# Stack Tecnologico SOC: Guida Dettagliata

*Daniele Speziale - Responsabile IT*

*Febbraio 2025*

## 1. CORE: OpenSearch + OpenSearch Dashboards

### *OpenSearch*

OpenSearch è un motore di ricerca e analisi open-source derivato da Elasticsearch e Kibana. Nel contesto di un SOC, funge da repository centralizzato per tutti i dati di sicurezza.

**Caratteristiche Principali:**

OpenSearch è una piattaforma distribuita e scalabile in grado di gestire petabyte di dati. Supporta ricerche full-text ad alta velocità usando indici invertiti e tecnologie di caching avanzate. Offre API RESTful per ingestion, ricerca e analisi. Ha supporto nativo per machine learning, anomaly detection e forecasting. Fornisce plugin per sicurezza (encryption, autenticazione, RBAC), audit logging completo, e compressione dati intelligente per ridurre storage footprint.

**Architettura:**

Un cluster OpenSearch consiste di nodi di diverso tipo: nodi data (memorizzano indici e shard), nodi master (gestiscono lo stato del cluster), nodi ingest (pre-processano i documenti), nodi coordinating (smistano le richieste). Tipicamente un deployment aziendale ha 3+ nodi master (per alta disponibilità), 5+ nodi data (per volume e performance), e nodi ingest dedicati per heavy lifting di parsing/enrichment.

**Ingestion e Parsing:**

I dati entrano in OpenSearch principalmente tramite

Logstash (processing engine potente),

Beats (agenti leggeri), o API dirette.

Logstash legge da sorgenti diverse, le trasforma tramite filter plugin (grok per parsing, mutate per modifica campi, geo-ip per enrichment geografico,

lookup per correlazioni), e le scrive in OpenSearch con output plugin.

Un tipico pipeline Logstash per firewall log: INPUT (legge syslog), FILTER (estrae source IP, destination IP, action via regex grok, arricchisce con geolocation, normalizza timestamp), OUTPUT (scrive in indice OpenSearch).

**Indexing Strategy:**

I dati vengono organizzati in indici. La best practice è creare indici per sorgente e tempo: **firewall-logs-2025.10.17**, **windows-events-2025.10.17**, **network-flows-2025.10.17**. Questo facilita ricerche rapide, retention policies, e pulizia dati. Usa Index Lifecycle Management (ILM) per automatizzare transizioni: dati hot (0-7 giorni, su SSD veloce), warm (7-30 giorni, compressione), cold (30+ giorni, storage economico), delete (dopo retention policy). Ogni indice ha configurazione di sharding: numero di shard determina parallelismo (maggiori shard = ricerche più veloci ma più overhead), replica determina disponibilità (solitamente 2 repliche per production).

**Query Capabilities:**

OpenSearch supporta Query DSL (Domain Specific Language) per ricerche complesse. Esempi: **GET /firewall-*/_search** con body DSL per cercare tutti i firewall log, filtrare per action="drop", aggregare per source_ip per identificare attacker patterns. Range query per time range, term query per esatto match, match_phrase per frasi, bool query per combinare logica complessa (must/should/must_not). Aggregations permettono calcoli: terms aggregation per top 10 IP, date histogram per trend temporale, stats aggregation per metriche (min/max/avg).

Esempio: index=main sourcetype=webserver-logs | stats count as requests

**Performance Tuning:**

Ottimizza performance tramite: refresh interval (quanto spesso i nuovi dati diventano searchable), segment merging (combinazione di segmenti per ridurre file), shard allocation awareness (distribuire shard su diversi data center per resilienza), query optimization (specificare index pattern, ridurre date range). Monitora JVM memory, disk I/O, network bandwidth. Un cluster ben configurato risponde a query complesse su miliardi di log in secondi.

## *OpenSearch Dashboards*

OpenSearch Dashboards è l'interfaccia di visualizzazione e analisi. È il "volto" del SOC.

**Componenti Principali:**

Discover: navigazione interattiva dei dati. Seleziona un indice, visualizza i campi disponibili, filtra e esplora. Un analista cerca "login failures" nei Windows events, vede lista dettagliata con timestamp, username, failure reason, source IP, e può drill-down per investigare.

Visualizations: grafici e metriche. Creazione di chart di diverso tipo: line chart per trend temporali (numero di login failures nel tempo), bar chart per comparazione (top 10 attacker IP), pie chart per distribuzione, heatmap per correlazione due dimensioni (ore del giorno vs giorno della settimana

per anomalia rilevamento), gauges per metriche critical (current threat level).

Dashboards: combinazione di visualizzazioni in un singolo schermo. Un SOC ha dashboard per: Overview generale (numero di alert, event count, top threat), Firewall Analysis (traffic patterns, blocked domains), Endpoint Security (malware detected, unusual processes), Login Activity (failed logins, privilege escalations), Network Flow Analysis (top talkers, unusual bandwidth consumers), User Activity (file access patterns, lateral movement indicators).

**Interattività e Drill-Down:**

I dashboard di OpenSearch Dashboards sono interattivi. Un click su un barra in un chart filtra il resto del dashboard per quel valore. Crea un dashboard che mostra "malware detections by endpoint". Clicca su un endpoint → tutta la visualizzazione si filtra per mostrare solo data di quel device, puoi drill-down nella Discover view per vedere ogni evento.

**Alerting:**

Configura alert rules direttamente in Dashboards. Esempio: "se count di 'event_id: 4625' (failed login) nei ultimi 5 minuti > 20, invia alert". Gli alert possono triggerare webhook, email, Slack, PagerDuty, webhook generici per integrazione con sistemi esterni.

**Canvas e Reporting:**

Canvas permette creazione di visual narratives custom: infografica, report styled, executive summary visualizzata in tempo reale. Reporting automatico: genera PDF/PNG di dashboard, invialidosi via email a stakeholder su schedule (settimanale, giornaliero).

**Role-Based Access Control (RBAC):**

Definisci ruoli per team diversi. SOC Manager vede tutto, SOC Analyst vede solo la propria area di responsabilità, Management vede solo dashboard executive, Compliance vede solo audit-related data. L'accesso ai dati è controllato a livello di indice, documento, e persino campo.

# 2. ML: OpenSearch ML (Built-in) + Custom Python Scripts

## *OpenSearch ML Built-in*

OpenSearch include moduli di machine learning per security-specific use cases.

**Anomaly Detection:**

Job di anomaly detection monitorano indici e identificano deviazioni dal comportamento normale. Il sistema usa algoritmi como Random Cut Forest (RCF), Isolation Forest, e Robust Principal Component Analysis (RPCA).

Come funziona: definisci un job che monitora l'indice "windows-events" cercando anomalie nel numero di "event_id: 4625" (failed login) aggregati per "source_ip" ogni 5 minuti. Il sistema apprende il comportamento normale (distribuzione temporale, valori attesi, variabilità), quindi ogni 5 minuti calcola un anomaly score (0-100) per ogni source_ip. Score 0 = perfettamente normale, score 100 = altamente anomalo. Se uno IP normalmente genera 2-5 failed login al minuto ma improvvisamente genera 50, score sale a 95.

Configurazione pratica: crei il job tramite OpenSearch Dashboards UI o API. Specifici: indice da monitorare, metrica aggregata (count, avg, sum di quale campo), granularità temporale (5 min, 1 hour), finestra di training (quanto tempo prima di iniziare a rilevare). Sistema addestra il modello su 2-4 settimane di dati storici, poi monitora continuamente.

Output: ogni periodo temporale, il job produce anomaly score per ogni entità (se aggregato per source_ip, uno score per IP). Risultati scritti in indice separato che puoi interrogare/visualizzare.

Casi d'uso comuni: login failures (brute force detection), network traffic volume anomaly, process execution frequency, database query patterns, file access frequency.

**Forecasting:**

Job di forecasting prevedono trend futuri basati su pattern storici. Usa algoritmi tipo ARIMA (AutoRegressive Integrated Moving Average), exponential smoothing.

Applicazione: monitora l'indice di network flow, aggreghi bandwidth totale ogni ora. Il sistema analizza trend: traffico più alto di giorno, più basso di notte, picco il lunedì. Genera forecast per le prossime 24-168 ore (1-7 giorni). Se il forecast predice 5 Mbps di traffico ma domani salgono a 50 Mbps senza motivo, è anomalia rilevante.

Utilità: capacità planning (se trending di storage log cresce, quando server è pieno?), detection di anomalie future (se baseline è X ma forecast dice Y molto diverso, indaga).

**Feature Engineering:**

OpenSearch può calcolare feature (trasformazioni di dati) utili per ML. Esempio: dato un timestamp, calcolaz giorno della settimana, ora del giorno, è weekend, è holiday. Dato un IP, lookup geolocalizzazione, reputation score da threat intel. Dato un username, lookup department, job role, critico o no. Queste feature vengono aggiunte a ogni documento, arricchendo i dati.

Nel Dashboards UI, definisci feature trasformazioni in Data Prep. Sistema applica automaticamente a nuovi dati ingestion.

## *Custom Python ML Scripts*

Per use case più sofisticati, integra custom Python script con librerie ML.

**Architettura di Integrazione:**

Opzione 1: Script Python standalone che legge da OpenSearch tramite python-opensearch library, esegue analisi, scrive risultati back a OpenSearch. Script esegue periodicamente via cron/scheduler esterno.

Opzione 2: Script eseguiti come parte di Logstash pipeline tramite filter plugin "ruby" o "exec", per processing real-time durante ingestion.

Opzione 3: OpenSearch ML Commons plugin permette registrazione e execution di modelli Python direttamente in cluster.

**Libraries Comuni:**

Scikit-learn: classification (Isolation Forest, Random Forest), clustering (K-means, DBSCAN), regression. Pandas: data manipulation, aggregazioni, feature engineering. NumPy: operazioni numeriche. Scipy: statistiche avanzate. TensorFlow/PyTorch: deep learning (reti neurali per pattern complessi). XGBoost: gradient boosting machine (molto accurato per classification/regression). LightGBM: alternative leggera a XGBoost, più veloce su grandi dataset.

**Esempio Implementativo: User Behavior Anomaly Detection**

## *Code: python*

```
from opensearchpy import OpenSearch
import pandas as pd
from sklearn.ensemble import IsolationForest
from datetime import datetime, timedelta

# Connessione
client = OpenSearch(['localhost:9200'])

# Estrai ultimo mese di login events
query = {
    "bool": {
        "must": [
            {"term": {"event_id": 4624}},  # Successful login
            {"range": {"@timestamp": {"gte": "now-30d"}}}
        ]
    }
}

# Paginated search per estrarre tutti i dati
scroll_size = 10000
page = client.search(index="windows-events-*", body={"query": query, "size": scroll_size})
hits = page['hits']['hits']
sid = page['_scroll_id']

all_data = []
while len(hits) > 0:
    all_data.extend(hits)
    page = client.scroll(scroll_id=sid, scroll='2m')
    hits = page['hits']['hits']

# Converti in DataFrame
df = pd.DataFrame([hit['_source'] for hit in all_data])

# Feature engineering
df['hour'] = pd.to_datetime(df['@timestamp']).dt.hour
df['day_of_week'] = pd.to_datetime(df['@timestamp']).dt.dayofweek
df['is_weekend'] = df['day_of_week'].isin([5, 6]).astype(int)

# Aggregate per user
user_features = df.groupby('user').agg({
    'source_ip': 'nunique',  # Numero di IP diversi
    'workstation_name': 'nunique',  # Numero di device diversi
    'hour': lambda x: len(x) / len(x.unique()),  # Average login per hour
    'is_weekend': 'mean'  # Fraction di weekend login
}).reset_index()

# Applica Isolation Forest
model = IsolationForest(contamination=0.1)  # Assume 10% anomalie
user_features['anomaly_score'] = model.fit_predict(user_features[['source_ip', 'workstation_name', 'hour',
user_features['anomaly_prob'] = 1 / (1 + model.score_samples(user_features[['source_ip', 'workstation_nam

# Scrivi risultati back a OpenSearch
for idx, row in user_features.iterrows():
    doc = {
        'user': row['user'],
        'anomaly_score': row['anomaly_score'],
        'anomaly_probability': row['anomaly_prob'],
        'num_different_ips': row['source_ip'],
        'num_different_devices': row['workstation_name'],
        'analysis_timestamp': datetime.now().isoformat()
    }
    client.index(index='user-anomaly-analysis', body=doc)
```

Questo script estrae login events dell'ultimo mese, calcola feature, applica Isolation Forest (algoritmo ML che identifica outlier), scrive i risultati in indice "user-anomaly-analysis". Un dashboard vizualizza i risultati, il SOC vede quali user hanno comportamento anomalo.

**Scheduling:**

Esegui script Python tramite: Airflow (orchestration framework, gestisci dependency e retry), cron job (semplice, esegui periodicamente), Kubernetes CronJob (per scalabilità), OpenSearch ML Jobs framework se possibile.

# 3. NLP: OpenAI API vs Open-Source (Llama)

## *OpenAI API (Closed-source, Commercial)*

OpenAI fornisce API REST per accedere ai modelli GPT (GPT-4, GPT-4 Turbo, GPT-3.5 Turbo).

**Vantaggi:**

Modelli altamente accurati e versatili, gestiti da OpenAI (niente infrastructure da mantenere), supportano vari task: classification, summarization, extraction, question-answering, code generation. Aggiornamenti modelli automatici, OpenAI investe continuamente in miglioramenti. Rate limiting e quotas gestiti in background.

**Svantaggi:**

Costi di subscription, calcoli per numero di token (una parola ≈ 1-2 token). Per SOC con migliaia di log, costi possono salire rapidamente. Dati inviati a server OpenAI (compliance/privacy concern in alcuni settori regolati). Latenza network (pochi secondi per risposta).

**Use Cases in SOC:**

1. Log Summarization: dai 10 log file di una investigation, genera summary esecutivo in 2 frasi. Prompt: "Summarize these security logs into 2 sentences highlighting the key threat: [log content]"

2. Threat Classification: classifica automaticamente alert. Dai un alert generico (es., "High CPU usage detected on server X"), GPT classifica se è: malware behavior, cryptomining, legitimate spike, misconfiguration. Prompt: "Classify this security alert: [alert]. Categories: [list]. Return only category name."

3. Incident Timeline Reconstruction: estrai sequenza di event da log non strutturati. Input: chaotic log messages, output: "Step 1: attacker scanned ports at 10:05, Step 2: exploitation attempt at 10:12, Step 3: lateral movement at 10:45". Prompt: "Extract the attack sequence from these logs: [logs]. Format as numbered list of steps with timestamps."

4. Remediation Recommendations: dato incident, suggeri passi remediation. Input: "Attacker accessed database and dumped user table", Output: "1) Change database credentials, 2) Audit access logs for other breaches, 3) Notify users of compromise". Prompt: "Given this security incident: [description], provide 5 remediation steps."

5. Natural Language Query: SOC analyst chiede "What IPs connected to our critical database last night?", API converte in OpenSearch DSL query, esegue.

**Implementation:**

## *Code: python*

```
import openai
import json

openai.api_key = "your-api-key"

# Summarize security logs
logs_content = """
Oct 17 10:05 - Source IP 192.168.1.100 scanned ports 22,80,443,3306 on target 10.0.0.50
Oct 17 10:12 - SSH brute force attempt from 192.168.1.100 to 10.0.0.50
Oct 17 10:45 - Successful SSH login from 192.168.1.100 to 10.0.0.50 with user 'admin'
Oct 17 11:30 - Large file transfer detected from 10.0.0.50 to external IP 1.2.3.4
"""

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {
            "role": "user",
            "content": f"Summarize this security incident: {logs_content}. Format as 3 bullet points."
        }
    ],
    temperature=0.7,
    max_tokens=200
)

summary = response['choices'][0]['message']['content']
print(summary)
# Output: "• SSH brute force attack from 192.168.1.100 • Attacker gained admin access • Exfiltrated data
```

Invoca API, GPT analizza log, ritorna summary. Scrivi result in OpenSearch, visualizza in dashboard.

## *Open-Source: Llama*

Llama è famiglia di large language model open-source da Meta.

**Versioni:**

Llama 2.xx (generazione precedente, 7B/13B/70B parametri, libero per commercial use). Llama 3 (generazione nuova, upgraded performance, 8B/70B). Llama 3.1 (version stabile attuale).

**Vantaggi:**

Completamente open-source, nessun costo, esecuzione locale (rispetto privacy e compliance), modelli versatili come GPT, comunità attiva.

**Svantaggi:**

Require GPU potente (Llama 70B richiede 2+ GPU A100 or 4 GPU RTX 4090), latenza di inference più alta che OpenAI, setup infrastructure, model meno polished che GPT-4.

**Running Llama Locally:**

Opzione 1: Ollama (semplifica deployment). Installa Ollama, esegui **ollama run llama2**, model scarica automatico e espone API locale.

### *Code: bash*

```
# Install Ollama (macOS/Linux/Windows)
# Then:
ollama run llama2:7b

# Model è accessibile via http://localhost:11434/api/generate
```

Opzione 2: vLLM (framework di inference ad alta performance). Setup Python, carica model, servì via FastAPI.

### *Code: python*

```
from vllm import LLM, SamplingParams

# Carica model
llm = LLM(model="meta-llama/Llama-2-7b-hf", gpu_memory_utilization=0.8)

# Inference
prompts = [
    "Summarize this attack: attacker used SQL injection to access user database",
    "What's the recommended mitigation for CVE-2024-1234?"
]

sampling_params = SamplingParams(temperature=0.7, top_p=0.9, max_tokens=200)
outputs = llm.generate(prompts, sampling_params)

for output in outputs:
    print(output.outputs[0].text)
```

## 4. SOAR: Splunk Phantom, Azure Sentinel

SOAR (Security Orchestration, Automation, and Response) è piattaforma che automatizza response a security incidents.

### *Splunk Phantom (ora Splunk SOAR)*

Phantom è SOAR enterprise da Splunk.

**Architettura:**

Phantom consiste di: User interface web per creazione playbook e monitoring, Database backend (PostgreSQL) che memorizza playbook, credentials, run history, Automation engine (Python-based) che esegue playbook, Connector framework che integra con sistemi external (OpenSearch, firewall, endpoint tools, etc).

**Playbook:**

Un playbook è workflow di automation. GUI drag-and-drop per creare step sequenziali. Ogni step è: decision point (if/then/else logic), action (esegui tool esterno), data transformation (manipula data con Python).

Esempio playbook di risposta a "malware detected on endpoint":

```
1. Trigger: Alert arrives da OpenSearch (malware detection)
2. Decision: Severity > high?
   - YES → proceed
   - NO → log and close
3. Action: Query OpenSearch per file hash + other indicators
4. Action: Lookup file hash in VirusTotal API
5. Decision: Confirmed malware?
   - YES → proceed
   - NO → investigate manual
6. Action: Isolate endpoint (esegui endpoint isolation via CrowdStrike API)
7. Action: Kill process (esegui command via EDR agent)
8. Action: Quarantine file (muovi file su isolated storage)
9. Action: Revoke user credentials (API call a AD/Okta)
10. Action: Notify security team (Slack message con summary)
11. Action: Create ticket (ServiceNow incident)
12. Action: Log incident to SIEM (scrivi event in OpenSearch)
```

Phantom esegue sequentially, con error handling e retry logic. Se step 6 (isolate endpoint) fallisce, playbook non procede ai step 7-8 (kill process, quarantine) per evitare inconsistency.

**Connectors:**

Phantom ha 500+ connectors pre-built per sistemi comuni. Connectors sono plugin che implementano API call a sistema esterno.

Connectors rilevanti per SOC: OpenSearch Connector (query, index data), Palo Alto Networks (firewall management), Crowdstrike (endpoint isolation), Splunk Connector (query Splunk), ServiceNow (ticket creation), Slack (notifications), Office365 (user/mailbox management), AWS/Azure/GCP (cloud resource management), VirusTotal (threat intel), MISP (threat intel), Netskope (cloud security), Proofpoint (email security).

**Data Flow:**

Un alert da OpenSearch triggerato da anomaly detection → webhook notification a Phantom → Phantom riceve alert, estrae metadata, passa a playbook → playbook esegue automaticamente

→ results loggati back a OpenSearch → dashboard mostra playbook execution history.

### *Azure Sentinel (Microsoft SIEM + SOAR)*

Azure Sentinel è SIEM da Microsoft che include SOAR capability.

**Architettura:**

Azure Sentinel è cloud-native SIEM, raccoglie dati da Office365, Azure, on-premises (via agent). Ha detection rules built-in, automation (chiamato "Automation rules" e "Playbooks"), query language KQL (simile a SQL), investigation graph per visualizzare relazione tra entities.

**Playbooks in Sentinel:**

Playbook in Sentinel sono basati su Logic Apps (Microsoft workflow engine). Creazione via GUI, trigger su alert o incident, azioni supportate: send email, create Teams message, query Log Analytics, invoke API webhook.

Esempio: quando detector rulein Sentinel trova login sospetto, triggera playbook che: estrae user, query Azure AD per activity recente, se anomalia confermata, invia escalation a SOC manager tramite email e Teams.

**Automation Rules:**

Sentinel ha "Automation Rules" che combinano detection rule + playbook. Rule fires → automation rule evaluates condition → triggerations playbook se condition met. Semplifica orchestrazione.

## 5. Threat Intelligence: MISP e AlienVault OTX

Threat Intelligence (TI) è informazione su minacce: indicatori (IP malicious, domain, file hash), tactics/techniques (MITRE ATT&CK), vulnerabilità (CVE), threat group profiles.

### *MISP (Malware Information Sharing Platform)*

MISP è piattaforma open-source per sharing e storage di threat intelligence.

**Componenti:**

Database centrale (MySQL/PostgreSQL) memorizza events (ogni event è set di attributes che descrivono una minaccia). Attributes sono: IP address, domain, URL, email, file hash (MD5/SHA1/SHA256), filename, regkey (Windows registry key), YARA rule, ecc. Tagging system:

tag event con taxonomies (Whitehat, Adversary, ecc.) per classificazione. Clustering: automaticamente raggruppa simile events per identificare campaign.

**Correlazione:**

MISP correlate events automaticamente. Se due event hanno stesso file hash, o stesso IP, o stesso malware family, MISP identifica correlation. SOC analyst può "follow the thread" per scoprire relazione tra incidents.

**Distribuzione:**

MISP supporta org feeds (feed di events da MISP community, condiviso gratuitamente). Subscribe a feed: "Ransomware feeds", "APT activity", "Botnet C2", ogni giorno ricevi updates di nuovi indicatori.

**Integration con OpenSearch:**

Script Python regularmente pull da MISP API, estrae indicatori (IP, domain, hash), indicizza in OpenSearch in indice "threat-intel-misp". Successivamente, durante log analysis, corri query che correlate. Esempio query:

```
GET /firewall-*,threat-intel-misp/_search
{
  "query": {
    "bool": {
      "must": [
        {"match": {"firewall.destination_ip": "threat-intel-misp.ip"}}
      ]
    }
  }
}
```

Se firewall log contiene connessione a IP noto