Project: Agentic SOC – Autonomous Incident Response on NVIDIA DGX

Author: Sunil Tiwari Date: January 2, 2026 Version: 2.0 (Final Release)

---

1. Executive Summary

This document details the architecture and implementation of an end-to-end Agentic Security Operations Center (SOC). The system enables autonomous threat detection, AI-driven investigation, and human-in-the-loop governance. It runs entirely on-premise using NVIDIA DGX Spark hardware, ensuring 100% data sovereignty by leveraging local Large Language Models (LLMs)

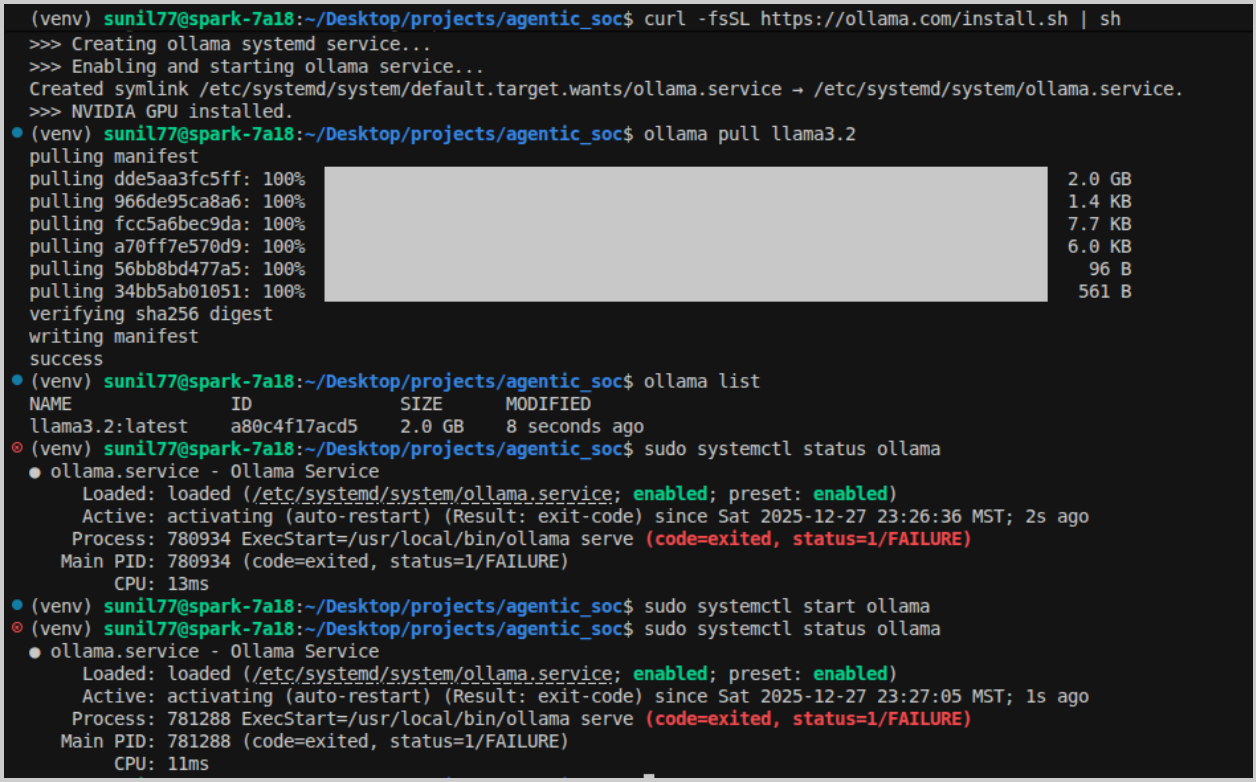
Part 1: The Core Infrastructure (The Foundation)

Below are the permanent components already running in my system (NVIDIA DGX Spark).

NVIDIA DGX (AARCH64/ARM64 Architecture): The high-performance compute platform unlike standard PCs (x86).

Wazuh (SIEM/XDR): It collects logs from the Linux endpoints, analyzes them against rules, and generates alerts. It provides the telemetry the AI agents will analyze.

Ollama: The inference engine. Allows me to run Large Language Models (LLMs) like Llama 3.2 locally. In my SOC environment, using Ollama ensures data privacy. Hence, the security alerts never leave the network.



CrewAI: This is the orchestration framework. It allowed me to define “Agents” with specific roles, goals, and backstories. It manages the Process, ensuring “Agent A” passes its findings to “Agent B” in a structured way.

Streamlit: It is an UI/Interface which provides a "Single Pane of Glass" dashboard for analyst governance.

Part 2: The Agentic Framework (The Logic):

Orchestration: We utilize CrewAI to define "Agents" with specific roles, goals, and backstories. LangChain acts as the bridge between these agents and the local Ollama API endpoints.

The Decision Matrix (OODA Loop): The system follows the military-grade OODA Loop (Observe, Orient, Decide, Act):

- Observe: Wazuh detects a brute force attack or anomaly.
- Orient: Triage Agent extracts entities (IPs, Users); Threat Researcher enriches data via AbuseIPDB.
- Decide: Incident Commander evaluates risk and proposes a mitigation strategy.
- Act: The proposal is pushed to the Analyst Dashboard for final authorization.

---

4. Project Life Cycle (Maturity Model)

Phase 1 - Static (Foundations)
Establishing core infrastructure by setting up Linux, Wazuh, and baseline log collection.

Phase 2 - Observable (Telemetry)
Enrolling endpoints and generating meaningful security telemetry and real-world security events.

Phase 3 - AI-Ready (Local Intelligence)
Deploying local LLMs (Ollama / Llama 3.2) on DGX systems to enable on-premise AI processing.

Phase 4 - Automated (Response Automation)
Implementing Python-based automation that detects alerts and triggers predefined response actions.

Phase 5 - Autonomous (Agentic SOC) (Current)
Operating a multi-agent SOC where AI agents collaborate, reason, and coordinate incident response.

Phase 6 - Intelligent (RAG & Memory)
Enhancing agents with retrieval-augmented generation and memory, enabling them to learn from past attacks and ingest external intelligence such as PDF threat reports.

Phase 7 - Optimized (Hardware): Fine-tuned inference parameters for the NVIDIA DGX architecture.

Phase 8 - Event-Driven (The Bridge): [New] Implemented a real-time wazuh_bridge.py listener to automate detection.

Phase 9 - Governance (The Dashboard): [New] Deployed a Streamlit Console for human-in-the-loop approval (NIST 800-53 Compliance).

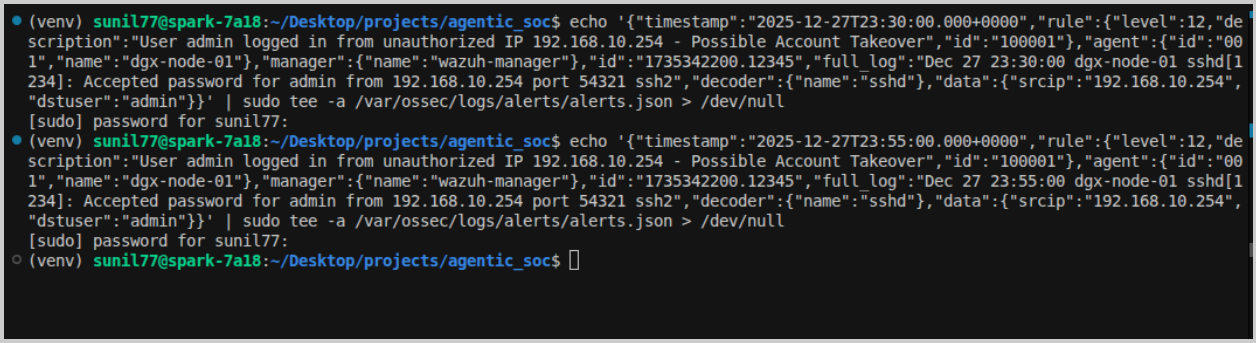
---

5. Agent Roles & Responsibilities

The "Brain" of the SOC consists of three specialized agents defined in agents.yaml:

1. Wazuh Alert Triage Specialist

- Goal: Extract key entities (Source IP, Rule ID, User) from raw JSON logs .



- Role: Acts as the "translator" between raw logs and the analysis team .

2. Cyber Threat Researcher

- Goal: Cross-reference entities against global threat intelligence (MITRE ATT&CK) .
- Role: Determines if an IP is a known scanner, botnet, or legitimate user .

3. SOC Incident Commander

- Goal: Make the final determination: BLOCK, WATCH, or IGNORE .
- Role: Balances security posture with business continuity. Generates the final RFC (Request for Change) .

---

6. Technical Implementation: The "Nervous System"

The Event Bridge (Phase 8)

To move beyond manual triggers, we implemented a custom Python Watchdog (wazuh_bridge.py).

- Logic: Uses a tail -f approach to monitor /var/ossec/logs/alerts/alerts.json.
- Filter: Ignores noise; only triggers on Rule Level >= 10.
- Action: Instantiates the CrewAI process immediately upon detection, passing the src_ip as a variable.

6.2 The Governance Console (Phase 9)

To ensure operational control, we deployed a Streamlit dashboard.

- Data Pipeline: Agents write investigation results to a structured active_incidents.json database.

- Frontend: The dashboard reads this file in real-time, displaying a "Card" for each incident.
- Controls: Analysts use "Approve Block" or "Monitor Only" buttons to execute the final response, closing the loop.

---

7. Validation & Testing

Adversary Emulation

We validated the pipeline using Atomic Red Team principles (Technique T1110).

- Attack: hydra -l root -p badpass 127.0.0.1 ssh
- Detection: Wazuh triggers Rule 5716 (SSHD Authentication Failed).
- Response: The Bridge detects the log entry -> Agents investigate -> Dashboard displays the alert.

Execution Trace Example

The following log demonstrates the Incident Commander's reasoning process:

- Final Report: Threat Assessment Complete * Analysis: The source IP (192.168.1.100) matches Rule "wazuh-2001" (Suspicious Activity) .
- Justification: The system detected repeated failed access attempts fitting a brute-force pattern. This warrants immediate containment . * Decision: I have determined the action required is to BLOCK the traffic until further investigation. * ACTION: BLOCK.

---

8. Technical Achievements

- Environment Isolation: Solved circular dependency conflicts between crewai, litellm, and openai using a strictly pinned virtual environment.
- Permission Bridging: Developed a secure execution strategy allowing user-space AI agents to read root-protected SIEM logs.
- Local Inference: Achieved <5s latency using on-premise Llama 3.2 on NVIDIA DGX GPUs.
- Full Stack Integration: Successfully linked a C++ based SIEM (Wazuh), Python AI backend, and Web-based Frontend into a unified workflow.