       waitForQualityGate abortPipeline: true
                   }
           }
      }
  }
```

5.3.5 OWASP Dependency Check Stage

Il secondo strumento utilizzato è OWASP Dependency-check [58], un tool open source di analisi delle dipendenze capace di scannerizzare tutte le librerie usate nel codice e identificare quelle che possiedono delle vulnerabilità note.
Anche in questo caso, è stato scelto perchè utilizzato nell'azienda in cui è stato svolto il presente lavoro di tesi.

Il programma scarica, al primo uso, tutti i CVE disponibili e confronta se in una dipendenza è presente un CPE, se è così riporta la corrispondente entry del CVE.

La configurazione di questo strumento richiede l'installazione del plugin OWASP Dependency-Check, che aggiungerà i comandi necessari ad eseguire l'analisi e utili per la creazione del corrispondente quality gate.

```
stage('OWASP Dependency-Check execution') {
    when {
            expression {
              return params.executeDependencyCheck
            }
       }
    steps {
        dir('app'){
            script {
                dependencyCheck additionalArguments:
                ' -o "./target/" -s "./" -f "HTML" ',
                 odcInstallation: 'Dependency-Check'
                if (params.enableQualityGates) {
                    dependencyCheckPublisher (
                    pattern:"target/dependency-check-report.html",
                                    failedTotalHigh: 0,
                                    failedTotalCritical: 0
                    )
                } else {
                    dependencyCheckPublisher (
                    pattern:"target/dependency-check-report.html",
                        )
                }

                archiveArtifacts artifacts:
                'target/dependency-check-report.html'
            }
        }
    }
}
```

Il comando per lanciare l'analisi è composto dalle istruzioni:

- "-o": la directory dove verrà salvato il report;
- "-s": che definisce la directory dove è contenuto il codice da analizzare;
- "-f": per indicare il tipo di formato del file di output.

A questo punto, per il salvataggio del report le strade percorribili sono due: la prima che considera la presenza del quality gate, che sarà trattata nel prossimo sottocapitolo, mentre la seconda si occuperà solamente di invocare il componente che pubblicherà i risultati nella destinazione definita nella stringa *pattern*.
I risultati memorizzati tramite questo metodo avranno come tempo di vita quello dell'esecuzione della pipeline, perciò non saranno più disponibili al termine della pipeline stessa. Per risolvere questo problema è stato aggiunto il comando *"archiveArtifacts"* per salvare in modo permanente il report appena prodotto.

5.3.6 OWASP Dependency-Check Quality Gate

Si passerà ora a coprire la condizione lasciata in sospeso nella sottosezione precedente: nel caso in cui l'analisi di Dependency-Check preveda il controllo da parte del quality gate, l'invocazione del componente per il salvataggio dei risultati deve essere corredato con i requisiti minimi per considerare l'analisi superata.

È in questo passaggio che viene prodotto il risultato, che verrà scritto nella variabile globale *"currentBuild.result"* se l'analisi ha avuto esito positivo, al contrario si salverà il valore "FAILURE".

Tra le varie soglie definite nel plugin di Dependency-Check [59] sono state considerate solo quelle che controllano il numero totale di vulnerabilità di tipo critico e alto.

Il codice operativo di questo stage effettua un controllo sulla variabile sopracitata e, se questa riporta il fallimento, lancia un errore per bloccare preventivamente la pipeline.

```
stage('DependencyCheck Quality Gate') {
    when {
        expression {
            return params.enableQualityGates &&
            params.executeDependencyCheck
        }
    }
    steps {
        dir('app') {
            script {
                if (currentBuild.result == 'FAILURE') {
                    error('Dependency Check Quality Gate:
                    found at least 1 Critical or High
                    vulnerability in one dependency')
                }
            }
        }
    }
}
```

Con questo stage, si chiude la parte di analisi del codice sorgente Java per passare alla ricerca di vulnerabilità nella costruzione dei containers e delle immagini.

5.3.7 Trivy DockerFile Stage

Il tool che è stato considerato per l'analisi sui file Docker che compongono l'applicazione è Trivy [60], uno strumento molto potente sviluppato da AquaSecurity. Esso permette di rintracciare delle vulnerabilità di software all'interno di containers e artifacts.
In questo stage verrà utilizzato per effettuare l'analisi sui file di IaC (Infrastructure as a Code) [61] ovvero i file che tramite del codice caratterizzano le configurazioni dell'infrastruttura, come ad esempio il DockerFile oppure il file yaml per il deployment sul cluster.

Rispetto ad altri software disponibili online come, ad esempio, Docker-Bench Security [62] oppure Checkov [63], Trivy è in grado di generare un output più comprensibile rispetto agli altri due perchè riporta le vulnerabilità con il loro grado e una possibile soluzione, al contrario gli altri due restituiscono una serie di test con il rispettivo risultato, ma la mole di controlli e la similitudine tra i nomi dei controllori possono generare confusione.

Anche qui, ricorre l'uso della variabile per saltare il controllo, mentre per avviare il tool bisogna avere precedentemente installato Trivy sulla macchina host per poi lanciarne l'esecuzione con un comando sh. In questo stage, l'analisi dei file IaC è possibile grazie alla modalità *config* di Trivy il cui output poi, verrà rediretto, all'interno di un file testuale.

```
stage("Trivy Dockerfile Scan"){
      when {
      expression {
        return params.executeTrivyIAC
      }
    }
      steps {
          dir('app') {
              script {
                  sh "trivy config ./ > ./target/trivyConfig.txt"
              }
          }
      }
  }
```

5.3.8 Trivy DockerFile Quality Gate

Una limitazione di Trivy è che, nell'ambiente di Jenkins, non è definita una vera e propria entità quality gate come per i precedenti tools, quindi per la creazione di questo stage è stato pensato uno script che verifica la presenza all'interno del report di vulnerabilità di tipo CRITICAL oppure HIGH.

Per raggiungere l'obbiettivo si è usato il comando Linux *grep* con un'espressione regolare per identificare se, dopo la stringa "CRITICAL" o "HIGH", è presente un numero diverso da zero, se così è, salva nelle variabili "criticalVuln" e "highVuln" il valore 1.
Successivamente, se una delle due variabili assume quel valore, viene lanciato l'errore che termina la pipeline.

```
stage('Trivy Config Quality Gate') {
    when {
        expression {
            return params.enableQualityGates &&
            params.executeTrivyIAC
        }
    }
    steps {
        dir('app/target') {
            script {
                criticalVuln = sh(
                    script:
                        "grep --regexp='CRITICAL: [1-9][0-9]*'
                            ./trivyConfig.txt",
                        returnStatus: true
                 )
                highVuln = sh(
                    script:
                        "grep --regexp='HIGH: [1-9][0-9]*'
                            ./trivyConfig.txt",
                        returnStatus: true
                )
                if (criticalVuln == 1 || highVuln == 1) {
                    error('Trivy IaC Quality Gate: found at least 1
                    Critical or High vulnerability in one dependency'
    )
                }
            }
        }
    }
}
```

5.3.9 Docker Build Stage

Dopo aver controllato la correttezza delle istruzioni del DockerFile si può procedere con la creazione dell'immagine che verrà poi analizzata nuovamente per cercare altre vulnerabilità.

```
stage("Build image"){
    when {
        expression {
          return params.executeTrivyScan
        }
    }
    steps {
        dir('app') {
            script {
                sh "docker build -t ${IMAGE_NAME} ./"
            }
        }
    }
}
```

Si procederà all'esecuzione del comando docker build definendo un tag per assegnare il nome all'immagine e il path dove è presente il Dockerfile.
Questo passaggio è vincolato dalla stessa variabile parametrica che definisce la non esecuzione dello stage di analisi dell'immagine.

5.3.10 Trivy Image Scan Step

Nell'ultimo stage della pipeline si analizzerà l'immagine generata precedentemente.

I tools con queste capacità sono molto vari, perciò per fare una prima scrematura si è deciso di considerare solo quelli open-source e tra questi si è preso in considerazione Clair, Anchore Engine e Trivy [64].

Clair [65], software sviluppato da RedHat. Le sue limitazioni riscontriamo: un'accuracy non perfetta, l'incapacità di analizzare le librerie utilizzate per la creazione dell'immagine, la difficile integrazione in un CI/CD. Inoltre, il tool risulta non semplice da utilizzare perchè basato su vari client, di cui la maggior parte deprecati.
Ancore Engine [66] è molto simile a Clair ma ha la capacità di analizzare le dipendenze. Rimane, però, una non semplice esecuzione dell'analisi e una grande limitazione sul non riportare nel resoconto le vulnerabilità che non hanno una correzione.
Trivy invece presenta molte ottime possibilità, come ad esempio: la semplicità d'uso, l'ottima integrazione all'interno di un CI/CD, un output facilmente comprensibile e la possibilità di analizzare anche le dipendenze.

Per chiarezza espovitiva verrà fornita una tabella riassuntiva con il resoconto dei confronti tra questi tre sofware:

TOOL	**ANALISI DELLE DIPENDENZE**	**FACILITÀ D'USO**	**INTEGRAZIONE CON CI/CD**
Trivy	8 linguaggi supportati	molto facile	perfetta
Clair	X	difficile	difficile
Ancore Engine	4 linguaggi supportati	semplice	perfetta

Tabella 5.1: Tabella confronti tools

Considerati i punti di forza e di debolezza dei tools precedentemente descritti, la scelta è ricaduta su Trivy, che verrà utilizzato per effettuare l'analisi sull'immagine e verificare che la sua costruzione non sia partita da delle base images corrotte.

```
stage("Trivy Image Scan"){
    when {
    expression {
      return params.executeTrivyScan
        }
    }
    steps {
        dir('app') {
                script {
                   sh "trivy image ${IMAGE_NAME} >
                   ./target/trivyAnalisys.txt "
                }
        }
    }
}
```

È stato lanciato il comando Trivy in modalità *image* inserendo anche il nome dell'immagine e, come fatto anche nella sezione 5.3.8, è stato rediretto l'output su un file testuale per salvare il resoconto dell'analisi.

5.3.11 Trivy Scan Quality Gate

Lo stage finale della pipeline consiste nello stesso quality gate visto nella sezione 5.3.8. La sola differenza sta nel nome del file da leggere, che nel caso precedente ispeziona il file *"trivyConfig.txt"*, mentre in questo esamina il file *"trivyAnalisys.txt"*.

A questo punto, se il quality gate non blocca l'analisi, si può affermare che la pipeline è terminata con successo e si può procedere con il deploy del container nel cluster Kubernetes. Però, farlo senza prima aver verificato che l'intera infrastruttura sia sicura, potrebbe vanificare tutti gli sforzi impiegati per l'analisi dell'applicazione. Per tale motivo, prima dell'effettivo deploy, si procederà quindi all'analisi infrastrutturale.

5.4 Analisi dell'infrastruttura

Come appena descritto, l'analisi dell'infrastruttura è uno step non trascurabile se non si vuole correre il rischio di vanificare tutto il lavoro fin qui svolto. Inoltre, gestire correttamente un cluster Kubernetes non è semplice ed è soprattutto molto facile cadere in piccoli errori che possono comprometterne l'integrità. È dunque molto importante analizzare quello che è l'ambiente di sviluppo cloud.

Anche per questo tipo di analisi i tools a disposizione sono numerosi, sono stati considerati, ma non scelti, due prodotti creati da AquaSecurity: Kube-Bench [67] e Kube-Hunter [68]. Il primo permette di analizzare tutti file che compongono l'infrastruttura del cluster alla ricerca di vulnerabilità o cattive configurazioni, mentre il secondo identifica le debolezze del cluster Kubernetes sulla base delle informazioni contenute all'interno del MITRE ATT&CK, ovvero una libreria di tutti i possibili pattern d'attacco.

Le limitazioni di Kube-Bench interessano l'output che, come nel caso di Docker-Bench Security, è poco organizzato e che può portare a confusioni. Invece per Kube-Hunter, la limitazione principale è quella di concentrarsi solo sui controlli del MITRE.

Per questa ricerca, si è scelto quindi di utilizzare un tool che permettesse di avere molte funzionalità e che offrisse anche una dashboard: Kubescape di Armo [69], si è rivelato uno dei più validi tra quelli scoperti.

Il tool offre una dashboard altamente user-friendly e, una volta eseguito il corrispondente comando, si connette al cluster per effettuarne l'analisi.
L'indagine consisterà nell'applicare dei controlli usati di frequente nel panorama di DevSecOps e dei controlli generati da frameworks sviluppati da Armo, MITRE e NSA. Tale analisi sarà utile per individuare rischi di sicurezza nel cluster, nei file yaml o Helm e nelle immagini utilizzate nei pods.

Per effettuare l'analisi basta creare un profilo nel portale di Armo e installare Kubescape sull'host connesso al cluster. Poi, tramite il successivo comando, copiabile dalla dashboard, si procederà all'analisi che genererà il report successivamente messo a disposizione nella propria pagina personale del portale.

```
kubescape scan --submit --account=[ARMOAccountID]
```

Come è possibile osservare dalla figura successiva, il risultato dell'analisi verrà raccolto in una tabella, in cui ogni entry sarà composta dal nome del controller, da una descrizione del controllo, dallo stato (failed o passed), dal grado di criticità e da una possibile remediation.
Kubescape permetterà quindi di scoprire i punti deboli del cluster Kubernetes e offrirà anche delle possibili soluzioni.

STATUS	ID	CONTROL NAME	DESCRIPTION	FAILED	SEVERITY	REMEDIATION
Failed	C-0050	Resources CPU limit and request	This control identifies all Pods for which the CPU...	26	High	Set the CPU limit or use exception mechanism to...
Failed	C-0004	Resources memory limit and request	This control identifies all Pods for which the memo...	24	High	Set the memory limit or use exception mechanism to...
Failed	C-0074	Containers mounting Docker socket	Mounting Docker socket (Unix socket) enables...	1	Medium	Remove docker socket mount request or define ...
Failed	C-0056	Configured liveness probe	Liveness probe is intended to ensure that workload...	13	Medium	Ensure Liveness probes are configured wherever...
Failed	C-0044	Container hostPort	Configuring hostPort requires a particular port...	1	Medium	Avoid usage of hostPort unless it is absolutely...
Failed	C-0018	Configured readiness probe	Readiness probe is intended to ensure that...	18	Low	Ensure Readiness probes are configured wherever...
Passed	C-0061	Pods in default namespace	It is recommended to avoid running PODs in cluster...	0	Low	Create necessary namespaces and move a

Figura 5.6: Tabella generata da Kubescape

Capitolo 6

Creazione e analisi di un applicativo

Il ciclo di sviluppo di un applicativo è fondamentale per la creazione di un prodotto che sia affidabile e sicuro, e spesso, per realizzarlo sono necessari tanto tempo e tante risorse.

Per questo lavoro di ricerca, si è deciso di simulare la creazione di un'applicazione web e della sua infrastruttura che ha come scopo principale quello di validare il corretto comportamento della pipeline e dell'analisi infrastrutturale.

L'organizzazione di questo capitolo verterà su due punti: il primo descriverà come è stata creata l'applicazione e il suo ambiente cloud, mentre il secondo riporterà i risultati delle analisi.

6.1 Creazione di un'applicazione

Si definirà come è stato creato l'applicativo di test e, per comodità espositiva, verrà diviso in sottosezioni, diverse per ognuno dei passi principali della generazione. Oltre a questo, verranno definite delle vulnerabilità che sono state volutamente inserite all'atto di sviluppo.

6.1.1 Codice sorgente

Per questa trattazione è stata creata una *"To-Do Application"* capace di definire e salvare in un database dei tasks. Il tutto utilizzando Java Spring Boot come linguaggio.

Il framework Spring Boot è in grado di offrire ottime astrazioni, chiamate annotazioni [70], che permettono di aggiungere informazioni al programma e semplificano di molto lo sforzo nella programmazione; ed esempio, l'annotazione @GetMapping mappa una richiesta di tipo HTTP GET al corrispondente handler che la andrà a gestire.

La classe principale è la "ToDo-Item class" che rappresenta i tasks da definire e che ha come attributi: un Long come identificativo, una stringa come descrizione del e un booleano che definisce se il task è stato portato a termine o meno.

Una volta definito l'elemento base, si passa alla creazione dei componenti [71], ovvero delle entità che si occuperanno della logica dell'applicazione:

1. Il primo componente è la Repository, generata tramite l'annotazione @Repository, un persistent storage che si occuperà del salvataggio dei dati;
2. Il secondo è il Controller, con annotazione @Controller, il cui compito è interfacciarsi con le richieste dei client ed esporrà delle API.
3. Servirà inoltre un componente chiamato Service, definito tramite la corrispondente annotazione @Service, che si occupa della logica operativa tra servizio e client. In particolare, i suoi ruoli principali sono: fornire i dati al controller che li esporrà ai client e salvare le informazioni inviate dalle richieste.

Grazie alla corretta comunicazione tra questi tre componenti un client potrà quindi effettuare diversi tipi di richieste HTTP al server, definite all'interno della classe controller, identificate dalle rispettive annotazioni:

- @GetMapping("/tasks"): per ottenere la lista di tutti i tasks;
- @GetMapping("/tasks/id"): risponde ad una GET con un identificativo nel payload, per ottenerne l'analogo task;
- @DeleteMapping("/tasks/id"): cancella dal database il task con quell'identificativo;
- @PutMapping("/tasks/id"): che consente la modifica del flag del task corrispondente all'id;

- @PutMapping("/description/id"): per cambiare la descrizione del task;
- @PostMapping("/task"): consente di creare un task, specificandone i campi all'interno del payload della richiesta.

Le prime vulnerabilità che sono state inserite in questo passaggio sono generate dall'uso della libreria H2 non aggiornata all'ultima versione, ma rimasta a quella 1.4.200 [72].
Questa libreria contiene due vulnerabilità critiche di tipo Remote Code Execution e una con grado alto di tipo XML External Entity [73]. Per l'uso di questa libreria rispetto alle altre è stato specificato il numero della versione nel campo "version" del file di gestione dei pacchetti Maven ("pom.xml"), come si può osservare dall'immagine successiva.

```
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
        <scope>compile</scope>
</dependency>
```

Figura 6.1: Libreria H2 nel file "pom.xml"

Inoltre, è stata inserita una vulnerabilità di tipo strutturale, generata da una cattiva configurazione delle classi e delle API in Java Spring Boot. In particolare, la presenza dell'autobinding tra i componenti e l'uso della classe "ToDo-Item" all'interno delle richieste permette ad attaccanti di leggere e modificare il contenuto del database [74].
Nell'immagine a pagina seguente, si può vedere questa configurazione errata in cui: il componente "Service" viene direttamente ritornato al client e la classe principale viene usata nella richiesta di creazione del task.

```java
@RestController
public class TodoListController {

	@Autowired
	private TodoService todoService;

	@GetMapping("/tasks")
	private List<TodoItem> getAllTasks(){
		return todoService.getAllTasks();
	}

	@GetMapping("/tasks/{id}")
	private TodoItem getTaskById(@PathVariable("id") Long id) {
		return todoService.getTaskByID(id);
	}

	@DeleteMapping("/tasks/{id}")
	private void deleteTaskById(@PathVariable("id") Long id) {
		todoService.deleteTask(id);
	}

	@PutMapping("/tasks/{id}")
	private void completeTask(@PathVariable("id") Long id) {
		TodoItem task = todoService.getTaskByID(id);
		if(task!=null) {
			task.setCompleted(true);
			todoService.createTask(task);
		}
	}

	@PutMapping("/description/{id}")
	private void modifyTask(@PathVariable("id") Long id, @RequestBody String description) {
		TodoItem task = todoService.getTaskByID(id);
		if(task!=null) {
			task.setDescription(description);
			todoService.createTask(task);
		}
	}

	@PostMapping("/task")
	private void createTask(@RequestBody TodoItem task) {
		 todoService.createTask(task);
	}

}
```

Figura 6.2: Codice sorgente del Controller

6.1.2 DockerFile

Dopo la programmazione della parte operativa si può passare alla definizione del file per la creazione dell'immagine Docker che, partendo da una base image e dai file Jar generati dalla build del codice sorgente, genera l'immagine finale "todoapp".

Per la scrittura di questo DockerFile si è deciso di usare come immagine di partenza "openjdk:17-alpine" [75] che soffre di alcune vulnerabilità tra cui quelle ad alta criticità elencate di seguito:

1. Buffer Overflow all'interno del modulo *openssl/libcrypto1.1* e *openssl/libssl1.1* [76] che permette di effettuare un attacco di buffer overflow con un payload composto da 62 byte necessari per riempire il buffer e i restanti contenenti il vettore d'attacco;
2. Out-of-bounds Read nel modulo *apk-tools* [77], che dà la possibilità di leggere byte al di fuori del limite massimo;
3. Out-of-bounds Write nel modulo *zlib*, che non era presente durante la prima consultazione, ma comparso poi in un'analisi successiva [78]. Anche in questo caso permette un attacco di buffer overflow.

Oltre ad aver volutamente inserito queste vulnerabilità, si è anche scelto di non rispettare la best practice che permette di limitare i privilegi di un utente. In questo modo, sarà possibile eseguire il container con i privilegi di root

Verranno presentate ora le istruzioni [79] di cui è composto il DockerFile. Esse possono essere varie e, solitamente, per comporre questo file si parte con l'istruzione FROM che permette di definire la base image, in seguito si definiscono i file Jar da usare per creare l'immagine tramite le istruzioni ARG e COPY. Il successivo comando esplicita su quale porta sarà esposto il servizio, invece l'ultimo elemento consente di avviare il container come eseguibile con il comando "docker run". Una possibile realizzazione del DockerFile basato sulle istruzioni riportate in precedenza, può essere:

```
FROM openjdk:17-alpine
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 3314
ENTRYPOINT ["java","-jar","/app.jar"]
```

A conclusione del processo di sviluppo del codice sorgente (sezione 6.1.1) e del DockerFile sono stati caricati tutti i file creati all'interno di un repository personale di GitHub.

6.1.3 Kubernetes e AWS

A questo punto si può proseguire con la costruzione dell'architettura che, nonostante sia composta da due argomenti (Kubernete e AWS), verrà descritta in un'unica sottosezione per motivi puramente discorsivi.

Per creare una vera e propria infrastruttura cloud bisogna affidarsi ad un cloud service provider, in questo caso AWS, che servirà ad ospitare il cluster creato grazie al tool Kops.
L'idea è stata quella di creare un cluster composto da due istanze AWS EC2 con risorse hardware di tipo t2.micro (composto da 1 vCPU da 1 GiB [80] di RAM) e t2.medium (con 2 vCPU e 4 GiB di RAM) che avrebbero svolto rispettivamente i ruoli di nodo master e di nodo worker.

Il primo passo da seguire è installare sia Kops che AWS CLI [81], un software open-source che permette di interagire con i servizi AWS tramite comandi eseguiti in un terminale. Grazie a questo strumento sarà quindi possibile collegare l'account AWS con il terminale tramite il comando:

```
aws configure
```

Ovviamente, per concludere correttamente la procedura c'è bisogno di inserire le credenziali dell'utenza AWS.

Per l'effettiva creazione del cluster Kubernetes, oltre alle istanze EC2, sono necessari due servizi in particolare:

- Amazon VPC (Virtual Private Cloud) [82] che permette di creare una rete virtuale simile a quelle tradizionali, definendone degli indirizzi IPv4 e IPv6 o delle sottoreti. Per ogni account è incluso un VPC di default già pronto per la connessione alle istanze EC2.
- Amazon S3 (Simple Storage Service) [83], anche detto bucket S3, è un servizio che permette l'archiviazione di oggetti all'interno di AWS. Questo componente è necessario per il funzionamento di Kops, perchè in questo database verranno salvate tutte le configurazioni del cluster.

Verrà quindi utilizzato il VPC di default per definire la rete, invece bisognerà creare un bucket S3 in quanto non offerto di default da AWS; per farlo basterà lanciare un comando dal terminale precedentemente configurato tramite l'AWS CLI:

```
aws s3 mb s3://"storage.dev.yourdomain.com"
```

Dopo questo comando, bisogna che Kops riesca ad accedere al servizio appena definito e per farlo si deve esportare la variabile d'ambiente "KOPS_STATE_STORE" tramite il comando *export*, assegnandole come valore il nome del bucket creato in precedenza.

In seguito alle operazioni preliminari, si può procedere con l'effettiva creazione del cluster tramite i comandi di Kops. In particolare, bisognerà definire: il tipo di CSP, la zona o le zone nelle quali si vuole schierare il cluster, il nome da assegnargli, la chiave pubblica corrispondente all'account AWS, il nome del bucket S3 e il numero dei master e dei worker con le rispettive specifiche hardware. Tutte queste informazioni sono contenute nel comando seguente:

```
kops create cluster --cloud=aws --zones=eu-west-3a \
 --name= [cluster name] --ssh-public-key ~/.ssh/id_rsa.pub \
--state s3://"storage.dev.yourdomain.com" \
--master-count 1 --master-size=t2.micro \
--node-count 1 --node-size=t2.medium
```

Infine, per rendere operativo il cluster bisognerà lanciare il comando *kops update cluster –name ..... –yes –admin*, che andrà ad avviare le due istanze EC2. Si potrà quindi, affermare che il cluster è stato creato correttamente.

Continuando nella configurazione del cluster, si è inoltre deciso di abilitare la dashboard di Kubernetes [84] per avere una migliore integrazione grafica con il cluster e, soprattutto, per inserire una misconfiguration nell'architettura.

La Kubernetes dashboard è molto semplice da avviare, infatti, basta solo installare il software e eseguire il comando *kube proxy*, ma per effettuare l'accesso è necessario inserire un token e, per farlo, serve creare un nuovo utente tramite il meccanismo del Service Account in Kubernetes.
Tramite il comando successivo, viene definito un Service account, ovvero un'utenza che verrà assegnata al pod *kubernets-dashboard*, che autorizzerà il pod stesso a contattare le API di Kubernetes.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

Successivamente si dovrà procedere con l'assegnazione dei privilegi all'utenza appena creata e, per fare ciò sarà utile definire il concetto di Role-Based Access Control (RBAC) [85].

Tale servizio consente di regolare gli accessi alle risorse in base ai ruoli dei singoli utenti ed introduce, inoltre, quattro tipi di entità: *Role, ClusterRole, RoleBinding e ClusterRoleBinding*.
I primi due descrivono delle regole che definiscono un insieme di permessi, il Role identifica permessi solo all'interno di un preciso namespace, mentre i previlegi impostati in un ClusterRole hanno valenza a prescindere dal namespace utilizzato; i secondi due permettono di assegnare i privilegi descritti nelle precedenti entità ad uno o più utenti e, anche in questo caso, un RoleBinding garantisce a quell'utente di utilizzare i permessi solo all'interno di un preciso namespace, mentre il ClusterRoleBinding all'interno di tutto il cluster.

Ai fini della trattazione, si è quindi deciso di non rispettare il principio del Least Privilege e di inserire una misconfiguration, che consiste nell'assegnare al Service Account un ClusterRole automaticamente definito durante la creazione del cluster tramite Kops. Tale ruolo, chiamato *cluster-admin*, ha i privilegi di "super-user" e verrà collegato tramite il seguente ClusterRoleBinding:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Si è deciso, infine, di produrre un file yaml per il deployment dell'applicativo che consente di descrivere il ciclo di vita dell'applicazione definendone, ad esempio, il numero di repliche, la versione dell'immagine da utilizzare o quale porta esporre. Il file mostrato di seguito verrà configurato in modo scorretto, non specificando dei limiti di utilizzo della CPU e della memoria.

```
latestapiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: todoapp
  name: todoapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: todoapp
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: todoapp
    spec:
      containers:
      - image: dockerID/todoapp:latest
        name: todoapp
        resources: {}
status: {}


---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: todoapp
  name: todoapp
spec:
  ports:
  - name: 80-80
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: todoapp
  type: ClusterIP
status:
  loadBalancer: {}
```

Con la costruzione di questo file termina la parte di descrizione dello sviluppo dell'applicativo e dell'inserimento delle vulnerabilità. Il cluster è attivo e l'applicazione è in esecuzione, si quindi può quindi procedere con l'avvio della pipeline e l'analisi infrastrutturale proposta nel capitolo 5.

6.2 Ricerca delle vulnerabilità

In questa sezione si ripercorreranno i passi già definiti nel capitolo 5 ma applicati sull'applicazione descritta in precedenza: si utilizzeranno tutti gli strumenti analizzati per ricercare i possibili problemi nel servizio.

L'organizzazione della trattazione sarà divisa in sottocapitoli e, in ognuno di essi, verrà preso in considerazione il report generato dal tool, per poi identificarne le diverse sorgenti di errore.

6.2.1 Report di Sonarqube

Dopo che il codice sorgente dell'applicazione è stato scaricato dal repository ed è stata effettuata la build, il primo tool di analisi che si incontra è Sonarqube.

Il risultato dell'analisi ha prodotto il seguente output con quattro vulnerabilità di livello "Major":

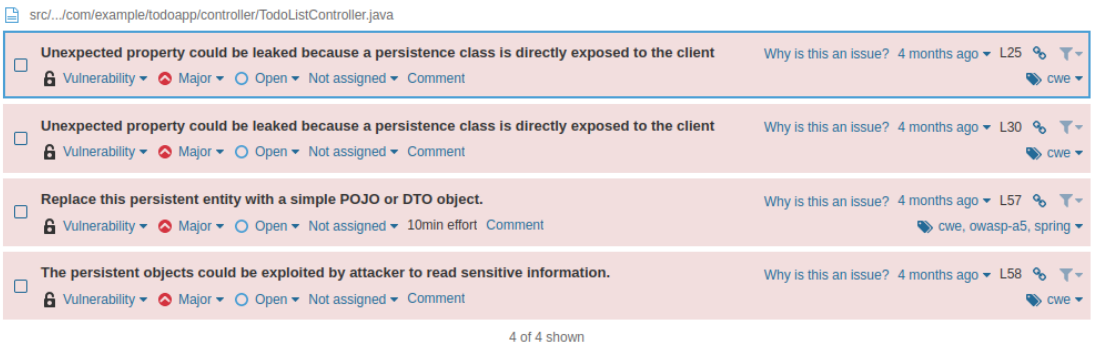


Figura 6.3: Sonarqube report

Questo output è stato generato da una da una vulnerabilità inserita durante lo sviluppo del codice sorgente che, tramite una non ottimale definizione della struttura delle API, permette ad un attaccante di eseguire del codice non autorizzato oppure modificare dati sensibili e variabili del programma, come definito nel corrispondente CWE [74].

Il terzo errore identifica ancora meglio il problema: è indicato di sostituire la persistent entity con un oggetto di tipo POJO o DTO, questo perchè all'atto di definizione delle API veniva utilizzato sia dalle GET che dalla POST direttamente l'oggetto "ToDo-Item" cosa che implica i problemi sopraccitati.

Le quattro vulnerabilità trovate da Sonarqube sono rintracciabili nel codice presente nell'immagine 6.2 alle righe 25, 30, 57 e 58 e si rifanno tutte allo stesso problema strutturale volutamente inserito durante lo sviluppo. Per correggere queste vulnerabilità, il primo passo è stato aggiungere una nuova classe "TaskDTO", avente gli stessi attributi della classe "ToDo-Item" e servirà per non esporre direttamente quest'ultima.
Oltre a questa aggiunta è stato modificato il Controller come segue:

```
@RestController
public class TodoListController {

        @Autowired
        private TodoService todoService;

        private TodoItem convertToEnt(TaskDTO taskDto) {
                TodoItem task= new TodoItem(taskDto.getId(),taskDto.getDescription(),taskDto.isCompleted());
                return task;
        }

        private TaskDTO convertToDTO(TodoItem task) {
                TaskDTO taskDto= new TaskDTO(task.getId(),task.getDescription(),task.isCompleted());
                return taskDto;
        }

        @GetMapping("/tasks")
        private List<TaskDTO> getAllTasks(){
                return todoService.getAllTasks().stream().map(x->convertToDTO(x)).collect(Collectors.toList());
        }

        @GetMapping("/tasks/{id}")
        private TaskDTO getTaskById(@PathVariable("id") Long id) {
                TodoItem task = todoService.getTaskByID(id);
                return convertToDTO(task);
        }

        @DeleteMapping("/tasks/{id}")
        private void deleteTaskById(@PathVariable("id") Long id) {
                todoService.deleteTask(id);
        }

        @PutMapping("/tasks/{id}")
        private void completeTask(@PathVariable("id") Long id) {
                TodoItem task = todoService.getTaskByID(id);
                if(task!=null) {
                        task.setCompleted(true);
                        todoService.createTask(task);
                }
        }

        @PutMapping("/description/{id}")
        private void modifyTask(@PathVariable("id") Long id, @RequestBody String description) {
                TodoItem task = todoService.getTaskByID(id);
                if(task!=null) {
                        task.setDescription(description);
                        todoService.createTask(task);
                }
        }

        @PostMapping("/task")
        private void createTask(@RequestBody TaskDTO taskDto) {
                TodoItem task = convertToEnt(taskDto);
                todoService.createTask(task);
                return;

        }

}
```

Figura 6.4: Nuovo Controller

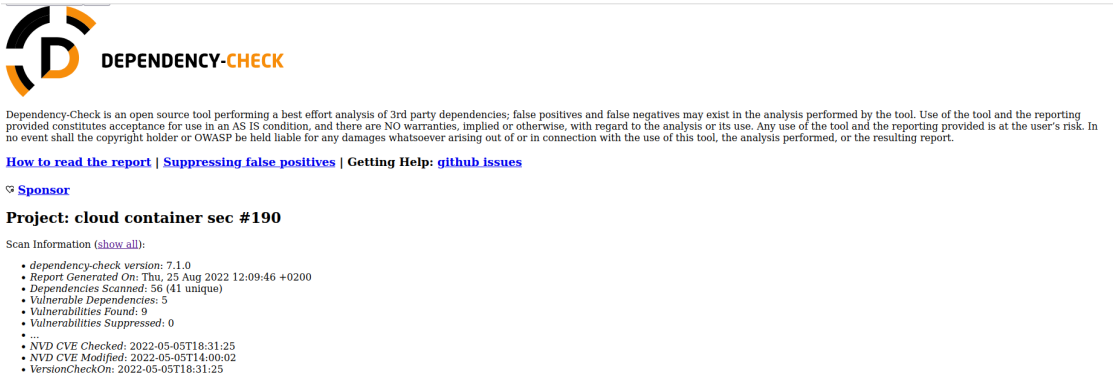
Si può notare la presenza di due nuovi metodi "convertToEnt" e "convertToDTO" che permettono la conversione da "TaksDTO" a "ToDo-Item" e viceversa. Inoltre, è cambiato il corpo delle richieste a rischio identificate dall'analisi di Sonarqube:

- @GetMapping("/tasks") usa ancora il Service per ottenere la lista di "ToDo-Item" nel database, ma poi la converte in una lista di "TaskDTO". In questo modo non vengono più esposti nè il Service, nè la classe principale;
- @GetMapping("/tasks/id"), come nel caso precedente, effettua la conversione prima di ritornare l'oggetto per ottenere lo stesso risultato;
- @PostMapping("/task") adesso riceve un "TaskDTO" nel payload e ne effettua la conversione a "ToDo-Item" prima di salvare nel database il nuovo task.

Grazie a queste modifiche è stato possibile correggere l'errore strutturale nel codice e, quindi, rieseguendo l'analisi di Sonarqube, queste vulnerabilità non compariranno più.

6.2.2 Report di OWASP Dependency-check

Si procede con l'analisi delle dipendenze tramite OWASP Dependency-check, il secondo tool della pipeline, che ha riportato i seguenti risultati generici: con cinque dipendenze vulnerabili e nove vulnerabilità trovate.


DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: cloud container sec #190

Scan Information (show all):

- *dependency-check version*: 7.1.0
- *Report Generated On*: Thu, 25 Aug 2022 12:09:46 +0200
- *Dependencies Scanned*: 56 (41 unique)
- *Vulnerable Dependencies*: 5
- *Vulnerabilities Found*: 9
- *Vulnerabilities Suppressed*: 0
- ...
- *NVD CVE Checked*: 2022-05-05T18:31:25
- *NVD CVE Modified*: 2022-05-05T14:00:02
- *VersionCheckOn*: 2022-05-05T18:31:25

Figura 6.5: Risultato dell'analisi di OWASP Dependency-check

Tra tutte le vulnerabilità si focalizzerà l'attenzione sulle tre contenute nella libreria del database H2 catalogate come "Critiche". Di seguito, è possibile vedere il risultato della scansione:

Figura 6.6: Vulnerabilità della libreria H2

OWASP Dependency-check è riuscito ad identificare tutte le vulnerabilità critiche che erano state volutamente inserite definendo una versione non aggiornata della libreria H2:

- una vulnerabilità di tipo XML External Entity [86];
- una vulnerabilità di tipo Remote Code Execution [87];
- una vulnerabilità ancora sottoposta ad analisi, di tipo Remote Code Execution [88].

La correzione di questi errori è direttamente legata all'aggiornamento della libreria H2, che può essere effettuato in due modi: il primo consiste nel modificare il comando Maven visto nello stage di build alla sezione 5.3.3, oppure al posto della versione specificata nella figura 6.1 si possono usare le keyword "RELEASE" e "LATEST", che indicano rispettivamente l'ultima release e l'ultima versione disponibili. Per quest'ultima soluzione è consigliabile l'uso della keyword RELEASE perchè identifica una versione più stabile, mentre per la LATEST possono esserci ancora dei problemi da risolvere.

6.2.3 Report di Trivy

Continuando con l'esecuzione della pipeline, il tool successivo è Trivy che è stato usato per analizzare sia il contenuto del DockerFile che l'immagine generata da esso. Per questioni discorsive i risultati delle due analisi verranno raccolti entrambi in questo sottocapitolo.

Per l'analisi del Dockerfile, Trivy ha riportato un errore di configurazione che introduce una problematica molto frequente nel panorama dello sviluppo a container. Infatti, come riportato da Sysdig nel suo annuale report [89], il 76% delle immagini vengono eseguite con i privilegi di root. L'analisi del DockerFile è stata fondamentale per riuscire ad identificare questa misconfiguration:

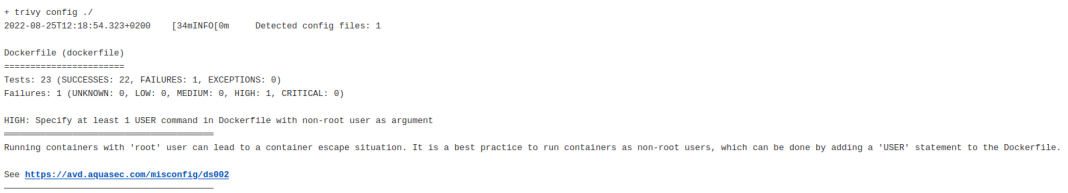


Figura 6.7: Risultato dell'analisi del DockerFile

Dopo l'analisi del file di configurazione, come già detto nel capitolo della pipeline, si procede con la creazione dell'immagine Docker che, come base image, utilizzava una versione vulnerabile di alpine basata sulla 3.14.0, ovvero *"openjdk:17-alpine"*, che come è possibile vedere nell'immagine successiva presenta al suo interno molte vulnerabilità:

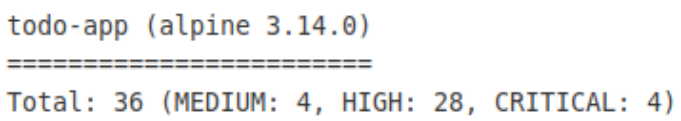


Figura 6.8: Risultato dell'analisi dell'immagine

L'attenzione si concentrerà, però, sulle quattro identificate come critiche, che come si può vedere dalla figura 6.9, corrispondono alle tre già evidenziate durante la creazione dell'applicativo.
Si noti come Trivy divide i risultati in base alle librerie vulnerabili quindi, come nel caso della vulnerabilità di *libcrypto e libssl*, nonostante il CVE sia lo stesso, vengono contate come due vulnerabilità distinte.

Per quanto riguarda le altre criticità con grado più basso, anch'esse volutamente inserite all'atto di sviluppo [75], sono state tutte rintracciate da Trivy, anche se i numeri delle occorrenze non coincidono per il motivo mostrato in precedenza.

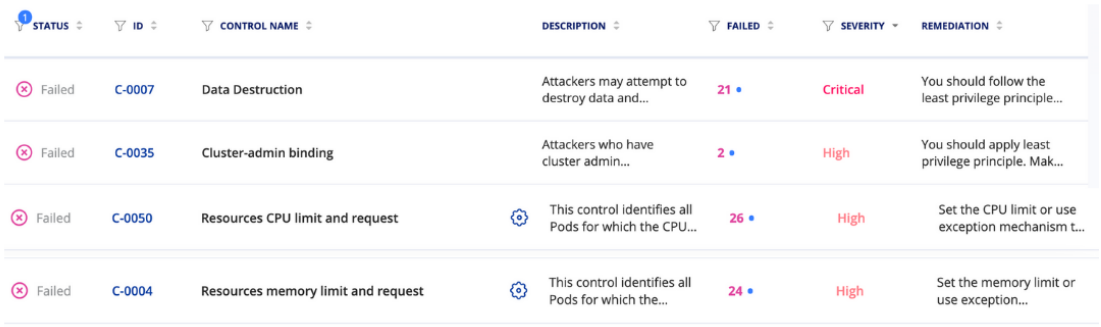
Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
apk-tools	CVE-2021-36159	CRITICAL	2.12.5-r1	2.12.6-r0	libfetch before 2021-07-26, as used in apk-tools, xbps, and other products, mishandles... https://avd.aquasec.com/nvd/cve-2021-36159
libcrypto1.1	CVE-2021-3711	CRITICAL	1.1.1k-r0	1.1.1l-r0	openssl: SM2 Decryption Buffer Overflow https://avd.aquasec.com/nvd/cve-2021-3711
libssl1.1	CVE-2021-3711	CRITICAL	1.1.1k-r0	1.1.1l-r0	openssl: SM2 Decryption Buffer Overflow https://avd.aquasec.com/nvd/cve-2021-3711
zlib	CVE-2022-37434	CRITICAL	1.2.11-r3	1.2.12-r2	zlib: a heap-based buffer over-read or buffer overflow in inflate in inflate.c... https://avd.aquasec.com/nvd/cve-2022-37434

Figura 6.9: Vulnerabilità critiche nell'immagine

Questo report conclude i resoconti delle analisi generate dall'esecuzione della pipeline.

6.2.4 Report di Kubescape

L'ultimo passo dell'analisi sull'applicativo consiste nel rintracciare le misconfigurations all'interno dell'infrastruttura stessa.
Grazie a Kubescape è stato possibile riconoscere, tra i tanti altri errori di configurazione, due coppie di segnalazioni: la prima generata dall'assegnazione dei massimi privilegi al ServiceAccount, mentre la seconda riporta che non è stata rispettata la best practice di impostare limiti di CPU e memoria.



STATUS	ID	CONTROL NAME	DESCRIPTION	FAILED	SEVERITY	REMEDIATION
Failed	C-0007	Data Destruction	Attackers may attempt to destroy data and...	21	Critical	You should follow the least privilege principle...
Failed	C-0035	Cluster-admin binding	Attackers who have cluster admin...	2	High	You should apply least privilege principle. Mak...
Failed	C-0050	Resources CPU limit and request	This control identifies all Pods for which the CPU...	26	High	Set the CPU limit or use exception mechanism t...
Failed	C-0004	Resources memory limit and request	This control identifies all Pods for which the...	24	High	Set the memory limit or use exception...

Figura 6.10: Risultato dell'analisi di Kubescape

6.3 Tabella riassuntiva delle analisi

Dopo aver effettuato le varie analisi sull'applicativo e sulla sua infrastruttura, è stato possibile stilare una tabella riassuntiva, dove saranno riportate le vulnerabilità, come è stato possibile rintracciarle e una possibile mitigation strategy.
La struttura della tabella presenterà i seguenti campi:

- **Ambiente**: che identifica il contesto in cui si sviluppa il problema;
- **Vulnerabilità**: che riporta le vulnerabilità volutamente inserite all'atto di sviluppo e che sono state rintracciate durante l'analisi;
- **Sezione**: abbreviata in Sez, indica la sezione dove è stata descritta la vulnerabilità;
- **Tool**: definisce il tool che ha permesso di rintracciare la vulnerabilità in questione;
- **Rischio**: che riporta, appunto, la classificazione del rischio;

AMBIENTE	VULNERABILITÀ	SEZ	TOOL	RISCHIO
Java Spring Boot	-Code injection -Sensitive Data Exposure	6.1.1	Sonarqube	critico
Java Spring Boot	Libreria H2 vulnerabile	6.1.1	OWASP Dependency-check	- alto - basso in caso di patches
Docker image	Immagine corrotta	6.1.2	Trivy (image)	critico
DockerFile	Container con privilegi root	6.1.2	Trivy (DockerFile)	alto
Kubernetes	-Misconfiguration degli IaC -Privilegi eccessivi	6.1.3	Kubescape	da basso a critico

Tabella 6.1: Tabella riassuntiva analisi

Tutte le analisi effettuate in precedenza sono utili a rintracciare vulnerabilità ed errori di configurazione, ma questi accorgimenti da soli potrebbero non essere sufficienti. Ad esempio, può essere scoperta una zero-day vulnerability sfruttabile, a cui non è possibile applicare una correzione, oppure può succedere che il servizio venga attaccato.
Pertanto, si è supposto che l'inserimento del response engine nel framework e dunque, di una soluzione che permetta di intercettare gli eventi verificabili in tempo reale, garantisse una migliore protezione del servizio.

Capitolo 7

La run-time security

Il capitolo presenterà una possibile applicazione della run-time security.
Nella prima parte verranno descritti due elementi fondamentali, Falco e Falco Sidekick, il primo applica interamente il concetto di run-time security mentre il secondo amplia le possibilità di Falco stesso.

Si continuerà definendo un componente pienamente integrabile nel framework: il Response Engine, di cui verranno illustrate l'architettura e il funzionamento.

A concludere il capitolo, verranno ripercorsi i passaggi di una simulazione di attacco su un servizio protetto dal response engine.

7.1 Falco e Falco Sidekick

Il concetto di run-time security raccoglie tutte le azioni messe in atto per rintracciare e rispondere alle minacce che possono abbattersi sui containers durante la loro esecuzione.
Questa novità permette quindi di aumentare i controlli di sicurezza all'interno del cloud, coprendo i casi in cui il servizio sta svolgendo correttamente il suo compito.
Tale possibilità non è di poco conto, infatti, la maggior parte dei tools che applicano il concetto della run-time security sono disponibili solo a pagamento, tra questi possiamo trovare: Sysdig Secure [90], Qualys [91] e Datadog [92].

Gli strumenti citati in precedenza, seppur offrano una suite completa di strumenti, però possono avere costi elevati.
In questo scenario si inserisce Falco [93]: l'unico software completamente open source, realizzato da Sysdig e poi donato al CNCF, che permette di fare run-time security.

L'uso di Falco sta riscuotendo un notevole successo tra gli sviluppatori tanto da essere attualmente considerato il "de facto Kubernetes threat detection engine", come appunto riporta il sito ufficiale.

Falco è in grado di ricevere ed elaborare degli eventi provenienti da diverse sorgenti: le systemcalls, i Kubernetes audit logs e i logs provenienti dal cloud. Se uno di questi eventi viola una delle regole di Falco, viene generato un alert che conterrà le informazioni necessarie per identificare la sorgente d'errore.

Le regole di Falco possono essere personalizzabili oppure di default, e consentono di rilevare un evento che verrà poi confrontato con una condizione, se la condizione è violata viene generato un allarme. Per questa trattazione è stato sufficiente il solo uso delle rules di default.

La limitazione principale di Falco è che gli output possono essere spediti ad una ristretta tipologia di destinatari composta da file, endpoint HTTP o webserver. Per ampliare il ventaglio delle possibilità è stato utilizzato Falco Sidekick [94]: un progetto, anche esso open source, in grado di fornire una lista più ricca di possibili destinatari e un'interfaccia web che raccoglie tutti gli alerts.

La configurazione di questi due elementi è molto semplice, possono infatti essere direttamente deployati in un apposito namespace tramite Helm: il gestore di pacchetti di Kubernetes; come dimostrano i comandi successivi:

```
$ helm install falco falcosecurity/falco --namespace falco -f
config.yaml

$ helm install falcosidekick falcosecurity/falcosidekick --
namespace falco -f sidekick.yaml
```

Ovviamente, per farlo è necessario aver installato Helm[95] ed aver scaricato la repo contenente gli Helm charts corrispondenti. Da notare, inoltre, il passaggio di un file yaml, che permette di definire tutte le personalizzazioni dei due tools.

In questa ricerca si è deciso di definire la seguente configurazione:

1. Falco inoltrerà i suoi output su un endpoint HTTP;
2. Falco Sidekick sarà in ascolto su quell'endpoint, riceverà gli eventi e sarà in grado di inoltrarli ad altre destinazioni.

Questi due strumenti saranno la parte fondamentale del response engine sviluppato e permetteranno di intercettare gli eventi e di distribuire gli alerts.

7.2 Il Response Engine

Un response engine è un meccanismo che può essere impegato all'interno di un cluster Kubernetes e che permette di attuare delle contromisure nel caso in cui vengano intercettati eventi particolari durante l'esecuzione.
Le contromisure che è possibile definire sono basate sul concetto di serverless [96], ovvero un nuovo paradigma che si sta sviluppando negli ultimi anni, che consente nuovamente di rendere ancora più modulari i microservizi, scindendoli in funzioni ognuna delle quali eseguita come un container a sé.

L'adozione di un response engine potrebbe portare molti vantaggi in uno scenario reale: ad esempio può essere usato per aumentare la sicurezza generale del cluster, oppure per notificare il team di sviluppo in caso di problemi, o per investigare sugli attacchi, etc.

Il meccanismo, definito in questa tesi, si pone di soddisfare i seguenti requisiti:

- essere in grado di identificare gli eventi che possono colpire i containers;
- offrire un servizio di notifica per avvertire il team di sviluppo riguardo il verificarsi degli eventi potenzialmente pericolosi all'interno del cluster;
- essere in grado di prendere una contromisura nel caso in cui l'alert abbia un alto grado di criticità.

La scelta di inserire un response engine con questi requisiti all'interno del framework è stata dettata dall'intenzione di voler aumentare il livello di sicurezza del servizio.

7.2.1 L'architettura

L'architettura di un response engine non è unicamente definita, ma è possibile impiegare diverse soluzioni e raggiungere comunque l'obiettivo. Per la parte di identificazione di eventi, sono spesso utilizzati Falco e Falco Sidekick, mentre per la parte di gestione dei servizi serverless la scelta può essere più varia.
I possibili tools utilizzabili in questa parte sono definiti dagli output che offre Falco Sidekick [94]:

- AWS Lambda;
- GCP Cloud Run;
- GCP Cloud Functions;

- Fission;
- KNative;
- Tekton;
- Kubeless;
- OpenFaaS.

Tra tutti questi, il primo è disponibile a pagamento, i successivi due oltre ad essere anche loro a pagamento non sono compatibili con AWS, dei restanti invece si è posta maggiore attenzione verso l'ultimo perchè sembrerebbe essere il prodotto più maturo tra gli open-source elencati in precedenza.

OpenFaas [97] è, appunto, uno strumento che permette la creazione e la gestione di servizi di tipo serverless, ovvero servizi che si basano sull'esecuzione di funzioni invocate da eventi asincroni.
Questo framework è molto facile da utilizzare e offre molte possibilità di personalizzare il response engine, in quanto permette di definire di servizi Faas (Function as a Service)[98]: funzioni deployate nell'infrastruttura all'interno di un container, che verranno invocate al verificarsi di un evento particolare nel cluster.
Un'altra caratteristica particolare di questo software è che, se il servizio Faas dovesse essere invocato più di una volta nello stesso istante, OpenFaas effettuerà lo scaling del pod che contiene la funzione, creandone le repliche necessarie ad eseguire tutti i compiti.

OpenFaas verrà impiegato per creare una Faas, scritta in linguaggio Go [99], capace di ispezionare l'alert inviatole da Falco Sidekick, per poi indentificare il pod sotto attacco e il suo namespace provvedendo, nel caso in cui l'evento sia critico, a sospendere il pod stesso.

In questa ricerca, oltre ai software descritti in precedenza è stato utilizzato anche Slack [100], un software di collaborazione aziendale che viene usato dai team per comunicare. Si è scelto di utilizzare proprio questo software per la messaggistica perchè già usato sia in ambito universitario, sia in ambito lavorativo per l'organizzazione dei compiti all'interno del team della società di cui sono co-fondatore: Staseera SRL [101].

Dopo aver introdotto i requisiti e i componenti che sono stati scelti, si può definire una possibile architettura del response engine; a questo scopo l'immagine seguente mostra la rappresentazione grafica di come verrà realizzato questo meccanismo di risposta:

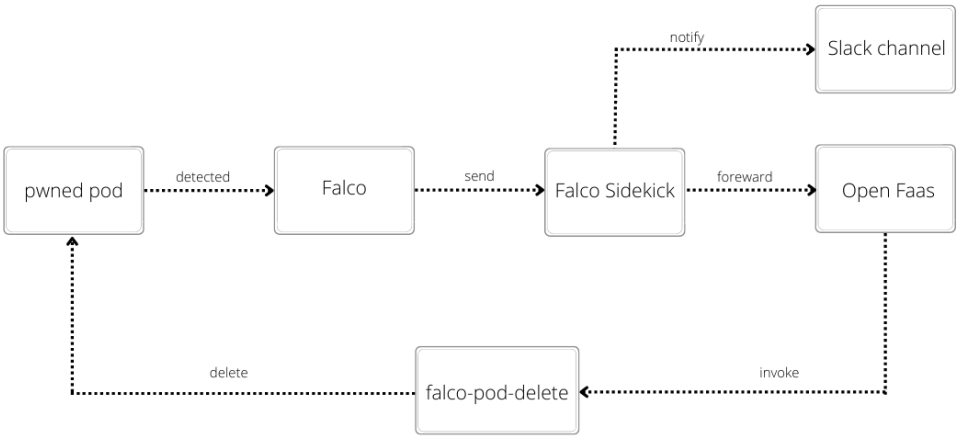


Figura 7.1: Rappresentazione dell'architettura

7.2.2 Funzionamento del response engine

Il funzionamento di questo meccanismo può essere descritto cosi:

1. si parte da una situazione potenzialmente pericolosa all'interno di un pod, come ad esempio una systemcall lanciata per scaricare dei dati;
2. a questo punto, Falco intercetta la systemcall e la valuta tramite le sue rules. Se anche solo una delle condizioni viene violata si genera un alert che verrà inviato verso l'endpoint HTTP;
3. Falco Sidekick riceve immediatamente l'alert e lo inoltra verso il canale Slack e verso OpenFaas;
4. OpenFaas, ricevuto l'alert, analizza il messaggio ricevuto e, se la criticità è superiore a "Warning", invoca il servizio FaaS definito nel pod "falco-pod-delete";
5. La funzione appena invocata legge dai campi del messaggio il nome del pod attaccato e il suo namespace di appartenenza per poi sospendere il pod.

Il tutto avviene in un lasso di tempo brevissimo, nell'ordine dei millisecondi, permettendo così di bloccare la minaccia ed evitare una possibile fuga di informazioni.

7.3 Costruzione del Response Engine

Si può procedere ora, con il ripercorrere i passi principali che sono stati seguiti per la creazione di questo componente.

Si segnala che nella prima versione del response engine, era stato pensato di utilizzare un cluster Kubernetes di prova come Minikube [102], Kind [103] o K3s [104], eseguito in una sola istanza AWS EC2.
Si è visto che questi tre software offrono l'uso di un cluster utilizzando Docker come driver, e quindi, creano un container nel quale simulano il cluster Kubernetes. Utilizzando questi software si è visto che gli alerts di Falco riportavano, come pod sorgente dell'errore, il nome del container a livello più alto: "minikube" se veniva usato un cluster Minikube, "k3s" o "kind" se si utilizzavano gli altri due.
Per risolvere questo problema le possibili scelte erano tre:

- Passare alla creazione di una VM, con la possibilità di utilizzare il driver di Virtualbox per effettuare una nested virtualization. Così facendo però si sarebbero esclusi dalla trattazione eventuali alerts generati da AWS;
- Utilizzare come AMI della EC2 un particolare tipo di istanza chiamata "Bare Metal" [105] capace di eseguire le virtualizzazioni annidate. Però i prezzi per questo tipo di istanza sono molto alti;
- Creare un cluster Kubernetes, nel quale definire il response engine. L'ultima possibilità è stata identificata come la soluzione migliore.

Si partirà, dunque, utilizzando l'ambiente definito per l'applicazione di test e si procederà con l'installazione e la configurazione dei software scelti.

Il primo elemento da configurare è OpenFaas. Per installarlo è necessario l'uso di Arkade [106], software simile ad Helm per la gestione di servizi aggiuntivi per Kubernetes. Dopo aver lanciato il comando:

```
arkade install openfaas
```

Sarà creato un namespace chiamato "openfaas" nel quale saranno presenti i pods che si occuperanno della gestione delle funzioni FaaS [107], in particolare:

- un OpenFaas Gateway che si servirà delle API per il controllo delle funzioni, delle metriche e dello scaling;
- un pod NATS [108] per la gestione asincrona delle funzioni;
- Prometheus [109] che provvederà alle metriche e all'auto-scaling delle FaaS.

In seguito si può passare all'istallazione di Falco e Falco Sidekick tramite i comandi Helm visti nella sezione 7.1, passando come argomento i file yaml di configurazione.
Per questi file le possibilità di personalizzazione sono varie e sono basate su una lunga serie di campi modificabili, per non appesantire la trattazione verranno quindi citati i file di partenza e verranno mostrate solo le modifiche effettuate per rendere funzionante il meccanismo di risposta descritto nella trattazione.

Falco

Partendo dal file iniziale *values.yaml* [110], presente nel repository ufficiale, sono stati modificati i seguenti elementi:

- Si è specificata la versione più recente dell'immagine, che alla definizione del componente era la 0.32.0:

```
image:
    registry: docker.io
    repository: falcosecurity/falco
    tag: 0.32.0
    pullPolicy: IfNotPresent
    pullSecrets: []
```

- Si è impostato l'endpoint HTTP per inviare gli alerts di Falco a Falco Sidekick, portando a true il flag "httpOutput" e fornendo l'URL di destinazione:

```
httpOutput:
    enabled: true
    url: "http://falcosidekick:2801"
    userAgent: "falcosecurity/falco"
```

Falco Sidekick

Per quanto riguarda la configurazione di questo plug-in si è partiti dal file *values.yaml* [111], impostando la comunicazione con i vari elementi del response engine:

- Dopo aver creato uno spazio di lavoro Slack e un canale dove si andranno ad inoltrare gli alert, si può procedere con l'attivazione del webhook, inserendo nel file il corrispondente URL. Inoltre, è stato deciso che fosse conveniente limitare il numero dei messaggi inviati sul canale alle sole situazioni più critiche, perciò è stato modificato il campo "minimumpriority" per considerare solo gli alerts con criticità minima di tipo "Warning".

```
slack:
    webhookurl: "https://hooks.slack.com/services/[channelID
]"
    footer: ""
    icon: ""
    username: ""
    outputformat: "all"
    minimumpriority: "warning"
    messageformat: ""
```

- La successiva modifica è necessaria per identificare la funzione FaaS che deve essere invocata al verificarsi di un evento, in questo caso è stato specificato il nome del servizio *"falco-pod-delete* che verrà definito successivamente:

```
openfaas:
    functionname: "falco-pod-delete"
    functionnamespace: "openfaas-fn"
    gatewayservice: "gateway"
    gatewayport: 8080
    gatewaynamespace: "openfaas"
    minimumpriority: ""
    mutualtls: false
    checkcert: true
```

- Per una migliore visualizzazione degli alerts, si è deciso di attivare la user interface di Falco Sidekick [112], che verrà deployata come pod e sarà accessibile alla porta 2802. Sarà molto utile perchè offrirà una dashboard che permetterà di accedere a tutti i dettagli degli eventi:

```
webui:
    enabled: true
```

A questo punto la parte che si occupa della run-time security sarà attiva e capace di intercettare ed inoltrare gli output sia sul canale Slack che ad OpenFaas, si può quindi procedere con la creazione del pod contenente la funzione di cancellazione pod.
OpenFaas sarà in grado di fornire un primo template della funzione tramite il comando [113]:

```
faas-cli new falco-pod-delete --prefix [DockerID] --lang golang-
middleware
```

La funzione creata leggerà gli alerts messi a disposizione da OpenFaas tramite JSON e ne farà l'unmashalling, ovvero ne tradurrà la sintassi, definendo una struttura chiamata Alert composta da tutti i campi necessari per l'identificazione del problema e della sorgente. Ad esempio, questa struttura identificherà: la Falco rule violata, un timestamp, la priorità e tutti i vari campi che definiscono l'origine, come il nome del pod, il namespace, il processo che ha generato l'errore, etc.
Dopo aver effettuato la traduzione, controllerà se la priorità dell'evento è superiore a "Warning", e se così è, invocherà il client Kubernetes per sospendere il pod identificato dal nome e dal namespace.

Per consentire al pod di effettuare tale operazione, è necessario ricorrere nuovamente al servizio RBAC per definire un ServiceAccount, per il servizio *falco-pod-delete*, un ClusterRole con i permessi di "get", "list" e "delete" solo sulle risorse di tipo "pod" ed infine realizzare un ClusterRoleBinding per assegnare i privilegi al pod.

Si può procedere con la definizione del file yaml per OpenFaas [114] composto da un campo provider dove verrà indicato l'URL del gateway e tutte le informazioni per la funzione da creare, come ad esempio, il linguaggio da utilizzare, l'handler della funzione e l'immagine da utilizzare.

Il file "falco-pod-delete.yaml" sarà così composto:

```
provider:
  name: openfaas
  gateway: http://[gatewayIP]:8080
functions:
  falco-pod-delete:
    lang: go
    handler: ./handler.go
    image: [DockerID]/falco-pod-delete:latest
    annotations:
      com.openfaas.serviceaccount: falco-pod-delete
    build_args:
      GO111MODULE: on
```

Infine, per l'effettivo build, push sul repository Docker e deploy nel cluster, si userà OpenFaas lanciando il seguente comando nella directory contenete il file yaml:

```
faas-cli up
```

A questo punto la parte di costruzione del response engine può considerarsi completata e può essere effettuato un test per verificare che l'infrastruttura sia capace di attuare le contromisure desiderate.

7.4 Simulazione di un attacco

Successivamente alla definizione e alla realizzazione del componente, si è voluto simulare un attacco contro il servizio, per verificare che la sicurezza generale è migliorata.
L'attacco è stato progettato sulla base di un pattern utilizzato dai teams che sfruttano il cloud per lanciare malware per il crypto-mining.

Come già in parte delineato nella sottosezione 3.2.2, questo tipo di attacco inizia tramite la ricerca di una misconfiguration o una vulnerabilità e, una volta trovata, si procede con sfruttarla per ottenere il controllo su un pod.
Dopo che la prima parte dell'operazione ha avuto successo si tenta di lanciare dei comandi che permetteranno di scaricare dalla rete tutti i file necessari per avviare il malware.

Sulla base di questo comportamento, si simulerà l'attacco a cui verrà aggiunta una semplificazione: si supporrà che l'attaccante sia già riuscito a rintracciare e sfruttare una vulnerabilità e che voglia creare un nuovo container privilegiato dove scaricare i file del malware.
Si è fatto uso di questa semplificazione perchè, per effettuare questo tipo di attacco è necessario avere una maggiore esperienza che per motivi di tempo, non è stato possibile raggiungere.

La simulazione, quindi, inizierà con l'utilizzo del comando kubectl, il command line tool di Kubernetes che permette di eseguire dei comandi all'interno del cluster. Com'è possibile notare dall'immagine successiva, il comando permetterà di creare un nuovo pod con privilegi di root, a partire da un'immagine nginx e all'interno del default namespace.

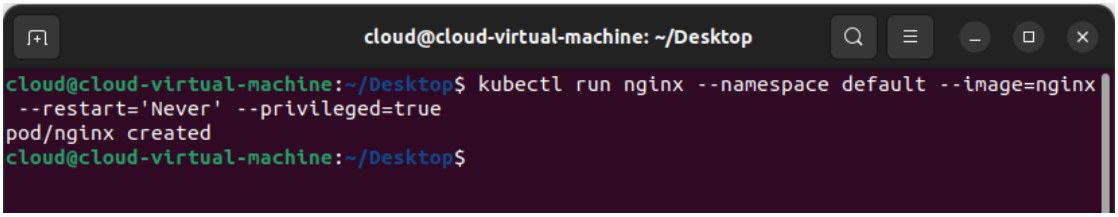


Figura 7.2: Creazione del pod privilegiato

A conferma del fatto che il risultato del comando ha avuto effetto, nella Kubernetes Dashboard compare il nuovo pod come si evince dalla figura 7.3.
Dalla figura 7.4 si può anche notare che Falco ha comunque intercettato l'evento e che ha generato un alert di bassa priorità salvato poi da Falco Sidekick all'interno del database.

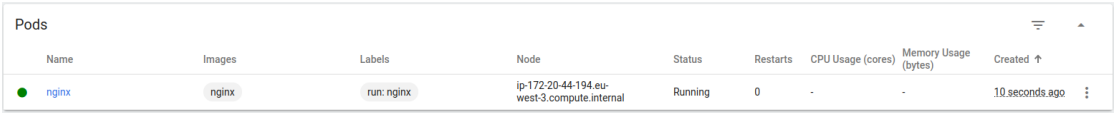


Figura 7.3: Nuovo pod nella Kubernetes Dashboard

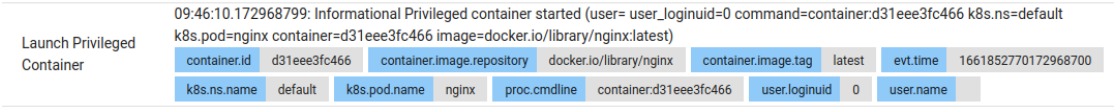


Figura 7.4: Alert generato dal nuovo pod

Lo step successivo sarà quello di aprire un terminale all'interno del pod nginx per scaricare i file del malware utilizzando nuovamente il comando kubectl:

```
$ kubectl exec —stdin —tty nginx — /bin/bash
```

L'esecuzione di questo comando permetterà effettivamente l'apertura del terminale, ma come è possibile vedere nella figura 7.5, esso viene immediatamente chiuso con l'avviso "command terminated with exit code 137".

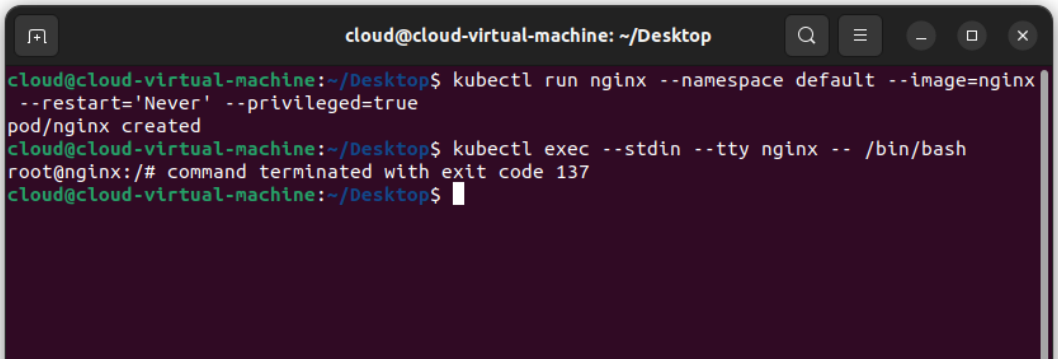


Figura 7.5: Apertura terminale bash

Il messaggio generato, comparso subito dopo l'apertura del terminale, ha bisogno di essere analizzato più attentamente, in quanto Falco ha degli alert non trascurabili.

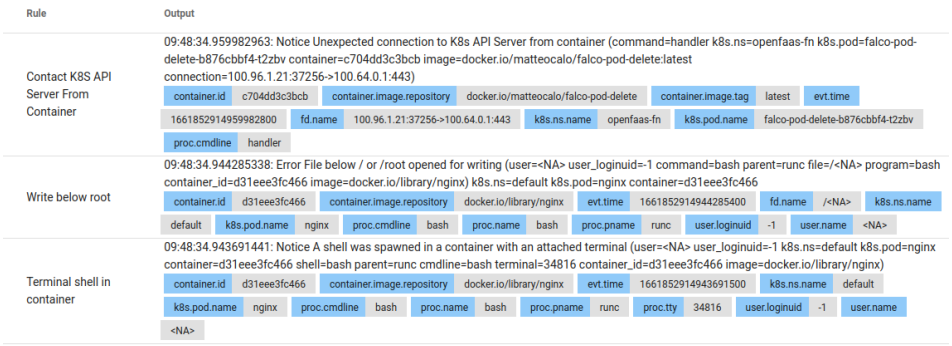


Figura 7.6: Alert collezionati da Falco Sidekick

Come mostra l'immagine precedente gli alerts identificati da Falco sono:

- Terminal shell in container: Falco ha identificato, attraverso questo alert, che è stato aperto un terminale bash nel container nginx. La priorità di questo output è di tipo "Notice", quindi ha una severità di livello basso.
- Write below root: Falco ha riconosciuto che un processo ha aperto per la scrittura un file al di sotto della directory "/" o "/root". Questo tipo di evento ha criticità alta, ovvero di tipo "Error", per questo motivo l'alert è stato anche propagato verso il canale Slack.

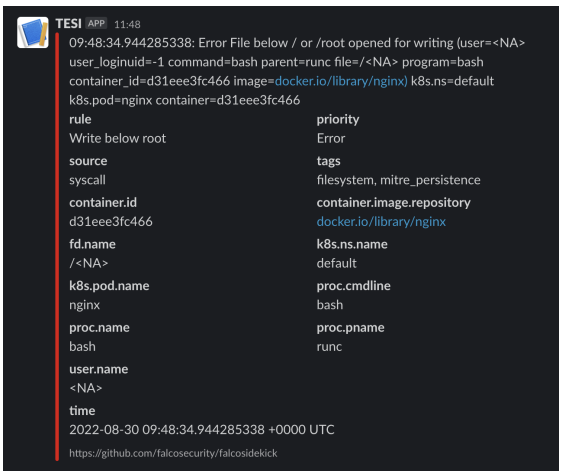


Figura 7.7: Notifica sul canale Slack

- Contact K8S API Server From Container: indica che l'alert precedente è stato ricevuto correttamente da OpenFaas, che ha valutato il rischio come alto e di conseguenza ha deciso di invocare la funzione *"falco-pod-delete"*.

Osservando la Kubernetes Dashboard, si può vedere infine che il pod nginx, che era stato creato appositamente per iniziare l'attacco, sia stato effettivamente terminato grazie alla contromisura applicata dal response engine.

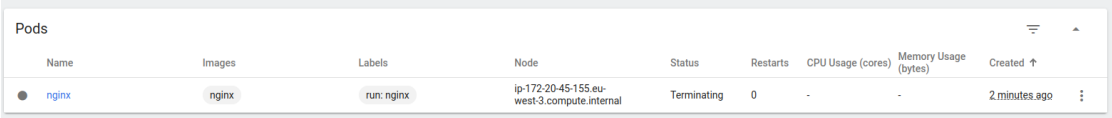


Figura 7.8: Pod in terminazione

Dopo aver studiato ed analizzato i dati e le informazioni raccolte, si procederà alla stesura delle conclusioni del presente lavoro di ricerca.

Capitolo 8

Conclusioni

Lo scopo primario di questo lavoro di ricerca è stato quello di creare un framework per lo sviluppo sicuro di un applicativo containerizzato, in modo da definire i migliori strumenti e comportamenti che un'azienda dovrebbe seguire.

Durante la parte di pianificazione, si è da subito deciso di mantenere la trattazione il più generica possibile, per consentirne l'applicazione a qualsiasi caso d'uso.
In particolare, si è riuscito ad ottenere un valido insieme di strumenti mantenendo i costi molto bassi.
Questo perchè si puntava a coinvolgere soprattutto le aziende che non vogliono investire capitale per tools commerciali, rendendole comunque, capaci di offrire un buon livello di sicurezza del servizio.

Il processo di definizione del framework ha coinvolto due componenti principali: il primo basato su una pipeline di sicurezza Jenkins, mentre il secondo sulla creazione di un meccanismo per la difesa in tempo reale.
Ognuno di questi due elementi è stato messo alla prova con uno use case che rispecchiasse il più possibile la realtà.

La creazione di un'applicazione e di un'infrastruttura cloud con delle vulnerabilità volutamente inserite è stata molto utile per testare l'efficacia della pipeline di sicurezza che ha riportato i risultati attesi. Grazie all'esecuzione della prima parte del framework sull'applicativo, è stato possibile rintracciare la totalità delle vulnerabilità e degli errori di configurazione introdotti durante lo sviluppo.

Per quanto riguarda invece la seconda parte del framework, il response engine era stato pensato come meccanismo per incrementare la sicurezza generale di un servizio a container, offrendo la capacità di prendere delle contromisure in caso di eventi inaspettati nel cluster.

Dopo aver sottoposto l'intera infrastruttura ad un caso reale di un attacco, il meccanismo di difesa è stato in grado di rispondere correttamente alla situazione, bloccandone sul nascere ogni possibilità di riuscita.
Perciò, è possibile affermare che l'aggiunta di questo componente all'interno dell'ambiente cloud ha effettivamente contribuito ad un aumento di affidabilità.

In conclusione, si può confermare che sia la prima, che la seconda parte del framework hanno raggiunto i risultati attesi e, di conseguenza, è possibile considerare l'intero framework come valido.

8.1 Sviluppi futuri del response engine

Seppur il response engine sia riuscito a rispettare le aspettative preposte, si è deciso di fornire uno spunto per uno sviluppo futuro che sia in grado di ampliarne ulteriormente le possibilità di personalizzazione.

Per ragioni di tempo non è stato possibile continuare lo studio di queste possibilità che, essendo così varie, possono addirittura fornire un punto di partenza per un successivo lavoro di ricerca.
Si è deciso però, di pensare ad una possibile definizione di un ulteriore servizio FaaS che, in base agli alerts ricevuti, sia in grado di differenziare quale contromisura adottare.

A questo punto, l'architettura definita nella figura 7.1 subirà una leggera modifica, perchè, al posto della funzione di cancellazione del pod sarà presente una funzione che chiameremo *"invoker"*, che riceverà ed analizzerà l'alert proveniente da Falco Sidekick, per poi invocare la funzione Faas che meglio si adatta a quella situazione.

Per questa versione del meccanismo è necessario, però, partire da uno studio che abbia già rintracciato tutte le più frequenti incidenze d'attacco che si sono verificate su un servizio.
Da queste incidenze, si possono quindi costruire varie funzioni Faas capaci di applicare delle contromisure per bloccare o mitigare quel particolare attacco.

La funzione *invoker*, in base alla regola violata, effettuerà infine una POST HTTP all'url **"http://gateway-openfaas:8080/function/[funzione da invocare]"** inoltrando nel payload l'alert stesso, in modo che la FaaS chiamata abbia tutti i dettagli necessari per eseguire il suo compito.

Bibliografia

[1] RedHat. *Report sullo stato della sicurezza di Kubernetes*. 2022 (cit. a p. 3).

[2] *OWASP Top 10 - 2021*. URL: https://owasp.org/Top10/ (visitato il 13/04/2022) (cit. a p. 4).

[3] Wikipedia. *Metodologia agile*. URL: http://it.wikipedia.org/w/index.php?title=Metodologia_agile&oldid=127666865 (visitato il 18/07/2022) (cit. a p. 7).

[4] *Che cos'è DevOps?* URL: https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-devops/#devops-overview (visitato il 22/03/2022) (cit. a p. 7).

[5] RedHat. *Cosa si intende con CI/CD?* URL: https://www.redhat.com/it/topics/devops/what-is-ci-cd#panoramica (visitato il 22/03/2022) (cit. a p. 8).

[6] RedHat. *Cosa significa DevSecOps?* URL: https://www.redhat.com/it/topics/devops/what-is-devsecops (visitato il 22/03/2022) (cit. a p. 9).

[7] Angela Salgarelli, Kiratech. *DevSecOps cos'è e quali sono i relativi vantaggi*. URL: https://www.kiratech.it/blog/devsecops-cos-%C3%A8-e-quali-sono-i-relativi-vantaggi (visitato il 22/06/2022) (cit. a p. 9).

[8] Wikipedia. *Internet security awareness*. URL: https://en.wikipedia.org/wiki/Internet_security_awareness (visitato il 22/06/2022) (cit. a p. 10).

[9] Lorenzo Dascola e Claudio Opizzi, Partners4Innovation. *SecDevOps: perché è da preferire a DevSecOps e come introdurlo in azienda*. URL: https://www.cybersecurity360.it/soluzioni-aziendali/secdevops-perche-e-da-preferire-a-devsecops-e-come-introdurlo-in-azienda/ (visitato il 24/06/2022) (cit. a p. 10).

[10] Amazon Web Service. *Cos'è AWS*. URL: https://aws.amazon.com/it/what-is-aws/ (visitato il 06/04/2022) (cit. a p. 11).

[11] *Amazon EC2*. URL: https://aws.amazon.com/it/ec2/?trk=a8d72064-a55e-4a1a-b151-dc58801045e9&sc_channel=ps&s_kwcid=AL!4422!3!588221158288!e!!g!!amazon%20ec2&ef_id=CjwKCAjw4JWZBhApEiwAtJUNOH_Nge9m9s70GYWnLTKiyieUDoPeH_yUUYaTHsnGWEKyMXYaVfOd6hoCqlkQAvD_BwE:G:s&s_kwcid=AL!4422!3!588221158288!e!!g!!amazon%20ec2 (visitato il 09/04/2022) (cit. a p. 12).

[12] Amazon Web Service. *Istanze e AMI*. URL: https://docs.aws.amazon.com/it_it/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html (visitato il 04/04/2022) (cit. a p. 12).

[13] Amazon Web Service. *Tipi di istanze di Amazon EC2*. URL: https://aws.amazon.com/it/ec2/instance-types/ (visitato il 07/04/2022) (cit. a p. 12).

[14] *IAM: consente l'accesso in sola lettura alla console IAM*. URL: https://docs.aws.amazon.com/it_it/IAM/latest/UserGuide/reference_policies_examples_iam_read-only-console.html (visitato il 12/03/2022) (cit. a p. 13).

[15] Fugue. *The state of cloud security*. 2022, p. 5 (cit. a p. 14).

[16] *Access to Glassdoor's Infra (AWS) and BitBucket account through leaked repo*. URL: https://hackerone.com/reports/801531 (visitato il 16/05/2022) (cit. a p. 14).

[17] *Mobile Apps Exposing AWS Keys Affect 100M+ Users' Data*. URL: https://bevigil.com/blog/mobile-apps-exposing-aws-keys-affect-100m-users-data/ (visitato il 08/04/2022) (cit. a p. 14).

[18] *Mobile Apps Exposing AWS Keys Affect 100M+ Users' Data*. URL: https://bevigil.com/blog/mobile-apps-exposing-aws-keys-affect-100m-users-data/ (visitato il 08/04/2022) (cit. a p. 15).

[19] David Fiser, Alfredo Oliveira. *TeamTNT Continues Attack on the Cloud, Targets AWS Credentials*. URL: https://www.trendmicro.com/en_gb/research/21/c/teamtnt-continues-attack-on-the-cloud--targets-aws-credentials.html (visitato il 09/04/2022) (cit. a p. 15).

[20] *Kubernetes Project Journey Report*. URL: https://www.cncf.io/reports/kubernetes-project-journey-report/ (visitato il 16/04/2022) (cit. a p. 16).

[21] Kops. URL: https://kops.sigs.k8s.io/ (visitato il 15/04/2022) (cit. a p. 16).

[22] Taylor Smith. *5 common Kubernetes misconfigs and how to fix them*. URL: https://bridgecrew.io/blog/5-common-kubernetes-misconfigs-and-how-to-fix-them/ (visitato il 10/04/2022) (cit. a p. 16).

[23] *Resource Management for Pods and Containers*. URL: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ (visitato il 22/04/2022) (cit. a p. 17).

[24] Eviatar Gerzi. *Using Kubelet Client to Attack the Kubernetes Cluster*. URL: https://www.cyberark.com/resources/threat-research-blog/using-kubelet-client-to-attack-the-kubernetes-cluster (visitato il 18/04/2022) (cit. a p. 17).

[25] *Cos'è il cryptojacking? - Definizione e spiegazione*. URL: https://www.kaspersky.it/resource-center/definitions/what-is-cryptojacking (visitato il 16/04/2022) (cit. a p. 17).

[26] Jay Chen, Aviv Sasson and Ariel Zelivansky. *Hildegard: New TeamTNT Cryptojacking Malware Targeting Kubernetes*. URL: https://unit42.paloaltonetworks.com/hildegard-malware-teamtnt/ (visitato il 16/04/2022) (cit. a p. 17).

[27] *Cos'è Docker*. URL: https://www.ibm.com/it-it/cloud/learn/docker (visitato il 02/05/2022) (cit. a p. 19).

[28] Srinivas for InfoSEC. *Common container misconfigurations and how to prevent them*. URL: https://resources.infosecinstitute.com/topic/common-container-misconfigurations-and-how-to-prevent-them/ (visitato il 29/04/2022) (cit. a p. 19).

[29] Sebastian Walla. *Compromised Docker Honeypots Used for Pro-Ukrainian DoS Attack*. URL: https://www.crowdstrike.com/blog/compromised-docker-honeypots-used-for-pro-ukrainian-dos-attack/ (visitato il 09/05/2022) (cit. a p. 20).

[30] Team Nautilus from AquaSecurity. *Attacks in the Wild on the Container Supply Chain and Infrastructure*. 2021, p. 17 (cit. a p. 20).

[31] Augusto Remillano II. *Malicious Docker Hub Container Images Used for Cryptocurrency Mining*. URL: https://www.trendmicro.com/vinfo/it/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining (visitato il 04/05/2022) (cit. a p. 20).

[32] *Cos'è Java Spring Boot*. URL: https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-java-spring-boot/ (visitato il 02/07/2022) (cit. a p. 21).

[33] *CVE List*. URL: https://cve.mitre.org/cve/index.html (visitato il 18/06/2022) (cit. a p. 21).

[34] Eyal Katz. *Top 10 Most Common Java Vulnerabilities You Need to Prevent*. URL: https://spectralops.io/blog/top-10-most-common-java-vulnerabilities-you-need-to-prevent/ (visitato il 16/09/2022) (cit. a p. 22).

[35] Simon Maple. *Java Top 10 Security Vulnerabilities Disclosed [2019 – List]*. URL: https://snyk.io/blog/162-security-vulnerabilities-disclosed-in-javas-top-10-libraries-including-jackson-spring-and-jetty/ (visitato il 18/09/2022) (cit. a p. 23).

[36] Story Tweedie-Yates. *The Nightmare Before Christmas: Looking Back at Log4j Vulnerabilities*. URL: https://blog.aquasec.com/log4j-vulnerabilities-overview?_ga=2.133279306.594166770.1647939703-1938768683.1647335394 (visitato il 08/04/2022) (cit. a p. 23).

[37] *CVE-2021-44228 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-44228 (visitato il 15/07/2022) (cit. a p. 23).

[38] *Injection Attacks Explained*. URL: https://www.lifars.com/2020/04/injection-attacks-explained/ (visitato il 15/07/2022) (cit. a p. 24).

[39] Ars Technica Dan Goodin. *Gab's CTO Introduced a Critical Vulnerability to the Site*. URL: https://www.wired.com/story/gab-cto-critical-vulnerability/ (visitato il 27/09/2022) (cit. a p. 24).

[40] Ian Muscat. *What Are XML External Entity (XXE) Attacks*. URL: https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/ (visitato il 15/09/2022) (cit. a p. 24).

[41] *Recipe*. URL: https://plugins.jenkins.io/recipe/ (visitato il 19/09/2022) (cit. a p. 24).

[42] *CVE-2022-34793*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-34793 (visitato il 19/09/2022) (cit. a p. 24).

[43] *What is a denial of service attack (DoS) ?* URL: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos#:~:text=A%20Denial%2Dof%2DService%20(,information%20that%20triggers%20a%20crash. (visitato il 19/09/2022) (cit. a p. 25).

[44] *Directory traversal*. URL: https://portswigger.net/web-security/file-path-traversal (visitato il 19/09/2022) (cit. a p. 25).

[45] *CVE-2020-5410*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-5410 (visitato il 19/09/2022) (cit. a p. 25).

[46] *Buffer Overflow Attack*. URL: https://www.imperva.com/learn/application-security/buffer-overflow/ (visitato il 15/09/2022) (cit. a p. 25).

[47] *What is Shift Left Security?* URL: https://www.checkpoint.com/cyber-hub/cloud-security/what-is-shift-left-security (visitato il 14/05/2022) (cit. a p. 26).

[48] *Security by design: una procedura vantaggiosa per le aziende.* URL: https://www.it-impresa.it/blog/security-by-design/ (visitato il 15/05/2022) (cit. a p. 27).

[49] *Best practice per la sicurezza in IAM.* URL: https://docs.aws.amazon.com/it_it/IAM/latest/UserGuide/best-practices.html (visitato il 25/07/2022) (cit. a p. 27).

[50] Taylor Smith. *5 common Kubernetes misconfigs and how to fix them.* URL: https://bridgecrew.io/blog/5-common-kubernetes-misconfigs-and-how-to-fix-them/ (visitato il 25/05/2022) (cit. a p. 29).

[51] *Best practices for writing Dockerfiles.* URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (visitato il 25/05/2022) (cit. a p. 31).

[52] Giuseppe Tacconi. *Security assessment: cos'è e come farlo per valutare il grado di protezione del patrimonio aziendale.* URL: https://www.cybersecurity360.it/soluzioni-aziendali/security-assessment-cose-e-come-farlo-per-valutare-il-grado-di-protezione-del-patrimonio-aziendale/ (visitato il 26/07/2022) (cit. a p. 32).

[53] *Jenkins.* URL: https://www.jenkins.io/ (visitato il 22/03/2022) (cit. a p. 33).

[54] *Groovy.* URL: https://it.wikipedia.org/wiki/Groovy (visitato il 22/03/2022) (cit. a p. 33).

[55] Cameron McKenzie. *Declarative vs. scripted pipelines: What's the difference?* URL: https://www.theserverside.com/answer/Declarative-vs-scripted-pipelines-Whats-the-difference (visitato il 22/03/2022) (cit. a p. 33).

[56] *SonarQube Documentation.* URL: https://docs.sonarqube.org/latest/ (visitato il 25/03/2022) (cit. a p. 40).

[57] *Metric Definitions.* URL: https://docs.sonarqube.org/latest/user-guide/metric-definitions/ (visitato il 27/03/2022) (cit. a p. 41).

[58] *OWASP Dependency-check.* URL: https://owasp.org/www-project-dependency-check/ (visitato il 25/03/2022) (cit. a p. 44).

[59] *OWASP Dependency-check plugin.* URL: https://www.jenkins.io/doc/pipeline/steps/dependency-check-jenkins-plugin/#dependencycheckpublisher-publish-dependency-check-results (visitato il 26/03/2022) (cit. a p. 46).

[60] *Trivy*. URL: https://aquasecurity.github.io/trivy/v0.17.0/ (visitato il 25/03/2022) (cit. a p. 47).

[61] RedHat. *Cosa si intende con Infrastructure as Code (IaC)?* URL: https://www.redhat.com/it/topics/automation/what-is-infrastructure-as-code-iac (visitato il 20/09/2022) (cit. a p. 47).

[62] *Docker Bench for Security*. URL: https://github.com/docker/docker-bench-security (visitato il 26/03/2022) (cit. a p. 47).

[63] *Checkov*. URL: https://github.com/bridgecrewio/checkov (visitato il 25/03/2022) (cit. a p. 47).

[64] *Comparison with other scanners*. URL: https://aquasecurity.github.io/trivy/v0.17.0/comparison/ (visitato il 24/03/2022) (cit. a p. 50).

[65] RedHat. *What is Clair?* URL: https://www.redhat.com/en/topics/containers/what-is-clair (visitato il 25/03/2022) (cit. a p. 50).

[66] *Anchore Engine*. URL: https://github.com/anchore/anchore-engine (visitato il 25/03/2022) (cit. a p. 50).

[67] *Kube-Bench*. URL: https://github.com/aquasecurity/kube-bench (visitato il 15/07/2022) (cit. a p. 52).

[68] *Kube-Hunter*. URL: https://github.com/aquasecurity/kube-hunter#kuberentes-attck-matrix (visitato il 15/07/2022) (cit. a p. 52).

[69] Ben Hirschberg. *Kubescape: A Kubernetes open-source platfrom providing a multi-cloud Kubernetes single pane of glass*. URL: https://www.armosec.io/blog/kubescape-the-first-tool-for-running-nsa-and-cisa-kubernetes-hardening-tests/ (visitato il 18/07/2022) (cit. a p. 52).

[70] *Spring Boot Annotations*. URL: https://www.javatpoint.com/spring-boot-annotations (visitato il 27/03/2022) (cit. a p. 55).

[71] *La Dependency Injection in Spring*. URL: https://daviooh.com/blog/2017/08/19/spring-dependency-injection#:~:text=Si%20tratta%20degli%20oggetti%20dedicati,identificate%20dall'annotazione%20%40Repository%20. (visitato il 22/09/2022) (cit. a p. 55).

[72] *H2 Database Engine*. URL: http://www.h2database.com/html/download.html (visitato il 23/09/2022) (cit. a p. 56).

[73] *H2 : Security Vulnerabilities*. URL: https://www.cvedetails.com/vulnerability-list/vendor_id-17893/product_id-45580/H2database-H2.html (visitato il 14/06/2022) (cit. a p. 56).

[74] *CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes*. URL: https://cwe.mitre.org/data/definitions/915 (visitato il 15/06/2022) (cit. alle pp. 56, 64).

[75] *Docker openjdk:17-alpine*. URL: https://snyk.io/test/docker/openjdk%3A17-alpine#SNYK-ALPINE314-OPENSSL-1569445 (visitato il 16/04/2022) (cit. alle pp. 58, 69).

[76] *CVE-2021-3711 Detail*. URL: https://www.cve.org/CVERecord?id=CVE-2021-3711 (visitato il 16/04/2022) (cit. a p. 58).

[77] *CVE-2021-36159 Detail*. URL: https://www.cve.org/CVERecord?id=CVE-2021-36159 (visitato il 16/04/2022) (cit. a p. 58).

[78] *CVE-2022-37434 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2022-37434 (visitato il 20/09/2022) (cit. a p. 58).

[79] *Dockerfile reference*. URL: https://docs.docker.com/engine/reference/builder/ (visitato il 16/04/2022) (cit. a p. 58).

[80] Wikipedia. *Gibibyte — Wikipedia, L'enciclopedia libera*. 2021. URL: http://it.wikipedia.org/w/index.php?title=Gibibyte&oldid=118028398 (visitato il 20/09/2022) (cit. a p. 59).

[81] *Che cos'è ?AWS Command Line Interface?* URL: https://docs.aws.amazon.com/it_it/cli/latest/userguide/cli-chap-welcome.html (visitato il 20/04/2022) (cit. a p. 59).

[82] *Cos'è Amazon VPC?* URL: https://docs.aws.amazon.com/it_it/vpc/latest/userguide/what-is-amazon-vpc.html (visitato il 20/04/2022) (cit. a p. 59).

[83] *Che cos'è Amazon S3?* URL: https://docs.aws.amazon.com/it_it/AmazonS3/latest/userguide/Welcome.html (visitato il 20/04/2022) (cit. a p. 59).

[84] *Kubernetes Dashboard*. URL: https://github.com/kubernetes/dashboard (visitato il 02/05/2022) (cit. a p. 60).

[85] *Using RBAC Authorization*. URL: https://kubernetes.io/docs/reference/access-authn-authz/rbac/ (visitato il 02/05/2022) (cit. a p. 61).

[86] *CVE-2021-23463 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-23463 (visitato il 14/06/2022) (cit. a p. 67).

[87] *CVE-2021-42392 Detail*. URL: https://nvd.nist.gov/vuln/detail/cve-2021-42392 (visitato il 14/06/2022) (cit. a p. 67).

[88] *CVE-2022-23221 Detail*. URL: https://nvd.nist.gov/vuln/detail/CVE-2022-23221 (visitato il 15/06/2022) (cit. a p. 67).

[89] Sysdig. *Cloud-Native Security and Usage Report*. 2022 (cit. a p. 68).

[90] *Sysdig Secure*. URL: https://sysdig.com/products/secure/ (visitato il 26/04/2022) (cit. a p. 73).

[91] *Qualys Container Security*. URL: https://www.qualys.com/apps/container-security/ (visitato il 28/04/2022) (cit. a p. 73).

[92] *DataDog*. URL: https://www.datadoghq.com/ (visitato il 28/04/2022) (cit. a p. 73).

[93] *The Falco Project*. URL: https://falco.org/ (visitato il 26/04/2022) (cit. a p. 73).

[94] *Falco Sidekick*. URL: https://github.com/falcosecurity/falcosidekick (visitato il 26/04/2022) (cit. alle pp. 74, 75).

[95] *Helm*. URL: https://helm.sh/docs/ (visitato il 26/04/2022) (cit. a p. 74).

[96] Piero Todorovich. *Quali sono i vantaggi per il business e gli sviluppatori del serverless computing*. URL: https://www.zerounoweb.it/cloud-computing/serverless-computing-vantaggi/ (visitato il 27/04/2022) (cit. a p. 75).

[97] *OpenFaas*. URL: https://docs.openfaas.com/ (visitato il 27/04/2022) (cit. a p. 76).

[98] *Cos'è Function-as-a-Service (FaaS)?* URL: https://www.redhat.com/it/topics/cloud-native-apps/what-is-faas (visitato il 27/04/2022) (cit. a p. 76).

[99] *Go Language*. URL: https://go.dev/ (visitato il 29/09/2022) (cit. a p. 76).

[100] *Slack*. URL: https://slack.com/intl/it-it/features (visitato il 29/04/2022) (cit. a p. 76).

[101] *Staseera.it*. URL: https://pingiovani.regione.puglia.it/vincitori/staseera-it (visitato il 25/09/2022) (cit. a p. 76).

[102] *Minikube*. URL: https://minikube.sigs.k8s.io/docs/ (visitato il 29/05/2022) (cit. a p. 78).

[103] *Kind*. URL: https://kind.sigs.k8s.io/ (visitato il 04/06/2022) (cit. a p. 78).

[104] *K3s, Lightweight Kubernetes*. URL: https://k3s.io/ (visitato il 01/06/2022) (cit. a p. 78).

[105] *Introducing Amazon EC2 M5n, M5dn, R5n, and R5dn Bare Metal Instances*. URL: https://aws.amazon.com/it/about-aws/whats-new/2021/02/introducing-amazon-ec2-m5n-m5dn-r5n-and-r5dn-bare-metal-instances/ (visitato il 06/06/2022) (cit. a p. 78).

[106] *Arkade - The Open Source Marketplace For Kubernetes*. URL: https://github.com/alexellis/arkade (visitato il 27/04/2022) (cit. a p. 78).

[107] *OpenFaaS stack*. URL: `https://docs.openfaas.com/architecture/stack/#application-layer` (visitato il 28/04/2022) (cit. a p. 78).

[108] *NATS*. URL: `https://nats.io/` (visitato il 28/04/2022) (cit. a p. 79).

[109] *Prometheus*. URL: `https://prometheus.io/` (visitato il 28/04/2022) (cit. a p. 79).

[110] *Falco chart*. URL: `https://github.com/falcosecurity/charts/blob/master/falco/values.yaml` (visitato il 04/05/2022) (cit. a p. 79).

[111] *Falco Sidekick chart*. URL: `https://github.com/falcosecurity/charts/blob/master/falcosidekick/values.yaml` (visitato il 10/05/2022) (cit. a p. 80).

[112] *Falco Sidekick UI*. URL: `https://github.com/falcosecurity/falcosidekick-ui` (visitato il 15/05/2022) (cit. a p. 81).

[113] *OpenFaaS CLI*. URL: `https://github.com/openfaas/faas-cli` (visitato il 28/04/2022) (cit. a p. 81).

[114] *OpenFaaS YAML file reference*. URL: `https://docs.openfaas.com/reference/yaml/` (visitato il 30/04/2022) (cit. a p. 81).