Course 8: Monitoring, Tracing & Alerting in Cloud-Native Environments

1. Introduction

By now, you have deployed a scalable cloud-native application on your managed Kubernetes cluster.
You have implemented GitHub Actions runners, Helm deployments, and Identity Federation.
The next crucial step is **observability** — the ability to understand what's happening in your system.

In production environments, **monitoring**, **tracing**, and **alerting** are fundamental to:

- Detecting issues early.
- Understand system performance.
- Improve reliability and scalability.
- Reduce downtime and debugging time.

In this course, you will learn about **Prometheus**, **Grafana**, and **OpenTelemetry**, and how they integrate and interact between them and **cloud-native monitoring services**.

2. Observability Overview

Observability has three key pillars:

1. **Metrics** — Quantitative data (CPU, latency, request rate, etc.).
2. **Logs** — Event records (errors, operations, etc.).
3. **Traces** — Request paths through multiple services.

Your project’s observability must include all three pillars to ensure you can debug and analyze behavior from the **load balancer** down to individual **pods** and **runners**.

---

3. Prometheus

3.1 What is Prometheus?

Prometheus is an **open-source monitoring system** designed for Kubernetes and cloud-native infrastructures.
It collects metrics, stores them in a time-series database, and lets you query them using **PromQL**.

3.2 How It Works

- Prometheus **scrapes metrics endpoints** (usually `/metrics`) exposed by your applications or system components.
- Metrics are stored locally in a time-series database.
- Alerts are defined using **Prometheus alert rules**.
- Grafana or cloud dashboards visualize Prometheus metrics.

3.3 Integration with Kubernetes

In a Kubernetes cluster:

- Prometheus can be **deployed via Helm**.
- It automatically discovers targets using **Kubernetes service discovery**.
- Common targets include pods, nodes, and custom applications that expose `/metrics`.

3.4 Cloud-Native Integration

Both GCP and AWS offer **managed Prometheus services** that can be deployed within your cluster:

- **GCP:** Managed Service for Prometheus (integrated into Cloud Monitoring).
- **AWS:** Amazon Managed Service for Prometheus (AMP).

These services:

- Handle scaling and storage automatically.
- Allow querying metrics directly from the cloud console.
- Integrate with **cloud-native alerting** (Cloud Monitoring or CloudWatch).

However, you can still deploy your own prometheus using Helm.

---

4. Grafana

4.1 What is Grafana?

Grafana is a **visualization and dashboard tool** for metrics, logs, and traces.
It connects to data sources such as Prometheus, Cloud Monitoring, etc.

4.2 How It Works

- Grafana queries Prometheus (or another source) to retrieve metrics.
- You can create custom dashboards to visualize key indicators (CPU, memory, latency, etc.).
- Alerts can be configured directly from dashboards.

Example:
A dashboard showing:

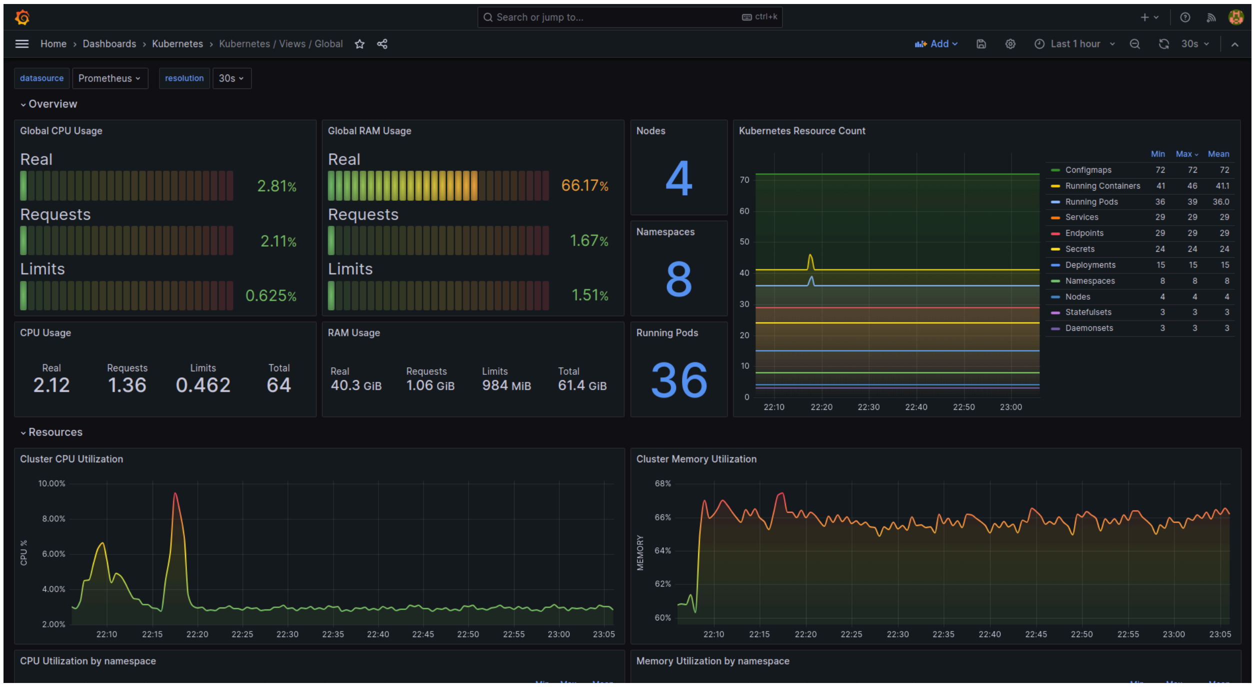
- Request latency from your app.
- CPU usage of self-hosted runners.
- Numbers of nodes and pods currently running for your runners and application.

4.3 Cloud-Native Integration

Once you've deployed a grafana instance in your cluster**,** it can connect to:

- Managed Prometheus.
- Cloud Logging / CloudWatch.
- OpenTelemetry traces.

Example:



datasource Prometheus
resolution 30s

Global CPU Usage

Real 2.81%
Requests 2.11%
Limits 0.625%

Global RAM Usage

Real 66.17%
Requests 1.67%
Limits 1.51%

Nodes 4

Namespaces 8

Running Pods 36

CPU Usage

Real 2.12
Requests 1.36
Limits 0.462
Total 64

RAM Usage

Real 40.3 GiB
Requests 1.06 GiB
Limits 984 MiB
Total 61.4 GiB

Kubernetes Resource Count

	Min	Max	Mean
Configmaps	72	72	72
Running Containers	41	46	41.1
Running Pods	36	39	36.0
Services	29	29	29
Endpoints	29	29	29
Secrets	24	24	24
Deployments	15	15	15
Namespaces	8	8	8
Nodes	4	4	4
Statefulsets	3	3	3
Daemonsets	3	3	3

Cluster CPU Utilization

Cluster Memory Utilization

5. OpenTelemetry

5.1 What is OpenTelemetry?

OpenTelemetry (OTel) is a set of APIs and SDKs that help you **collect traces, metrics, and logs** from your applications.
It’s an **open standard** supported by all major cloud providers.

5.2 How It Works

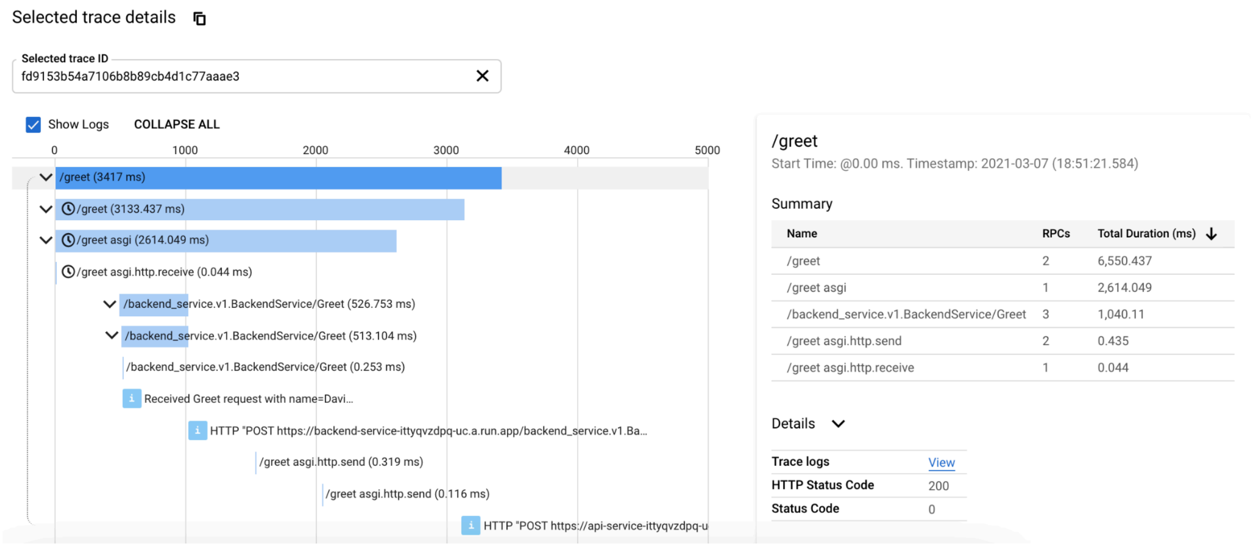
- You **instrument your code** (using OTel SDKs) to generate telemetry data.
- The data is sent to a **collector**, which exports it to tools such as Prometheus, Grafana, or cloud-native backends.

5.3 Cloud-Native Integration

- **GCP:** OpenTelemetry integrates with **Cloud Trace** and **Cloud Logging**.
- **AWS:** Integrates with **X-Ray**, **CloudWatch Logs**, and **AMP**.

This means that OTel traces and metrics can be visualized directly in cloud-native dashboards.

GCP:



Selected trace details

Selected trace ID: fd9153b54a7106b8b89cb4d1c77aaae3

☑ Show Logs   COLLAPSE ALL

```
/greet (3417 ms)
  /greet (3133.437 ms)
  /greet asgi (2614.049 ms)
    /greet asgi.http.receive (0.044 ms)
      /backend_service.v1.BackendService/Greet (526.753 ms)
      /backend_service.v1.BackendService/Greet (513.104 ms)
        /backend_service.v1.BackendService/Greet (0.253 ms)
        Received Greet request with name=Davi…
            HTTP "POST https://backend-service-ittyqvzdpq-uc.a.run.app/backend_service.v1.Ba…
                /greet asgi.http.send (0.319 ms)
                    /greet asgi.http.send (0.116 ms)
                                HTTP "POST https://api-service-ittyqvzdpq-u
```

/greet

Start Time: @0.00 ms. Timestamp: 2021-03-07 (18:51:21.584)

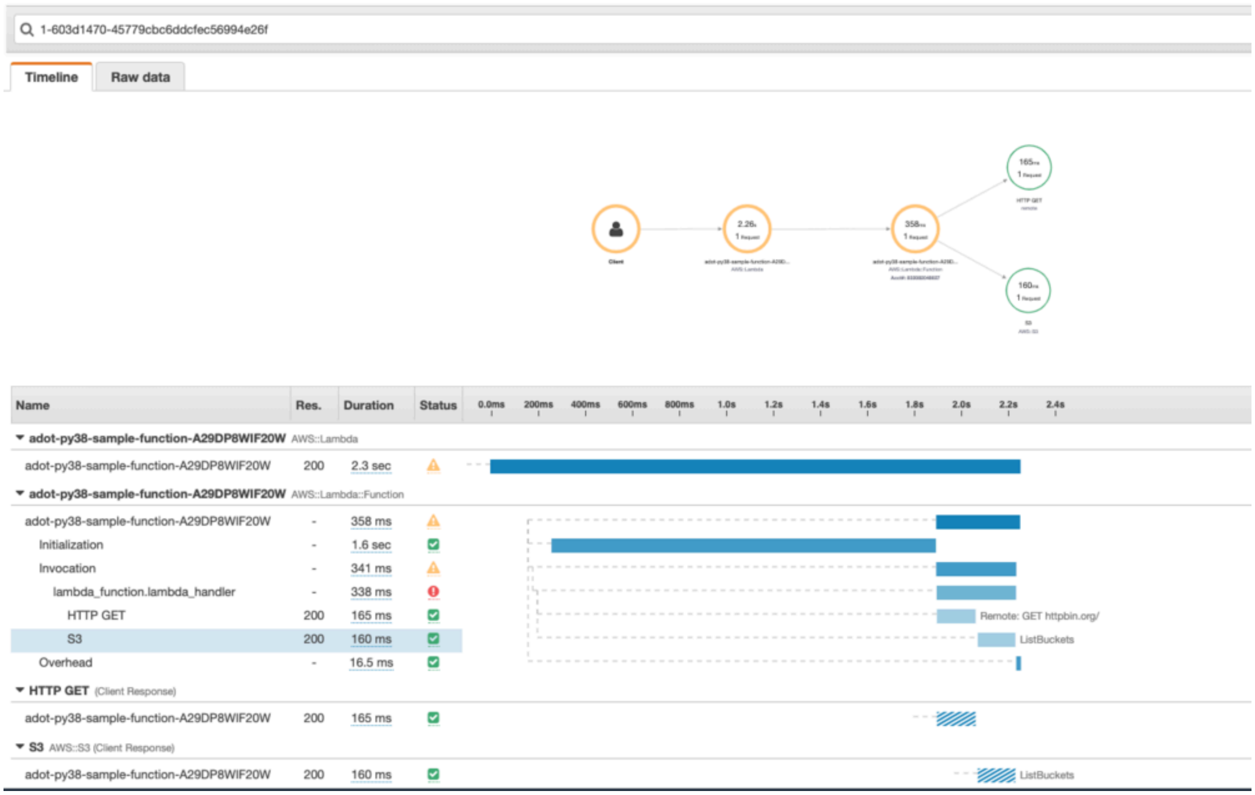
Summary

Name	RPCs	Total Duration (ms) ↓
/greet	2	6,550.437
/greet asgi	1	2,614.049
/backend_service.v1.BackendService/Greet	3	1,040.11
/greet asgi.http.send	2	0.435
/greet asgi.http.receive	1	0.044

Details

Trace logs	View
HTTP Status Code	200
Status Code	0

AWS:



1-603d1470-45779cbc6ddcfec56994e26f

Timeline | Raw data

Name	Res.	Duration	Status
▼ adot-py38-sample-function-A29DP8WIF20W AWS::Lambda			
adot-py38-sample-function-A29DP8WIF20W	200	2.3 sec	⚠
▼ adot-py38-sample-function-A29DP8WIF20W AWS::Lambda::Function			
adot-py38-sample-function-A29DP8WIF20W	-	358 ms	⚠
Initialization	-	1.6 sec	✓
Invocation	-	341 ms	⚠
lambda_function.lambda_handler	-	338 ms	❗
HTTP GET	200	165 ms	✓
S3	200	160 ms	✓
Overhead	-	16.5 ms	✓
▼ HTTP GET (Client Response)			
adot-py38-sample-function-A29DP8WIF20W	200	165 ms	✓
▼ S3 AWS::S3 (Client Response)			
adot-py38-sample-function-A29DP8WIF20W	200	160 ms	✓

6. Alerting

6.1 Why Alerting Matters

Monitoring without alerting is like logging errors no one ever reads.
Your system must **actively notify** you when something goes wrong.

6.2 How Alerts Work

You can define alerts based on:

- **Metrics thresholds** (CPU > 80%, errors > 5%).
- **Custom business metrics** (failed tasks, latency, etc.).
- **Tracing anomalies** (slow spans, missing dependencies).

6.3 Tools for Alerting

You can use:

- **Prometheus Alertmanager**
- **Grafana Alerting**
- **Cloud-native alerting**
  - **GCP Cloud Monitoring alerts**
  - **AWS CloudWatch alarms**

Alerts can be sent via many ways such as email, chats, webhooks, phone, etc.

---

7. Integration Between Tools

Component	Purpose	Example Integration
Prometheus	Collect metrics	Scrapes metrics from Kubernetes pods and nodes
Grafana	Visualize data	Connects to Prometheus or Cloud Monitoring
OpenTelemetry	Collect traces and logs	Exports data to Prometheus, Grafana, or cloud-native tools
Cloud Monitoring / CloudWatch	Managed alternative	Can aggregate and alert on Prometheus or OTel data

Together, they provide a full **observability stack** that spans from metrics to traces.

8. Project Requirements

For this project, you must:

- Implement observability for **runners and your application**.
- Use **Prometheus + Grafana + OpenTelemetry**, or equivalent tools of your choice.
- Collect **metrics**, **traces**, and **logs**.
- Set up **at least one alert** for:
  - Resource metrics (e.g., CPU, memory, latency)
  - Application-level issue (e.g., failed requests)
- Be able to **trace a single request** using `correlation_id` across your application.

You are free to use **cloud-managed** services or **self-managed** deployments via Helm.
Be ready to justify your architecture choices during the defense.

---

9. Key Takeaways

- **Prometheus** collects metrics.
- **Grafana** visualizes them and can create alerts.
- **OpenTelemetry** traces requests and enriches metrics with context.
- **Cloud-native integrations** simplify scaling, maintenance and observability.
- Observability is **not optional**, it’s what makes your system maintainable and reliable.