AIOps Advanced Training for Kubernetes - Detailed Reference

Duration: 24 Hours

Training Objectives

- Understand the fundamentals of AIOps and its necessity in modern Kubernetes environments.
- Learn how to integrate monitoring, logging, and observability tools with ML models.
- Apply AI/ML techniques for real-time anomaly detection, incident management, and root cause analysis.
- Forecast resource usage and implement predictive scaling with AI models.
- Implement open-source security tools with LLM-powered threat explanations.
- Build smart workflows for alert routing and AIOps automation using LangChain and Ollama.
- Deliver hands-on labs that simulate real-world operational challenges and AI-based solutions in Kubernetes.

Module 1: Introduction to AIOps & Kubernetes

Outcome: Understand AIOps concepts and set up a monitoring stack on Kubernetes.

- Theory: AIOps fundamentals, reactive vs proactive vs predictive ops. Kubernetes complexity: ephemeral workloads, scale, observability gaps.
- Tools: Prometheus, Grafana, Minikube, IsolationForest (Python), Prometheus, Grafana
- Lab: Deploy Prometheus and Grafana using Helm charts. Simulate CPU/memory usage using BusyBox or stress-ng. Observe and visualize cluster metrics in Grafana. Optionally export metrics for anomaly detection in Python.
- Outputs: Grafana dashboards showing CPU/memory usage patterns, sample anomaly metrics, Helm config files.

Module 2: Monitoring & Observability with AI

Outcome: Integrate observability tools and apply AI to label and summarize logs.

- Theory: Metrics (Prometheus), Logs (ELK/Loki), Traces (OpenTelemetry). Observability pipelines in cloud-native environments.
- Tools: OpenTelemetry, ELK Stack, FluentBit, ML/LLM (LangChain)
- Lab: Install OTEL Collector. Ingest logs. Label anomalies using ML or LLM.
- Outputs: Indexed logs, ML-labeled alerts, pipeline config files

Module 3: Incident Management & Root Cause Analysis

Outcome: Use graph models and LLMs to explain incidents and derive root causes.

- Theory: Incident correlation, noise reduction, NLP-based RCA using GPT. Graph modeling with Neo4j.
- Tools: Neo4j, Ollama (local LLM), LangChain, Kubernetes logs

- Lab: Load logs into Neo4j. Use GPT to explain alerts. Build cause-effect graph.

- Outputs: Root cause graph, GPT output summary, enriched logs

Module 4: Predictive Scaling & Optimization

Outcome: Predict workload patterns and scale Kubernetes clusters automatically.

- Theory: Limitations of HPA/VPA. Event-driven scaling (KEDA). Forecasting with Prophet for predictive autoscaling.

- Tools: KEDA, Prophet, Python, HorizontalPodAutoscaler

- Lab: Deploy KEDA. Simulate variable load. Forecast future usage and trigger autoscaling.

- Outputs: Forecast plot, scaling configs, KEDA deployment

Module 5: Security, Threat Detection & Drift Enforcement

Outcome: Detect and respond to runtime threats using behavioral analysis and LLMs.

- Theory: Behavioral anomaly detection, drift detection. Tools: Falco, Sysdig Secure, KubeArmor. GPT for threat explanation.

- Tools: Falco, KubeArmor, Ollama, Python, shell scripts

- Lab: Deploy Falco. Simulate attack. Classify threat with GPT. Auto-remediate (optional).

- Outputs: Falco logs, LLM classification, remediation YAMLs

Module 6: AIOps Platforms & Smart Workflows

Outcome: Build end-to-end AIOps workflows for alert handling using open-source tools.

- Theory: DIY AIOps with LangChain + OpenAI. Alert classification and routing. Market tools: Dynatrace, Moogsoft, Splunk.

- Tools: LangChain, Ollama API, Slack Webhooks, Python

- Lab: Create alert router using LangChain + GPT. Send output to Slack.

- Outputs: LangChain flow, alert messages, webhook config

Lab VM / Environment Requirements (Preferred/Ideal)

- OS: Ubuntu 22.04 LTS (native or WSL2)

- CPU: 8 cores (minimum for Ollama + K8s)

- RAM: 16 GB or more (essential for Ollama models + monitoring tools)

- Disk: 60 GB SSD or higher (Docker images, Ollama models, datasets)

- Tools: Docker, Minikube, Helm, kubectl, Ollama (installed via shell script)

- Python: 3.10+ with pip

- Editors: Jupyter Notebook or VSCode

- Python Libraries: pandas, numpy, matplotlib, scikit-learn, prophet, langchain, ollama
- Required:
  - Neo4j Desktop (open-source)
  - Ollama with the following models pulled:
    - llama3.2:1b
    - llama3.2:3b
    - gemma3:1b
    - gemma3:3b
- Alerting Options: Local Mattermost, file-based logging, or email