Kubernetes Observability & Detection Engineering

Corso di **Penetration Testing & Ethical Hacking**
Prof. Arcangelo Castiglione, a.a. 2023/2024

Un'analisi a cura di **Antonio Gravino**
github.com/antoniogrv/kube-observe

Indice dei contenuti

1 Introduzione e motivazioni progettuali

Questo documento, intitolato **Kubernetes Observability & Detection Engineering**, ha l'obiettivo di illustrare un lavoro progettuale di natura parzialmente compilativa e con una limitata componente pratica, descrivendo in maniera esaustiva e organica l'analisi svolta nel contesto in esame, e mostrando in maniera dettagliata e rigorosa i procedimenti svolti per ottenere l'applicativo realizzato.

Nella fattispecie, il progetto è relativo alla tipologia quindici (15) fra quelle selezionabili per il corso di Penetration Testing & Ethical Hacking, "*Sistemi SOAR (Security Orchestration, Automation and Response) e SIEM (Security Information and Event Management)*". I temi trattati sono laterali rispetto ai contenuti principali erogati durante il corso, ma sono in ogni caso fortemente legati ad argomenti di sicurezza, e che comunque rientrano nella traccia sopracitata.

Inoltre, poiché non è stato fornito un template predefinato da compilare, la stesura di questo documento (nella misura del suo formato e dei suoi contenuti) è a discrizione dell'autore, auspicando che risulti essere completa e soddisfacente.

La scelta di questa specifica traccia è dovuta ad un interesse personale dello studente nei confronti delle tecnologie in esame, specificamente **Kubernetes**, e il desiderio di esplorare nuovi anfratti del suo ecosistema, intersezionandosi ove possibile con i temi della traccia, per quanto le possibilità di lacciarsi a specifici tool o software studiati durante il corso sia limitato. Il lettore, dunque, consideri questa attività progettuale come un'analisi - o meglio, un'introduzione - ad alcuni angoli di un vasto strumento che, ormai, non è più considerabile "*secondario*", o "*ausiliare*"; e che, anzi, risulta essere il principale coltellino svizzero di un qualsiasi *DevSecOps Engineer* contemporaneo.

Kubernetes è spesso considerata una "*bestia*", uno strumento "*difficile*", "*capzioso*", profondamente ramificato nel mondo cloud-native e le cui articolazioni di produzione non possono esimersi dallo svolgere le dovute considerazioni di sicurezza, come previsto in qualsiasi sistema degno di una tale popolarità. Qualsiasi utente di Kubernetes deve fare i conti con la sicurezza dei propri container e del proprio cluster, a meno di applicativi estremamente semplici e poco affascinanti: capire come questa sicurezza può essere espletata, anche nella misura tale per cui ci si limita a *guardare* piuttosto che *agire*, è vitale alla compliance dovuta agli stakeholders e agli utenti del sistema.

Il corso di Penetration Testing insegna allo studente a scrutare una macchina virtuale, smembrarla, farla propria. Questo progetto, invece, punta ad illustrare *come capire che tutto questo sta accadendo*, e quali sono gli strumenti più adeguati per farlo; prima, però, viene descritto il funzionamento di un tipico cluster Kubernetes, mettendolo anche in correlazione agli ambienti didattici di virtualizzazione implementati durante il corso; vengono, poi, fornite opportunamente alcune definizioni critiche per il contesto affrontato (fra cui SIEM e SOAR) e, infine, viene proposta una piccola sperimentazione esemplificativa.

In particolare, la sperimentazione riguarda il bootstrap di un sistema SIEM su Kubernetes usando strumenti allo stato dell'arte, seguito da una serie di considerazioni di natura genuinamente pratica e tecnica sul perché il local development di sistemi di observability sia poco pratico, e su perché quindi la sperimentazione svolta non abbia davvero realizzato a pieno un SIEM, limitandosi alle fasi di bootstrapping.

Tutto ciò afferisce, in parte o in tutto, ad una branca del DevOps Engineering e della Cybersecurity spesso indicata come "Detection Engineering", in cui si pone particolare accento sulla questione dell'osservabilità ("observability"). L'esigenza che un enorme cluster composto da centinaia di macchine virtuali e ospitante migliaia di applicativi debba essere osservabile (in una misura o nell'altra) potrebbe risultare scontata; tuttavia, è facile dimenticare che il prezzo delle risorse (umane, economiche, computazionali) impiegate per realizzare sistemi di observability viene ripagato esclusivamente nei casi peggiori in cui questi sistemi arrivano a notificare disastri ed incidenti; fino ad allora, sembreranno non costituire alcun valore aggiunto. Ebbene, un incidente di sicurezza può davvero essere fatale per l'integrità e la confidenzialità dei dati, e per la disponibilità degli applicativi: fare in modo che questi ultimi siao costantemente sotto l'occhio vigile di Kubernetes, a cui ne sarà delegata la governance e la mise-en-place, è il minimo che possiamo fare.

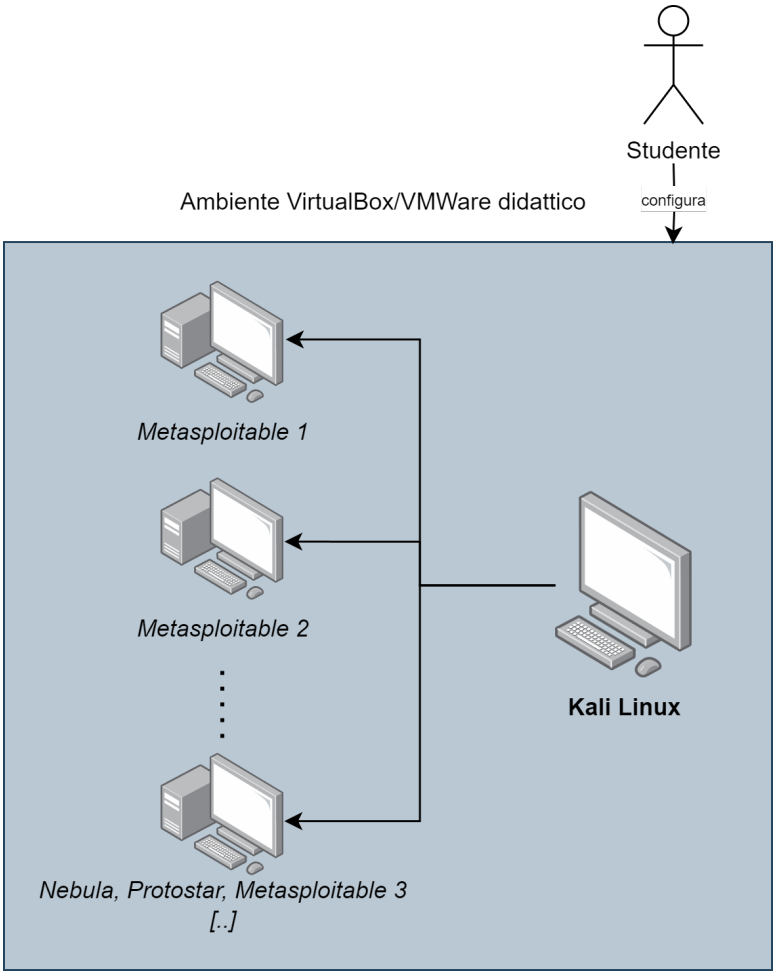
L'autore ha anche realizzato una tesi magistrale su Kubernetes, specializzando il proprio studio sull'orchestrazione di carichi di lavoro di Machine Learning (rif. Governance MLOps: Orchestrazione sicura di carichi di Machine Learning con Kubernetes, relatori: proff. Christiancarmine Esposito, Rocco Zaccagnino). Quindi, in termini strettamente personali, questo progetto è da considerarsi come un tentativo di colmare le lacune emerse durante tale lavoro di tesi, e riguardanti appunto l'osservabilità dei sistemi, che si intersezionano comodamente coi meccanismi SIEM e SOAR che andremo ad analizzare.

Tutti i riferimenti utilizzati sono raccolti nella bibliografia. I diagrammi nel Capitolo 2 sono stati creati di persona con Drawio. I screenshots del Capitolo 6 sono stati acquisiti di persona. Le immagini nei Capitoli 3, 4 e 5 provengono da Internet.

2 Approcciare applicativi del mondo reale

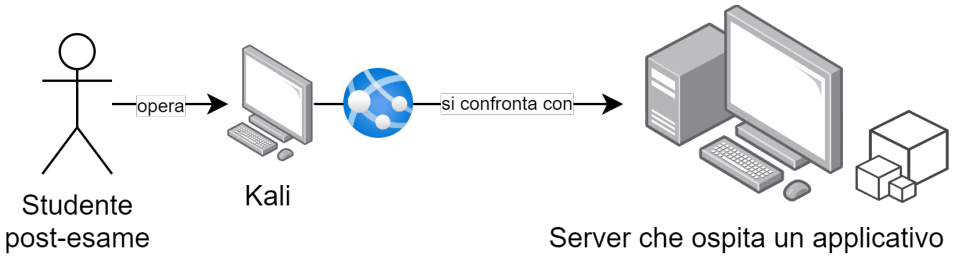
Un tipico studente del corso di Penetration Testing approccerà lo studio replicando, sotto istruzioni fornite dal docente, un ambiente virtualizzato in cui poter operare in maniera autonoma e sicura. L'ambiente, generalmente realizzato mediante i software di virtualizzazione VMWare o VirtualBox, consiste di macchine virtuali raggruppabili, ad esempio, in una rete NAT. Per la natura del corso, lo studente provvederà a caricare una macchina virtuali montante Kali Linux, e altrettante macchine virtuali vulnerable-by-design (da qui in poi "vulnerabili") da individuare, esplorare e infine soggiogare. Poiché queste macchine hanno l'esigenza di comunicare fra loro, e poiché non si vuole esporre le macchine virtuali verso l'esterno, allora lo studente realizzerà una rete NAT a cui afferiranno tutte le macchine virtuali (in genere, Metasploitable 1, 2, 3, e altre macchine tipicamente impiegate per il corso).

In questo modo, le macchine potranno accedere ad Internet venendo schermati dall'host dello studente; inoltre, le macchine saranno in grado di comunicare fra loro. L'obiettivo di questa configurazione è permettere che la macchina Kali riesca ad interfacciarsi con le macchine vulnerabili, e con le applicazioni che esse ospitano.



Chiaramente, si tratta di un ambiente di studio, propedeutico all'apprendimento dei contenuti erogati durante il corso.

Nella realtà, lo studente dovrà interfacciarsi con macchine configurate in maniera molto diversa. Tralasciando i dettagli di networking (che non sono trascurabili, ma che non rientrano nell'esame in oggetto), una macchina attaccante potrebbe dover interfacciarsi con una macchina target sita sulla rete pubblica, a prescindere dalla sua specifica locazione.

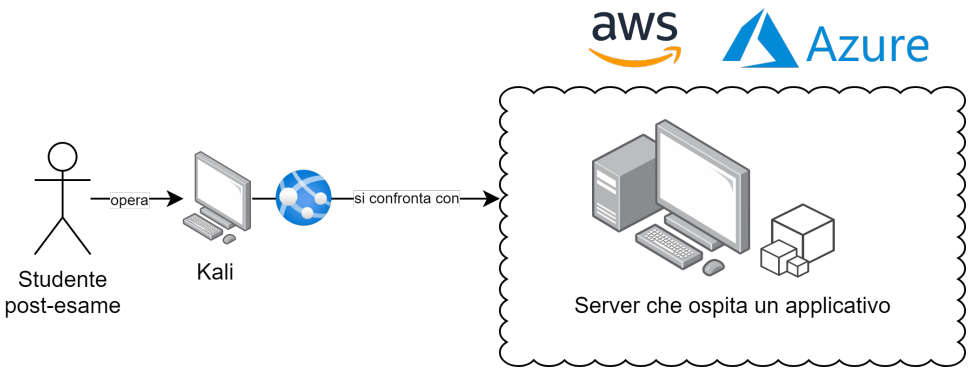


Nel nostro studio, la caratterizzazione del server target (i.e. se ha servizi vulnerabili, come li espone, come posso sfruttarli) in sé non è importante, se non per l'effettivo processo di Pentration Testing così come articolato durante il corso.

Indaghiamo, invece, la "cornice" attorno al target.

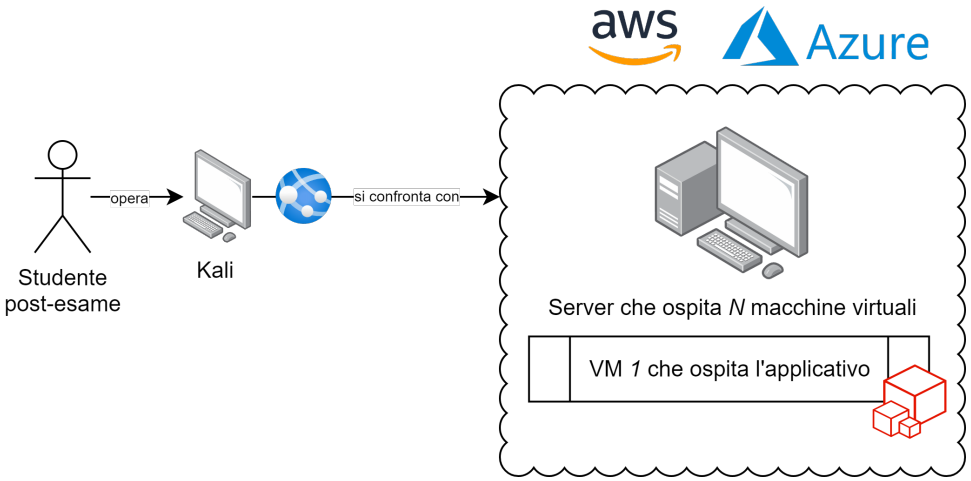
Plausibilmente, il server target è offerto (in leasing, o in altre forme commerciali) da un cloud provider, come Amazon Web Services (AWS), Microsoft Azure, o Google Cloud Provider. Questi tre provider, che sono i più noti e popolari, non sono comunque gli unici: alternative ben note sono rappresentate dai provider di IBM e Oracle, mentre in Italia viene tipicamente citato Aruba.

In alternativa, il server target potrebbe essere sito in colocazione ("colocation") in un data center di aziende che non fungono da cloud provider, come Hetzner. Questa soluzionesi differenzia dai server offerti dai cloud provider per i modelli di responsabilità adottati, e nei servizi offerti. In genere, l'impiego di un cloud provider non si limita al subaffitto di un server, ma implica l'utilizzo di strumenti accessori e servizi integrativi che permettono di sfruttare a pieno le capacità e le risorse del provider; nel caso di Hetzner, invece, ci si limiterebbe a prendere in carico il server.



Si osservi che sia nel caso del cloud provider che del colocator, il server è localizzato in un data center. In Italia, è di recente apertura il data center di Milano, occupato per lo più da Amazon Web Services.

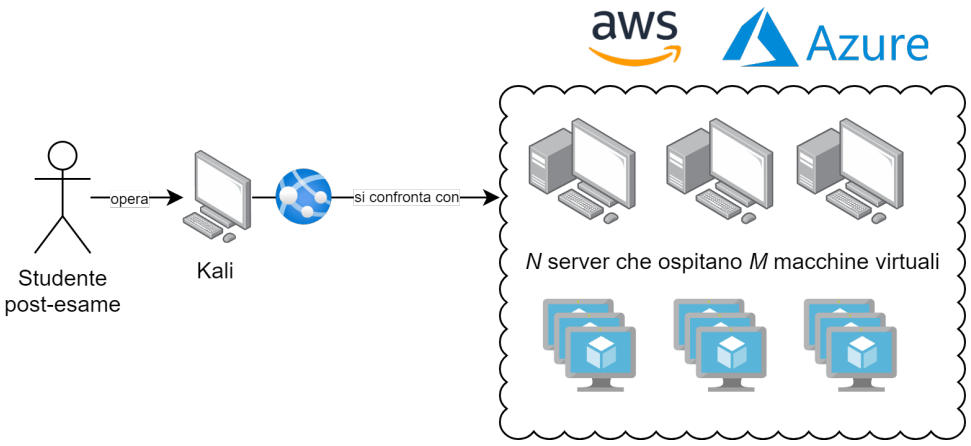
Il provider del server, in realtà, non offre un intero server: una delle idea fondamentali attorno al Cloud Computing è che il provider si limita, invece, ad offrire una certa *slice* del server, nella forma di una macchina virtuale. Il cliente, poi, è libero di innestare ulteriori macchine virtuali (gli applicativi) sulla propria macchina virtuale concessa dal provider, oppure usare soluzioni più snelle, moderne e compatte come i container – che, in ogni caso, sarebbero situate sulla macchina virtuale del provider.



Nella realtà, una singola macchina virtuale non è sufficiente. La scalabilità verticale degli applicativi ha degli hard caps non trascurabili, specialmente in termini di performance dell'hardware – pur facendo osservare che i provider permettono di modificare l'istanza della macchina virtuale (i.e. le caratteristiche hardware, spesso declinate in memoria e capacità di calcolo) con una certa facilità. Un pattern ben noto nel cloud computing, una regola contemporaneamente scritta e non scritta, riguarda l'esteso impiego della scalabilità orizzontale per bilanciare il carico fra le macchine virtuali.

AWS, così come gli altri cloud provider, mette a disposizione soluzioni commerciali e tecnologiche per implementare cluster di macchine virtuali, in qualche misura legate fra loro, e ospitanti gli stessi applicativi. Soluzioni del genere sono rappresentate da Elastic Container Service ed Elastic Beanstalk, fra le altre. Se, invece, ci si vuole limitare al provisioning di macchine virtuali, una soluzione tipica è rappresentata dagli EC2 Autoscaling Group, dove con "EC2" intendiamo una macchina virtuale offerta da AWS.

Quindi, sottointendendo la presenza di un Load Balancer (ala-Nginx, o una diversa soluzione open-source) che funga anche da proxy/reverse-proxy, l'attaccante Kali dovrà interfacciarsi spesso con un cluster di una certa quantità di macchine virtuali, ognuna situata potenzialmente su nodi fisici diversi (anche per assicurare affidabilità e high-avaibility in contesti multi-region), e ognuna contenente uno o più applicativi, vulnerabili o meno.



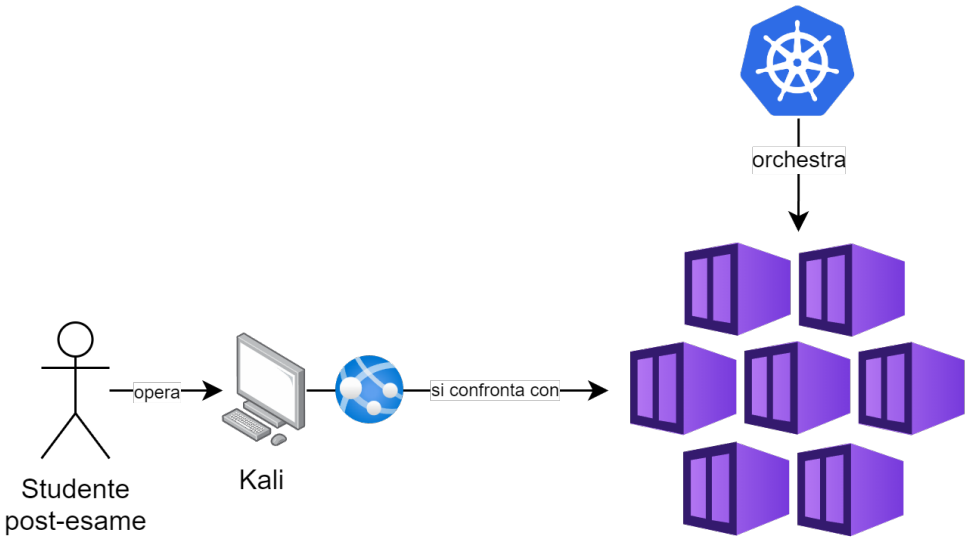
Nella pratica, il punto di incontro iniziale per l'attaccante è il Load Balancer. E' anche possibile che le macchine virtuali siano posizionate in segmenti diversi di rete, e a volte non sono raggiungibili *direttamente* dall'esterno. A volte si fa anche uso di un Bastion, o configurazioni di rete particolari, e potenzialmente vulnerabili, che aumentano notevolmente l'attack surface disponibile.

Le macchine virtuali vulnerabili usate durante il corso, come Metasploitabile 1 e 2, sono proviste di applicativi (tarate a specifiche versioni vulnerabili, in genere) come servizi FTP o applicazioni web fallaci (es. DVWA). Se, ad esempio, volessi bilanciare il carico dell'applicativo DVWA offerto su una macchina virtuale Metasploitable 2 schermata da un Load Balancer e su nodi offerti da un provider, avrei bisogno di un modo efficace ed efficiente di effettivamente replicare l'applicativo DVWA su tutte le macchine virtuali. Si osservi che l'attaccante Kali, in questa configurazione, potrebbe compromettere anche solo una delle istanze di DVWA, invece che tutte – dipenderebbe dallo scope del lavoro di penetration testing da svolgere.

Con una sola macchina virtuale, non subentrerebbe la necessità di replicare DVWA. Con più macchine virtuali, subentra invece la necessità di distribuire repliche di DVWA ad una platea considerevole di host, e di renderle rapidamente disponibili ad accogliere nuove richieste dal Load Balancer.

Un caso molto tipico è rappresentato da una situazione in cui DVWA non sarebbe l'unica applicazione sulle macchine virtuali. E' facile immaginare un sistema in cui la logica di business è espletata da molteplici applicativi, come sistemi di messaggistica (es. code, ala-Kafka), DBMS (es. MySQL), cache drivers (es. Redis), e così via. Per quanto la casistica dei database sia particolare, poiché implica l'esigenza di sincronizzare i dati fra le varie istanze, l'idea di avere più applicativi insieme su un certo cluster permane.

Emerge quindi la necessità di avere uno strumento atto a coordinare il cluster, governandone i nodi e le sue applicazioni. In questo contesto, lo stato dell'arte è rappresentato da **Kubernetes**.



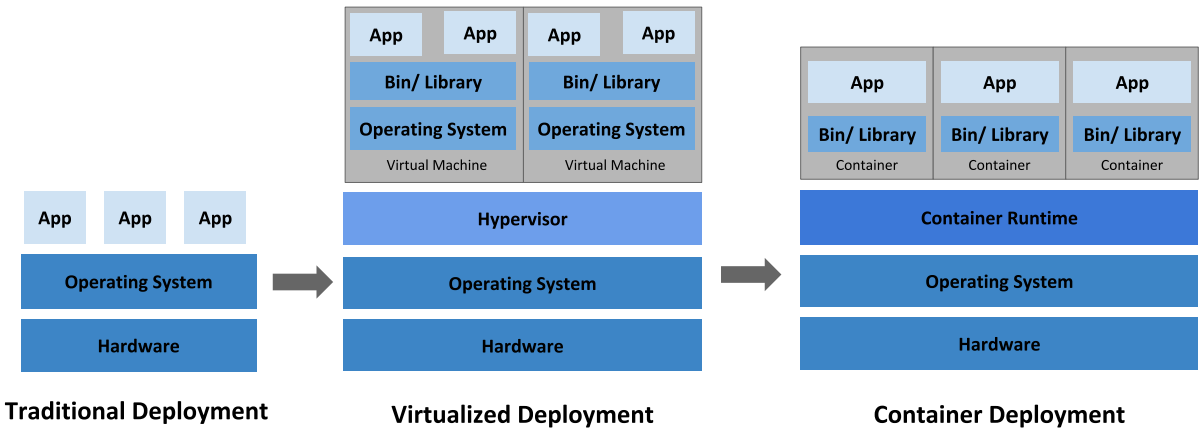
E' possibile fare tutto quello che fa Kubernetes senza Kubernetes, ma il costo (in termini economici!) è infinitamente più alto. Sempre più aziende, anche in Italia, stanno iniziando i primi timidi approcci a Kubernetes, visto come il miglior orchestratore disponibile ad oggi, e capace di sostituire la maggior parte dei servizi offerti dai cloud provider in maniera nativa ed open-source. Con Kubernetes, e le sue versioni fully-managed sui provider (es. Elastic Kubernetes Service, EKS), i provider si "riducono" a leaser di macchine virtuali, col vantaggio che il cliente non debba curarsi - fra le altre cose - delle condizioni fisiche dell'hardware.

3 Tecnologie di orchestrazione

Riprendendo l'esempio precedente, gli applicativi relativi a DVWA dovrebbero essere distribuiti su un notevole numero di nodi e quindi di macchine virtuali; il miglior modo di distribuirle, e cioè quello più rapido ed efficiente, è quello di containerizzare l'applicazione (nel caso di DVWA, in più container considerando che viene rilasciata con un DBMS MySQL ausiliario). Il container viene quindi replicato sulle macchine virtuali, e le porte del servizio (in casi tipici) vengono esposte verso l'esterno.

L'applicazione che l'attaccante dovrà affrontare, a questo punto, sarà rappresentata da un certo numero di container situati su un diverso numero di macchine virtuali e un altrettanto diverso numero di nodi fisici.

Non si discuteranno gli specifici vantaggi relativi alla containerizzazione a discapito della virtualizzazioni. Ci si limiterà a dire che queste due tecnologie vengono usate insieme, come vedremo; e che, in ogni caso, la distribuzione di applicazioni containerizzate piuttosto che virtualizzate è lo standard de-facto nell'industria.

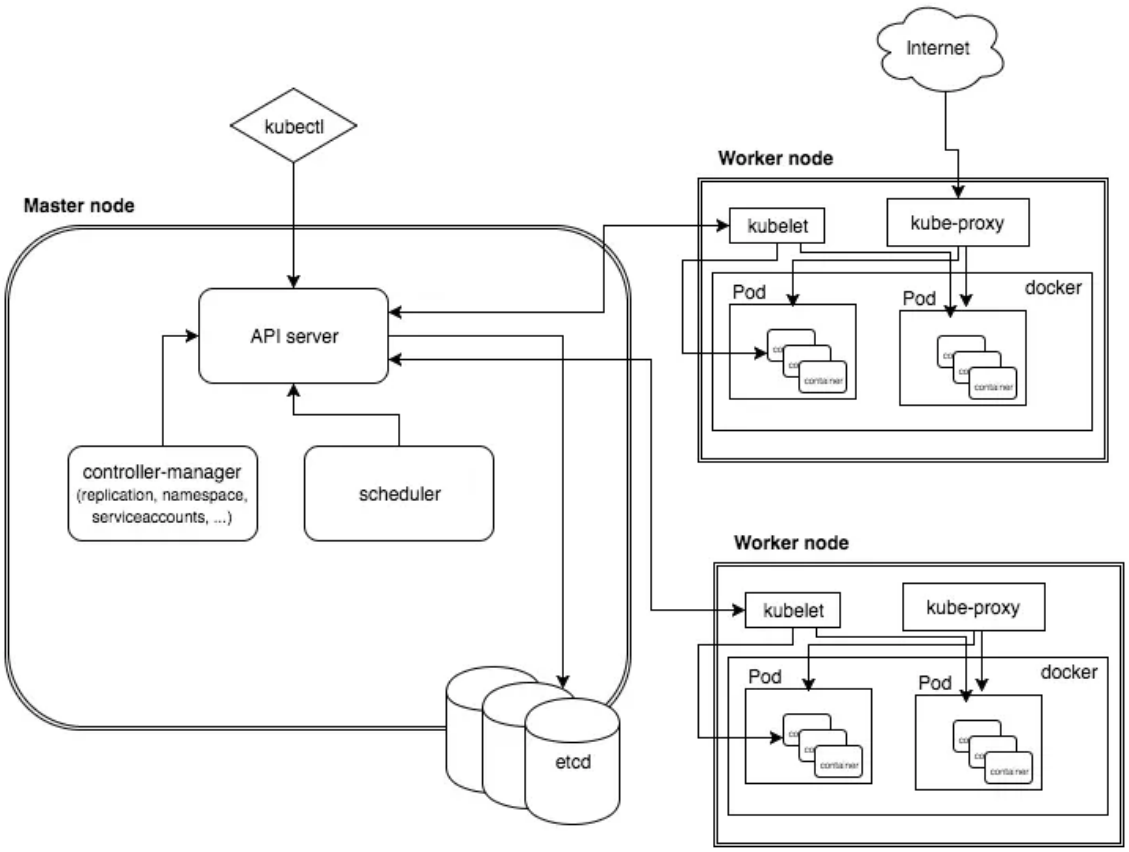


La distribuzione dei container sulle macchine virtuali può (e dovrebbe) avvenire tramite Kubernetes. Solitamente definito un "orchestratore", cioè un sistema in grado di orchestrare il ciclo di vita delle applicazioni sui nodi, Kubernetes permette di generare on-the-fly container anche molto complessi, distruggerli quando non sono più necessari, spostarli su diversi nodi per meglio bilanciare il carico, spegnerli, accenderli ed esporli verso l'esterno in base alle esigenze.

La definizione di Kubernetes presente sulla documentazione ufficiale descrive perfettamente il sistema.

"Kubernetes è una piattaforma portatile, estendibile e open-source per la gestione di carichi di lavoro e servizi containerizzati, in grado di facilitare sia la configurazione dichiarativa [degli applicativi, ndr] che l'automazione del ciclo di vita del software."

Analogamente, Kubernetes include una serie di meccanismi fondamentali per il corretto funzionametno di questo processo, come un sistema di esposizione dei servizi che consente di bilanciare il carico; l'auto-discovery degli applicativi sui nodi; la rapida mise-en-place di nuove versioni delle applicazioni ("rolling updates"), e **la capacità purgare gli applicativi infetti o compromessi in modo automatico**.



Nonostante Kubernetes sia un orchestratore di container, questi ultimi non sono i *first-class citziens* del sistema (i.e. "l'unità di lavoro di base"). Kubernetes preferisce, invece, lavorare con raggruppamenti logici di container chiamati "**Pod**".

Ad esempio, il Pod di DVWA potrebbe essere composto da due container: uno per il webserver PHP, e uno per MySQL.

Sia il webserver PHP che l'istanza MySQL dovrebbero essere containerizzati prima di essere orchestrati. Per fortuna, DVWA prevde già questa particolare casistica. Sulla repository GitHub del progetto, è infatti disponbiile una versione containerizzata del sistema, distribuita tramite Docker Hub (rif. DVWA on Docker Hub).

Concettualmente, avendo l'immagine dei container (i.e. le istruzioni per generarli), è

possibile istruire Kubernetes su come orchestrare l'applicazione, specificando la quantità e la qualità delle repliche, indicare i nodi da coinvolgere, aggiungere metadati.

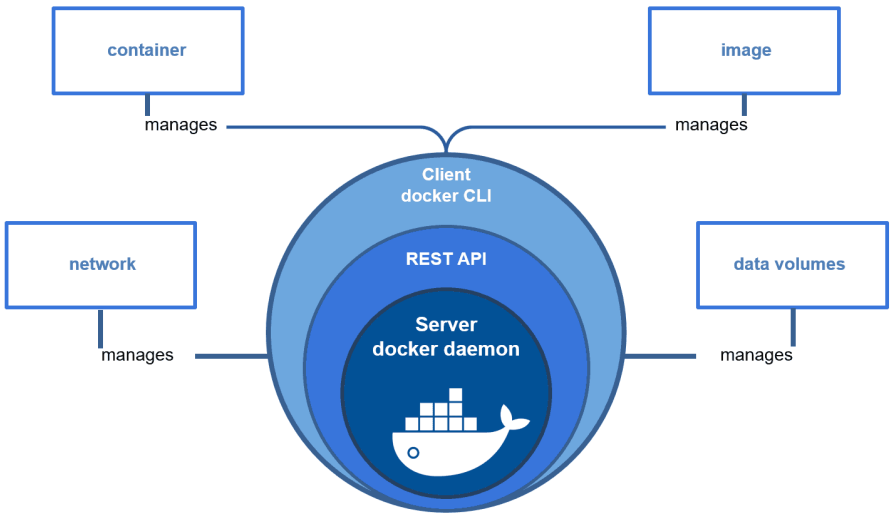
Difatto, lo sviluppatore dovrà interfacciarsi con la CLI di Kubernetes ("kubectl"). Istruire Kubernetes su cosa fare, e interrogare lo stato del cluster, implica l'inviare una richiesta HTTP verso il Kubernetes API Server, che è il centro nevralgico delle attività dell'orchestratore. L'API Server sfrutta alcuni meccanismi ausiliari, come uno Scheduler e dei Controller, per popolare il cluster di nuovi applicativi. Non si andrà nel dettaglio su quali tipi di "oggetti" sono disponibili, ma ci si limiterà a dire che tutti gli oggetti di Kubernetes (Service Accounts, IAM Roles, Namespaces, etc) collaborano per garantire il corretto funzionamento del sistema. In un cluster reale, la maggior parte degli oggetti coesisterebbe sul cluster, oltre all'applicativo DVWA opportunamente replicato.

Si aggiunge, però, che Kubernetes sfrutta un database distribuito ("etcd") per memorizzare lo stato del cluster.

Kubernetes, poi, non ha visione dei nodi a livello fisico, ma si interfaccia eslcusivamente sui "worker nodes", cioè sulle macchine virtuali. Agganciare e sganciare macchine virtuali dal cluster è immediato e non comporta sforzi. In pratica, "replicare DVWA" su Kubernetes significa effettuare una POST HTTP Request verso l'API Server, e attendere che gli oggetti di Kubernetes effettuino la replicazione. Under-the-hood, non siamo interessati a *come* questo viene fatto nel contesto di questo esame.

3.1 Containerizzazione tramite Docker

Gli applicativi orchestrati su Kubernetes devono essere containerizzati prima di essere orchestrati.



Un possibile runtime di containerizzazione è Docker, composto da un engine (il "Docker Daemon") e una CLI. Le immagini dei container vengono prodotti a partire dai Dockerfile, cioè un set di istruzioni (un "playbook") che istruisce il daemon su come realizzare, appunto, il container. Tipiche istruzioni riguardano l'utilizzo di certi software, l'installazione di dipendenze, e in generale tutto ciò che serve per il corretto funzionamento dell'applicativo. Si allega di seguito un esempio di Dockerfile dalla documentazione ufficiale di Docker.

```
# Questo esempio è ripreso dal Dockerfile ufficiale di Golang 1.22

FROM buildpack-deps:bookworm-scm

# install cgo-related dependencies
RUN set -eux; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
        g++ \
        gcc \
        libc6-dev \
        make \
        pkg-config \
    ; \
    rm -rf /var/lib/apt/lists/*

ENV GOLANG_VERSION 1.22.2

ENV GOTOOLCHAIN=local

ENV GOPATH /go
ENV PATH $GOPATH/bin:/usr/local/go/bin:$PATH
COPY --from=build --link /usr/local/go/ /usr/local/go/
RUN mkdir -p "$GOPATH/src" "$GOPATH/bin" && chmod -R 1777 "$GOPATH"
WORKDIR $GOPATH
```

Il lettore osserverà come un container creato con Docker è decisamente più snello di una macchina virtuale utilizzata per lo stesso scopo (cioè distribuire applicativi). Il tema della sicurezza apre margini di discussione, ma Docker rimane comunque un sistema ben collaudato.

Una volta containerizzati, gli applicativi devono essere caricati su un'image repository. I cloud provider offrono servizi di questo tipo, ma è anche possibile - ad esempio - sfruttare soluzioni open source, o banalmente usare un'istanza on-premise del Docker Registry.

Si fa osservare, comunque, che Kubernetes non usa Docker come container runtime, ma il software open-source "containerd". Entrambi usano comunque lo standard Open Con-

tainer Initiative per produrre le immagini (che sono fondamentalmente blob di dati), quindi risultano intercambiabili.

3.2 Local Kubernetes Development

Esistono numerosi strumenti per lavorare in locale con Kubernetes. Questi strumenti permettono di creare cluster locali (virtualizzati o containerizzati) e operare su di essi (in ambienti di "development") prima di procedere ad ambienti reali (di "staging", "pre-production", o "production").

Fra i tool open-source disponibili, si annoverano Kind, Minikube e Docker Desktop.

Docker Desktop integra, oltre al Docker Daemon, anche un engine integrato di Kubernetes che permette la creazione di un cluster mono-nodo. E' una recente introduzione del team di sviluppo di Docker e non risulta ad oggi particolarmente configurabile.

Kind permette invece di creare cluster multi-nodo, dove ogni nodo rappresenta un container su Docker. All'interno dei container dei nodi (i.e. macchine virtuali), sarebbero posizionati ulteriori container raggruppati in pod. Soluzione efficace e ben mantenuta dagli sviluppatori.

Minikube, invece, permette di creare cluster mono-nodo, retrocompatibili con precedenti versioni di Kubernetes, ed è il tool consigliato sulla documentazione stessa di Kubernetes.

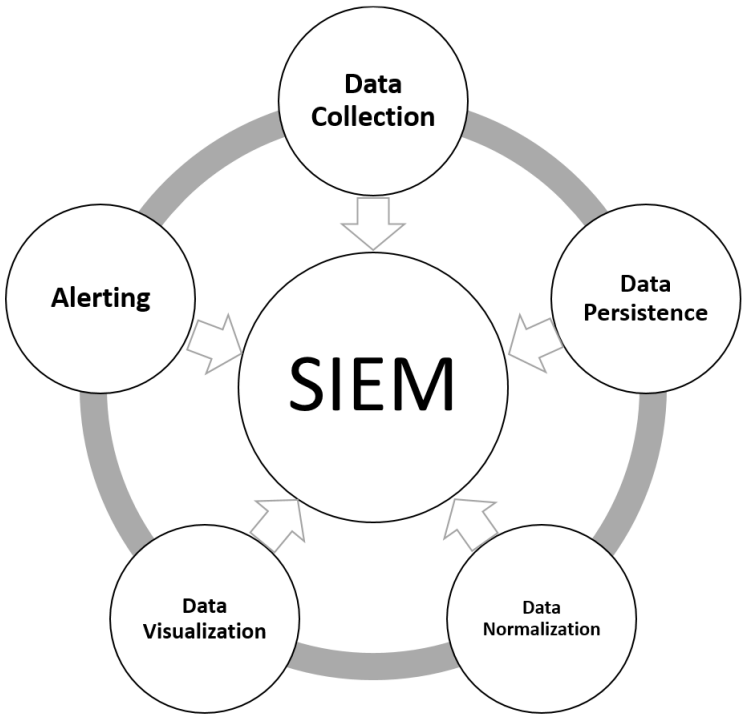
Tutti questi tool differiscono di poco fra loro, almneno per i fini di questo progetto. Si è scelto di utilizzare Minikube.

4 Sistemi SIEM e SOAR

I sistemi **SIEM** (Security Information and Event Management) e SOAR (Security Orchestration, Automation, and Response) sono fondamentali per la gestione della sicurezza informatica in ambienti complessi e sempre più minacciati. Il SIEM funge da sistema centrale per raccogliere, normalizzare e analizzare i dati provenienti da diverse fonti, come registri di sistema, eventi di sicurezza e log di rete. Questo consente agli analisti di sicurezza (in genere situati in un SOC) di individuare e rispondere rapidamente alle minacce, identificando schemi anomali o comportamenti sospetti.

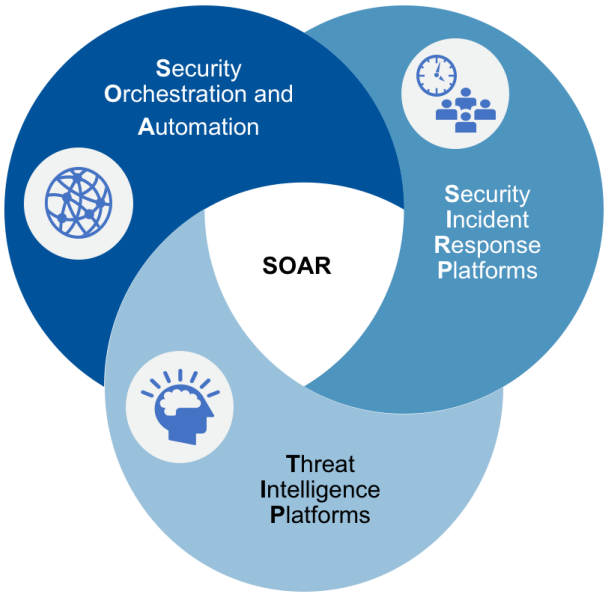
D'altra parte, il **SOAR** estende le capacità del SIEM integrando l'automazione e l'orchestrazione delle risposte alla sicurezza (da qui il nome). Questo significa che non solo il SOAR può individuare gli incidenti di sicurezza (grazie al feedback ricevuto dal SIEM, oppure in maniera autonoma), ma può anche prendere azioni automatiche per mitigare il rischio e risolvere i problemi di sicurezza. Ciò include l'avvio di risposte predefinite ("playbooks"), come l'isolamento di dispositivi compromessi, il blocco di indirizzi IP o l'avvio di investigazioni approfondite.

Insieme, SIEM e SOAR formano un ecosistema potente per la sicurezza informatica, fornendo agli operatori preposti gli strumenti necessari per rilevare, rispondere e mitigare le minacce in modo tempestivo ed efficiente. Questa integrazione aiuta le organizzazioni di grandi dimensioni a rafforzare le loro difese digitali e a proteggere i loro dati, riducendo al contempo il tempo di risposta agli incidenti e migliorando la loro capacità di adattarsi a nuove e sempre più sofisticate minacce informatiche.



In genere, con SIEM intendiamo una *dashboard* che raccoglie informazioni da una vasta gamma di fonti; con SOAR, invece, intendiamo un engine (cioè un software) che automatizza le risposte ai problemi di sicurezza – problemi individuati grazie ad informazioni pervenute dal SIEM, da sistemi terzi o da una componente del SOAR stesso.

Nella realtà, queste definizioni sono molto "liquide" e tendono a sovrapporsi. Si parla per lo più di "architetture di sicurezza" che estraggono le informazioni da diverse fonti, permettendo quindi il monitoring dello stato del sistema da parte degli umani (SIEM) e che nutriscono anche il bacino informativo di un engine che permette risposte automatizzate (SOAR).



SOAR = SOA + SIR + TIP

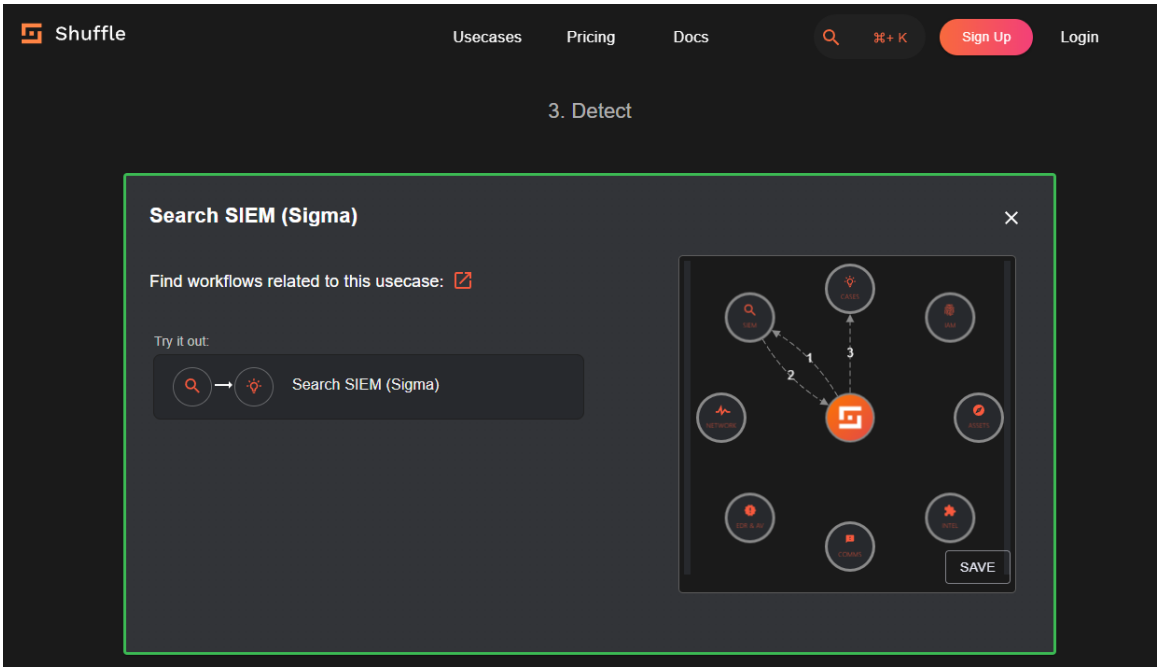
D'altra parte, il SOAR è caratterizzato da molteplici "knowledge bases" che permettono di definire in maniera più organica i protocolli di risposta. Una knowledge base può contenere, ad esempio, le metodologie di attacco più usuali e tipiche per il sistema in osservazione, con relative vulnerabilità ben note (es. CVE) e metodologie di risposta e mitigazione. Le specifiche implementazioni di un SOAR dipendendono dal vendor e dal software in uso (anche se in genere parliamo di molteplici software che coesistono e collaborano), ma nella sua forma classica permette comunque l'automatizzazione di risposte basate su protocolli ben definiti in funzione di feedback ricevuti da terzi.

L'implementazione di un SOAR può anche ridursi ad "se accade questo, fai quello" – nella pratica, è questo ciò che accade; tuttavia, in un ambiente enterprise, il "se accade questo" implicherebbe un feedback loop composto da pipeline più o meno articolate: le knowledge bases si incrociano per formare una fonte di verità da cui far scaturare una risposta. Ad esempio, i log del webserver PHP e i log di MySQL potrebbero convergere

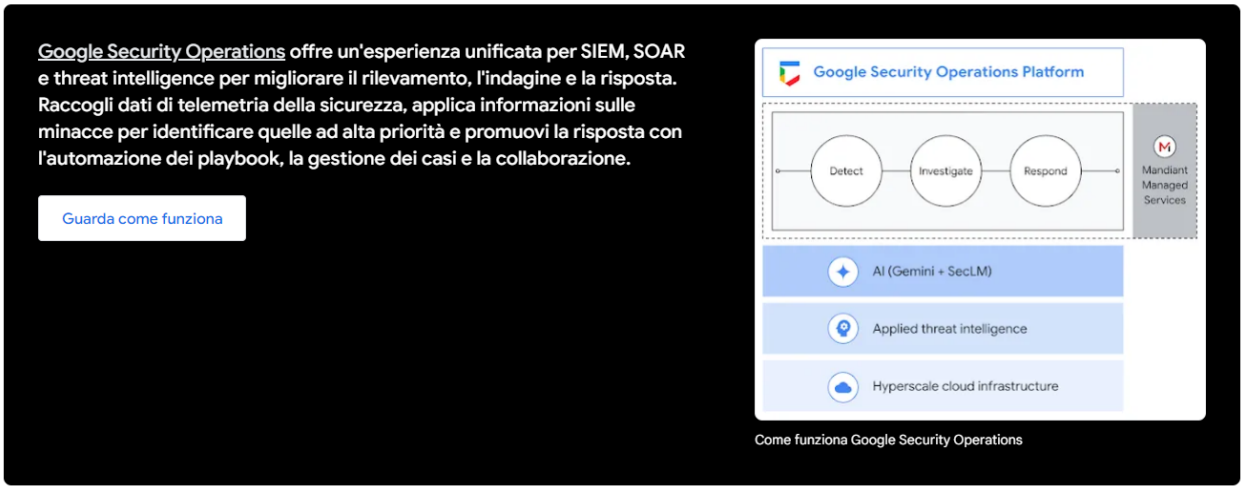
in una pipeline unificata che normalizzerebbe i dati, aggregandoli e componendoli in una misura tale che il successivo step (il "fai quello") sia in grado di prendere una decisione puntuale e informata.

Concretamente, spesso si parla di un sistema unificato – specialmente su Kubernetes, dove il principale sistema di osservabilità open-source, cioè l'ELK Stack, è contemporaneamnete definito SIEM e SOAR: banalmente, è sufficiente immaginare un sistema plug-and-play, in cui molteplici servizi possono essere agganciati ad un orchestratore (come ELK, che indagheremo in dettaglio più avanti), e che governano i feedback loops fra il feeding di nuove informazione (e le relative dashboard per gli utenti, come previsto dai SIEM), e l'automazione di risposte in funzione di tali informazioni (il SOAR), con tutto ciò che può esserci nel mezzo (pipeline di aggregazione, flussi dati, macchine a stati, validatori).

Un esempio pratico è dato dal SOAR enterprise "Shuffle". Shuffle permette di integrare svariate knowledge bases, fra cui alcuni sistemi SIEM, come il "Search SIEM" dell'azienda *SearchInform* (si tratta di soluzioni orientate ai business).



Accorpare le definizioni SOAR e SIEM è in realtà una pratica consolidata. E' sufficiente consultare il Google Security Operations Center (del cloud provider di Google, GCP). Questo servizio integra le funzionalità di dashboarding (SIEM) e automation (SOAR) in un unico, pratico SaaS offerto dal provider, e "augmented" (potenziato, migliorato) da servizi terzi del provider, come large-language models e storaging.



Come funziona Google Security Operations

Anche le aziende italiane apprezzato le soluzioni "all-in-one". Ad esempio, l'azienda milanese SGBox offre un SaaS modulare che integra le funzionalità di SIEM e SOAR, anche in questo caso potenziato da intelligenze artificiali nel contesto della network detection e della log analysis.

In generale, tutte queste soluzioni prevedono un sistema di logging ("ricordarci cos'è successo"), un sistema di detection ("capire se accade qualcosa fuori dalla norma"), un sistema di alerting ("notificare gli addetti ai lavori quando qualcosa è fuori posto"), coadiuvate da dashboard in real-time, e da moduli integrativi con engine di automazione ala-SOAR. OpenSearch (di proprietà di AWS) definisce con più precisione queste "componenti" (detection, logging, alerting) nella documentazione del suo SIEM SaaS "Security Analytics".

L'autore ha trovato particolarmente interessante la presentazione "Automated Cloud-Native Incident Response with Kubernetes and Service Mesh" dell'americano Matt Turner e l'italiano Francesco Beltramini alla Cloud Native Computing Foundation, in cui si presenta lo specifico caso d'uso dell'osservabilità su cluster Kubernetes, e come i tipici sistemi SOAR e SIEM SaaS oppure on-prem non si coniughino alla perfezione col modo di operare dell'orchestratore, proponendo invece strumenti adeguati alla natura cloud-native ed open-source di Kubernetes. In tal senso, una soluzione è rappresentata da ELK.

In ambito accademico, l'autore non ha rinvenuto alcun paper relativo all'impiego di SOAR e SIEM su Kubernetes.

5 Strumenti di osservabilità

Innanzitutto, si osserva che Elastic è stato brevemente citato durante il corso nell'ambito di un esempio pratico per la fase di Target Exploitation (ref. Target Exploitation - Parte 2, slide 48/49).

L'ELK Stack, che si basa su Elasticsearch e Kibana, offre un potente framework per l'analisi e la visualizzazione dei dati dei log.

- **Elasticsearch:** Questo è il cuore del sistema. Elasticsearch è un motore di ricerca distribuito e altamente scalabile che consente di memorizzare, indicizzare e cercare grandi quantità di dati in tempo reale. I dati vengono archiviati in cluster, dove ogni nodo contiene una parte del dataset. Elasticsearch fornisce API RESTful per interrogare i dati e supporta una vasta gamma di operazioni di ricerca e aggregazione.

  I dati da analizzare vengono inviati a Elasticsearch per l'indicizzazione. Questi dati possono provenire da una varietà di fonti, come log di applicazioni, eventi di sistema, metriche di performance, dati di sensori, e così via. L'ingestione dei dati può essere gestita tramite Logstash o tramite altre soluzioni di ingestione personalizzate.

- **Kibana:** Questa è l'interfaccia utente che consente agli utenti di esplorare, analizzare e visualizzare i dati memorizzati in Elasticsearch. Kibana fornisce un'ampia gamma di strumenti per la creazione di dashboard, grafici, mappe, e altre visualizzazioni interattive. Gli utenti possono eseguire query sui dati, creare filtri, eseguire analisi temporali e geospaziali, e personalizzare l'aspetto e la disposizione delle visualizzazioni.

  Gli utenti interagiscono con l'ELK Stack attraverso l'interfaccia utente di Kibana. Possono eseguire query sui dati, esplorare i log, identificare tendenze e anomalie, monitorare le metriche di sistema, e risolvere i problemi di prestazioni. Kibana fornisce anche funzionalità per la collaborazione, consentendo agli utenti di condividere dashboard e visualizzazioni con i loro colleghi.

Utilizzare l'ELK Stack come SIEM implica sfruttare le capacità di Elasticsearch per l'analisi dei dati dei log e Kibana per la visualizzazione dei risultati. In questo contesto, i dati provenienti da diverse fonti, come registri di sistema, eventi di sicurezza e log di rete, vengono inviati a Elasticsearch per l'indicizzazione. Utilizzando le potenti funzionalità di interrogazione e aggregazione di Elasticsearch, gli analisti di sicurezza possono individuare schemi anomali o comportamenti sospetti che potrebbero indicare un'attività malevola. Kibana fornisce un'interfaccia intuitiva per esplorare i dati, creare dashboard personalizzate e monitorare continuamente lo stato della sicurezza del sistema.

Per sfruttare l'ELK Stack come SOAR, invece, è possibile integrare l'automazione delle risposte agli incidenti (cioè l'engine SOAR in sé) direttamente nel flusso di lavoro di Elasticsearch e Kibana. Ad esempio, è possibile configurare avvisi basati su criteri predefiniti per rilevare automaticamente le minacce e avviare risposte programmate, come

l'isolamento di dispositivi compromessi, il blocco di indirizzi IP sospetti o l'avvio di investigazioni dettagliate. Utilizzando gli strumenti di visualizzazione di Kibana, gli operatori possono monitorare le azioni automatizzate e ottenere una panoramica completa delle operazioni di risposta agli incidenti.

Si fa osservare al lettore che tutto questo sarebbe realizzato attraverso specifiche integrazioni – che, purtroppo, risultano *per lo più* vendor-locked (i.e. a pagamento o con necessità di API keys). Alternative FOSS (Free & Open Source Software) esistono ma non si prestano al caso d'uso di Elastic.

L'utilizzo dell'ELK Stack come SIEM e SOAR (in simultanea) offre un'approccio integrato e altamente personalizzabile alla gestione della sicurezza informatica di sistemi complessi. Grazie alle sue capacità di analisi avanzate, automazione delle risposte e visualizzazione dei dati, l'ELK Stack consente alle organizzazioni di rilevare, rispondere e mitigare le minacce in modo tempestivo ed efficiente, migliorando così la sicurezza complessiva del loro ambiente IT.

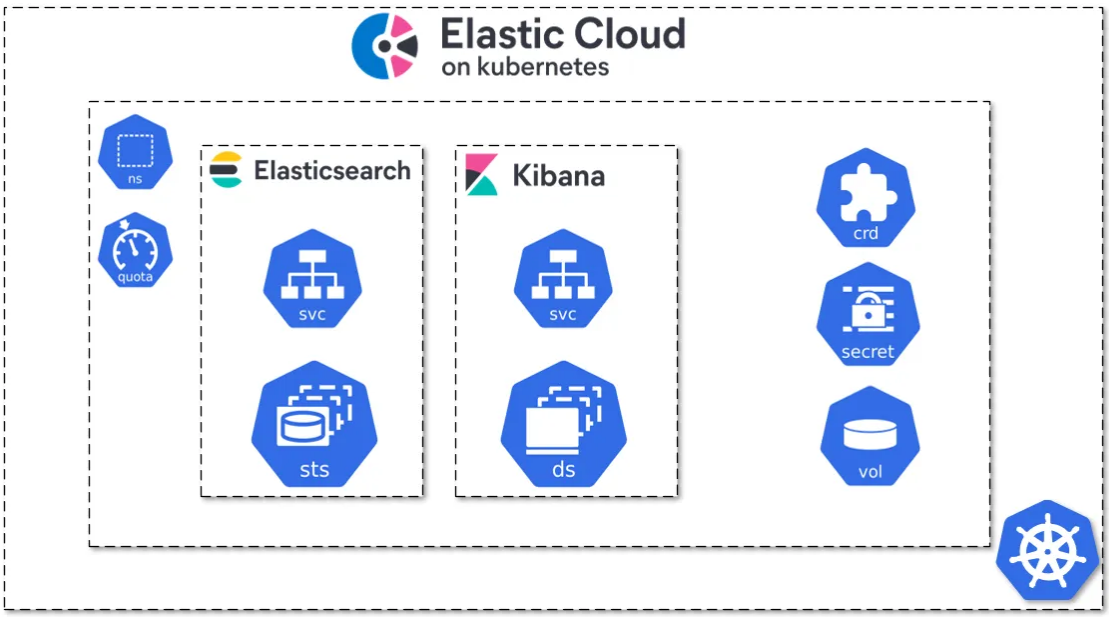
Come postilla, per aggiungere automazione delle risposte agli incidenti utilizzando l'ELK Stack come SOAR, è possibile integrare la funzionalità di rilevamento degli eventi di Elasticsearch con script di automazione personalizzati (usando linguaggi ben noti).

Segue un semplice esempio (non funzionante, solo concettuale) su come configurare Elasticsearch per creare un alert in base ad un endpoint HTTP di accesso.

```
PUT _watcher/watch/login_failed_alert
{
  "trigger": {
    "schedule": { "interval": "1m" }
  },
  "input": {
    "search": {
      "request": {
        "indices": ["logs"],
        "body": {
          "query": {
            "bool": {
              "must": [
                { "match": { "event": "login_failure" }},
                { "range": { "@timestamp": { "gte": "now-1m" }}}
              ]
            }
          }
        }
      }
    }
  },
```

```
  "condition": {
    "compare": {
      "ctx.payload.hits.total": { "gt": 5 }
    }
  },
  "actions": {
    "notify": {
      "logging": {
        "text": "Troppe richieste di login."
      }
    },
    "webhook": {
      "method": "POST",
      "host": "example.com",
      "port": 8080,
      "path": "/automated_response",
      "body": "Troppe richieste di login."
    }
  }
}
```

L'integrazione tra l'ELK Stack e Kubernetes consente di raccogliere, analizzare e visualizzare i dati dei log generati dai container (es. DVWA) e dalle applicazioni in esecuzione su un cluster Kubernetes. Potrebbe anche essere possibile integrare i probes di Kubernetes all'interno di Elastic.



Segue un esempio di come agganciare un sidecar container per effettuare il logging di un pod Kubernetes.

```
apiVersion: v1
kind: ElasticsearchLogger
metadata:
  name: elasticsearch
  namespace: kube-logging
spec:
  selector:
    app: elasticsearch
  ports:
    - name: http
      port: 9200
```

Successivamente, definiamo un pod che includa il nostro container principale insieme a un sidecar container che invia i log a Elasticsearch. In questo esempio, utilizzeremo un'applicazione web Python:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  namespace: default
spec:
  containers:
  - name: myapp-container
    image: myapp-image:v1
    ports:
    - containerPort: 8080
  - name: log-forwarder
    image: log-forwarder-image:v1
    env:
    - name: ELASTICSEARCH_HOST
      value: "elasticsearch.kube-logging.svc.cluster.local"
    - name: ELASTICSEARCH_PORT
      value: "9200"
```

Nel pod, il container myapp-container è l'applicazione web che vogliamo monitorare, mentre il container log-forwarder è il sidecar container che invia i log a Elasticsearch. Questo container può essere implementato utilizzando un'immagine personalizzata che utilizza strumenti come Fluentd o Logstash per l'invio dei log a Elasticsearch.

Infine, possiamo applicare queste configurazioni al cluster Kubernetes utilizzando il comando kubectl apply -f <nome-file.yaml> per creare il servizio e il pod. Una volta

applicate, i log generati dall'applicazione web saranno inviati a Elasticsearch e potranno essere esplorati e visualizzati attraverso Kibana per l'analisi e il monitoraggio.

6 Bootstrap di un SIEM su Kubernetes

L'obiettivo sperimentale di questa attività progettuale è effettuare il bootstrap di un SIEM usando l'ELK Stack su Kubernetes. Di conseguenza, l'obiettivo risulta essere impostare e configurare adeguatamente un nuovo cluster Kubernetes per accogliere le integrazioni di Elastic al fine ultimo di centralizzare la raccolta delle informazioni e l'automazione delle risposte, come abbiamo visto in precedenza.

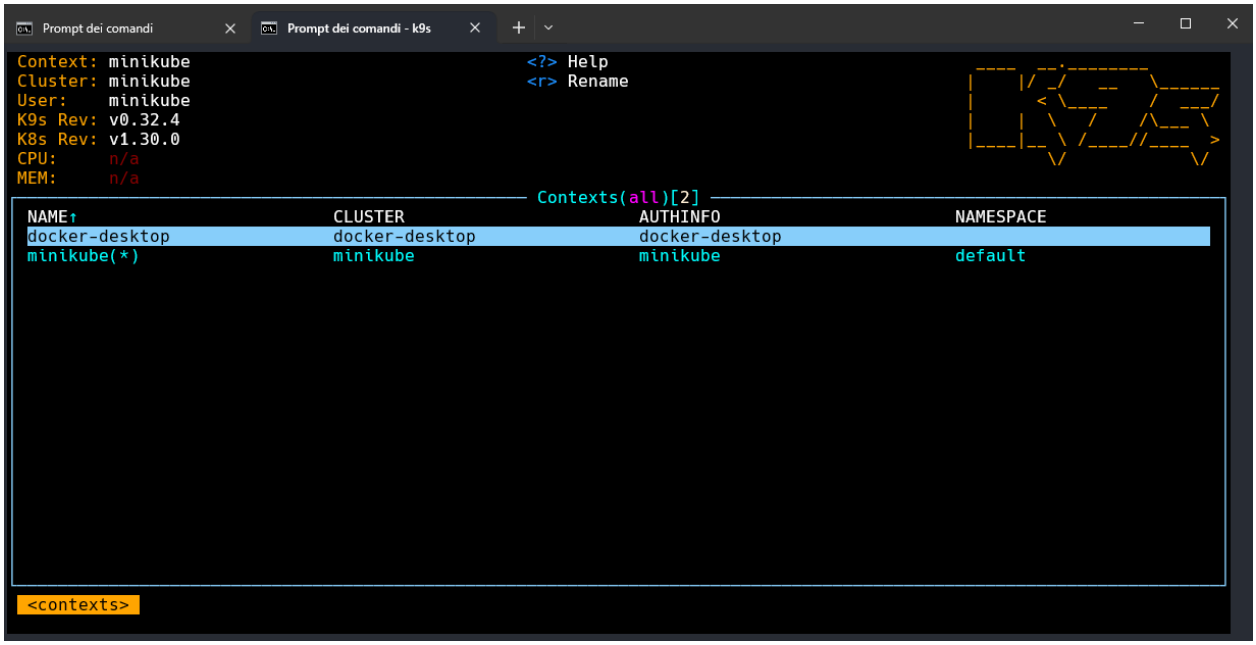
Nella fattispecie, creeremo con Minikube un cluster mono-nodo che ospiterà il SIEM. Per via delle specifiche necessità di Elastic, potremo creare un cluster usando la CLI di Minikube e configurando adeguatamente l'hardware virtuale allocato al container:

```
minikube start --cpus 4 --memory 8192
```

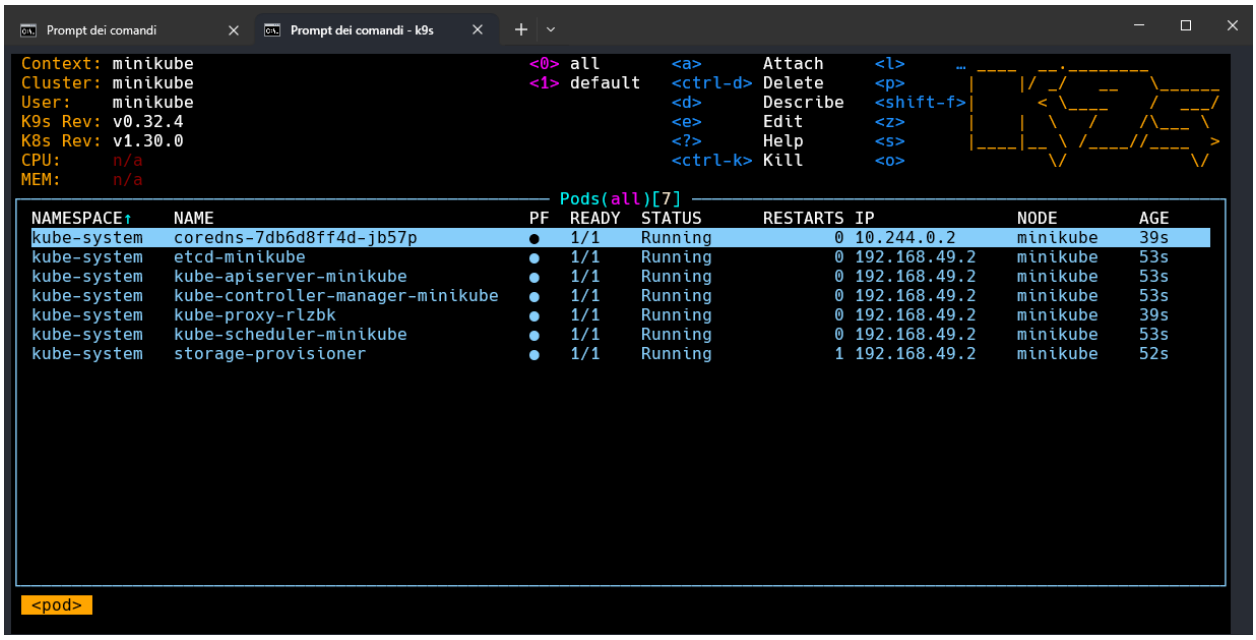
```
~
❯ minikube start --cpus 4 --memory 8192
W0503 10:42:40.518836   15492 main.go:291] Unable to resolve the current Docker CLI context "default": context "default"
: context not found: open C:\Users\19vie\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a3
3f0688f\meta.json: Impossibile trovare il percorso specificato.
* minikube v1.33.0 on Microsoft Windows 10 Home 10.0.19045.4291 Build 19045.4291
* Automatically selected the docker driver. Other choices: virtualbox, ssh
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.43 ...
* Creating docker container (CPUs=4, Memory=8192MB) ...
```

Kubernetes usa dei "contesti" per permettere la gestione dei cluster multipli. Lanciare il comando sopraindicato cambierà automaticamente il contesto predefinito al nuovo contesto "minikube". Possiamo quindi verificare il corretto funzionamento dell'API Server di Kubernetes, che è il fulcro del sistema. E' anche possibile utilizzare strumenti ausiliari come K9s per visualizzare in maniera più comoda il cluster.

```
kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:56326
CoreDNS is running at https://127.0.0.1/api/v1/namespaces/kube-system
 ↪  [...]
```



Con K9s possiamo accedere al contesto e verificare i pod attualmente presenti. Poiché l'installazione è andata a buon fine, inizialmente avremo solo i pod del namespace kube-system, necessari all'API Server per governare il cluster.



A questo punto possiamo agganciare Elastic al cluster Kubernetes mediante la CLI. Come accennato in precedenza, effettuare un "deploy" su Kubernetes significare effettuare una POST HTTP Request verso l'API Server, indicando un file YAML come input. Di contro,

Kubernetes genererà adeguatamente una serie di oggetti nel cluster per espletare le funzioni richieste. I file YAML di configurazioni usati sono rinvenibili nella directory /scripts della repository del progetto.

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: es-logging
  name: es-logging
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: es-logging
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: es-logging
    spec:
      containers:
      - image: elasticsearch:7.8.0
        imagePullPolicy: IfNotPresent
        name: elasticsearch
        ports:
        - containerPort: 9200
          name: http
          protocol: TCP
        - containerPort: 9300
          name: transport
          protocol: TCP
        env:
        - name: discovery.type
          value: single-node
        volumeMounts:
        - name: elasticsearch-logging
          mountPath: /data
        livenessProbe:
```

```
          httpGet:
            port: http
            path: /_cluster/health
          initialDelaySeconds: 40
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /_cluster/health
            port: http
          initialDelaySeconds: 30
          periodSeconds: 10
      volumes:
      - name: elasticsearch-logging
        emptyDir: {}
```

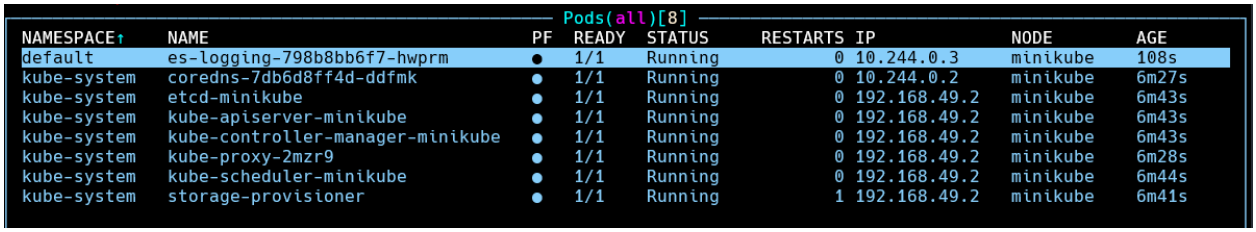
```
---
apiVersion: v1
kind: Service
metadata:
  name: es-service
  labels:
    app: es-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "Elasticsearch"
spec:
  type: NodePort
  ports:
  - port: 9200
    protocol: TCP
    targetPort: http
  selector:
    app: es-logging
```

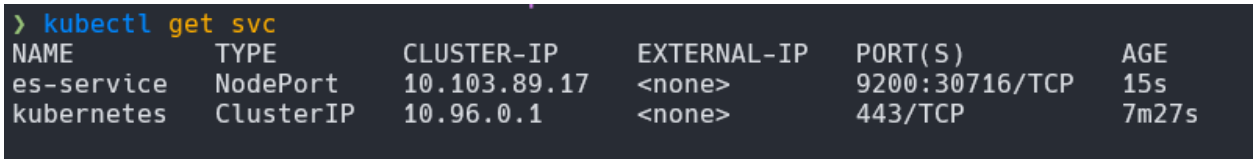
L'applicazione dei due componenti sopracitati, cioè il "Deployment" di Elastic (i.e. il backend) e il "Service" (l'endpoint con cui parlerà Kibana) può essere svolto tramite il seguente comando:

```
kubectl apply -f es-deployment.yaml -f es-service.yaml
```

Rispettivamente, il primo serve ad espletare la "logica di business", e il secondo ad esporla fuori e dentro al cluster.

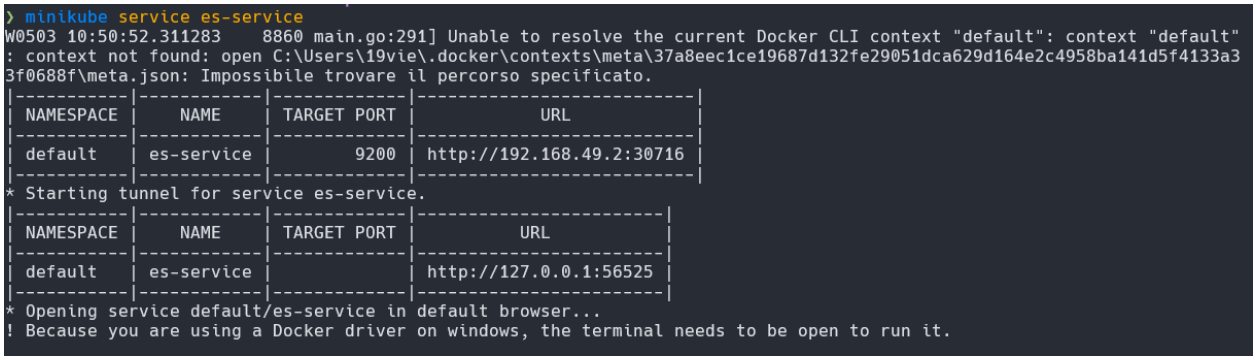
Verifichiamo la corretta creazione dei pod su K9s e dalla CLI.



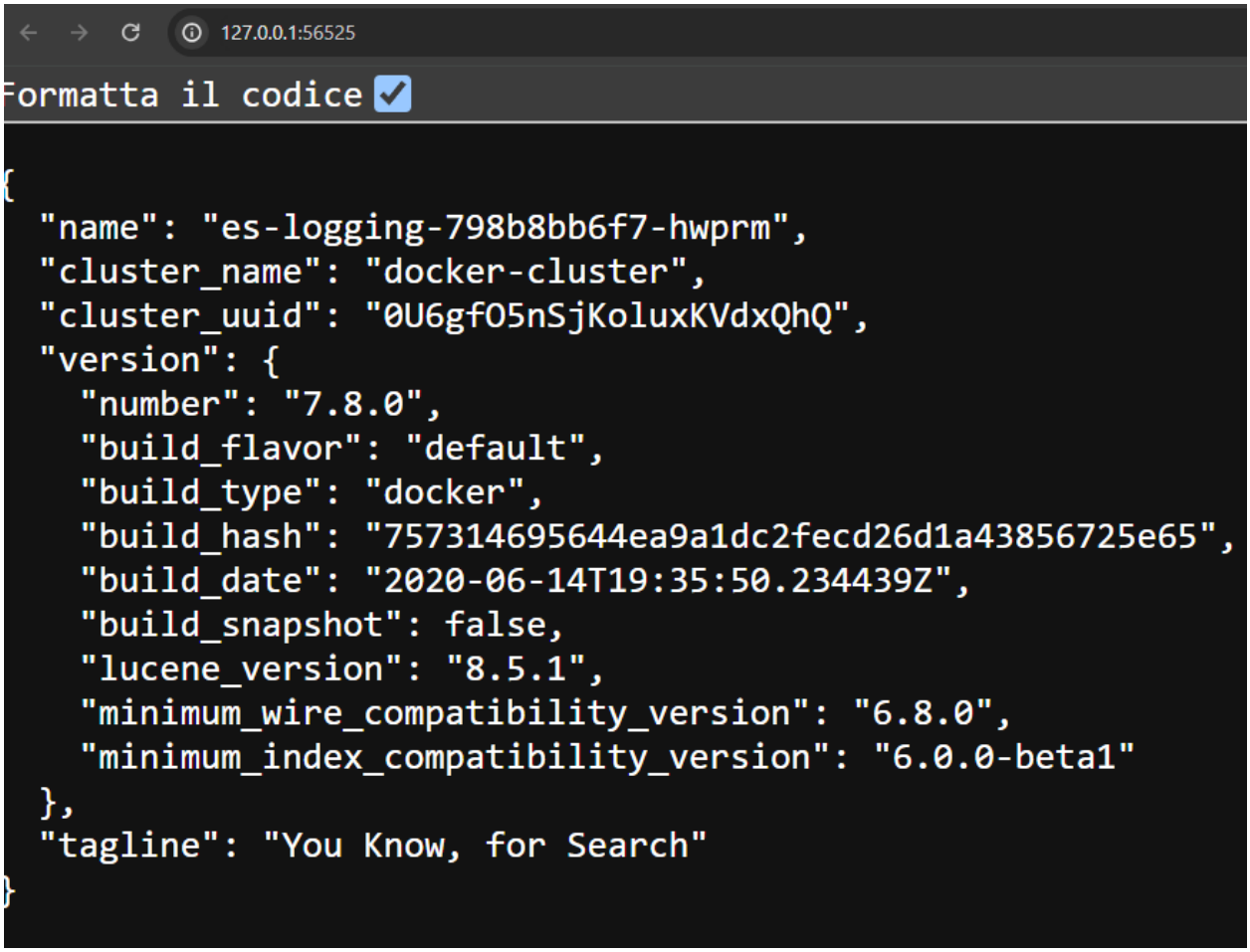


Possiamo adesso esporre Elastic verso l'esterno (fuori da Kubernetes) effettuando un port-binding fra la porta del servizio nel cluster, e una porta disponibile sull'host.

```
minikube service es-service
```



Se tutto è andato a buon fine, otterremo un JSON di risposta effettuando una GET alla porta 56526 del localhost.



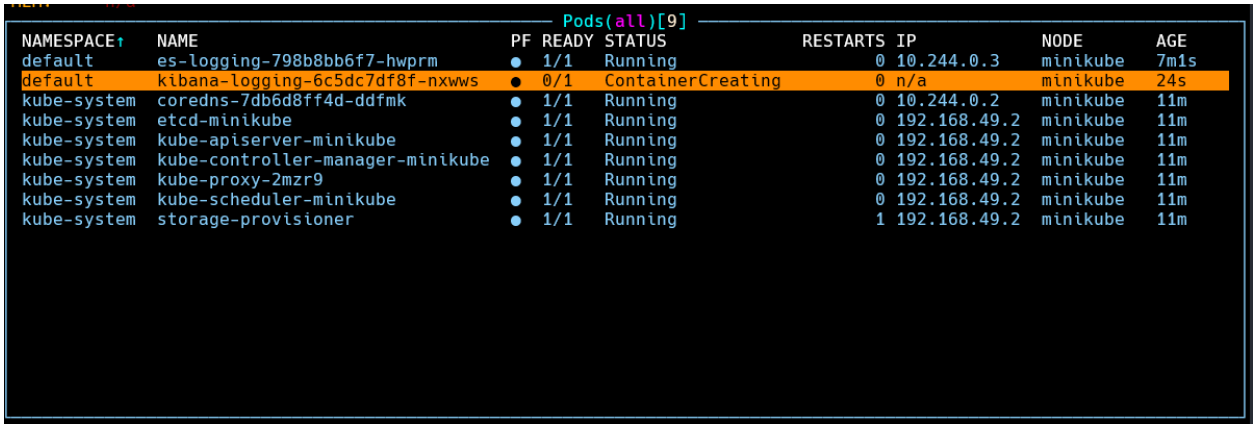
Possiamo reiterare lo stesso procedimento con Kibana. Questo mostra quanto è facile e immediato "deployare" applicativi su Kubernetes. Fondamentalmente, l'intero processo è astratto esclusivamente ai file YAML necessari a configurare i servizi.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: kib-manual
  name: kibana-logging
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kib-manual
  template:
    metadata:
      labels:
        app: kib-manual
```

```
    spec:
      containers:
      - image: kibana:7.8.0
        name: kibana
        ports:
        - containerPort: 5601
          name: ui
          protocol: TCP
```

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: kib-manual
  name: kibana-service
spec:
  ports:
  - port: 5601
    protocol: TCP
    targetPort: 5601
  selector:
    app: kib-manual
  type: NodePort
```

Verifichiamo il pod di Kibana con K9s.



Inoltre, il deployment produce un Service per esporre il servizio verso l'esterno e internamente nel cluster.

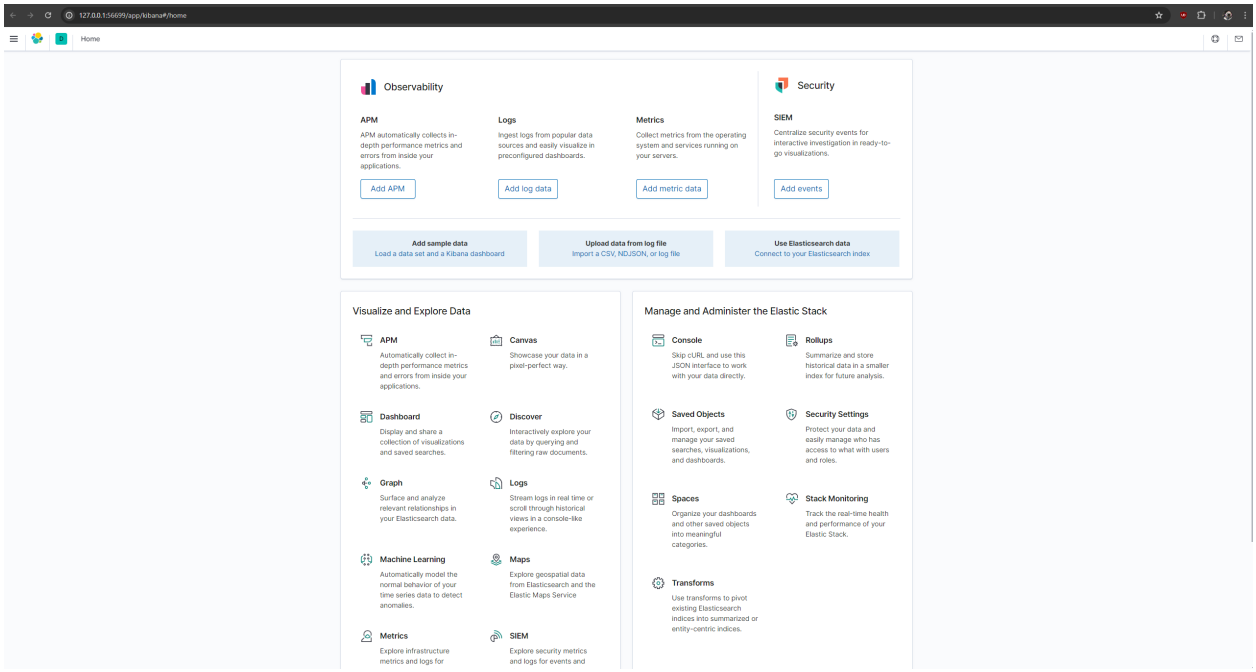
```
❯ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
es-service        NodePort    10.103.89.17    <none>        9200:30716/TCP   9m7s
kibana-service    NodePort    10.109.99.249   <none>        5601:30769/TCP   5m35s
kubernetes        ClusterIP   10.96.0.1       <none>        443/TCP          16m
```

Poiché Kibana è sostanzialmente un frontend per Elastic, che funge da backend, bisogna indicare l'host di Elastic (in riferimento al suo "ClusterIP", cioè l'IP di Elastic dentro il cluster) nelle variabili d'ambiente di Kibana.

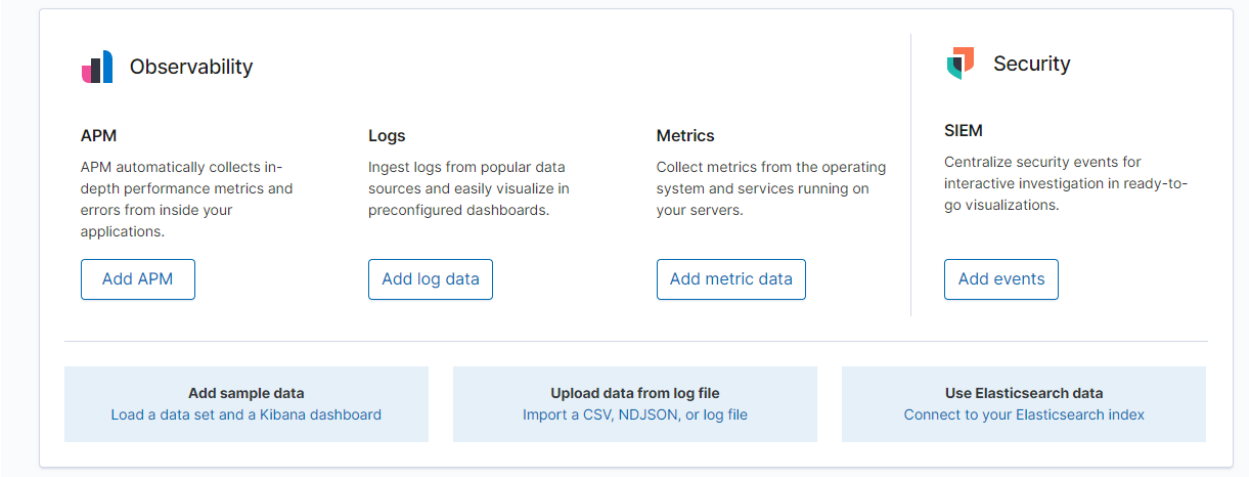
```
1 kubectl set env deployments/kibana-logging
   ↪  ELASTICSEARCH_HOSTS=http://10.103.89.17:9200
2 deployment.apps/kibana-logging env updated
```

Da K9s possiamo accedere ai log del pod di Kibana. Se tutto è andato a buon fine, un messaggio di successo ci notificherà che il server di Kibana è in esecuzione sulla porta 5601.

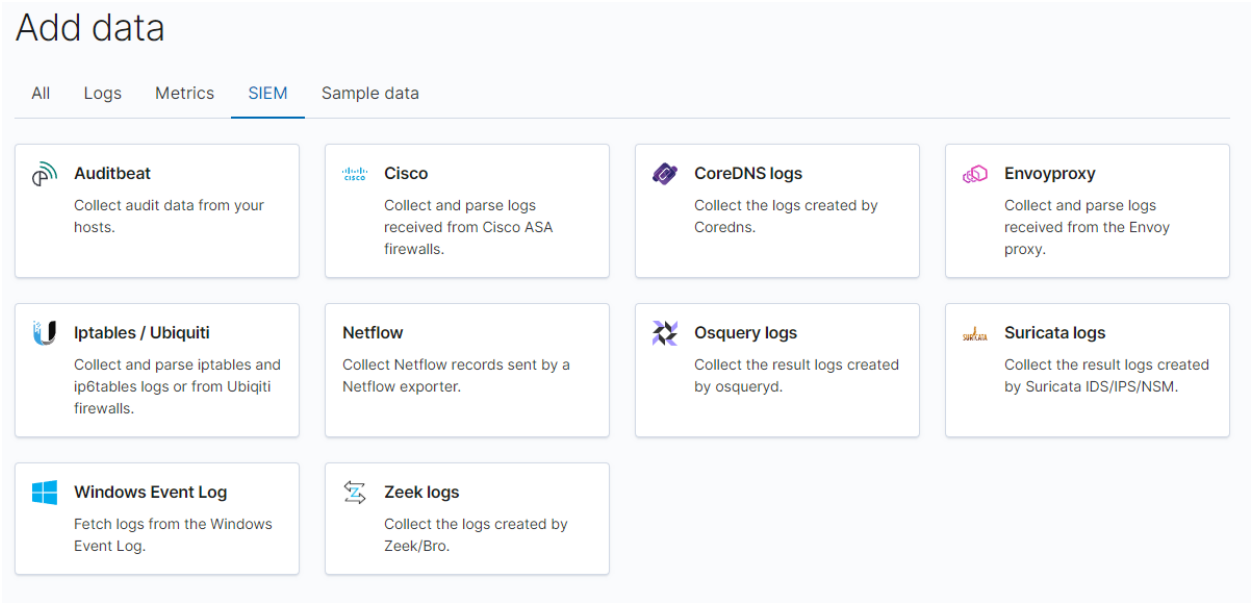
Infatti, se ci rechiamo su localhost:56699 dopo aver effettuato il port-fowarding tramite Minikube, otterremo il risultato auspicato:



Quello che vediamo è effettivamente definibile come un "Observability Center" che, con i moduli opportuni, può trasformarsi in un SIEM. Elastic è un servizio plug-and-play: basta attivare le integrazioni desiderate per raccogliere i dati che ci interessano. In particolare, sono disponibili le seguenti categorie di integrazioni:



Accedendo alla voce "SIEM" otteniamo più dettagli sulle integrazioni disponibili.



Come osserviamo, le integrazioni disponibili ad oggi riguardano prevalentemente sistemi enterprise, come Cisco (networking), Ibiquiti (firewalls), Suricata (IDS/IPS) e così via. Naturalmente, nulla vieta al developer di sviluppare una propria integrazione ad-hoc per le proprie esigenze.

Quindi, è stato opportunamente effettuato il bootstrap di un SIEM con Kubernetes, pur non attivando nuove integrazioni poiché quelle disponibili riguardano solo soluzioni enterprise, che richiedono l'attivazione di API Keys (in genere a pagamento).

Si aggiunge, infine, che l'autore ha provato ad attivare integrazioni alternative, come ad esempio Heartbeat per monitorare l'health status dei nodi, ma l'attivazione implica anche in questo caso API Keys o l'uso di credenziali particolari.

In genere, le aziende che intendono usare Kubernetes ed ELK come SIEM provvedono a creare un cluster su un cloud provider (es. AWS), attivare delle API Keys su Elastic (e con le rispettive integrazioni) e lavorano quindi in ambienti multi-cluster (un cluster per il development, un cluster per la produzione e così via) usando però cluster reali, non affidandosi a Minikube, Kind o altre soluzioni (gratuite, per uno studente) di local development.

7 Considerazioni conclusive

In termini compilativi, questo progetto ha permesso di esplorare più a fondo Kubernetes, i concetti dietro ai sistemi SIEM e SOAR, così come le possibili integrazioni fra Kubernetes ed Elastic (come collante per meccanismi di observability). In termini sperimentali, è stato condotto il bootstrap di un sistema SIEM (i.e. Elastic, Kibana) su Kubernetes, descrivendo in maniera del tutto riproducibile le operazioni svolte.

Le fasi successive al bootstrapping di Elastic e Kibana implicherebbero agganciare un certo numero di integrazioni al sistema: in questo caso, si parlerebbe comunque di integrazioni che dipenderebbero molto dalla configurazione di rete e dalla tipologia di applicativi che si dovrebbero monitorare, e che in ogni caso sarebbero blindati dietro soluzioni a pagamento, nella maggior parte dei casi.

L'autore ha anche ipotizzato l'utilizzo di Grafana e Prometheus come "alternative pienamente gratuite ed open-source", ma sono sistemi che si discostano troppo dal concetto di "SIEM" e quindi non avrebbe avuto molto senso perseguire questa strada.

Lo studio condotto, quindi, rappresenta molte delle possibili considerazioni iniziali che un addetto alla sicurezza dovrebbe porsi prima di mettere in campo un sistema di observability su Kubernetes, a partire da "cosa si sta facendo" (monitorare), "su quale piattaforma lo si sta facendo" (Kubernetes), "quali strumenti mi servono" (SIEM e SOAR), "come avvio un sistema del genere" (bootstrap SIEM), "quali sono i prossimi passi" (acquistare le integrazioni adatte).

In conclusione, si rimarca che sulla repository del progetto sono presenti tutti gli script YAML utilizzati, e che il materiale investigato per produrre questo documento è presente nella bibliografia.

8 Bibliografia e riferimenti

1. Documentazione di Kubernetes
2. Documentazione di Docker
3. Documentazione di ELK
4. Documentazione di Minikube
5. AWS - Elastic Kubernetes Service
6. OpenSearch Security Analytics
7. Google Security Operations
8. SGBox SIEM-SOAR
9. SearchInform SIEM
10. Shuffle Automation SOAR
11. CNCF - Automated Cloud-Native Incident Response with Kubernetes
12. Kubernetes Probes