| **Component** | **Role** | **Description** |
| --- | --- | --- |
| 🟩 **API / Microservice** | Source | Emits logs, traces, and metrics via OpenTelemetry SDK or Spring Actuator |
| 🟦 **OpenTelemetry Collector** | Collector | Collects telemetry, processes, and exports to backends |
| 🟨 **Prometheus** | Metrics DB | Scrapes and stores time-series metrics for performance analysis |
| 🟪 **Tempo** | Trace Storage | Stores and queries distributed traces |
| 🟥 **Jaeger** | Trace Viewer | UI for visualizing spans, dependencies, and performance bottlenecks |
| 🟧 **Grafana** | Dashboard | Unifies metrics (Prometheus) and traces (Tempo/Jaeger) for visualization |
| 👩‍💻 **User** | Viewer | Monitors health, latency, and bottlenecks through dashboards |

**🟦 OpenTelemetry Collector**

**What it does:**  
Acts as a **data router and processor** for all observability data — metrics, logs, and traces.

**Usage:**

* You configure your Spring Boot app with opentelemetry-exporter-otlp to send telemetry data (metrics/traces/logs) to the **Collector**.
* The collector then forwards:
  + **Metrics → Prometheus**
  + **Traces → Tempo/Jaeger**
  + **Logs → Loki (if you had it)**

**Think of it as:**  
A middleman that normalizes and routes your app’s monitoring data.

**How to check:**  
Go to your Collector logs — you should see messages like  
"Exporting metrics to prometheus" or "Exporting traces to tempo".

**🟨 Prometheus**

**What it does:**  
Stores **metrics** (numbers over time like CPU usage, memory, requests/sec).

**Usage:**

* Prometheus **scrapes metrics** from your app (or from the OpenTelemetry Collector).
* Metrics endpoint: http://<your-app>:8080/actuator/prometheus
* You can query metrics using **PromQL** (Prometheus Query Language).

**Try it:**  
Visit http://<prometheus-host>:9090 → Go to “Graph” → try queries like:

http\_server\_requests\_seconds\_count

system\_cpu\_usage

jvm\_memory\_used\_bytes

Prometheus must be connected to other microservice via actuator endpoint to pull this information this is defined in config.xml

global:  
 scrape\_interval: 5s  
 evaluation\_interval: 5s  
  
rule\_files:  
 - /etc/prometheus/alerts/prometheus-alerts.yml  
  
scrape\_configs:  
 - job\_name: 'order-service'  
 metrics\_path: /actuator/prometheus  
 static\_configs:  
 - targets: ['host.docker.internal:8080','order-service:8080']  
 - job\_name: 'customer-service'  
 metrics\_path: /actuator/prometheus  
 static\_configs:  
 - targets: ['host.docker.internal:8081','customer-service:8081']  
 - job\_name: 'gateway-service'  
 metrics\_path: /actuator/prometheus  
 static\_configs:  
 - targets: ['host.docker.internal:8085','gateway-service:8085']  
 - job\_name: 'discovery-service'  
 metrics\_path: /actuator/prometheus  
 static\_configs:  
 - targets: ['host.docker.internal:8761','discovery-service:8761']  
 - job\_name: 'config-service'  
 metrics\_path: /actuator/prometheus  
 static\_configs:  
 - targets: ['host.docker.internal:8888','config-service:8888']

And can be seen through targets page

A screenshot of a computer

AI-generated content may be incorrect.

**🟪 Tempo**

**What it does:**  
Stores **distributed traces** (used to follow a request through multiple services).

**Usage:**

* Traces come from OpenTelemetry (via Collector).
* Each trace shows how a request flows between services and how long each span took.

**Think of it as:**  
A **timeline** of your request journey across microservices.

**How to view:**

* Usually viewed **through Grafana**, not directly.
* In Grafana → add Tempo as a data source → open “Explore” → select Tempo → search for traces.

**🟥 Jaeger**

**What it does:**  
Also stores **traces**, like Tempo.  
You can use **either Tempo or Jaeger** (you have both, so maybe Tempo is backend and Jaeger is UI).

**Usage:**

* OpenTelemetry Collector can export traces to Jaeger.
* Each trace shows service name, operation, and duration.

**Try it:**  
Visit http://<jaeger-host>:16686 → Search by serviceName (e.g., spring-boot-app).

**In short:**  
Jaeger gives a **UI to visualize trace spans** (how long each call took, parent-child relationships, etc.).

**🟧 Grafana**

**What it does:**  
Acts as the **visualization and correlation hub**.  
You can connect all others (Prometheus, Tempo, Jaeger, Loki) to it.

**Usage:**

* Add **Prometheus**, **Tempo**, and **Jaeger** as data sources.
* Create dashboards for:
  + Metrics (CPU, JVM, requests/sec)
  + Traces (via Tempo)
  + Logs (if you add Loki later)
* Use **“Explore”** to correlate metrics ↔ traces ↔ logs.

**Try it:**  
Visit http://<grafana-host>:3000  
→ Log in (default admin/admin)  
→ Add data sources  
→ Import dashboard ID 4701 (Spring Boot Micrometer) from Grafana.com

**🧩 Quick Recap Table**

| **Tool** | **Type** | **Example UI** | **What You Get** |
| --- | --- | --- | --- |
| 🟦 OpenTelemetry Collector | Pipeline | — | Routes metrics/traces/logs |
| 🟨 Prometheus | Metrics DB | :9090 | CPU, JVM, request rates |
| 🟪 Tempo | Traces backend | via Grafana | Distributed traces |
| 🟥 Jaeger | Trace viewer | :16686 | Trace visualization |
| 🟧 Grafana | Dashboard/Correlation | :3000 | Unified dashboards |