Research Title

Threat-Aware Autoscaling Using ML-Based Anomaly Detection in Kubernetes

----

What is the research about?

The research focuses on making Kubernetes clusters smarter and more secure by integrating **machine learning-based anomaly detection** into the **autoscaling process**. In simple terms:

- Kubernetes automatically scales applications up or down based on resource usage (like CPU or memory).
- But what if the traffic coming to the application is **malicious or abnormal**? Autoscaling blindly would waste resources and even amplify an attack.
- So, the goal is to **detect whether incoming traffic is legitimate or suspicious** and use that information to make better autoscaling decisions.

Core Component: Threat-Aware Anomaly Detection

Instead of traditional classification (where you train models on both normal and attack traffic), this research uses **anomaly detection**. Why?

- In real-world systems, attack patterns change constantly, and collecting all possible attack data is hard.
- Anomaly detection trains models only on **legitimate traffic**. Anything that deviates significantly from this normal pattern is flagged as an anomaly.

Data Collection Process

To detect anomalies, we need data about incoming traffic. Here’s how it works:

1. **Log Collection**:
    - Kubernetes pods generate logs about requests.
    - A **log shipping agent (FluentD)** collects these logs.
2. **Log Processing**:

    - Logs are sent to the **ELK Stack (Elasticsearch, Logstash, Kibana)** for storage and analysis.

3. **Feature Extraction**: From these logs, we extract meaningful features that describe traffic behavior. Examples:
    - **Geo-location** of the request (country, region, IP origin).
    - **Request rate** (number of requests per second).
    - **HTTP method** (GET, POST, etc.).
    - **Response status codes** (200 OK, 404 Not Found, 500 Internal Server Error).
    - **User-Agent string** (browser or bot info).
    - **Payload size** (size of request body).
    - **Time-based patterns** (spikes during unusual hours). These features help the ML models understand what “normal” traffic looks like.

Machine Learning Models Used

We use **five different anomaly detection models** to compare performance:

1. **Autoencoder**:
    - A neural network that learns to compress and reconstruct normal traffic patterns.
    - If reconstruction error is high, the traffic is likely anomalous.
2. **Isolation Forest**:
    - Works by randomly partitioning data.
    - Anomalies are easier to isolate because they behave differently from normal points.
3. **One-Class SVM**:
    - Learns a boundary around normal data.
    - Anything outside this boundary is considered an anomaly.
4. **K-Nearest Neighbors (KNN)**:

    - Compares each point to its nearest neighbors.
    - If a point is far from others, it’s likely an anomaly.
5. **Gaussian Mixture Model (GMM)**:
    - Assumes normal data follows a Gaussian distribution.
    - Points with very low probability under this distribution are anomalies.

Model Training

- All models are trained **only on legitimate traffic**.
- This makes them sensitive to deviations without needing attack samples.

Evaluation and Comparison

We compare models based on:

- **Accuracy** (how well they detect anomalies).
- **Precision & Recall** (to avoid false positives and negatives).
- **Latency** (speed of detection).
- **Resource usage** (important for Kubernetes environments).

The best-performing model is **automatically selected** for deployment.

Integration with Autoscaling

Once anomalies are detected:

- If traffic is legitimate → normal autoscaling happens.
- If traffic is suspicious → autoscaling decisions are adjusted (e.g., prevent scaling up unnecessarily during an attack).

This makes Kubernetes **threat-aware**, saving resources and improving security.

Novelty of the Research

- Combines **security (anomaly detection)** with **performance optimization (autoscaling)**.
- Uses **multiple ML models** on the same dataset for comparison.
- Integrates with **Kubernetes**, which is widely used in cloud-native environments.