PRJ-DOP-023: Advanced Monitoring with Prometheus & Grafana on ECS

Certification: AWS Certified DevOps Engineer – Professional
Domain: Monitoring and Logging

---

1. Project Overview

This project demonstrates how to set up a modern, scalable, and fully-managed monitoring stack for containerized applications using the popular open-source tools **Prometheus** and **Grafana**. While CloudWatch is a powerful service, many organizations prefer Prometheus and Grafana for their rich query language (PromQL), powerful visualization capabilities, and extensive ecosystem of exporters.

We will deploy a containerized application on **Amazon ECS** and use the **AWS Distro for OpenTelemetry (ADOT)** as a sidecar container to scrape Prometheus metrics. These metrics will be sent to **Amazon Managed Service for Prometheus (AMP)**, a fully managed, highly available, and scalable time-series database. We will then use **Amazon Managed Grafana (AMG)** to query the metrics from AMP and build insightful dashboards for monitoring our application's health and performance.

Key Objectives

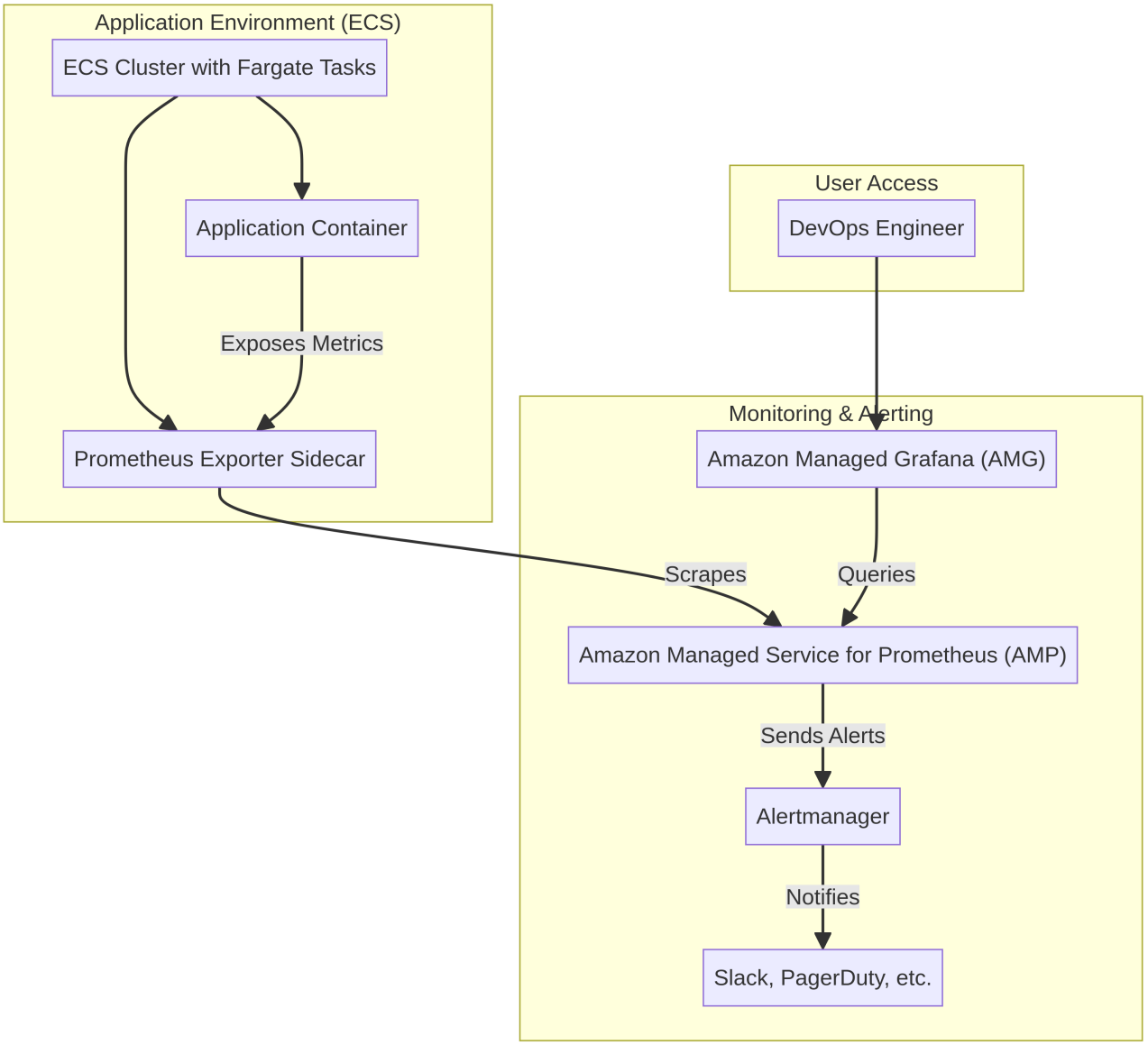
- Set up and configure Amazon Managed Service for Prometheus (AMP) and Amazon Managed Grafana (AMG).
- Instrument a sample application to expose custom metrics in the Prometheus format.
- Deploy an application on ECS Fargate with an OpenTelemetry sidecar container for metrics collection.
- Configure the sidecar to scrape the application's metrics and remote-write them to AMP.

- Connect AMG to the AMP workspace as a data source.
- Build a Grafana dashboard to visualize key application metrics.

---

2. Architecture

The architecture uses a sidecar pattern to decouple metrics collection from the application container, with AWS managed services providing the backend for storage and visualization.



Monitoring Flow:

1. **Application & Exporter:** An application running in an **ECS Fargate task** is instrumented with a Prometheus client library. It exposes a `/metrics` endpoint with its current metrics.
2. **Metrics Collection (Sidecar):** Alongside the application container in the same ECS task, we run the **AWS Distro for OpenTelemetry (ADOT) Collector** as a sidecar container. This collector is configured to:
    - Scrape the `/metrics` endpoint of the application container (e.g., `localhost:8080/metrics`).
    - Remote-write the collected metrics to the Amazon Managed Prometheus (AMP) workspace.
3. **Metrics Storage (AMP): Amazon Managed Service for Prometheus** receives and stores the time-series data. AMP handles the ingestion, storage, and querying of metrics at scale without requiring you to manage the underlying infrastructure.
4. **Visualization (AMG): Amazon Managed Grafana** is a fully managed Grafana service. We configure our AMP workspace as a data source within AMG. DevOps engineers can then log in to the AMG workspace to build dashboards and query metrics using PromQL.
5. **Alerting:** Alerting rules can be configured within AMP. These rules are evaluated against the incoming metrics. When an alert fires, it is sent to **Alertmanager**, which can be configured to route notifications to services like Slack, PagerDuty, or email.

---

3. Prerequisites

- An AWS account with administrative permissions.
- The AWS CLI installed and configured.
- Docker installed on your local machine.
- A sample application that can be instrumented with a Prometheus client library.

---

4. Step-by-Step Implementation Guide

Step 4.1: Set Up AMP and AMG Workspaces

1. **Create AMP Workspace:**
    - Go to the **Amazon Managed Service for Prometheus console -> Create workspace**.
    - Give it an alias (e.g., `app-monitoring-workspace`) and create it. It will take a few minutes to become active.
    - Once active, copy the **Endpoint - remote write URL**.
2. **Create AMG Workspace:**
    - Go to the **Amazon Managed Grafana console -> Create workspace**.
    - Give it a name and choose **AWS Single Sign-On (SSO)** for authentication (this is the recommended method).
    - In the **Configure settings** page, select your AMP workspace as a data source.
    - Finish creating the workspace. This can take several minutes.
    - Once created, assign a user (e.g., yourself) as an admin in the Grafana workspace.

Step 4.2: Instrument the Application

Here is an example of a simple Node.js Express app instrumented with the `prom-client` library.

`app.js`

```
const express = require("express");
const client = require("prom-client");
const app = express();

// Create a Registry to register the metrics
const register = new client.Registry();

// Add a default label (service name) to the registry
register.setDefaultLabels({ serviceName: "my-app" });

// Enable the collection of default metrics
client.collectDefaultMetrics({ register });

// Create a custom counter metric
const httpRequestsTotal = new client.Counter({
  name: "http_requests_total",
  help: "Total number of HTTP requests",
  labelNames: ["method", "path"],
});
register.registerMetric(httpRequestsTotal);

// Middleware to count requests
app.use((req, res, next) => {
  httpRequestsTotal.inc({ method: req.method, path: req.path });
  next();
});

app.get("/", (req, res) => res.send("Hello World!"));

// Expose the /metrics endpoint
app.get("/metrics", async (req, res) => {
  res.set("Content-Type", register.contentType);
  res.end(await register.metrics());
});

app.listen(8080, () => console.log("App listening on port 8080"));
```

Create a `Dockerfile` and push the image to ECR.

Step 4.3: Configure the OpenTelemetry Collector

Create a configuration file for the ADOT collector named `adot-config.yaml`.

```
receivers:
  prometheus:
    config:
      scrape_configs:
        - job_name: "my-app-scrape"
          scrape_interval: 15s
          static_configs:
            - targets: ["localhost:8080"] # Scrapes the app container on
port 8080

exporters:
  awsprometheusremotewrite:
    endpoint: "<YOUR_AMP_REMOTE_WRITE_URL>" # Replace with your AMP endpoint
    aws_auth:
      region: "<YOUR_AWS_REGION>"
      service: "aps"

service:
  pipelines:
    metrics:
      receivers: [prometheus]
      exporters: [awsprometheusremotewrite]
```

Important:

- Replace `<YOUR_AMP_REMOTE_WRITE_URL>` and `<YOUR_AWS_REGION>`.
- This configuration file needs to be injected into the ADOT sidecar container. The best way to do this is to store it in **AWS Systems Manager Parameter Store**.

Step 4.4: Create the ECS Task Definition with Sidecar

1. Go to **ECS -> Task Definitions -> Create new task definition**.
2. **Name:** `app-with-adot-sidecar`
3. **Task role:** Create or use a task role that has the `aps:RemoteWrite` permission for your AMP workspace.
4. **Containers:** Add two containers.

   Container 1: `my-app`

- **Name:** `my-app`
- **Image:** The ECR URI of your application image.
- **Port mappings:** 8080

Container 2: `adot-collector`

- **Name:** `adot-collector`
- **Image:** `public.ecr.aws/aws-observability/aws-otel-collector:latest`
- **Essential:** True
- **Environment variables:**
  - `AOT_CONFIG_CONTENT`: This is where you will inject the config from Parameter Store. The value should be a reference to the SSM parameter.

5. Create the task definition.

Step 4.5: Deploy and Verify

1. **Deploy the ECS Service:** Create a new ECS service using the `app-with-adot-sidecar` task definition.
2. **Verify in Grafana:**
   - Log in to your **Amazon Managed Grafana** workspace.
   - Go to **Configuration -> Data Sources**. You should see your AMP workspace already configured.
   - Go to **Explore**.
   - Select the AMP data source.
   - In the **Metrics** browser, you should now see the metrics from your application (e.g., `http_requests_total`).
   - Run a query like `sum(rate(http_requests_total[5m]))` to see the request rate.
3. **Build a Dashboard:** Create a new dashboard and add panels that query your application metrics from AMP.

---

5. Alerting with Alertmanager

1. **Create Alerting Rules:** In the AMP console, you can create a “Rule groups namespace” where you define your alerting rules in a YAML file, using standard Prometheus rule syntax.

   `alert-rules.yaml` “`yaml groups:

   ```
    - name: MyAppAlerts
   rules:
     - alert: HighErrorRate
       expr: sum(rate(http_requests_total{status_code=~"5.."}[5m])) > 10
       for: 1m
       labels:
         severity: critical
       annotations:
         summary: "High HTTP 5xx error rate"
   ```

   „“`

2. **Configure Alertmanager:** You need to run an Alertmanager instance (this can also be run on ECS). In the AMP console, you provide the endpoint of your Alertmanager. AMP will then send any firing alerts to it.

3. Alertmanager is then configured to route these alerts to your desired notification channels.

---

6. Cleanup

1. **Delete the Amazon Managed Grafana workspace.**
2. **Delete the Amazon Managed Prometheus workspace.**
3. **Delete the ECS service and cluster.**
4. **Delete the ECS task definition.**
5. **Delete the ECR image** for your application.
6. **Delete the ADOT config** from SSM Parameter Store.

7. **Delete the IAM roles** created for the ECS task and Grafana.

Business Context

The Problem

Development teams face slow deployment cycles, manual testing bottlenecks, and security vulnerabilities introduced during the CI/CD process. Traditional deployment methods lack security scanning, leading to vulnerabilities reaching production. Manual processes create inconsistency and human error.

The Solution

Secure CI/CD pipeline with integrated security scanning, automated testing, and infrastructure as code. Implements shift-left security with SAST, DAST, and dependency scanning. Automates deployments while enforcing security gates and compliance checks at every stage.

Business Value

- **Faster Time to Market:** Automated pipelines reduce deployment time from weeks to hours
- **Improved Security Posture:** Catches vulnerabilities before production deployment
- **Reduced Manual Effort:** Eliminates 80%+ of manual deployment tasks
- **Audit Trail:** Complete deployment history and security scan results for compliance

Risk Mitigation

Prevents deployment of vulnerable code, hardcoded secrets, misconfigured infrastructure, and non-compliant resources to production environments.

GRC Mapping

Compliance Frameworks

- **NIST CSF:** PR.IP-1 (Baseline configuration), PR.DS-6 (Integrity checking), DE.CM-4 (Code analysis)
- **ISO 27001:** A.12.1 (Operational procedures), A.14.2 (Security in development), A.12.4 (Logging)
- **CIS Controls:** Control 16 (Application Software Security), Control 11 (Secure Configuration)
- **OWASP DevSecOps:** Security testing integration, Secret management, Dependency checking

Security Controls Implemented

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Software Composition Analysis (SCA)
- Secrets scanning and prevention
- Infrastructure as Code security validation

Audit Evidence

- Pipeline execution logs with security scan results
- Code commit and approval records
- Security test reports (SAST/DAST/SCA)
- Deployment approval and rollback records

Regulatory Alignment

- **SOX:** Section 404 (Change management controls)
- **PCI DSS:** Requirement 6.3 (Secure development), Requirement 6.5 (Common vulnerabilities)

- **GDPR:** Article 25 (Data protection by design), Article 32 (Security of processing)
- **SOC 2:** CC8.1 (Change management), CC7.2 (System monitoring)