**Architecture Decision Record: Monitoring AWS Resources with Self-Managed Free Grafana**

**Date:** September 26, 2025  
**Status:** Proposed  
**Stakeholders:** NewLine SRE & SWE Team

**Context**

We need comprehensive monitoring for AWS applications and infrastructure using self-managed (on-premises) free Grafana OSS. The monitoring scope includes:

* **Applications:** Lambda, ECS, EC2
* **Databases:** DynamoDB, Aurora
* **Other AWS Components:** Various AWS services

This ADR evaluates implementation paths to achieve this without paid services, focusing on built-in capabilities, custom plugins, and OpenTelemetry (OTEL) integration for white-box observability.

**Decision**

**Adopt the Grafana Complete Approach** (OTEL Collectors with Prometheus) for full white-box monitoring, as it provides the most comprehensive observability while remaining feasible with free tools.

**Consequences**

**Benefits**

* Enables end-to-end metrics, traces, and logs
* Avoids vendor lock-in
* Supports custom instrumentation
* Leverages open standards for future-proofing

**Challenges**

* Requires significant initial setup effort
* Ongoing maintenance burden for collectors
* Potential scaling challenges for large environments

**Alternatives Considered**

**Option 1: Grafana Base Approach**

*Direct CloudWatch integration using built-in data source*

**Overview**

Use built-in features to directly query AWS resources via Grafana's CloudWatch data source, which is available in the free OSS version. AWS exposes metrics for Lambda, ECS, EC2, DynamoDB, Aurora, and other services through CloudWatch, and Grafana OSS includes a CloudWatch data source plugin.

**Implementation Steps**

1. Install Grafana OSS on a server
2. Configure the CloudWatch data source with AWS access keys or IAM role
3. Query metrics (e.g., CPU utilization for EC2, read/write capacity for DynamoDB)
4. Build or import dashboards (community dashboards available)
5. Set up alerts if needed

**Estimated Effort:** 5-7 steps, 2-4 hours for basic setup (more for custom dashboards)

**Limitations**

* Restricted to CloudWatch-exposed metrics only
* No direct support for traces/logs without additional data sources
* Not true white-box monitoring for application internals

**Option 2: Grafana Enhanced Approach**

*Custom Grafana plugins for direct AWS API integration*

**Overview**

Create custom Grafana plugins to query AWS APIs directly, extending beyond built-in capabilities.

**Benefits Over Base Approach**

* Custom queries to AWS APIs (e.g., deeper DynamoDB insights via DescribeTable)
* Lambda logs via invocation details
* Tailored visualizations
* Integration of non-CloudWatch data

**What's Possible with Free OSS**

* Develop and install custom data source plugins using Grafana's plugin SDK
* Call AWS SDK methods directly
* Create custom visualizations

**What's Not Possible**

* Features requiring Grafana Enterprise (advanced RBAC, reporting)
* Bypassing AWS API limits
* Accessing restricted data without proper permissions

**Implementation Steps**

1. Scaffold plugin with @grafana/create-plugin tool
2. Implement data source logic using AWS SDK
3. Test against AWS services
4. Build and sign plugin
5. Install in Grafana
6. Create dashboards

**Estimated Effort:** 10-15 steps per plugin, 20-40 hours (more for multiple services, testing, and iteration)

**Why Not Selected**

* High development effort for marginal gains over base approach
* Increased maintenance burden
* Potential security risks from credential management in plugins
* Time investment for updates when AWS APIs change

**Option 3: Grafana Complete Approach ✓ [Selected]**

*OTEL Collectors pushing to self-hosted Prometheus*

**Overview**

Use OpenTelemetry Collectors to push data into self-hosted Prometheus, then visualize in Grafana with custom plugins if needed.

**Benefits Over Base Approach**

* Enables white-box monitoring with application-level metrics/traces/logs
* Goes beyond CloudWatch aggregates
* Supports correlation across services

**Benefits Over Enhanced Approach**

* Less custom coding (leverages OTEL standards and AWS Distro for OTEL)
* Better scalability for distributed tracing
* Future-proofing with open standards

**What's Possible with Free OSS**

* Full setup using open-source OTEL Collector (or AWS ADOT)
* Prometheus for storage
* Grafana's Prometheus data source
* Collectors receiving from AWS services via CloudWatch exporter or direct instrumentation

**What's Not Possible**

* Seamless auto-instrumentation for all AWS managed services without configuration
* Requires application code changes for deepest insights

**Implementation Steps**

1. Install OTEL Collectors on relevant hosts/instances
2. Configure receivers/exporters
   * ECS via task definitions
   * Lambda extensions
3. Set up Prometheus to receive from OTEL
4. Install Grafana and add Prometheus data source
5. Develop minimal custom plugins if gaps exist
6. Build dashboards and alerts

**Estimated Effort:** 15-20 steps, 10-20 hours (more for production hardening and full coverage)

**Related Decisions**

None at this time. Future decisions may address:

* Scaling Prometheus for larger environments
* Log aggregation and integration strategies
* Advanced tracing capabilities

**Appendix**

**Technology Stack**

* **Grafana OSS:** Visualization layer
* **Prometheus:** Time-series database
* **OpenTelemetry Collectors:** Data collection and forwarding
* **AWS CloudWatch:** Native AWS metrics source
* **AWS Distro for OpenTelemetry (ADOT):** Optional AWS-optimized collector

**References**

* [Grafana OSS Documentation](https://grafana.com/docs/)
* [OpenTelemetry Documentation](https://opentelemetry.io/docs/)
* [AWS Distro for OpenTelemetry](https://aws-otel.github.io/docs/)
* [Prometheus Documentation](https://prometheus.io/docs/)