

# Kubernetes Observability & Detection Engineering



## Candidato

Antonio Gravino (05225 01502)

## Repository

[github.com/antoniogriv/kube-observe](https://github.com/antoniogriv/kube-observe)

Corso di *Penetration Testing & Ethical Hacking*

Prof. Arcangelo Castiglione, a.a. 2023/2024



1

Ambienti reali  
e ambienti virtuali

2

Orchestrazione  
con Kubernetes

3

Incidenti e risposte



4

Sperimentazione  
esemplificativa

# Motivazioni legate al progetto



- Interesse personale che prescinde dal corso e che ha modo di intersecarsi con i temi trattati
- Kubernetes precedentemente affrontato per la tesi di laurea magistrale
  - **Governance MLOps: Orchestrazione di carichi di lavoro di Machine Learning con Kubernetes**
  - Relatori: prof. Christiancarmine Esposito, prof. Rocco Zaccagnino
  - [github.com/antoniogrvc/kube-gf](https://github.com/antoniogrvc/kube-gf)
- Analisi per lo più compilativa con sperimentazioni limitate anche dalla *freschezza* degli strumenti

Università degli Studi di Salerno  
Dipartimento di Informatica

---

Corso di Laurea Magistrale in Informatica

**Governance MLOps:**  
Orchestrazione sicura di carichi di  
Machine Learning con Kubernetes

---

**Relatori**  
Prof. Rocco Zaccagnino  
Prof. Christiancarmine Esposito

**Candidato**  
Antonio Gravino  
Matricola: 05225 01502

---

Anno Accademico 2023/2024

- Installazione del sistema
  - Dipendenze
  - Provisioning
- Eseguire la pipeline
  - Caricare le immagini Docker
    - Dataset Generation Docker Image
    - Model Training & Testing Docker Image
  - Compilare le pipeline
    - Compilazione con Docker
    - Compilazione con Miniconda

Q Go to file t

15d9f

Update diagrams

Sanitize CDK Kube conf.

Sanitize Dockerfiles: remove unused .gitkeep

Add GF monolith Dockerfile

Update README.md

kubernetes

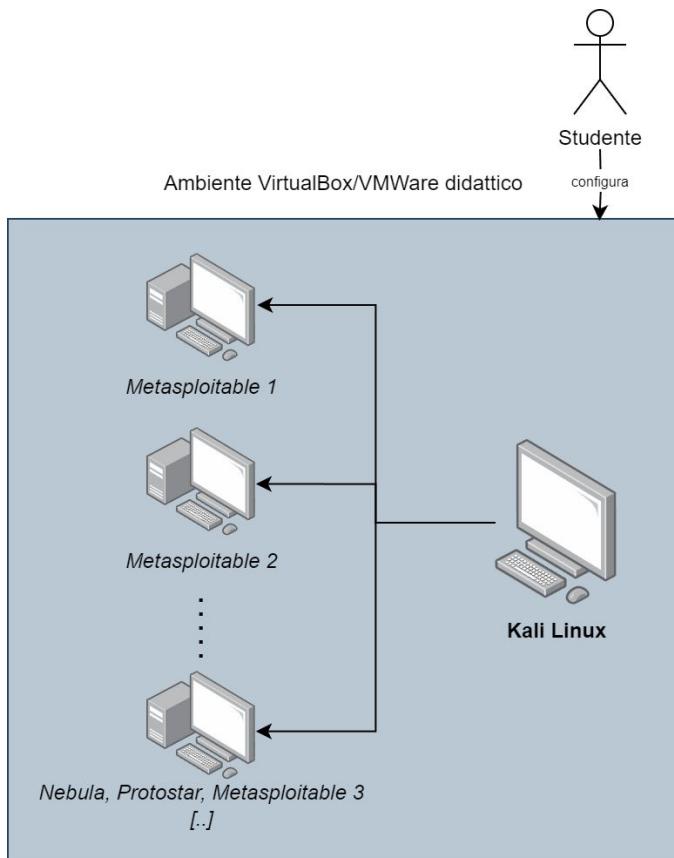
Fusion over Kubernetes

Ops, MLSecOps, Kubernetes, Kubeflow



- Gli aspetti pratici del corso sono stati affrontati in un **contesto controllato e di piccole dimensioni**, anche per favorire la didattica
- Il corso ci insegna, fra le altre cose, ad operare come agente Kali contro uno o più target vulnerabili all'interno di un ambiente virtualizzato
  - Tipicamente 1 VM Kali, e alcune VM vulnerable-by-design
    - Metasploitable 1, 2, 3 [...]
- Su alcune di queste macchine virtuali sono situati applicativi vulnerabili e applicativi non vulnerabili
  - Ad esempio, Metasploitable 2 ospita DVWA

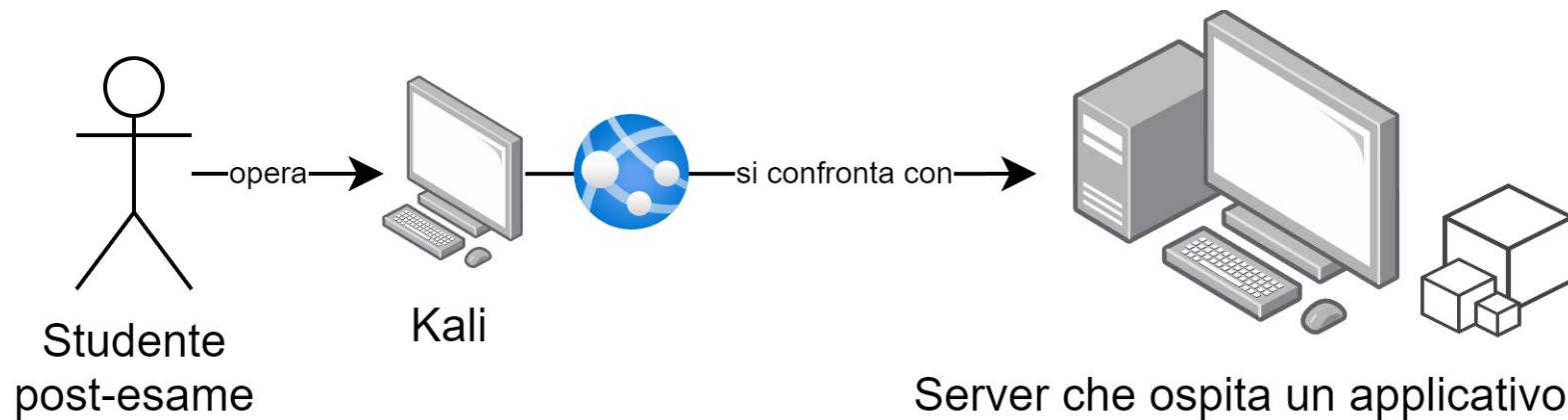
# Ambienti reali e ambienti virtuali



- Lo studente probabilmente studierà per l'esame utilizzando la seguente architettura di rete virtualizzata su VirtualBox o VMWare
- Efficace per i fini didattici del corso, riproduce un contesto controllato in cui sfruttare a pieno Kali contro target ben noti
- Rete NAT inclusiva della macchina virtuale di Kali e delle VM vulnerabili su cui addestrarsi

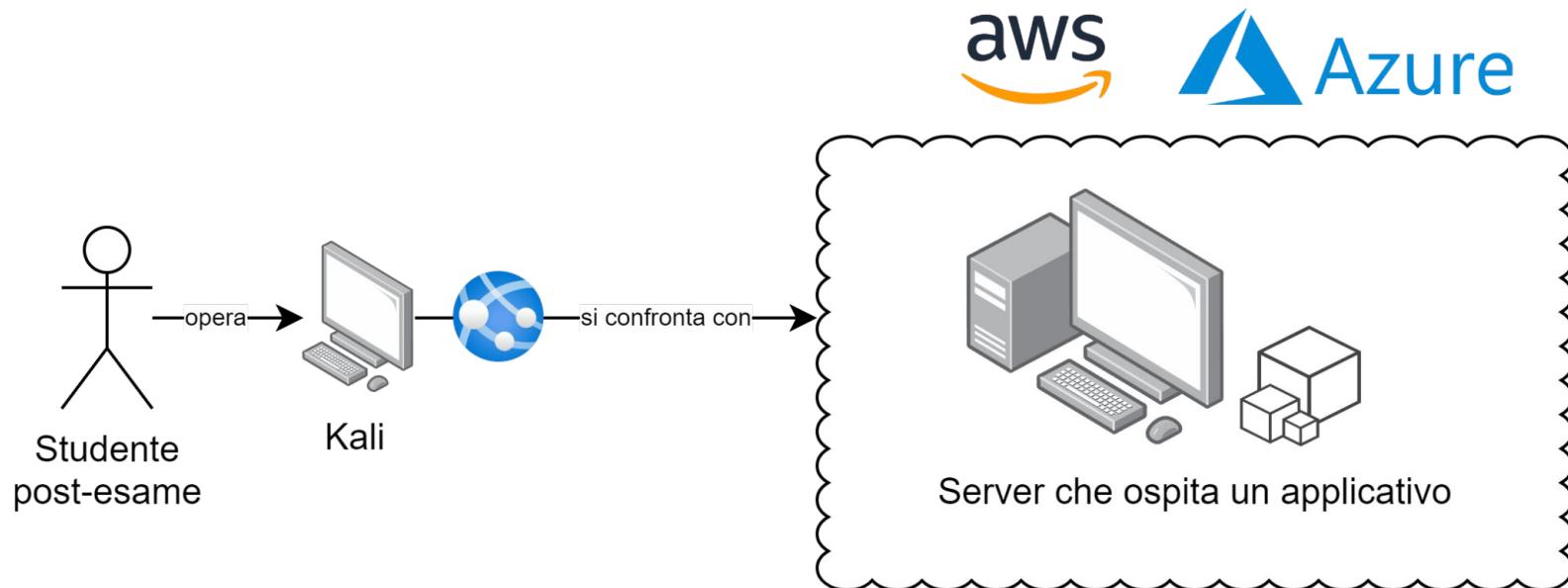


- Trasliamo l'ambiente di lavoro al “*mondo reale*” e tralasciamo i dettagli di networking. Lo studente si confronterà con *un* certo server che ospiterà un applicativo da analizzare secondo le metodologie studiate durante il corso.
- Quindi, qualcosa del genere?





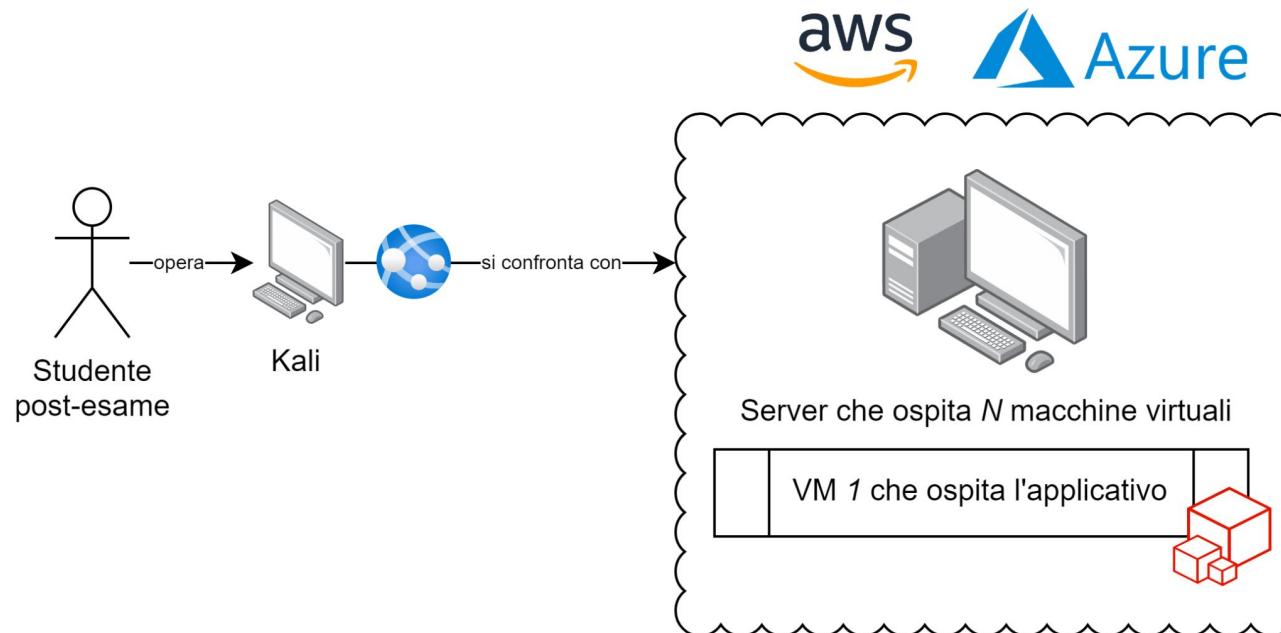
- Il server in questione è *plausibilmente* gestito da un **cloud provider** (es. AWS, Stati Uniti) o da un co-locator bare-metal (es. Hetzner, Germania)
- L'alternativa è una soluzione **on-premise in-house**, che comporta oneri aggiuntivi



# Ambienti reali e ambienti virtuali

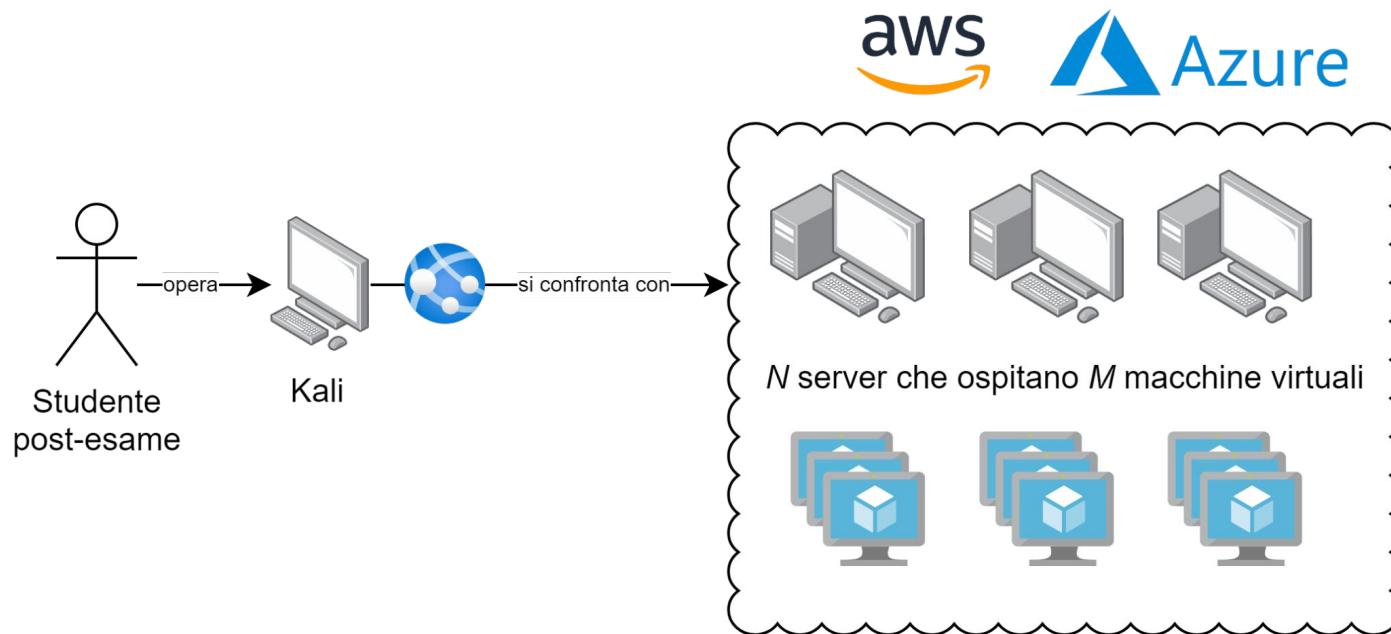


- Nella pratica, AWS ti disporrebbe uno *slice* del server nella forma di una macchina virtuale, in co-locazione virtuale con VM di altri clienti
- **L'applicativo giacerebbe sulla VM**, schermata dal resto del server



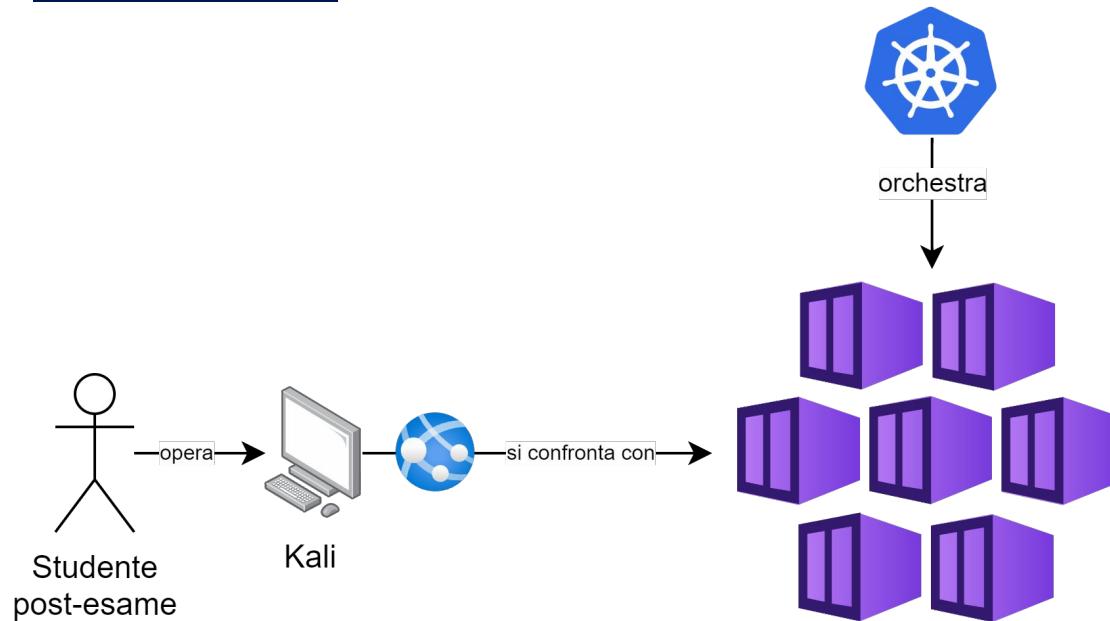


- Inoltre, il **carico di lavoro** di un tipico applicativo viene generalmente **distribuito orizzontalmente** su un **cluster di macchine virtuali**, ospitate su server gestiti dal provider



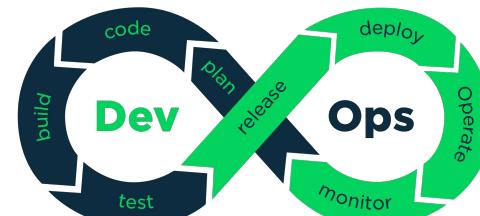


- La scalabilità e la governance di un cluster di VM è un tema complesso
- Richiede strumenti efficaci, specialmente se il cluster contempla **più** applicativi
- Parliamo di orchestrazione





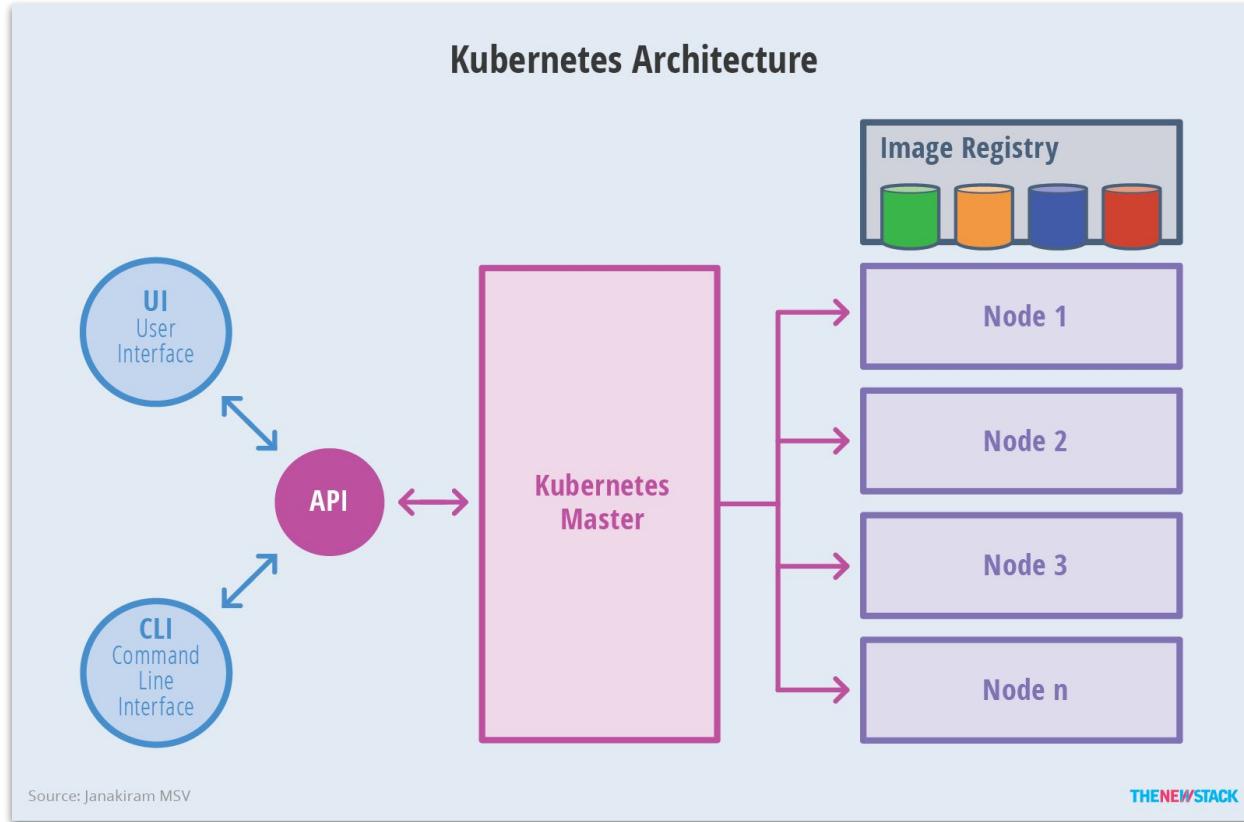
- Kubernetes è un software di orchestrazione e governance di cluster di macchine virtuali basato su container
- Creato da Google, fornisce supporto **multi-cloud** per realizzare **architetture scalabili, resilienti, affidabili e con zero-downtime**
- Tramite Kubernetes è possibile governare dinamicamente molteplici applicativi diversi, da **load balancers** ad **HTTP web servers**
- Si inserisce in una giovane branca dell'informatica legata al cloud-native
  - “**DevOps Engineering**”





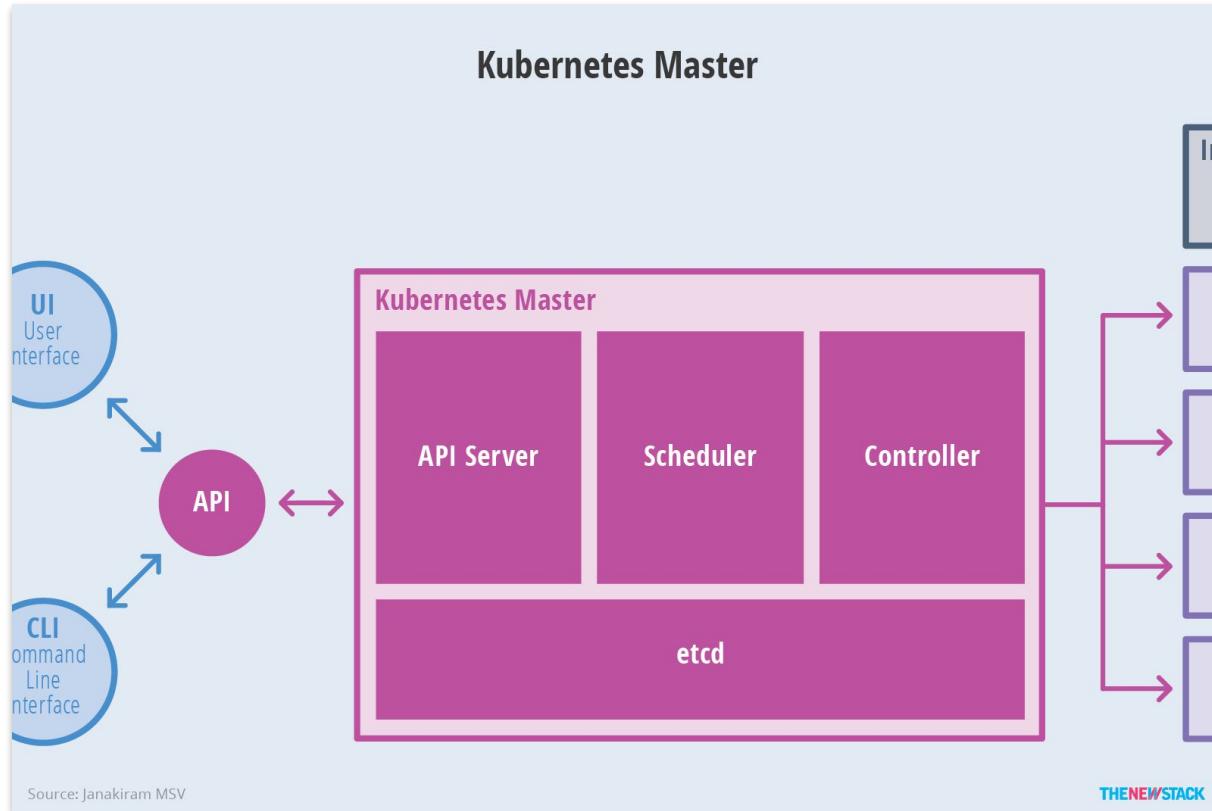
- La SEC su Kubernetes è un aspetto critico, ma è in “avvenire”
  - Software giovane, ma ciò non implica che è poco sicuro (IAM, Service Accounts, JWTs, ...)
- **I temi di Disaster Recovery e Incident Response su cluster gestiti da Kubernetes sono ancora molto aperti**, per quanto ci sia qualche euristica
- “*Non usare Kubernetes perché non è ancora pronto*”
  - **Le alternative a Kubernetes implicano eccessive operazioni manuali da parte dell'ingegnere** (che è error-prone)
  - In genere usato assieme a strumenti più maturi (es. Ansible)
  - Supportato da Google: difficile avere migliori garanzie

# Orchestrazione con Kubernetes



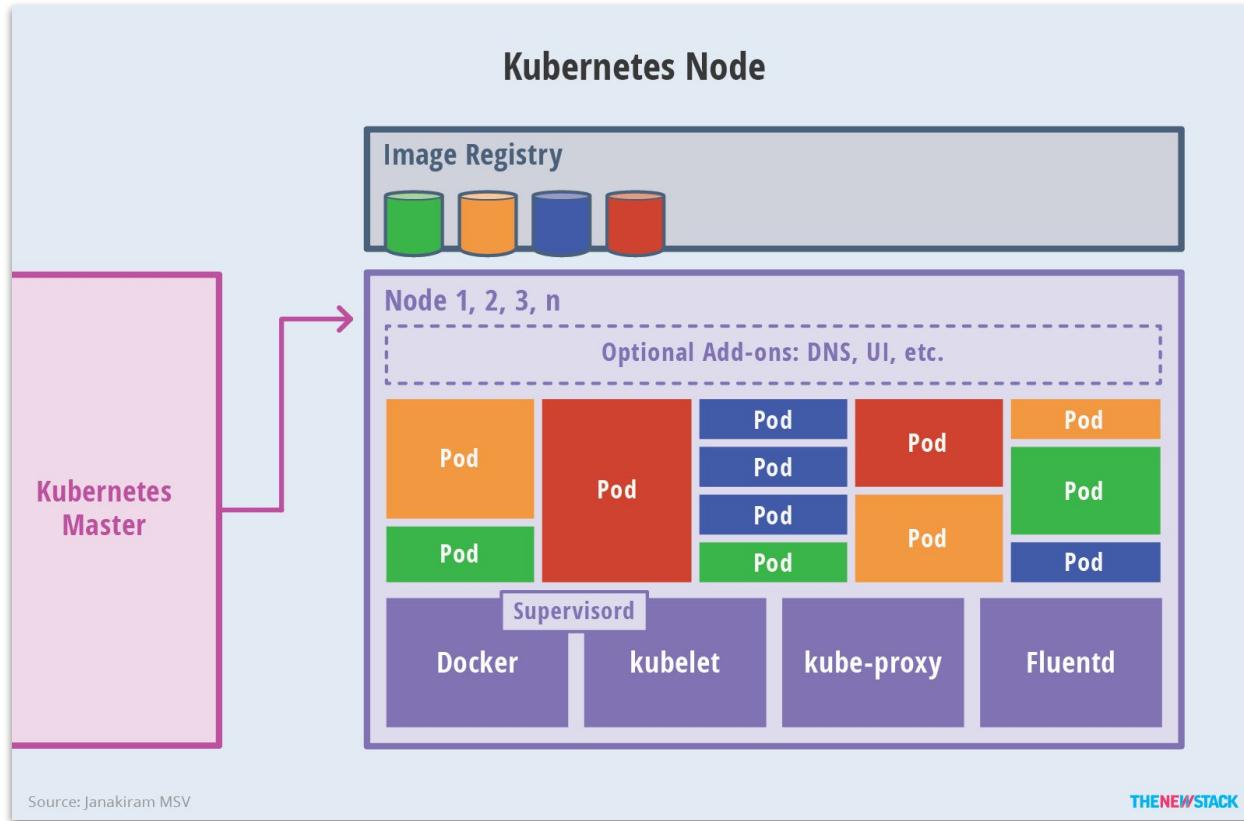
Fonte: diagrammi di Red Hat (Germania)

# Orchestrazione con Kubernetes



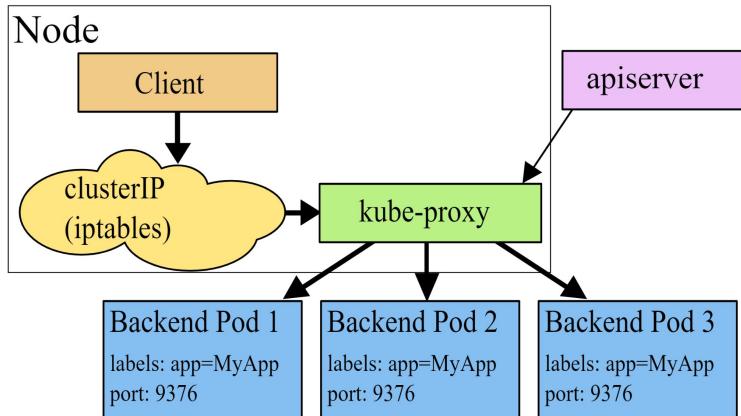
Fonte: diagrammi di Red Hat (Germania)

# Orchestrazione con Kubernetes



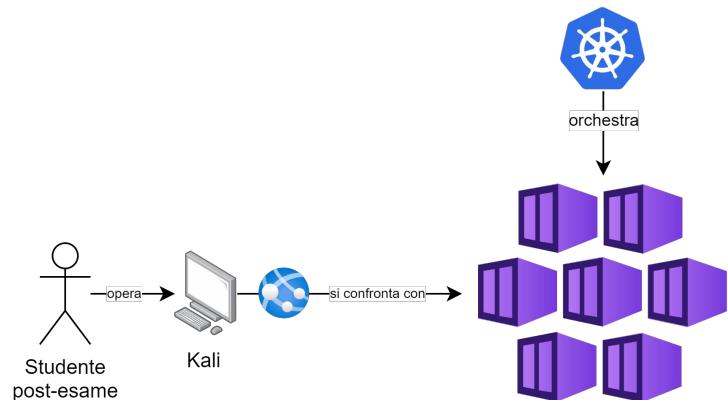
Fonte: diagrammi di Red Hat (Germania)

# Orchestrazione con Kubernetes



- L'utente si interfaccia con un proxy (es. un Load Balancer ala-Nginx)
- Il proxy bilancia il carico trasmettendo le richieste HTTP a **N pod** in funzione del servizio richiesto
  - service1.ilmiobackend.it
  - service2.ilmiobackend.it
- Ogni pod ospita **M container**. L'utente è ridirezionato su un certo pod solo se esso ospita il container richiesto (i.e. l'applicazione).
- Gli **N pod** sono situati su **K macchine virtuali**, a loro volta localizzati su **Z server**, possibilmente in **J aree geografiche** diverse (es. Milano)
- Esempi di container sono un qualsiasi webserver, un'applicazione Python, e così via

# Orchestrazione con Kubernetes



- Da Kali, attaccare un cluster Kubernetes in realtà significa interfacciarsi con un **kube-proxy** (es. Nginx), il quale ridirezionerebbe le richieste (se legittime) ai pod
- Dinamiche possono variare in base alla mise-en-place degli elementi nel cluster
- Al di là della configurazione del proxy, devo tener conto che **il pod potrebbe contenere un container vulnerabile**
  - Il proxy espone le applicazioni su Internet
  - Vulnerabilità individuabili come di consueto con Nessus, OpenVAS, nmap/exploitdb...



Tipicamente, quando parliamo di sicurezza (anche su Kubernetes) utilizziamo un vocabolario ben definito

- **IoC - Indicatore di Compromesso** (es. conseguenze di payload, syscalls...)
  - Durante il corso abbiamo visto diversi payload da iniettare su vettori di vario tipo (exploit)
- **SOC (Security Operations Center)** - Un luogo fisico che ospita analisti
- **SIEM (Security Information and Event Management)**
  - Generalmente una dashboard che raccoglie segnalazioni/alerts/eventi
- **SOAR (Security Orchestration, Automation, Response)**
  - Un *engine* di qualche tipo regolato da “*playbooks*”, ie. protocolli da seguire in caso si verifichi un incidente (segnalato dal SIEM)

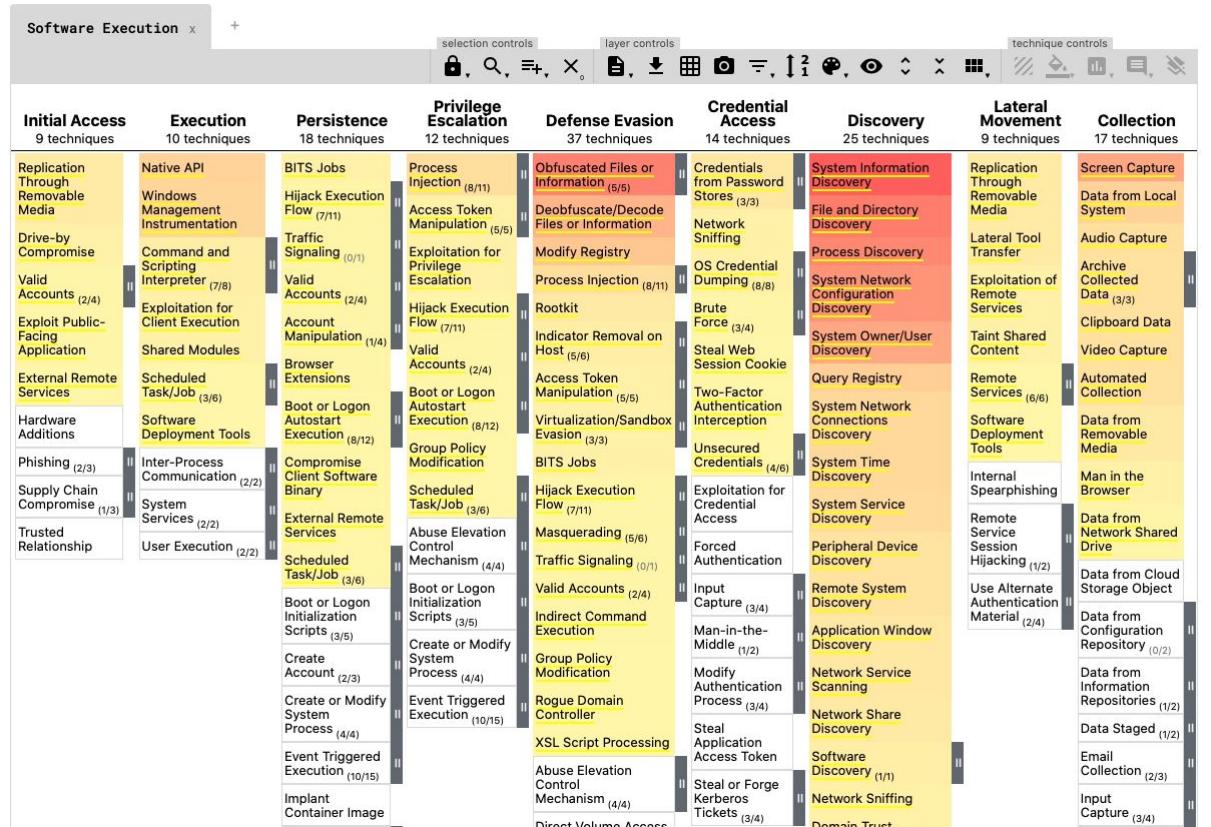


- Esistono vari framework per gestire gli incidenti di sicurezza. Ci rifacciamo al modello più popolare.
- **NIST Incident Response**
  - Step 1/2: **Detection**, Analysis
    - Identifica il payload tramite gli indicatori di compromesso
  - Step 3: **Containment**, Eradication, Recovery
    - Isola il payload, evita che si propaghi nel resto del cluster, distruggi il pod infetto, aggiungi regole al firewall, etc.
    - Ripristina l'ammontare corretto di pod
  - Step 4: Post-Incident Activity
    - Capire le origini dell'incidente



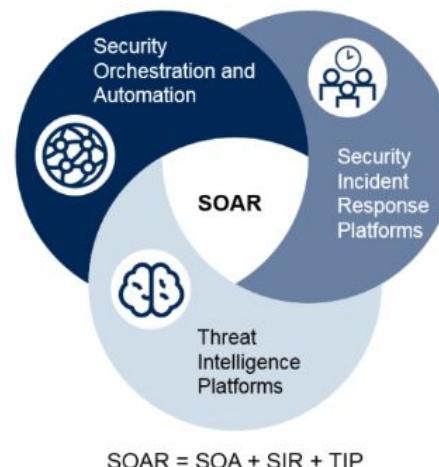
- Si tratta però di un approccio reattivo, mentre una salda SEC si basa su **approcci proattivi**.
- In questo contesto, è fondamentale assicurarsi di avere una vasta banca dati di *Indicatori di Compromesso* di qualità.
- Com'è implementabile? Non ci sono regole ben definite, ma fra gli standard è possibile individuare il **MITRE ATT&CK** (basato su "matrici")
  - **Collezione di tecniche che permettono l'individuazione** e il contenimento di queste vulnerabilità
  - Ma anche una **collezione di metodologie di attacco** ad ampio spettro
    - Informazioni con cui possiamo "nutrire" (feeding) il SOAR
    - Altre "knowledge bases" sono, ad esempio, i CVE

# Incidenti e Risposte





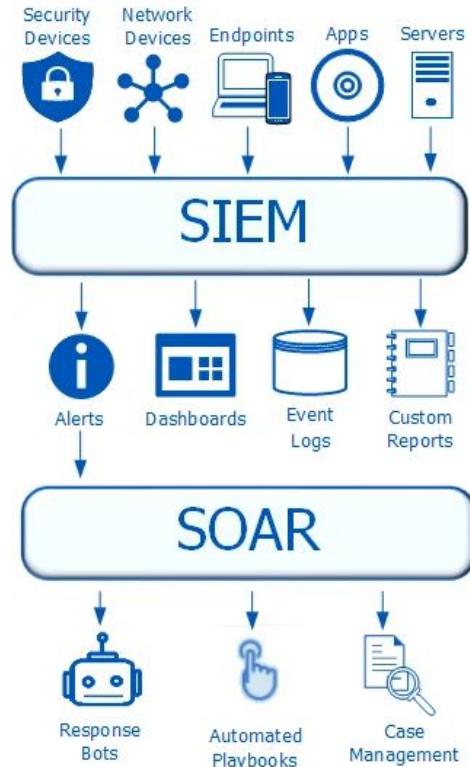
- **Il SOAR è un engine (un software) posizionabile in un pod che riceve feedback dal SIEM e interviene in caso di necessità.** In genere costituito da diversi software interconnessi.
  - Su Kubernetes, il SOAR spesso costituisce un namespace a sé
  - In generale, realizzabile con vari software open-source ed enterprise



# Incidenti e Risposte



- **Il SOAR ha bisogno di accesso totale all'infrastruttura** per operare efficacemente e autonomamente
  - Cosa fa? Una volta notificato dal SIEM, opera in autonomia senza che un ingegnere debba condurre la mitigazione
  - Perché non basta l'ingegnere? Banalmente, tempi di reazione
- **E' quindi imprescindibile che il SOAR abbia contezza del cluster, e che il cluster sia osservato in maniera limpida e integra**
  - Attaccare i vettori di observability per "ingannare" il SOAR?





- Nell'ambito dell'Incident Response, **Kubernetes offre alcuni vantaggi** nativamente
  - Rescheduling/Application Recovery
  - Custom Operators per estendere le capacità di osservazione
    - Esempio di Operator: Grafana, Prometheus...
  - Container effimeri
  - Checkpointing

Con Kubernetes il “**Recovery**” è automatico

- L'API Server osserva costantemente il database etcd e, in caso di pod fallace, provvede a rinnovarlo. Possibile tecnica: rimuovo un pod, e attendo che Kubernetes lo rigeneri

Anche il “**Contenimento**” è automatizzabile

- Kubernetes implementa delle “sonde”, lanciate ogni tot secondi che verificano l'integrità di un pod. Appena la sonda fallisce, posso distruggere il pod e ricrearlo.



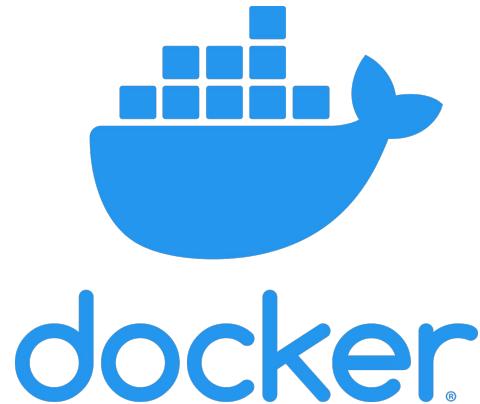
- In maniera *barebones*, le sonde di Kubernetes potrebbero notificare lo stato di un pod (indicante o meno irregolarità) e trasmetterlo ad un sistema SIEM
- A quel punto, il SIEM analizzerebbe lo stato del pod, e istruirebbe il SOAR sul da farsi
- Il SOAR opererebbe immediatamente sul pod (o i pod).
  - Un intervento potrebbe essere banale come eliminare il pod `kubectl delete pod ${pod}`
  - O più complesso ed effettuare una procedura di restart, negare alcuni permessi a certi pod o eliminare i Service Accounts (pur di non eliminare i pod infetti). Ha senso quando la rigenerazione di un pod è onerosa.
- Nella pratica, i classici SIEM/SOAR non vengono usati, a favore di “Operatori” (es. Istio) di Kubernetes che però svolgono un ruolo analogo



- Obiettivo sperimentale: effettuare il setup di un SIEM con **Elastic**, **Kibana** e **Docker**.
  - Realizzazione di un SOAR molto più complessa, implicherebbe *Istio* e le *Kubernetes Service Meshes*.



elastic





- Più nello specifico, si utilizzerà **Minikube** (su WSL) per **realizzare in locale un cluster Kubernetes mono-nodo che ospiterà il SEAM**.
  - Necessarie diverse pre-configurazioni per tarare adeguatamente l'hardware virtuale

```
1 [wsl2]
2 memory=10GB
3
```

A screenshot of a terminal window titled '.wslconfig'. The window contains three lines of configuration: line 1 sets the distribution to 'wsl2', line 2 specifies a memory allocation of '10GB', and line 3 is empty.

- Tramite la CLI di “**minikube**” possiamo creare il cluster, da interrogare tramite “**kubectl**”.
- Elastic richiede specificamente un minimo di 8gb di RAM. Poiché **stiamo “creando un nodo”** (i.e. una finta VM) **sotto container Docker**, dobbiamo tarare il file **.wslconfig**.



## Non funzionante

```
~> minikube start
W0503 10:25:00.468436    1892 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=2, Memory=4000MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.0.1 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

## Funzionante

```
~> 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 “Contexts” per permettere la gestione di cluster multipli.
- Creando un cluster con Minikube, viene settato nuovo un context locale. Come possiamo vedere, il Control Plane (API Server) è in esecuzione sulla porta 56326

```
> kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:56326
CoreDNS is running at https://127.0.0.1:56326/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

- Possiamo visualizzare con più contezza il cluster usando **strumenti open source** ad-hoc come **K9s**.

# Sperimentazione esemplificativa



K9s ci mostra i “contesti” attualmente disponibili.

The screenshot shows the K9s application interface. At the top, there are two tabs: "Prompt dei comandi" and "Prompt dei comandi - k9s". The "Prompt dei comandi - k9s" tab is active. On the left, there is a sidebar with system information:

Context:	n/a
Cluster:	n/a
User:	n/a
K9s Rev:	v0.32.4
K8s Rev:	n/a
CPU:	n/a
MEM:	n/a

On the right, there is a help menu:

- <?> Help
- <r> Rename

In the center, there is a tree diagram representing the Kubernetes cluster structure. Below it, a pink bar displays the message "Plugins load failed!".

Below the sidebar, a table titled "Contexts(all)[1]" is displayed:

NAME↑	CLUSTER	AUTHINFO	NAMESPACE
docker-desktop	docker-desktop	docker-desktop	

At the bottom left, there is a yellow button labeled "<contexts>".

# Sperimentazione esemplificativa



## Nuovo contesto locale generato da Minikube

The screenshot shows a terminal window with two tabs: "Prompt dei comandi" and "Prompt dei comandi - k9s". The "Prompt dei comandi" tab displays the following output:

```
Context: minikube
Cluster: minikube
User: minikube
K9s Rev: v0.32.4
K8s Rev: v1.30.0
CPU: n/a
MEM: n/a
```

The "Prompt dei comandi - k9s" tab shows a table titled "Contexts(all)[2]". The table has columns: NAME, CLUSTER, AUTHINFO, and NAMESPACE. It lists two contexts:

NAME	CLUSTER	AUTHINFO	NAMESPACE
docker-desktop	docker-desktop	docker-desktop	
minikube(*)	minikube	minikube	default

At the bottom of the terminal window, there is a yellow bar with the text "<contexts>".

# Sperimentazione esemplificativa



Pod attualmente in esecuzione. Di default, Kubernetes genera alcuni pod ausiliari.

```
Context: minikube
Cluster: minikube
User: minikube
K9s Rev: v0.32.4
K8s Rev: v1.30.0
CPU: n/a
MEM: n/a
```

<0> all <a> Attach <l> ...
<1> default <ctrl-d> Delete <p> ...
<d> Describe <shift-f> ...
<e> Edit <z> ...
<?> Help <s> ...
<ctrl-k> Kill <o> ...

NAMESPACE	NAME	PF	READY	STATUS	RESTARTS	IP	NODE	AGE
kube-system	coredns-7db6d8ff4d-jb57p	●	1/1	Running	0	10.244.0.2	minikube	39s
kube-system	etcd-minikube	●	1/1	Running	0	192.168.49.2	minikube	53s
kube-system	kube-apiserver-minikube	●	1/1	Running	0	192.168.49.2	minikube	53s
kube-system	kube-controller-manager-minikube	●	1/1	Running	0	192.168.49.2	minikube	53s
kube-system	kube-proxy-rlzbk	●	1/1	Running	0	192.168.49.2	minikube	39s
kube-system	kube-scheduler-minikube	●	1/1	Running	0	192.168.49.2	minikube	53s
kube-system	storage-provisioner	●	1/1	Running	1	192.168.49.2	minikube	52s

<pod>

# Sperimentazione esemplificativa



Procediamo al deploy di Elastic. Fondamentalmente, “deployare” qualcosa su Kubernetes implica “applicare un manifesto”. In pratica, viene effettuata una POST verso l’API Server, il quale imporrà agli “addetti ai lavori” (Controller, Scheduler) di generare nuovi “oggetti Kubernetes” (es. Pods)

```
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
```

```
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: apps/v1
kind: Deployment
metadata:
  labels:
    app: es-logging
  name: es-logging
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: es-logging
```

# Sperimentazione esemplificativa



Tramite K9s e kubectl osserviamo il corretto funzionamento dei nuovi oggetti creati

NAMESPACE↑	NAME	Pods(all)[8]						AGE
		PF	READY	STATUS	RESTARTS	IP	NODE	
default	es-logging-798b8bb6f7-hwprm	●	1/1	Running	0	10.244.0.3	minikube	108s
kube-system	coredns-7db6d8ff4d-ddfmk	●	1/1	Running	0	10.244.0.2	minikube	6m27s
kube-system	etcd-minikube	●	1/1	Running	0	192.168.49.2	minikube	6m43s
kube-system	kube-apiserver-minikube	●	1/1	Running	0	192.168.49.2	minikube	6m43s
kube-system	kube-controller-manager-minikube	●	1/1	Running	0	192.168.49.2	minikube	6m43s
kube-system	kube-proxy-2mzr9	●	1/1	Running	0	192.168.49.2	minikube	6m28s
kube-system	kube-scheduler-minikube	●	1/1	Running	0	192.168.49.2	minikube	6m44s
kube-system	storage-provisioner	●	1/1	Running	1	192.168.49.2	minikube	6m41s

```
> kubectl describe deployment es-logging
Name:           es-logging
Namespace:      default
CreationTimestamp: Fri, 03 May 2024 10:47:58 +0200
Labels:          app=es-logging
Annotations:    deployment.kubernetes.io/revision: 1
Selector:        app=es-logging
Replicas:       1 desired | 1 updated | 1 total | 0 available
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=es-logging
  Containers:
    elasticsearch:
      Image:      elasticsearch:7.8.0
      Ports:      9200/TCP, 9300/TCP
      Host Ports: 0/TCP, 0/TCP
      Liveness:   http-get http://:http/_cluster/health delay=40s timeout=40s
      Readiness:  http-get http://:http/_cluster/health delay=30s timeout=30s
```



Osserviamo che il “deployment” di Kubernetes effettua il binding di specifiche porte (9200, 9300/TCP). Una volta esposte all'esterno, saranno soggetti ad eventuali enumerazioni da parte di attaccanti.

```
> kubectl describe deployment es-logging
Name:           es-logging
Namespace:      default
CreationTimestamp:   Fri, 03 May 2024 10:47:58 +0200
Labels:          app=es-logging
Annotations:    deployment.kubernetes.io/revision: 1
Selector:        app=es-logging
Replicas:       1 desired | 1 updated | 1 total | 0 available
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=es-logging
  Containers:
    elasticsearch:
      Image:      elasticsearch:7.8.0
      Ports:      9200/TCP, 9300/TCP
      Host Ports: 0/TCP, 0/TCP
      Liveness:   http-get http://:http/_cluster/health delay=40s timeout=5s
      Readiness:  http-get http://:http/_cluster/health delay=30s timeout=5s
```

# Sperimentazione esemplificativa



In particolare, gli oggetti Kubernetes che “espongono servizi” sono detti “Services/Ingress”.

```
> kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
es-service	NodePort	10.103.89.17	<none>	9200:30716/TCP	15s
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	7m27s

```
> minikube service es-service
W0503 10:50:52.311283    8860 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.
-----|-----|-----|-----|
| NAMESPACE |     NAME      | TARGET PORT |          URL           |
-----|-----|-----|-----|
| default   | es-service  |         9200  | http://192.168.49.2:30716 |
-----|-----|-----|-----|
* Starting tunnel for service es-service.
-----|-----|-----|-----|
| NAMESPACE |     NAME      | TARGET PORT |          URL           |
-----|-----|-----|-----|
| default   | es-service  |             | http://127.0.0.1:56525 |
-----|-----|-----|-----|
* Opening service default/es-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```



Una volta effettuato il port-forwarding (fra porta del container - 9200 - e una porta disponibile su localhost - come la 56526, tipica di Elastic) possiamo verificare il funzionamento del software.

```
curl 127.0.0.1:56526
Formatta il codice 
{
  "name": "es-logging-798b8bb6f7-hwprm",
  "cluster_name": "docker-cluster",
  "cluster_uuid": "0U6gf05nSjKoluxKVdxQhQ",
  "version": {
    "number": "7.8.0",
    "build_flavor": "default",
    "build_type": "docker",
    "build_hash": "757314695644ea9a1dc2fecfd26d1a43856725e65",
    "build_date": "2020-06-14T19:35:50.234439Z",
    "build_snapshot": false,
    "lucene_version": "8.5.1",
    "minimum_wire_compatibility_version": "6.8.0",
    "minimum_index_compatibility_version": "6.0.0-beta1"
  },
  "tagline": "You Know, for Search"
}
```

# Sperimentazione esemplificativa



A questo punto, possiamo procedere con il deployment di Kibana, il quale sfrutterà Elastic per offrire servizi di monitoring. In sostanza, Kibana è il frontend ed Elastic è il backend.

NAMESPACE	NAME	PF	READY	STATUS	RESTARTS	IP	NODE	AGE
default	es-logging-798b8bb6f7-hwprm	●	1/1	Running	0	10.244.0.3	minikube	7m1s
default	kibana-logging-6c5dc7df8f-nxwvs	●	0/1	ContainerCreating	0	n/a	minikube	24s
kube-system	coredns-7db6d8ff4d-ddfmk	●	1/1	Running	0	10.244.0.2	minikube	11m
kube-system	etcd-minikube	●	1/1	Running	0	192.168.49.2	minikube	11m
kube-system	kube-apiserver-minikube	●	1/1	Running	0	192.168.49.2	minikube	11m
kube-system	kube-controller-manager-minikube	●	1/1	Running	0	192.168.49.2	minikube	11m
kube-system	kube-proxy-2mrz9	●	1/1	Running	0	192.168.49.2	minikube	11m
kube-system	kube-scheduler-minikube	●	1/1	Running	0	192.168.49.2	minikube	11m
kube-system	storage-provisioner	●	1/1	Running	1	192.168.49.2	minikube	11m

```
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
```

# Sperimentazione esemplificativa



Per funzionare, il frontend (Kibana) deve sapere come comunicare col backend (Elastic). Per farlo, tramite kubectl è possibile impostare manualmente delle variabili d'ambiente all'interno del container (non del pod!)

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

```
Logs(default/kibana-logging-bbfc5d599-d6gdk:kibana)[tail] -----  
Autoscroll:On      FullScreen:Off      Timestamps:Off      Wrap:Off  
:01:49Z", "tags": ["warning", "plugins", "reporting"], "pid": 7, "message": "Chromium sandbox provides an additional layer o  
:01:49Z", "tags": ["warning", "reporting"], "pid": 7, "message": "Enabling the Chromium sandbox provides an additional laye  
:01:50Z", "tags": ["status", "plugin:reporting@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from un  
:01:50Z", "tags": ["status", "plugin:spaces@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from unin  
:01:50Z", "tags": ["status", "plugin:security@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from uni  
:01:50Z", "tags": ["status", "plugin:dashboard_mode@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed fr  
:01:50Z", "tags": ["status", "plugin:beats_management@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed  
:01:50Z", "tags": ["status", "plugin:maps@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from uniniti  
:01:50Z", "tags": ["info", "plugins", "taskManager", "taskManager"], "pid": 7, "message": "TaskManager is identified by the K  
:01:50Z", "tags": ["status", "plugin:task_manager@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from  
:01:50Z", "tags": ["status", "plugin:encryptedSavedObjects@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status cha  
:01:50Z", "tags": ["status", "plugin:apm_oss@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from unin  
:01:50Z", "tags": ["status", "plugin:console_legacy@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed fr  
:01:50Z", "tags": ["status", "plugin:region_map@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from u  
:01:50Z", "tags": ["status", "plugin:ui_metric@7.8.0", "info"], "pid": 7, "state": "green", "message": "Status changed from un  
:01:50Z", "tags": ["listening", "info"], "pid": 7, "message": "Server running at http://0:5601"}  
:01:51Z", "tags": ["info", "http", "server", "Kibana"], "pid": 7, "message": "http server running at http://0:5601"}
```



A questo punto effettuiamo il port-forwarding di Kibana.

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	

```
> minikube service kibana-service
W0503 11:02:41.845172 20048 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.
-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
-----|-----|-----|-----|
| default | kibana-service | 5601 | http://192.168.49.2:30769 |
-----|-----|-----|-----|
* Starting tunnel for service kibana-service.
-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
-----|-----|-----|-----|
| default | kibana-service | | http://127.0.0.1:56699 |
-----|-----|-----|-----|
* Opening service default/kibana-service in default browser...
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

# Sperimentazione esemplificativa: risultati



The screenshot shows the Kibana home page at the URL 127.0.0.1:5669/app/kibana#/home. The page is divided into several sections:

- Observability** section:
  - APM**: APM automatically collects in-depth performance metrics and errors from inside your applications. Buttons: [Add APM](#).
  - Logs**: Ingest logs from popular data sources and easily visualize in preconfigured dashboards. Buttons: [Add log data](#).
  - Metrics**: Collect metrics from the operating system and services running on your servers. Buttons: [Add metric data](#).
  - SIEM**: Centralize security events for interactive investigation in ready-to-go visualizations. Button: [Add events](#).
- Add sample data**: Load a data set and a Kibana dashboard.
- Upload data from log file**: Import a CSV, NDJSON, or log file.
- Use Elasticsearch data**: Connect to your Elasticsearch index.
- Visualize and Explore Data** section:
  - APM**: Automatically collect in-depth performance metrics and errors from inside your applications.
  - Canvas**: Showcase your data in a pixel-perfect way.
  - Dashboard**: Display and share a collection of visualizations and saved searches.
  - Discover**: Interactively explore your data by querying and filtering raw documents.
  - Graph**: Surface and analyze relevant relationships in your Elasticsearch data.
  - Logs**: Stream logs in real time or scroll through historical views in a console-like experience.
  - Machine Learning**: Automatically model the normal behavior of your time series data to detect anomalies.
  - Maps**: Explore geospatial data from Elasticsearch and the Elastic Maps Service.
  - Metrics**: Explore infrastructure metrics and logs for
  - SIEM**: Explore security metrics and logs for events and
- Manage and Administer the Elastic Stack** section:
  - Console**: Skip CURL and use this JSON interface to work with your data directly.
  - Rollups**: Summarize and store historical data in a smaller index for future analysis.
  - Saved Objects**: Import, export, and manage your saved searches, visualizations, and dashboards.
  - Security Settings**: Protect your data and easily manage who has access to what with users and roles.
  - Spaces**: Organize your dashboards and other saved objects into meaningful categories.
  - Stack Monitoring**: Track the real-time health and performance of your Elastic Stack.
  - Transforms**: Use transforms to pivot existing Elasticsearch indices into summarized or entity-centric indices.

# Sperimentazione esemplificativa: risultati



## Observability

**APM**  
APM automatically collects in-depth performance metrics and errors from inside your applications.

**Logs**  
Ingest logs from popular data sources and easily visualize in preconfigured dashboards.

**Metrics**  
Collect metrics from the operating system and services running on your servers.

[Add APM](#) [Add log data](#) [Add metric data](#)

---

[Add sample data](#)  
Load a data set and a Kibana dashboard

[Upload data from log file](#)  
Import a CSV, NDJSON, or log file

[Use Elasticsearch data](#)  
Connect to your Elasticsearch index

## Security

**SIEM**  
Centralize security events for interactive investigation in ready-to-go visualizations.

[Add events](#)

# Sperimentazione esemplificativa: risultati



## Add data

All   Logs   Metrics   **SIEM**   Sample data

 **Auditbeat**  
Collect audit data from your hosts.

 **Cisco**  
Collect and parse logs received from Cisco ASA firewalls.

 **CoreDNS logs**  
Collect the logs created by Coredns.

 **Envoyproxy**  
Collect and parse logs received from the Envoy proxy.

 **Iptables / Ubiquiti**  
Collect and parse iptables and ip6tables logs or from Ubiquiti firewalls.

 **Netflow**  
Collect Netflow records sent by a Netflow exporter.

 **Osquery logs**  
Collect the result logs created by osqueryd.

 **Suricata logs**  
Collect the result logs created by Suricata IDS/IPS/NSM.

 **Windows Event Log**  
Fetch logs from the Windows Event Log.

 **Zeek logs**  
Collect the logs created by Zeek/Bro.



- **Il setup è completo:** il centro di monitoring (che funge anche da SIEM) è pronto ed *up-and-running* su Kubernetes.
- A questo punto, è sufficiente attivare le integrazioni che ci interessano.
  - Esempio: Integrazione **CISCO** per il collezionamento degli eventi dei **firewall**
  - Esempio: Integrazione **CoreDNS** per il collezionamento dei rispettivi **logs**
  - Esempio: Integrazione **Suricata** per il collezionamento di eventi dagli **IDS**
- In un contesto industriale, questo processo sarebbe più articolato e verrebbero fatte opportune considerazioni per la messa in produzione.
  - Esempio: impostare un **account RBAC per Kibana**, affinché possa comunicare solo col pod di Elastic. Ora come ora, Kibana può comunicare con tutti i pod del cluster.

# Sperimentazione esemplificativa: risultati



- Idealmente, il progetto avrebbe dovuto prendere una deriva del tipo “provare ad usare una delle integrazioni sottostanti”. Il problema? Sono tutte **enterprise** e richiedono **API Keys per essere usate**.

Add data

All   Logs   Metrics   **SIEM**   Sample data

 Auditbeat

Collect audit data from your hosts.

 Cisco

Collect and parse logs received from Cisco ASA firewalls.

 CoreDNS logs

Collect the logs created by Coredns.

 Envoyproxy

Collect and parse logs received from the Envoy proxy.

 Iptables / Ubiquiti

Collect and parse iptables and ip6tables logs or from Ubiquiti firewalls.

 Netflow

Collect Netflow records sent by a Netflow exporter.

 Osquery logs

Collect the result logs created by osqueryd.

 Suricata logs

Collect the result logs created by Suricata IDS/IPS/NSM.

 Windows Event Log

Fetch logs from the Windows Event Log.

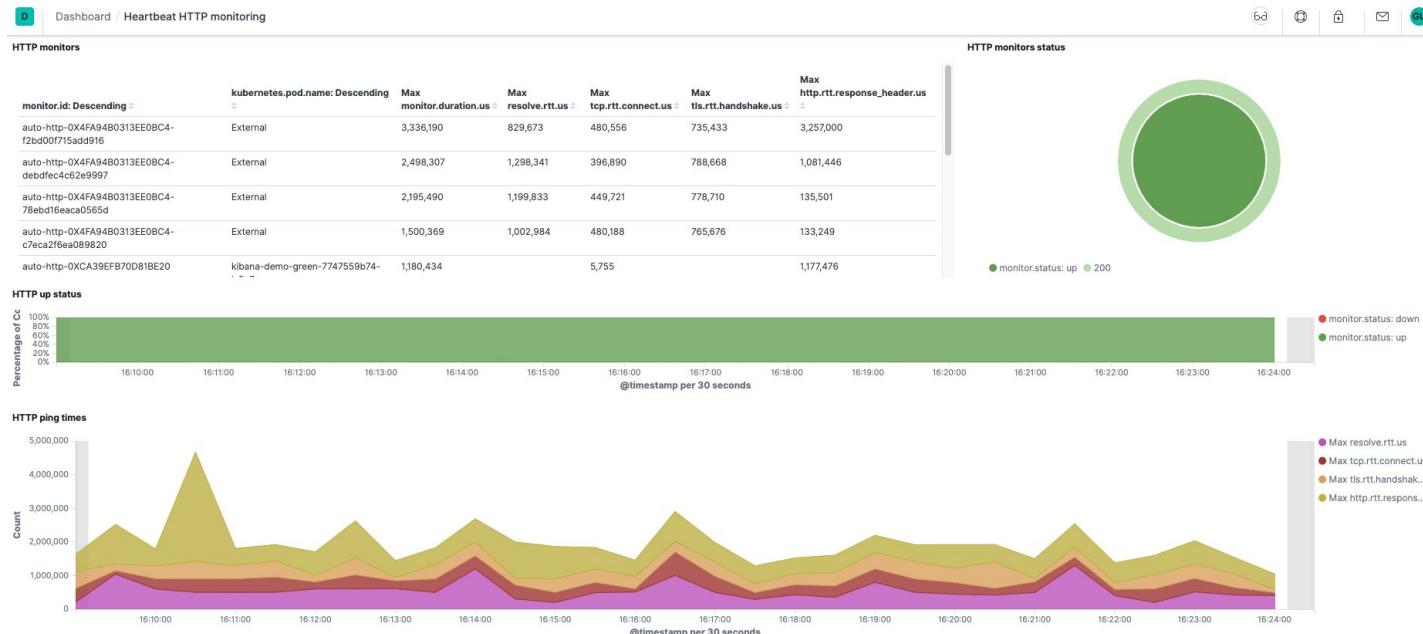
 Zeek logs

Collect the logs created by Zeek/Bro.

# Sperimentazione esemplificativa: risultati



- Quindi, ho pensato di provare ad integrare un servizio non-enterprise, cioè Heartbeat, una integrazione di HTTP Monitoring.



# Sperimentazione esemplificativa: risultati



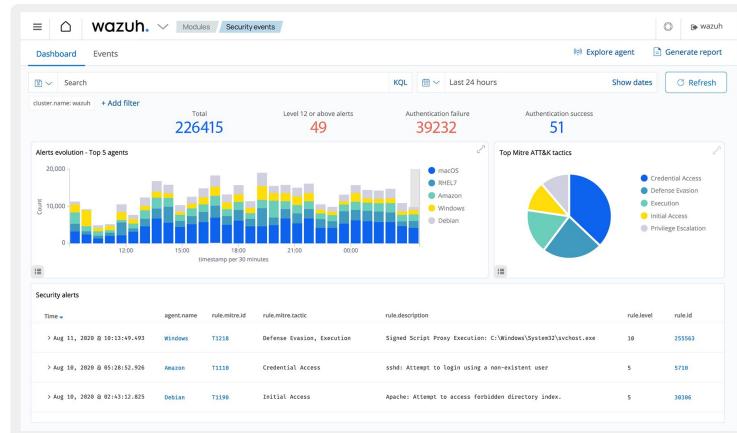
- **Il problema? Neanche queste integrazioni sono utilizzabili.**
  - Per qualsiasi integrazione è necessario generare una coppia `username:password` da dare in pasto alle integrazioni affinché riescano a parlare con Elastic.
- La versione locale di Elastic è disponibile solo in “*modalità anonima*”. Bisogna usare credenziali reali.
  - L’idea è che il developer dell’azienda usi delle credenziali reali anche in un contesto di development.
  - **In generale, molte aziende che lavorano con Kubernetes NON sviluppano in “locale”, ma usano dei cluster (reali, concreti) di “staging”.** Esempio: GARR

The screenshot shows a blog post from the elastic blog. The title is "Getting started with Elastic Cloud on Kubernetes: Deployment". It's dated 04 SEPTEMBER 2019 and categorized under "HOW TO". The author is Adam Quan. There are links for "Solutions", "Stack + Cloud", "News", "Customers", "Generative AI", "Culture", and a "Contact Sales" button. Below the title, there's a large image of interlocking gears and a steering wheel, symbolizing the integration of different technologies.





- **Elastic è, de facto, lo standard (open source) per il monitoring su Kubernetes, ma è ostico da implementare senza un cluster reale.**
  - Un developer che sta studiando lo strumento potrebbe utilizzare i cluster in free-trial su AWS... ma implica comunque indicare la propria carta di credito
- Alternative? *Wazuh* (open source), ma non sembra maturo come Elastic.
  - In ogni caso, le principali integrazioni plug-and-play sono disponibili su Elastic.





# Grazie per l'attenzione!