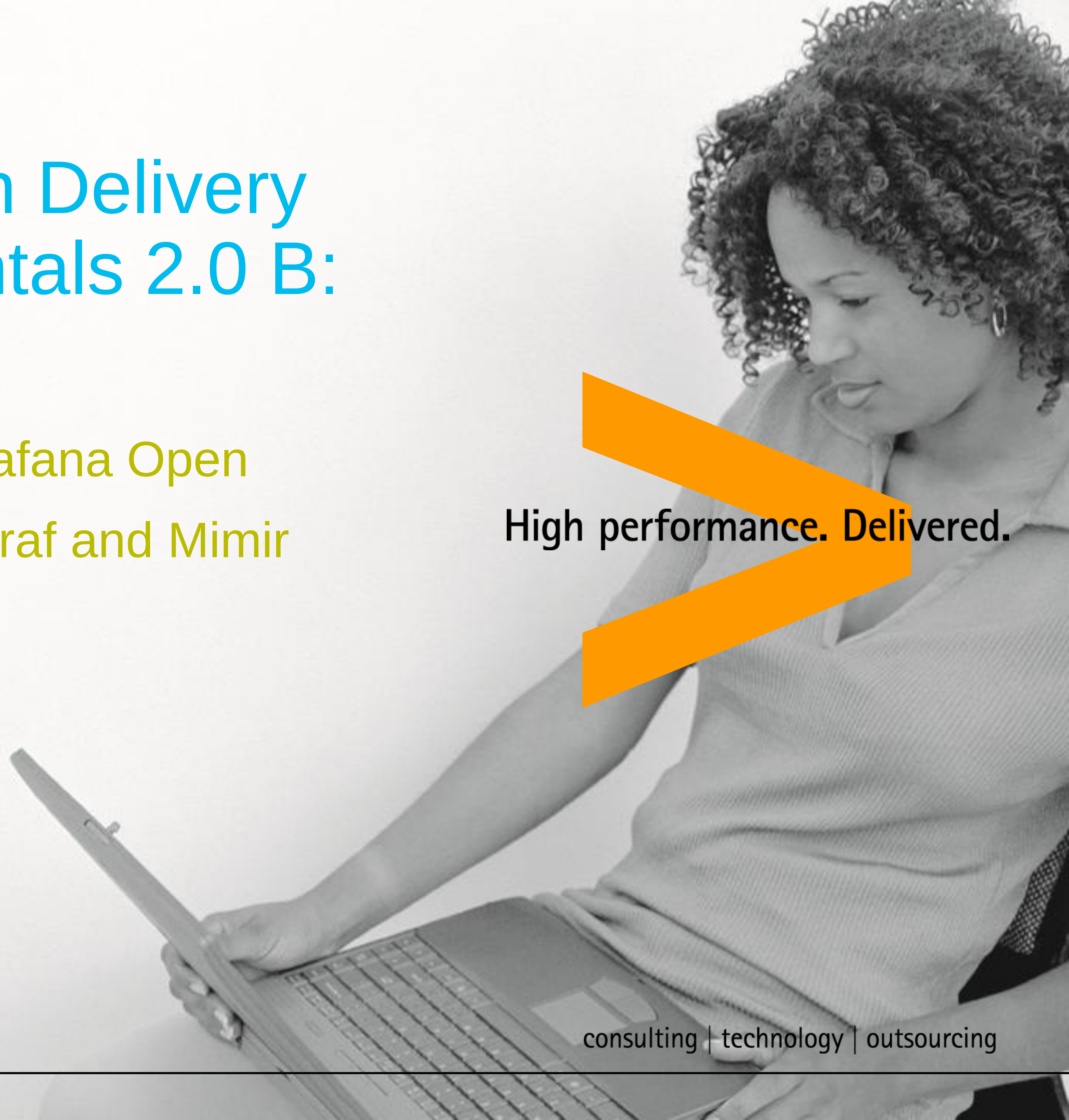
Application Delivery Fundamentals 2.0 B:

Prometheus Grafana Open Telemetry Telegraf and Mimir



Goals

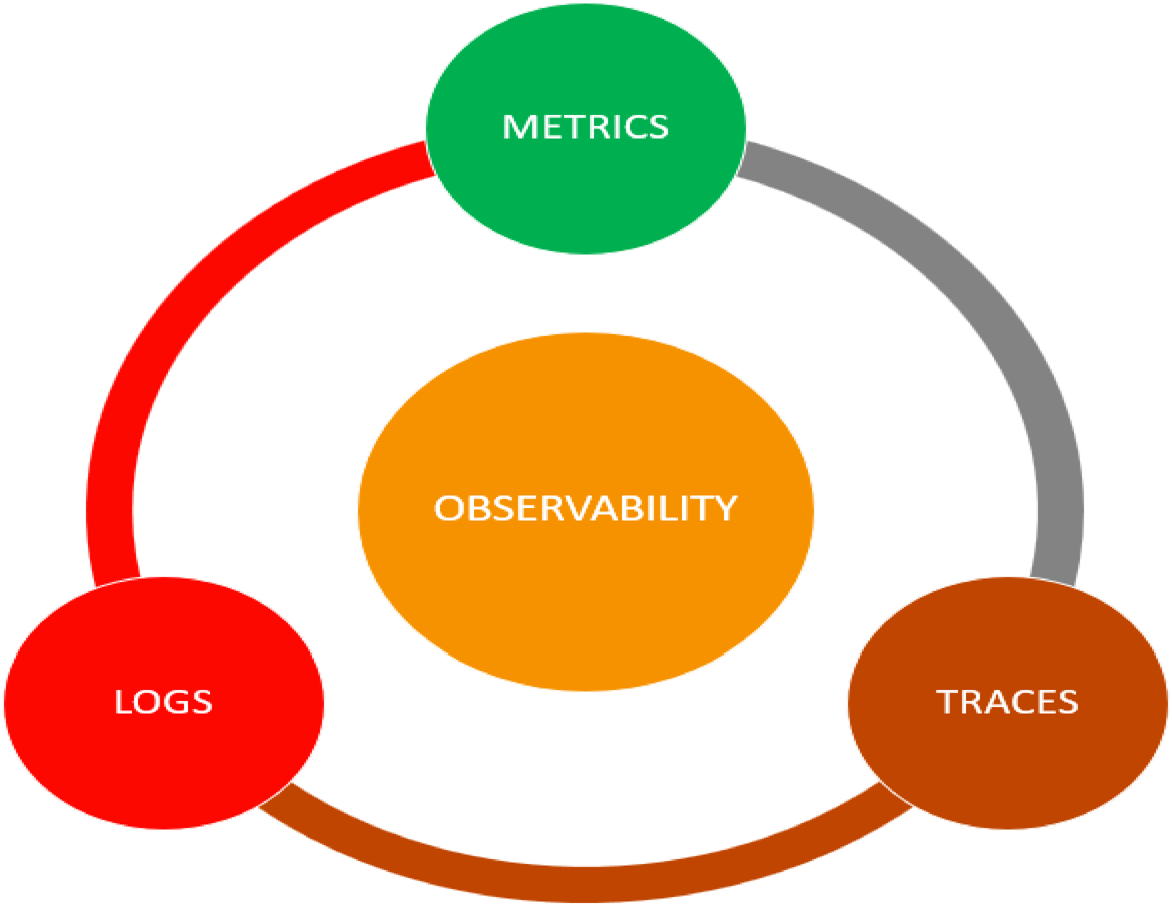
- Observability Concepts
- Overview of OpenTelemetry,Telegraf,Prometheus,Grafana and Mimir
- Prometheus fundamentals
- PromQL
- Instrumentation and Exporters
- Alerting & Dashboarding
- Alerting advanced
- Alert Manager HA
- Prometheus Federation
- Grafana

Observability

Microservices

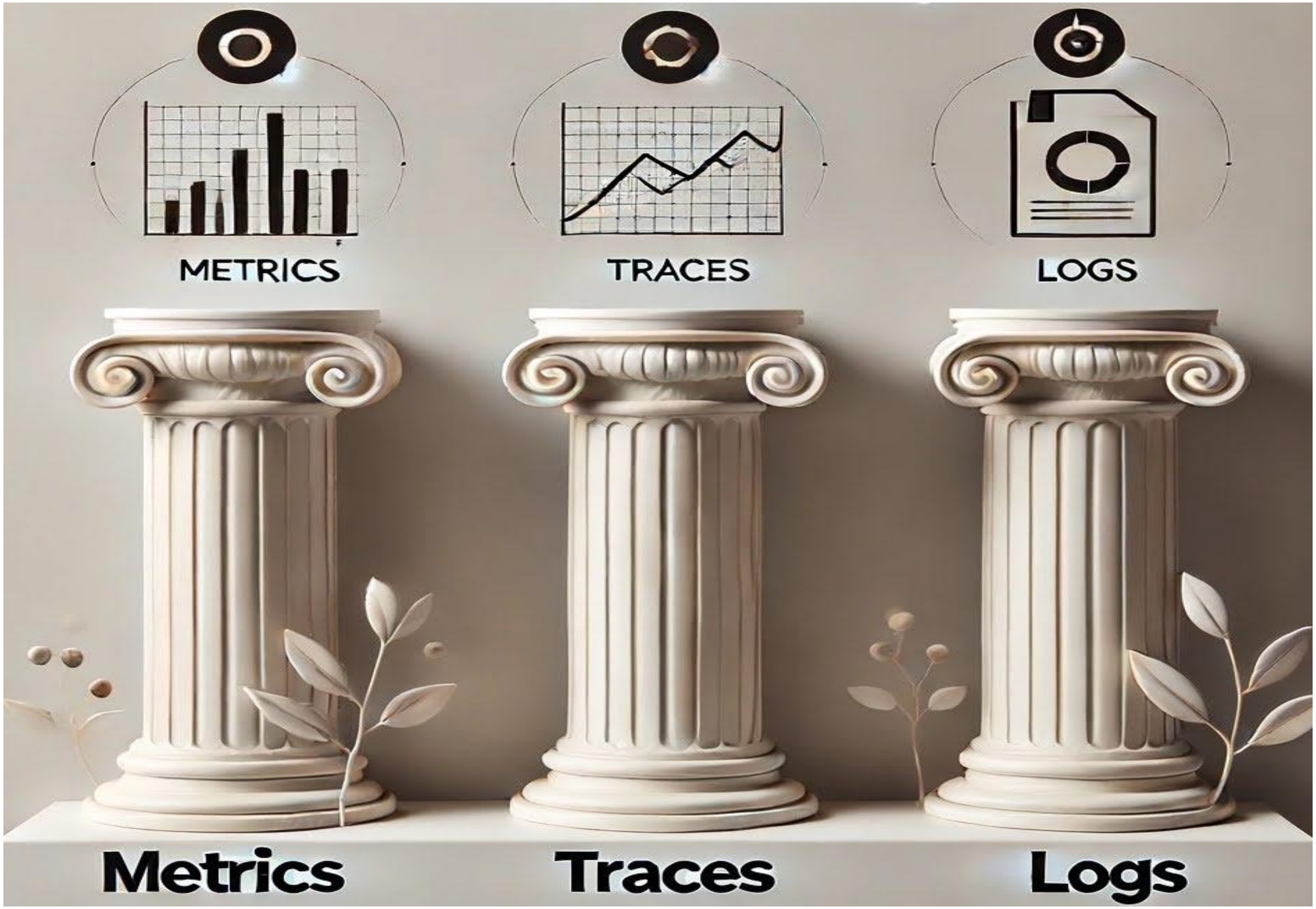
- Observability is the measure of how well the internal states of a system can be inferred from knowledge of its external outputs.
- Often confused with monitoring, observability goes a step further by identifying issues in a system and why those issues occur.
- It's from control theory and has become super crucial in software engineering, especially with the rise of complex, distributed systems and microservices.
- Observability collects data from metrics, logs, and traces, the three pillars, to give a complete view of a system's health and performance.

Observability



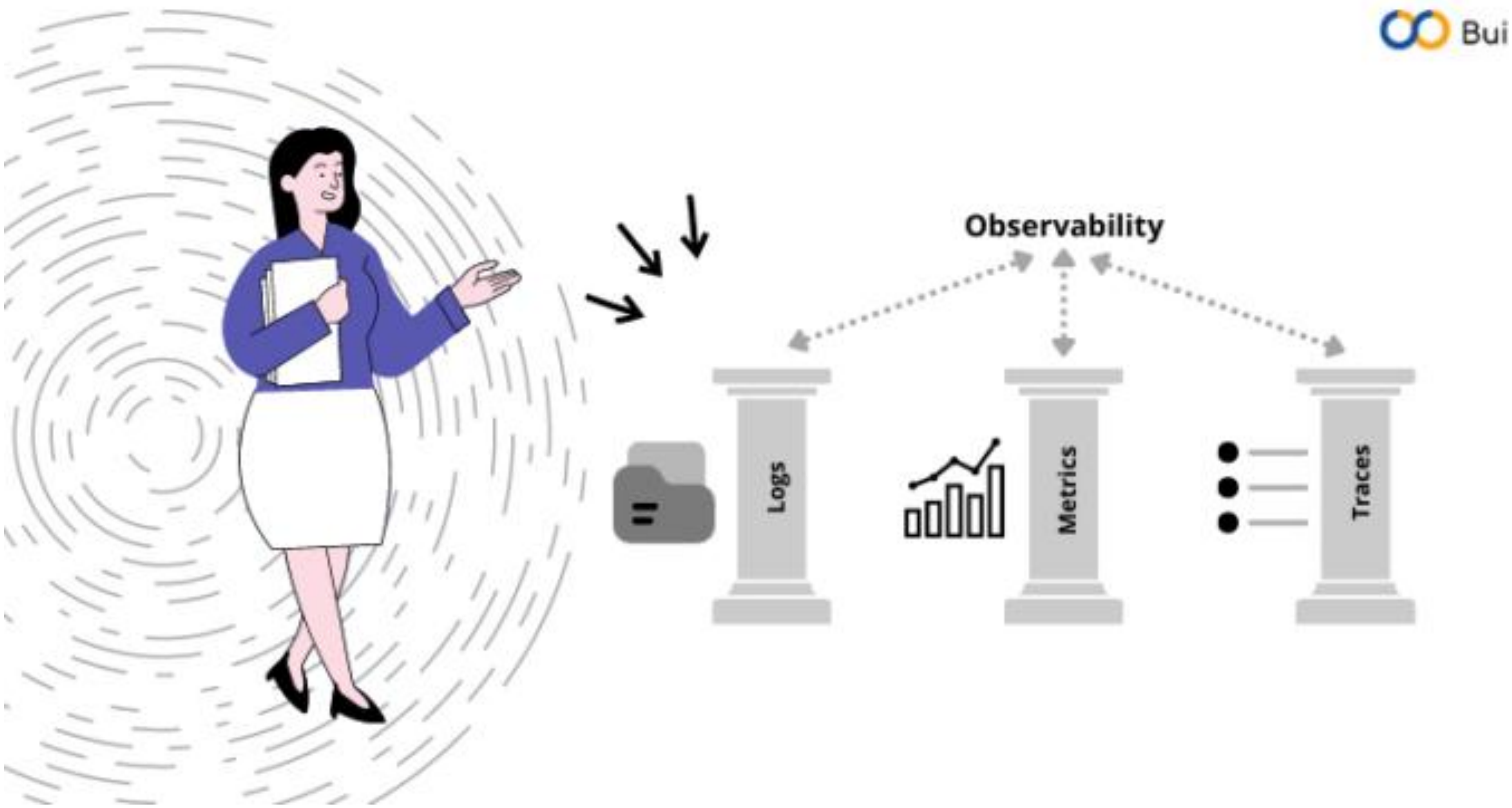
Observability

Microservices



Observability

Microservices



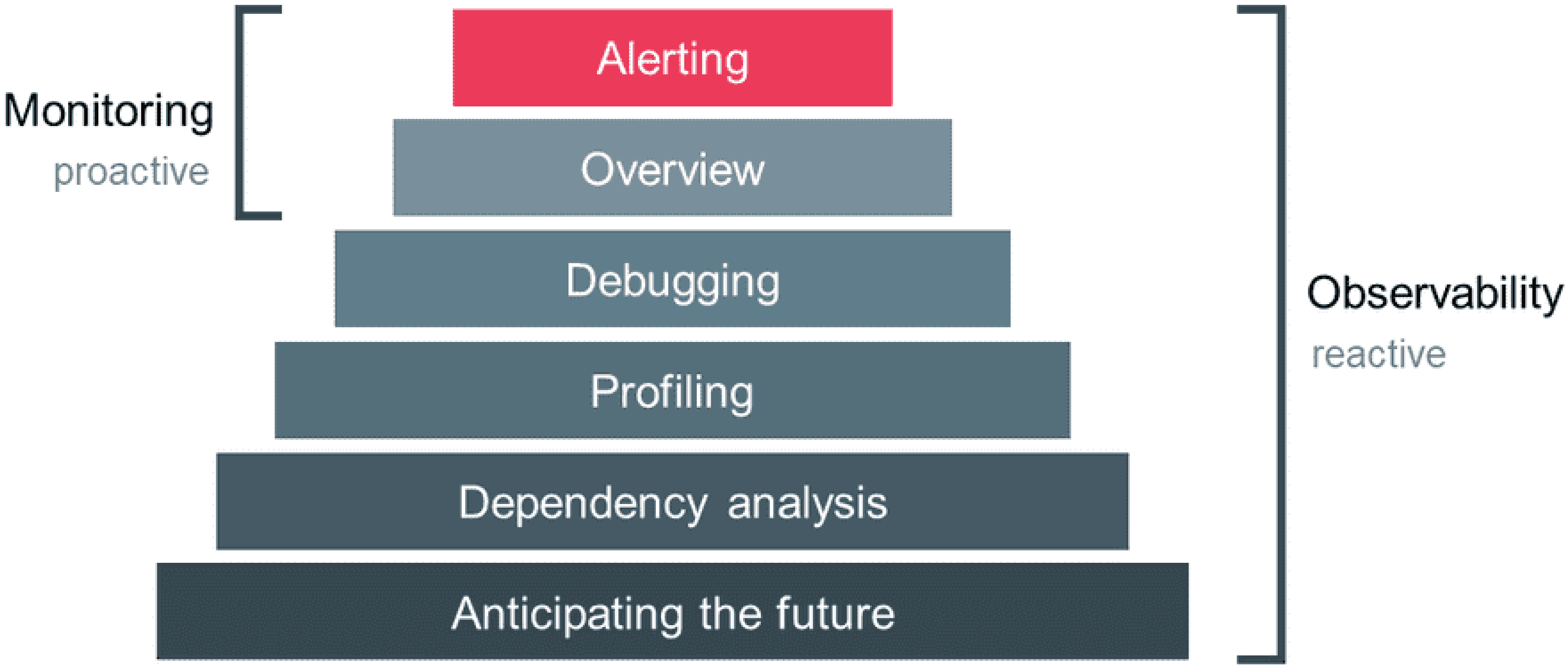
Need for Comprehensive Observability

Microservices

- Less downtime: Faster detection and fixing means reduced downtime and more customer service.
- Better decisions: With visibility into system performance, management can make better decisions on resource allocation and strategic improvements.
- Develop and run faster: Developers and ops teams can catch bottlenecks or failures before they become big problems.
- Better customer experience: Smoother system operation means less customer friction and better service overall.

Observability vs Monitoring

Microservices



Observability vs Monitoring

Microservices

Observability

- Tells you why a system is at fault
- Acts as a knowledge base in defining what to monitor
- Focuses on giving context to the data
- Gives a more complete assessment of the overall environment
- Observability is a traversable map
- It gives you complete Information
- Observability creates the potential to monitor different events

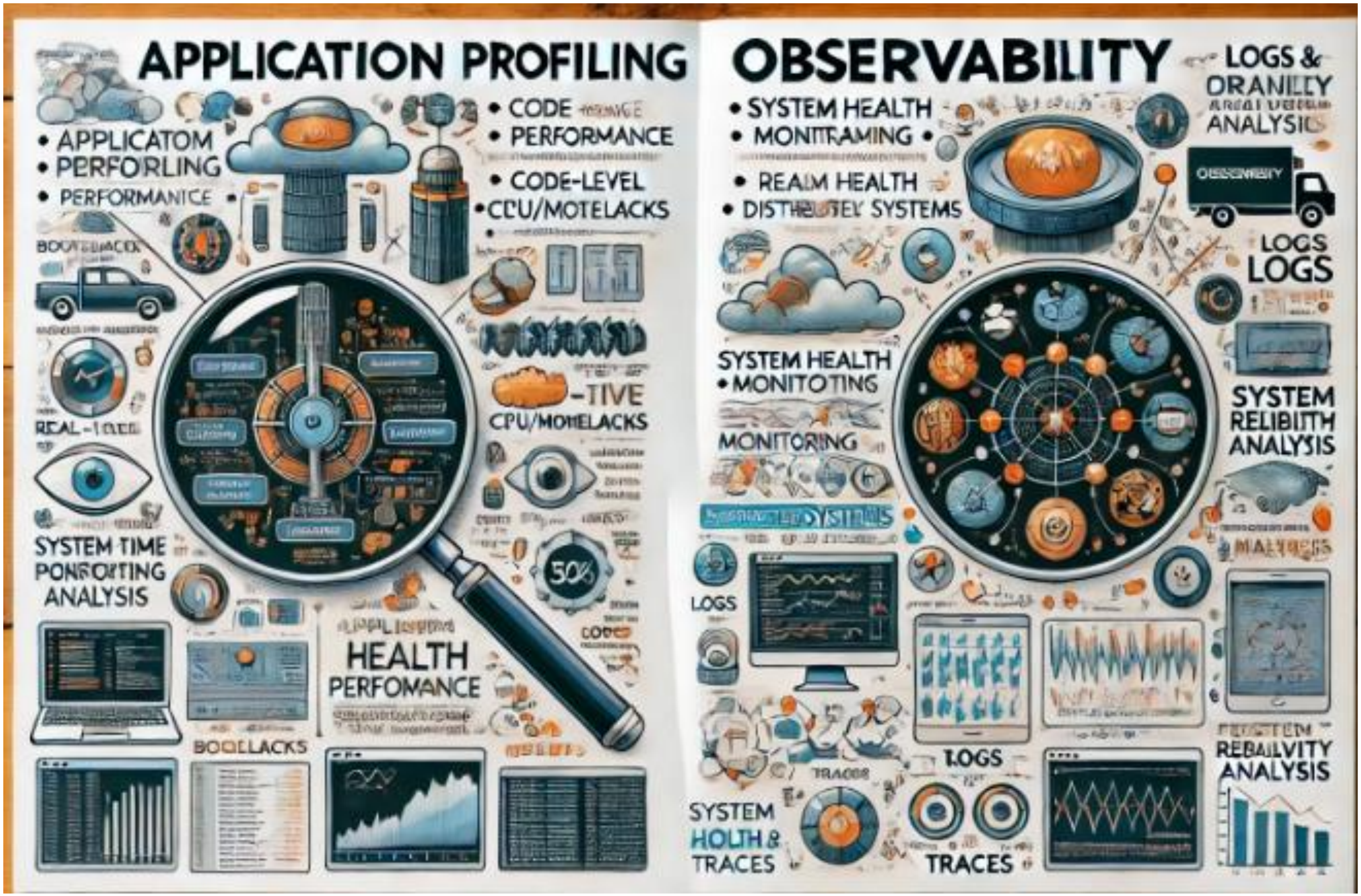
VS

Monitoring

- Notifies you that a system is at fault
- Focuses on monitoring the systems & discovering faults
- Focuses on collecting data
- Focuses on monitoring KPIs
- Monitoring is a single plane
- It gives you limited Information
- Monitoring is the process of using observability

Profiling

Microservices



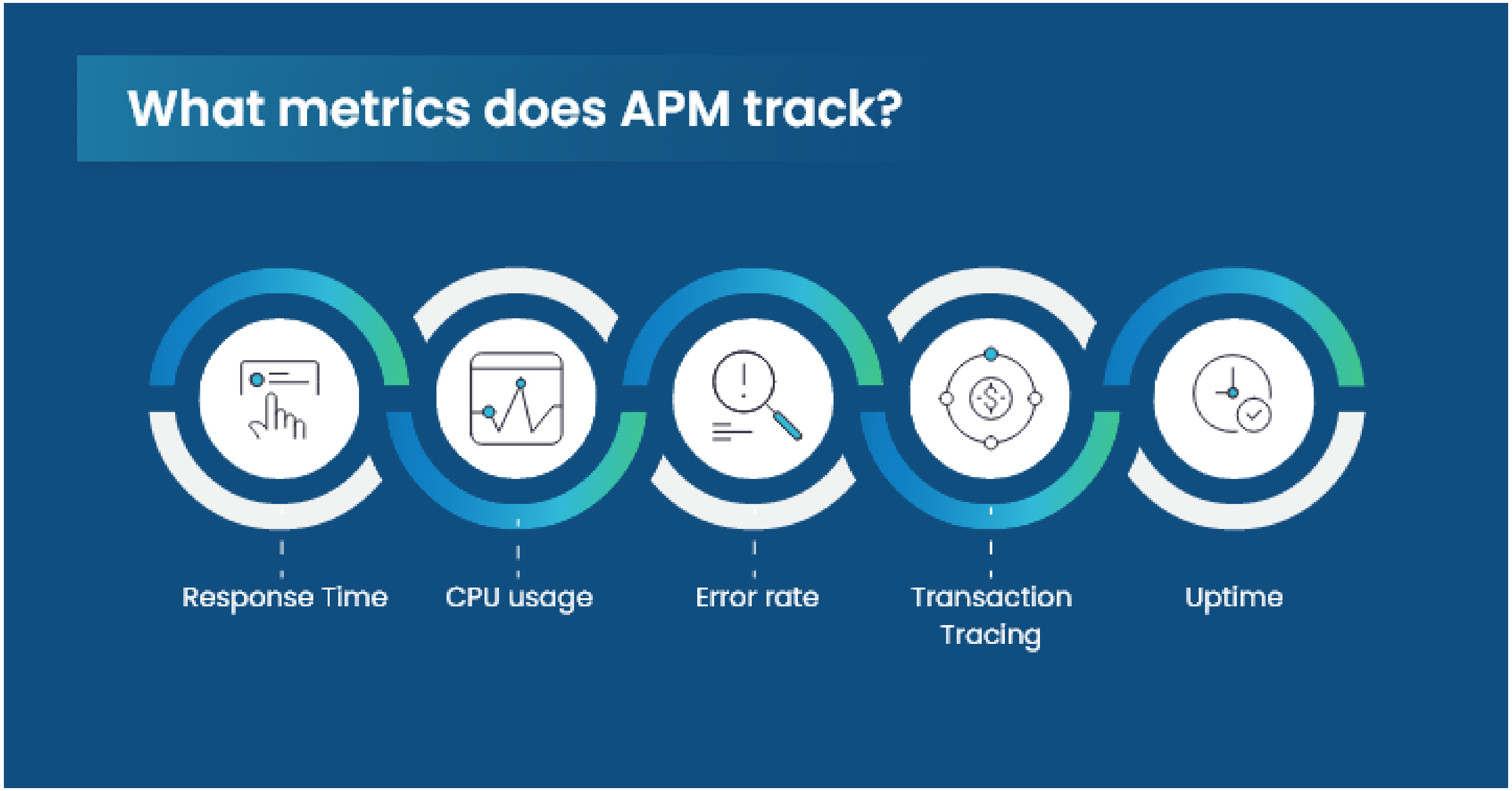
Profiling

Aspect	Application Profiling	Observability
Focus	Code-level performance and resource usage (CPU, memory, etc.)	System-wide behavior and health (logs, metrics, traces)
Purpose	Identify performance bottlenecks in specific functions or code	Provide real-time monitoring and insights into system health
Granularity	Highly granular, focused on individual threads or functions	Broader scope, focusing on entire applications or services
Timing	Typically done offline or in development	Continuous, real-time monitoring in production environments
Data Type	Low-level data such as CPU cycles, memory usage, and I/O	High-level data like logs, metrics, and distributed traces
Use Case	Debugging, performance optimization, resource tuning	Troubleshooting, detecting issues, and ensuring reliability
Tool Examples	Profilers (e.g., CPU profilers, memory profilers)	Monitoring tools (e.g., Prometheus, Grafana, Datadog)
Scope	Specific code segments or modules	Entire application stack or distributed systems
Insight	Provides detailed insights into code behavior	Provides insights into how the system behaves as a whole
Outcome	Identifies inefficiencies at the code level	Monitors and maintains system performance and availability

Metrics

- Metrics" refers to quantifiable measurements used to track and assess the performance or progress of a specific process, activity, or objective.
- Metrics are essential in various fields, including business, technology, healthcare, and education, to provide objective data for decision-making, analysis, and improvement.
- There are many types of metrics, depending on the context.

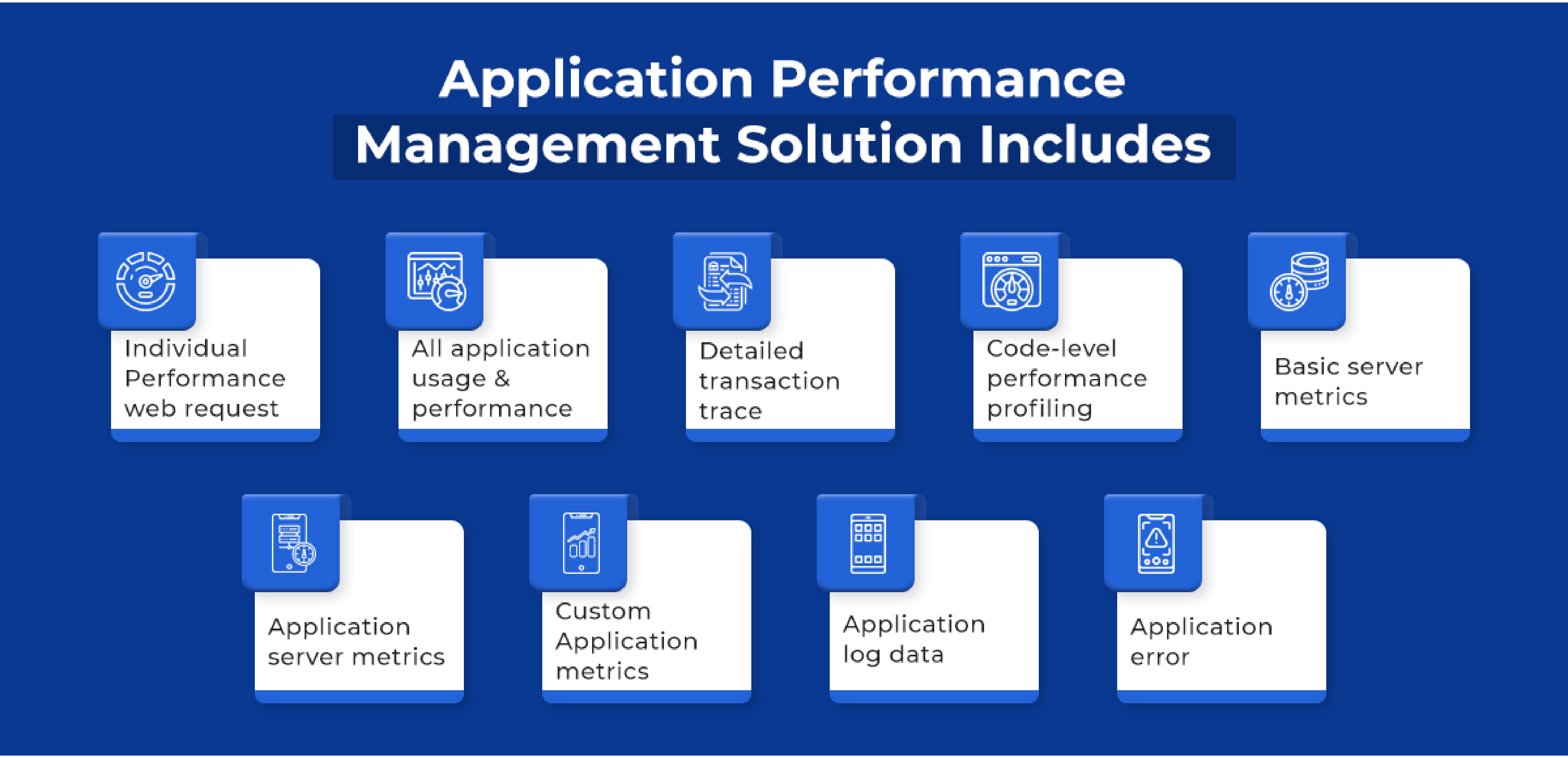
Metrics



What metrics does APM track?
Response Time
CPU usage
Error rate
Transaction Tracing
Uptime

Metrics

Microservices



Types of Metrics

Microservices

- **Business Metrics:**
  - **Revenue Growth:** Measures the increase in revenue over a certain period.
  - **Profit Margin:** Percentage of profit generated from total revenue.
  - **Customer Acquisition Cost (CAC):** The cost of acquiring a new customer.
  - **Net Promoter Score (NPS):** Measures customer satisfaction and likelihood to recommend a service/product.
  - **Churn Rate:** The percentage of customers who stop using a service/product.

Types of Metrics

- **Financial Metrics:**
  - **Return on Investment (ROI):** Measures profitability relative to the investment cost.
  - **Gross Profit Margin:** A company's total sales revenue minus its cost of goods sold (COGS).
  - **Debt-to-Equity Ratio:** A measure of a company's financial leverage.

Types of Metrics

- **Project Management Metrics:**
  - **On-time Delivery**: The percentage of projects delivered by the scheduled due date.
  - **Budget Variance**: The difference between the budgeted cost and the actual cost.
  - **Scope Changes**: Number or percentage of changes made to the project scope.

Types of Metrics

- **Technology/Software Development Metrics:**
  - **Velocity:** Measures the amount of work completed during a sprint (agile metric).
  - **Mean Time to Repair (MTTR):** Average time to fix a system after a failure.
  - **Defect Density:** The number of defects found in each amount of code.

Types of Metrics

- **Healthcare Metrics:**
  - **Patient Satisfaction Score:** Measures patient satisfaction with healthcare services.
  - **Average Length of Stay (ALOS):** The average number of days a patient spends in the hospital.
  - **Readmission Rates:** The percentage of patients who return to the hospital after discharge.

Types of Metrics

Microservices

- **Marketing Metrics:**
  - **Conversion Rate:** The percentage of visitors who take a desired action (e.g., purchasing).
  - **Click-through Rate (CTR):** The percentage of people who clicked on a link after seeing an ad or email.
  - **Cost Per Lead (CPL):** The average cost of acquiring a lead.

Types of Metrics

- **Human Resources Metrics:**
  - **Employee Turnover Rate:** The percentage of employees who leave a company over a given period.
  - **Absenteeism Rate:** Measures employee absences as a percentage of the workforce.
  - **Employee Engagement Score:** A measure of employees' emotional investment in their work.

Importance of Metrics

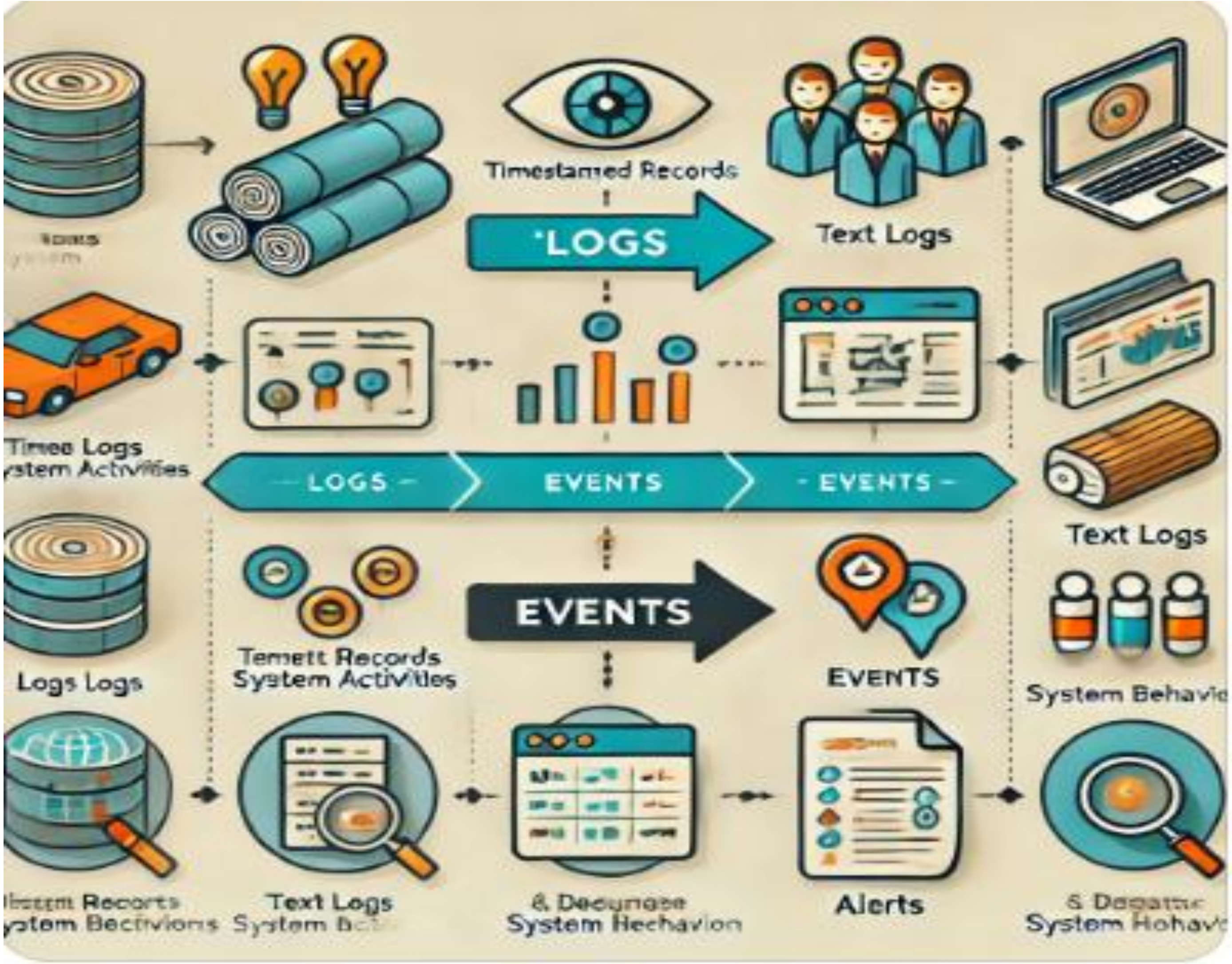
- Performance Measurement:
  - Metrics allow organizations to gauge how well they are meeting objectives or standards.
  - Decision-Making: With data from metrics, businesses can make informed decisions regarding strategy, resource allocation, and priorities.
  - Benchmarking: Metrics help compare performance over time or against competitors.
  - Continuous Improvement: Metrics enable tracking of progress and identification of areas needing improvement.

Understand logs and events

- Logs and events are essential components in monitoring and analyzing systems, networks, and applications.
- They provide critical insights into the performance, security, and operational status of infrastructure, helping in troubleshooting, auditing, and improving overall system health.
- Understanding the distinction between logs and events and how they are used can help us better interpret system behaviors.

Understand logs and events

Microservices



Understand logs and events

Microservices

Aspect	Logs	Events
Definition	A record of system activities captured over time	A specific action or occurrence within the system
Content	Contains detailed information such as timestamps, messages, errors, etc.	Describes a significant change or state in the system, like an alert or notification
Granularity	Typically continuous and detailed, capturing all system activities	Typically discrete, representing a single instance of change
Purpose	Used for tracking, diagnosing, and understanding system behavior over time	Used to trigger alerts, notifications, or actions based on system state
Use Case	Analyzing system health, debugging, and root cause analysis	Monitoring system state, triggering automation, or handling real-time responses
Storage	Stored as log files or in log management systems	Stored in event logs, metrics, or monitoring systems
Example	"2024-10-11 10:45:32: Error connecting to database."	"User login successful" or "Server down at 10:45 AM"
Relation to Monitoring	Provides context for events, giving detailed system history	Indicates when something of importance occurs
Tool Examples	Logstash, Fluentd, Splunk	Prometheus, Nagios, AWS CloudWatch Events
Time Aspect	Often continuous and spans over a long period	Discrete and represents a point-in-time occurrence

Understand logs and events

- Logs:
  - A log is a detailed, time-stamped record of activity generated by systems, applications, devices, or users.
  - Logs capture various types of information and are used for debugging, auditing, monitoring, and analyzing the state of a system.

Understand logs and events

- **Characteristics of Logs:**
  1. **Time-stamped:** Logs are usually recorded with the exact time and date when the action or event occurred.
  2. **Detailed Information:** Logs typically contain detailed information such as user actions, system processes, requests, errors, and system messages.
  3. **Continuous Generation:** Logs are continuously generated by systems, often creating large amounts of data over time.
  4. **Text-based:** Most logs are stored as plain text or in structured formats like JSON or XML.
  5. **Retained for Auditing:** Logs can be stored for long periods to serve as an audit trail of system activities.

Understand logs and events

Microservices

- **Types of Logs:**
  - **System Logs:** Captures operating system-related activities such as boot sequences, services starting or stopping, and system errors.
  - **Application Logs:** Records events within an application, such as user requests, database queries, and application errors.
  - **Security Logs:** Includes information about user logins, permissions, access control, and failed login attempts.
  - **Network Logs:** Logs data about network traffic, firewall rules, and packet exchanges.

Understand logs and events

- **Examples of Logs:**
  - Web server access logs capturing incoming HTTP requests.
  - System logs capturing user logins, reboots, or file access.
  - Error logs generated by applications when an error or crash occurs.

Understand logs and events

- **Events:**
  - An event is a specific occurrence or action within a system or application, often defined by triggers or conditions.
  - Events represent key moments that may require attention, such as a security alert, system crash, or significant change in state.

Understand logs and events

- **Characteristics of Events:**
  1. **Discrete Occurrences:** An event is usually tied to a specific action or change, such as a system startup, security breach, or completed transaction.
  2. **Higher-Level Information:** Events tend to summarize important activities, unlike logs which may capture more granular data.
  3. **Trigger-Based:** Events are often triggered by specific conditions or thresholds, such as a system reaching a performance limit, or a user accessing restricted data.
  4. **Alerting and Monitoring:** Events are often tied to monitoring systems that generate alerts, notifications, or reports for administrators.
  5. **Structured Data:** Events are often captured in a more structured way, helping systems and humans process them efficiently.

Understand logs and events

- **Examples of Events:**
  - A user successfully logs in to a system (authentication event).
  - A network intrusion detection system identifies a potential attack (security event).
  - A scheduled backup process successfully completes (system event).
  - An application generates a "transaction completed" event when a purchase is made.

Understand logs and events

- **Differences Between Logs and Events:**
  - **Granularity:** Logs are more granular and detailed, capturing every action or change in a system, while events highlight more significant occurrences.
  - **Purpose:** Logs are used for detailed analysis, diagnostics, and audit trails, whereas events focus on triggering alerts, monitoring system health, and identifying critical issues.
  - **Data Structure:** Logs are often unstructured or semi-structured (e.g., text files), while events tend to be structured and can be more easily parsed by monitoring tools.
  - **Storage:** Logs are often stored for a long time to track past actions, while events are typically used in real-time for immediate responses or alerts.

Understand logs and events

- **Use Cases of Logs and Events:**
  - **Troubleshooting:** When a system issue occurs, logs help pinpoint the exact sequence of actions that led to the issue.
  - For example, an error log in a web server could reveal why a particular page failed to load.
  - **Security Monitoring:** Security logs and events help detect suspicious behavior such as failed login attempts, unauthorized access, or malware activity.
  - Event monitoring tools can raise an alert if certain thresholds are breached (e.g., multiple failed login attempts).

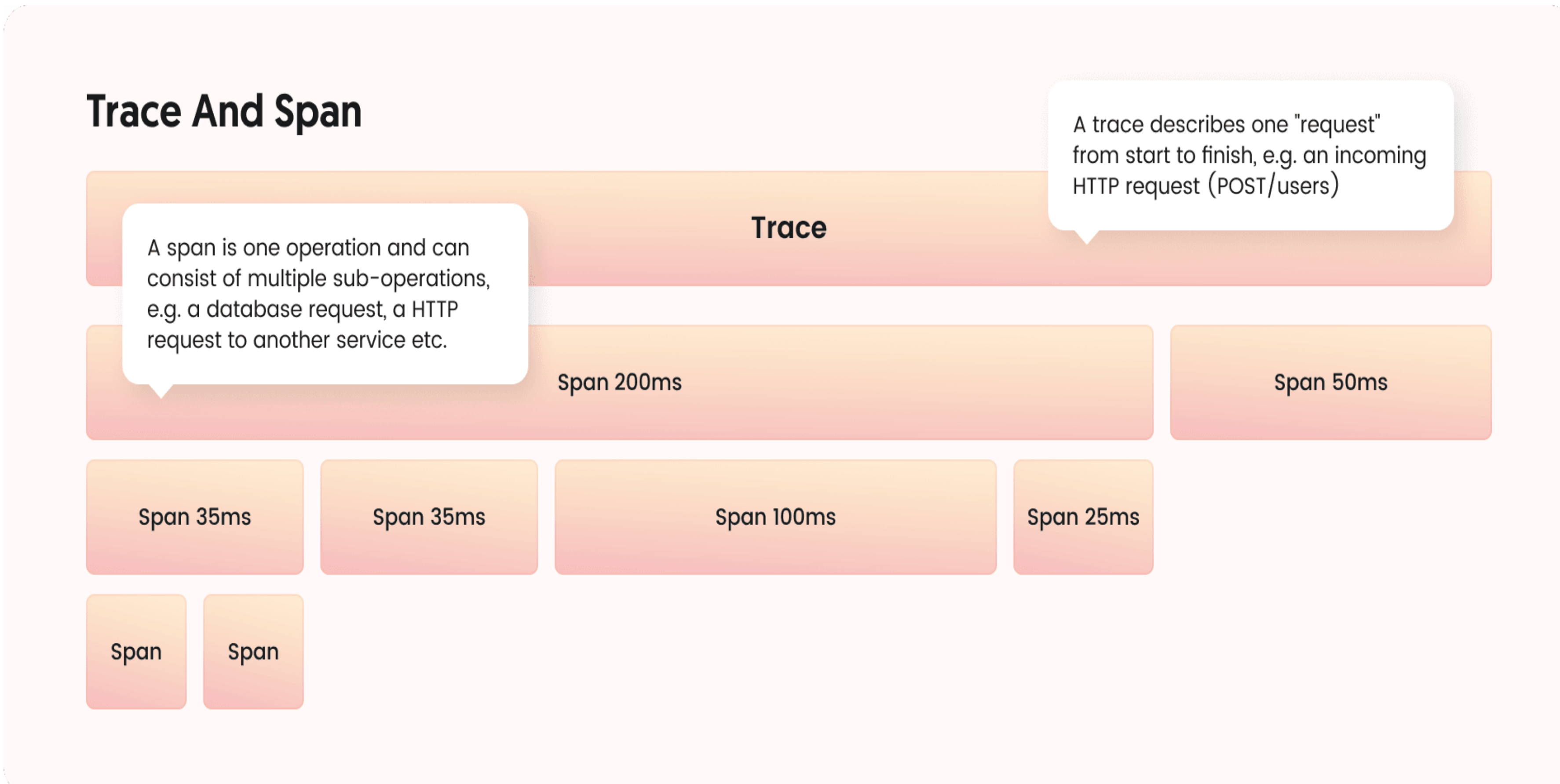
Understand logs and events

Microservices

- Performance Analysis:
  - Logs can capture metrics such as CPU usage, memory consumption, and network traffic, helping administrators detect performance bottlenecks or inefficient processes.
  - Compliance Auditing: Logs serve as a record of system activity, which is often required for regulatory compliance.
  - Logs can show who accessed data, when, and what actions were taken.
  - Real-Time Monitoring and Alerting: Event management systems (like SIEM or APM tools) continuously monitor systems for events that may require immediate action, such as security breaches or system outages.

Tracing and Spans

Microservices



Tracing and Spans

Microservices

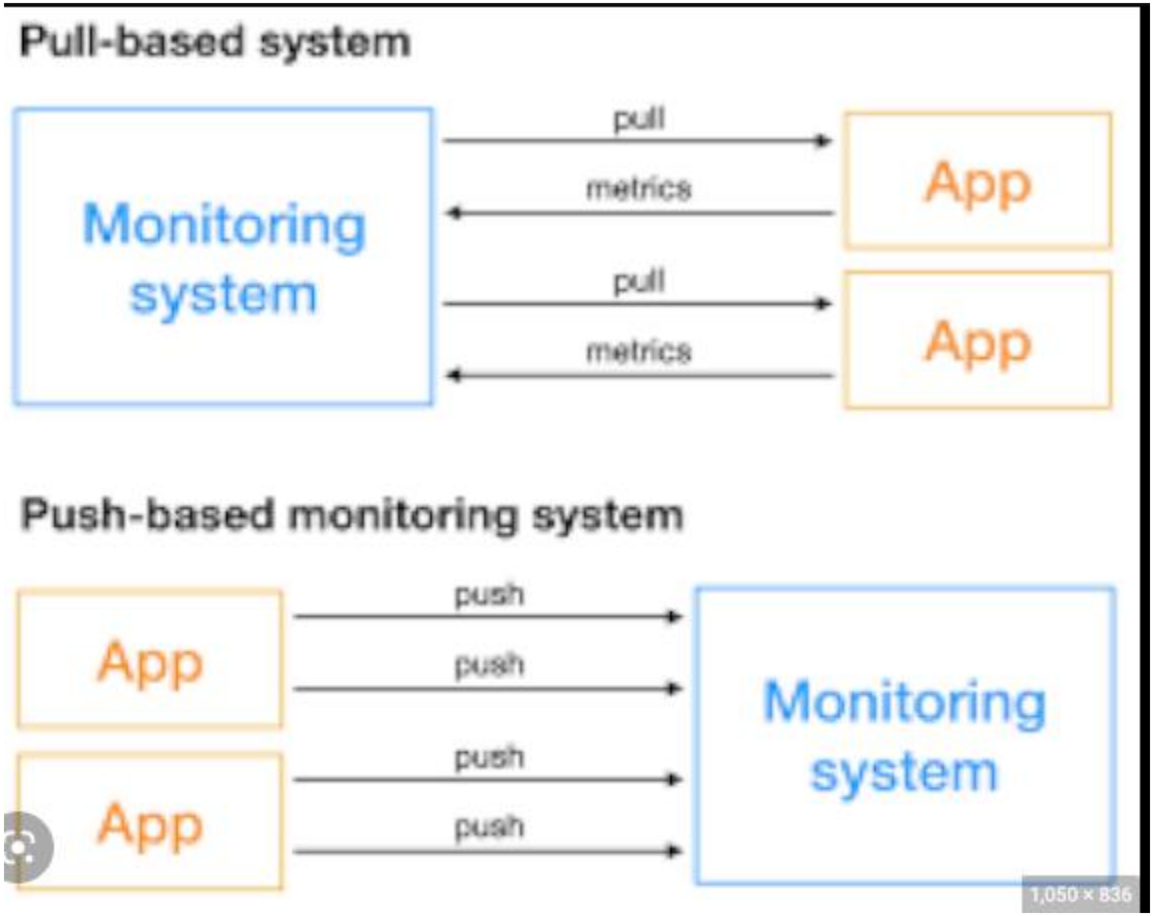
Aspect	Tracing	Spans
Definition	A method used to track the entire journey of a request as it moves through different services in a distributed system.	A single unit of work within a trace, representing a specific operation or step in the request's journey.
Scope	Captures the flow of a request across multiple systems or services, providing a high-level overview.	Represents a specific operation, such as a function call or database query, within the trace.
Granularity	High-level, focused on the end-to-end lifecycle of the request.	Fine-grained, focused on individual components or steps within the trace.
Purpose	To give visibility into how requests traverse through services, helping to identify bottlenecks, failures, or latency issues.	To break down the trace into specific operations, allowing detailed performance analysis for each step.
Hierarchy	A trace consists of multiple spans that represent the steps involved in processing a request.	A span is a segment of a trace, and spans can be nested or linked to show relationships between operations.

Tracing and Spans

Microservices

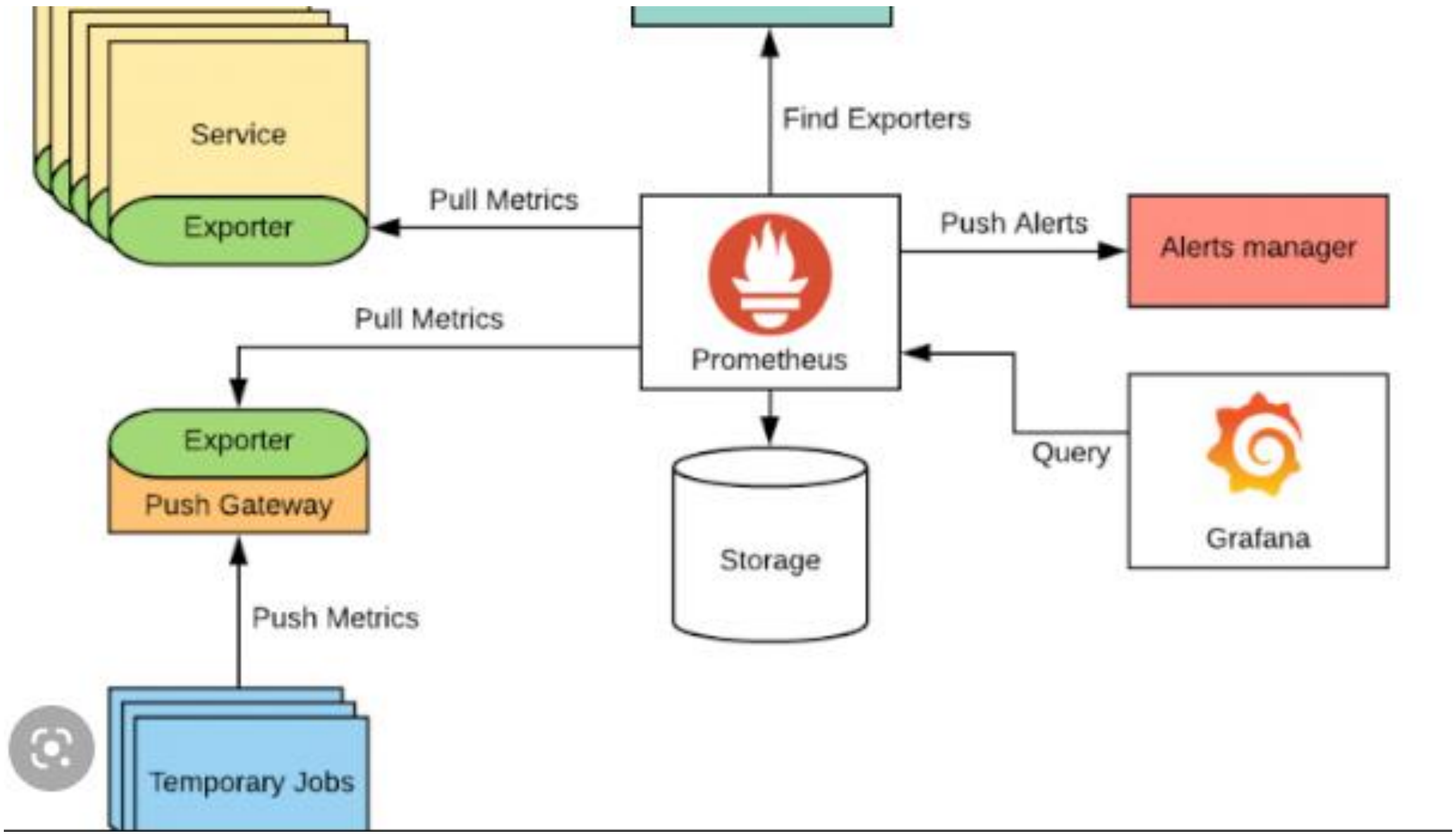
Data Captured	Collects metadata such as trace ID, timestamps, and relationships between spans.	Captures operation-specific details like start/end time, duration, and metadata (tags, logs).
Visualization	Typically visualized as a flow of connected spans, showing the end-to-end path of a request.	Visualized as individual nodes or blocks within the trace, often nested to show dependencies.
Use Case	Helps diagnose latency issues, track request flow, and identify problematic services in distributed architectures.	Helps to pinpoint delays or errors in specific operations within a service.
Example	A trace could show a user's request passing through a web server, an application server, and a database.	A span might represent the database query or the web server processing the request.
Tool Examples	Jaeger, OpenTelemetry, Zipkin	Spans are part of trace data in the same tools (Jaeger, Zipkin).

Push vs Pull



Pull-based system
Monitoring system
pull
metrics
pull
metrics
App
App
Push-based monitoring system
App
App
push
push
push
push
Monitoring system
1,050 × 836

Push vs Pull



Service
Exporter
Pull Metrics
Find Exporters
Push Alerts
Alerts manager
Pull Metrics
Prometheus
Exporter
Push Gateway
Storage
Query
Grafana
Push Metrics
Temporary Jobs

Push vs Pull

- Pull Model (Prometheus's Native Model)
- In the pull model, Prometheus regularly scrapes metrics from targets (services, servers, or applications) at specified intervals.
- It pulls metrics data by sending HTTP requests to a predefined endpoint (usually /metrics).
- How it works:
  - Prometheus server queries the /metrics endpoint of each target.
  - The target exposes metrics in a specific format that Prometheus understands.
  - Prometheus pulls data at intervals specified in its configuration.

Push vs Pull

Microservices

- Advantages:
  - Prometheus can control the frequency of data collection.
  - It can handle a dynamic list of targets and retry if one fails to respond.
  - Easier to monitor targets behind load balancers since Prometheus pulls from the available endpoint.
  - The target can remain stateless as it doesn't need to track which systems want its metrics.

Push vs Pull

- Disadvantages:
  - May not work well for short-lived jobs, as these may disappear before Prometheus scrapes their metrics.
  - Not ideal for environments where network access from Prometheus to the target is restricted.

Push vs Pull

- Push Model (Not Native to Prometheus but Achievable)
  - The push model involves the target sending (pushing) its metrics to a central location.
  - Prometheus does not natively support a push mechanism, but the Pushgateway component fills this gap.
  - The Pushgateway acts as an intermediary that receives metrics from applications (especially short-lived jobs) and exposes them to Prometheus.

Push vs Pull

- How it works:
  - Targets push their metrics to a Push gateway.
  - The Push gateway exposes a /metrics endpoint for Prometheus to scrape the pushed data.
  - The Push gateway stores the metrics until Prometheus pulls them.

Push vs Pull

- Advantages:
  - Suitable for short-lived jobs (e.g., batch jobs or CI jobs) that might finish execution before Prometheus can scrape them.
  - Can work in environments where the network setup restricts Prometheus from pulling metrics directly from targets (e.g., firewalls, private networks).
  - Useful when the target cannot keep an HTTP server running to expose metrics continuously.

Push vs Pull

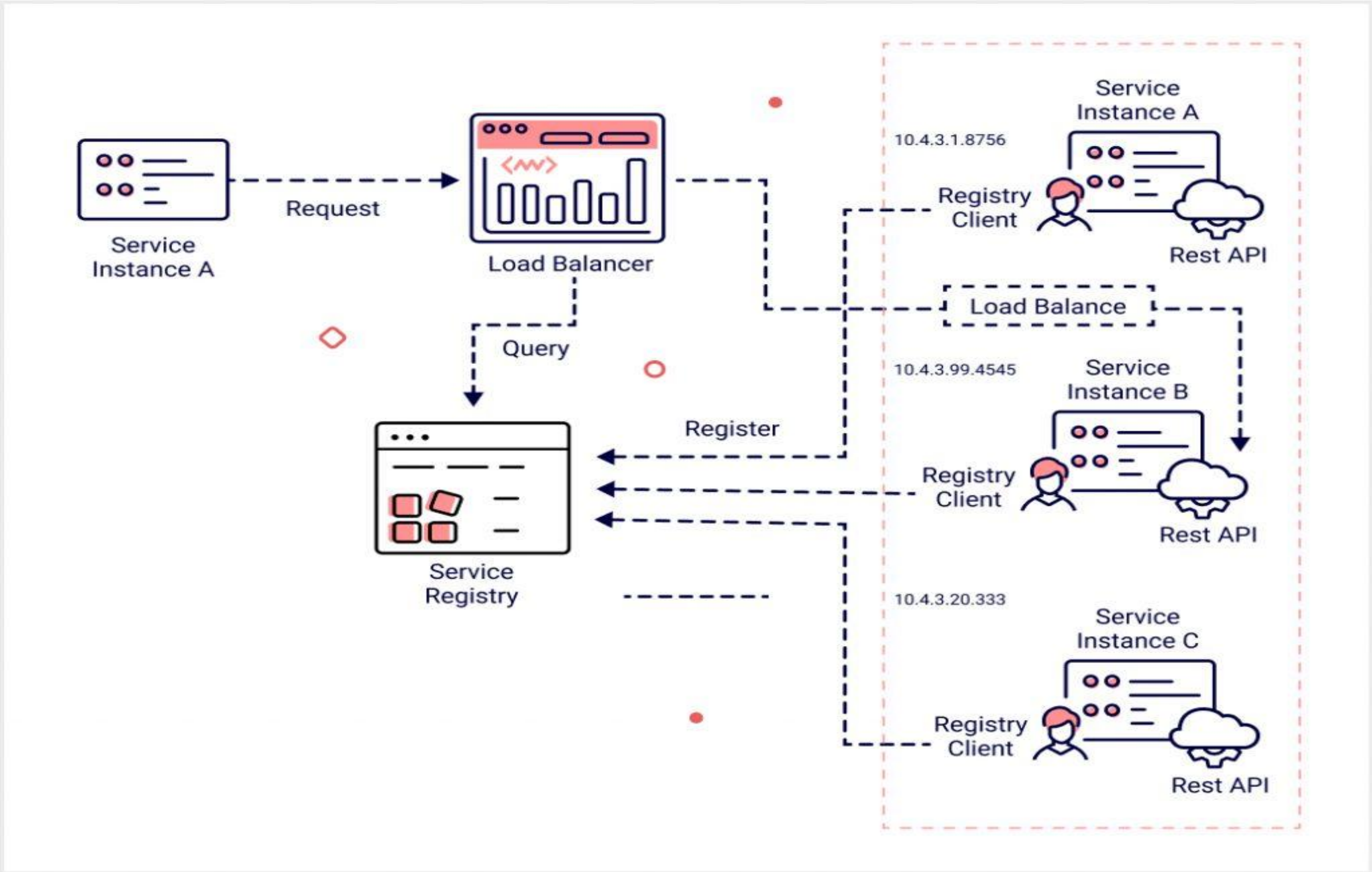
- Disadvantages:
  - Introduces an extra component (Push gateway), which adds complexity.
  - The state of the Push gateway must be managed, as it retains data until explicitly deleted.
  - Push gateway is not designed for continuous metrics (e.g., long-lived services); it's primarily intended for short-lived jobs.

Service Discovery

- Service discovery is a key concept in microservices and distributed systems.
- It refers to the process of automatically detecting and connecting microservices in a network.
- In these environments, services communicate with each other, and their locations (IP addresses, ports) may change dynamically due to scaling, failures, or deployments.
- Service discovery helps maintain seamless communication between services despite these changes.

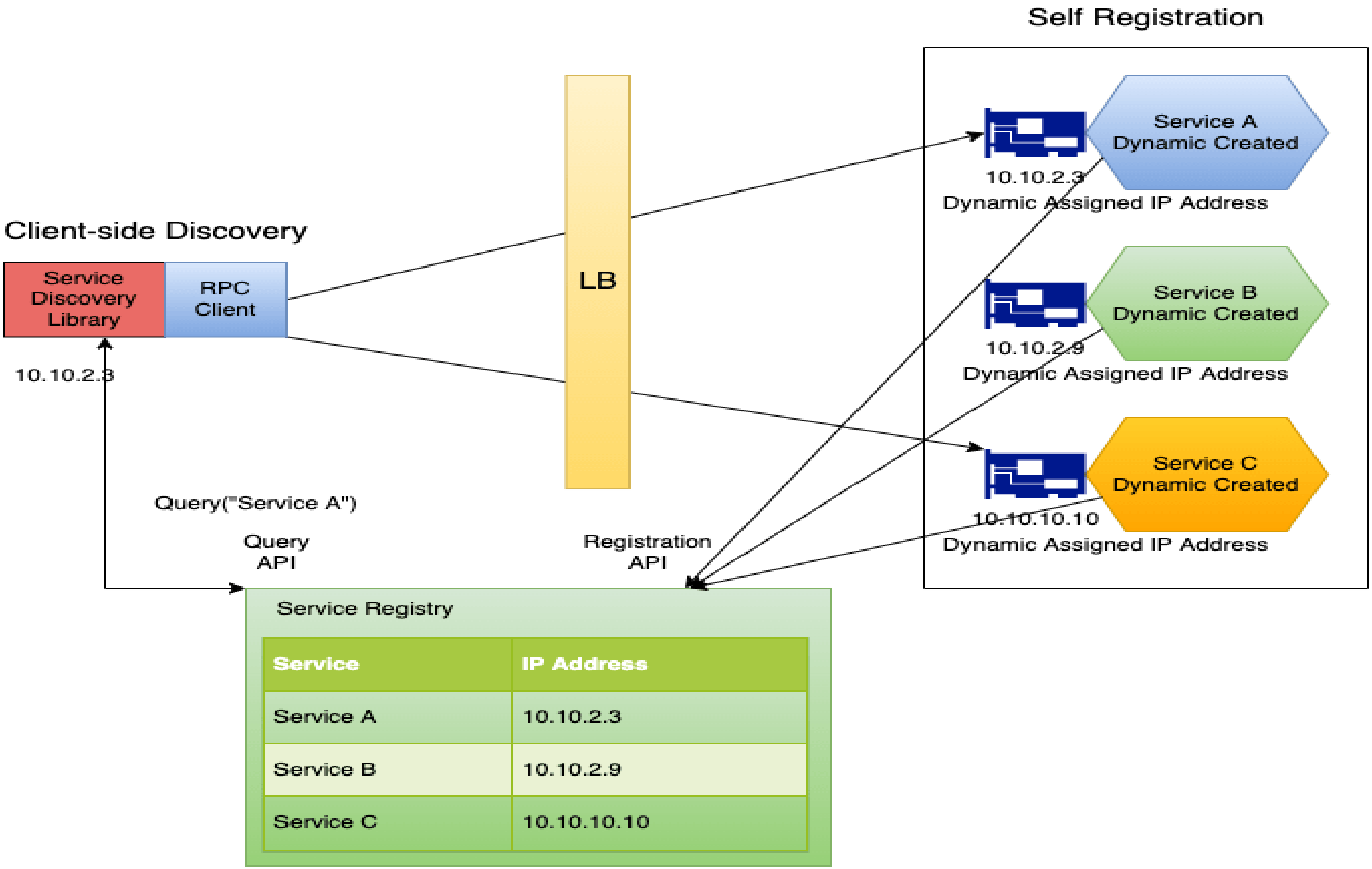
Service Discovery

Microservices



Service Discovery

Microservices



Service Discovery

- 1. Client-side discovery:
  - In client-side discovery, the client (the service that wants to make a request) is responsible for determining the location of service instances.
  - A service registry contains the addresses of all available services.
  - The client queries the registry to find a valid service instance and then sends a request directly to the selected instance.

Service Discovery

- **Components:**
  - **Service Registry:** A centralized database that tracks all instances of services.
  - **Client:** The service that queries the service registry and selects a service instance.
  - **Examples:**
  - Netflix Eureka (client-side service discovery)
  - Consul (can be used for client-side)

Service Discovery

- **Server-side discovery:**
  - In server-side discovery, the client sends a request to a load balancer, which in turn queries the service registry and forwards the request to an appropriate service instance.
  - The client is unaware of where the actual service instance is located; it simply sends the request to the load balancer.

Service Discovery

- **Components:**
  - **Service Registry:** Stores the locations of services.
  - **Load Balancer/Service Proxy:** Acts as an intermediary between the client and service instances.
  - **Examples:**
  - AWS Elastic Load Balancing (ELB)
  - Kubernetes (through the use of DNS or environment variables)

Service Discovery

- **Common Technologies for Service Discovery:**
  - **Eureka:** A service discovery tool developed by Netflix for client-side discovery.
  - **Consul:** A service mesh and service discovery tool from HashiCorp that supports both client-side and server-side discovery.
  - **Kubernetes:** Uses internal DNS or environment variables for service discovery within a cluster.
  - **Zookeeper:** Often used with Apache frameworks for service discovery.

Service Discovery

Microservices

- **How it Works:**
  - **Service Registration:** When a new service instance starts, it registers itself with the service registry, providing details like IP address, port, and metadata.
  - **Service Deregistration:** When a service instance stops, it deregisters itself from the registry to avoid sending traffic to unavailable instances.
  - **Health Checks:** The registry may periodically perform health checks to ensure that the registered services are available and healthy.

Service Discovery

- **Benefits of Service Discovery:**
  - **Scalability:** Helps services find and communicate with each other dynamically, supporting high levels of scaling.
  - **Resilience:** Allows systems to adapt to changes in the environment, such as instances going down or new ones being added.
  - **Flexibility:** Can support different deployment environments (cloud, containers, on-premises).

SLA vs SLO vs SLI

SLA → SERVICE LEVEL AGREEMENT
the agreement you make with your clients or users

SLOs → SERVICE LEVEL OBJECTIVES
the objectives your team must hit to meet that agreement

SLIs → SERVICE LEVEL INDICATORS
the real numbers on your performance

Basics of SLOs, SLAs, and SLIs

- SLA (Service Level Agreement):
  - An SLA is a formal agreement between a service provider and a customer.
  - It defines the specific measurable service performance standards that the provider is obligated to meet.
  - SLAs are typically legally binding and include consequences (such as penalties or service credits) if the service provider fails to meet the agreed-upon standards.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Sets expectations for service performance and outlines responsibilities between the provider and customer.
  - **Enforceability:** Often legally binding with penalties for non-compliance.
  - **Example:** "The service provider guarantees 99.9% uptime over a month. If this is not met, the customer will receive a 5% refund of the monthly service fee."

Basics of SLOs, SLAs, and SLIs

- **SLO (Service Level Objective):**
  - An SLO is a specific, measurable goal that defines the acceptable performance level of a service.
  - SLOs are part of the SLA but are less formal than the SLA itself.
  - They act as internal targets for the service provider, helping to ensure that the SLAs are met.
  - The SLO is used to define performance objectives like availability, latency, or response times.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Sets measurable objectives that help the service provider track and maintain service levels.
  - **Measurable:** Clearly defined metrics such as uptime percentage, error rates, or latency thresholds.
  - **Example:** "The system should have an uptime of 99.9% each month."

Basics of SLOs, SLAs, and SLIs

Microservices

- **SLI (Service Level Indicator):**
  - An SLI is the actual measurement or metric that indicates how well a service is performing against the SLOs.
  - SLIs are quantitative measurements, typically expressed as percentages or time units, that track specific characteristics of a service (such as availability, latency, throughput, or error rate).
  - SLIs provide the data that allows a provider to understand whether they are meeting their SLOs and SLAs.

Basics of SLOs, SLAs, and SLIs

- **Key Points:**
  - **Purpose:** Provides real-time metrics or measurements of service performance.
  - **Monitoring:** Often collected through monitoring tools and compared against SLOs.
  - **Example:** "The service achieved 99.95% uptime last month" (where uptime is the SLI).

Open Telemetry

Open Telemetry (OTel) is an **open-source observability framework** that provides tools, APIs, and SDKs for collecting **metrics**, **logs**, and **traces** from applications.

It's a **unified standard** for instrumenting your software to monitor performance and diagnose issues.

- **Backed by CNCF** (Cloud Native Computing Foundation)
- **Combines** the efforts of OpenTracing and OpenCensus.

Open Telemetry

a. Telemetry Data Types

- **Traces**: Represent the flow of requests through systems (Distributed Tracing).
- **Metrics**: Numeric values measured over time (e.g., CPU usage, request count).
- **Logs**: Textual records of events.

b. Signals

- Each telemetry data type is considered a **signal**.
- OTel supports **multi-signal collection**.

OpenTelemetry Components

a. SDKs & APIs

- Language-specific SDKs for **Java**, **Python**, **Go**, **C#**, etc.
- Used for manual instrumentation or auto-instrumentation.

b. OpenTelemetry Collector

- A **vendor-agnostic** agent that receives, processes, and exports telemetry data.
- Supports **receivers**, **processors**, **exporters**.
- Can **transform**, **filter**, or **batch** telemetry data.
- Can run as an **agent** (on host) or **gateway** (centralized).

Open Telemetry Components

a. SDKs & APIs

- Language-specific SDKs for **Java**, **Python**, **Go**, **C#**, etc.
- Used for manual instrumentation or auto-instrumentation.

b. OpenTelemetry Collector

- A **vendor-agnostic** agent that receives, processes, and exports telemetry data.
- Supports **receivers**, **processors**, **exporters**.
- Can **transform**, **filter**, or **batch** telemetry data.
- Can run as an **agent** (on host) or **gateway** (centralized).

Open Telemetry Components

c. Exporters

- Send telemetry data to backends like:
  - **Jaeger**, **Zipkin** (for traces)
  - **Prometheus**, **Grafana Mimir** (for metrics)
  - **ElasticSearch**, **Splunk**, **Datadog**, etc.

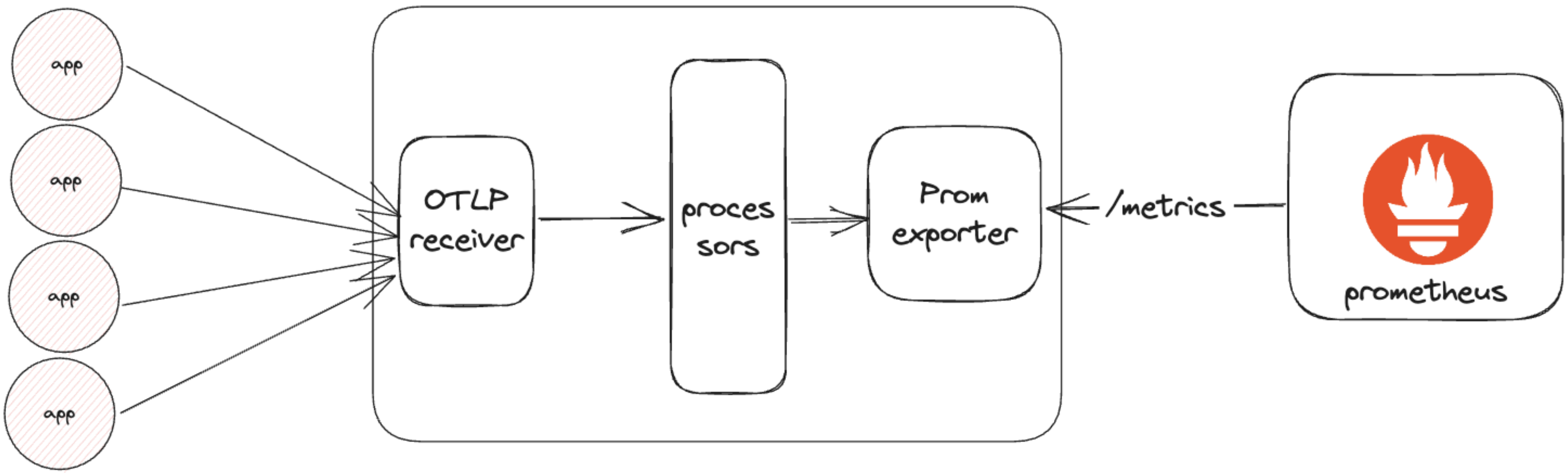
Open Telemetry Collector

- The Open Telemetry Collector is a vendor-agnostic service designed to receive, process, and export telemetry data like traces, metrics, and logs from applications and infrastructure.
- It's a key component in observability pipelines.

Open Telemetry Collector

Microservices

Open Telemetry Collector



Open Telemetry Collector

Objectives

- *Usability*: Reasonable default configuration, supports popular protocols, runs and collects out of the box.
- *Performance*: Highly stable and performant under varying loads and configurations.
- *Observability*: An exemplar of an observable service.
- *Extensibility*: Customizable without touching the core code.
- *Unification*: Single codebase, deployable as an agent or collector with support for traces, metrics, and logs.

Open Telemetry Collector

Microservices

Key Features:

1. **Receives Data**:
   1. Supports **OTLP**, **Jaeger**, **Prometheus**, **Zipkin**, **FluentBit**, etc.
   2. Handles **metrics, traces**, and **logs**.
2. **Processes Data**:
   1. Add attributes, filter, batch, transform telemetry.
   2. Supports pipelines with **processors** (e.g., sampling, filtering, batch).
3. **Exports Data**:
   1. To backend systems like **Prometheus**, **Grafana Mimir**, **Jaeger**, **Datadog**, **New Relic**, **Elasticsearch**, etc.
4. **Extensible**:
   1. You can add custom receivers, processors, or exporters.
   2. Distributions like **otel-contrib** include community-contributed components.

Open Telemetry Collector

Microservices

Core Components:

- **Receivers**: Ingest telemetry data.
- **Processors**: Modify/enrich telemetry.
- **Exporters**: Send telemetry to observability platforms.
- **Extensions**: Support services like health checks, pprof, zpages.

Open Telemetry Collector

Collector Modes:

1. **Agent**: Runs as a **sidecar** or on the host; close to the app.
2. **Gateway**: Runs as a **central service**; receives from multiple agents or apps.

Open Telemetry Collector

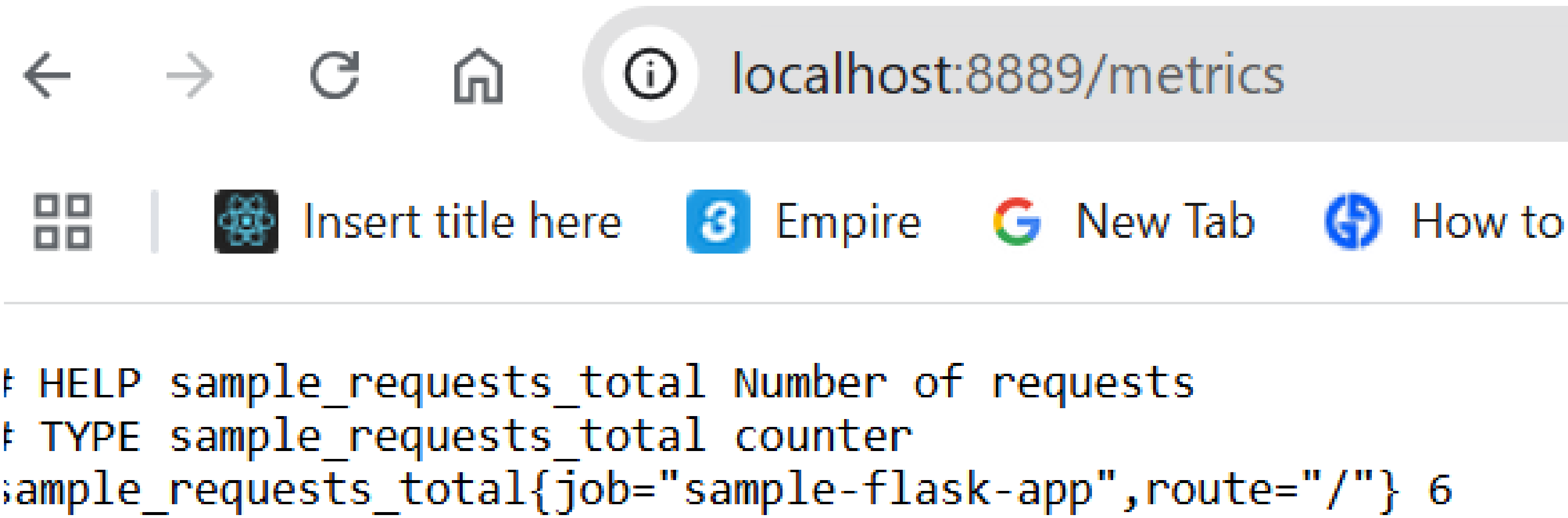
Microservices

Common Use Cases:

- Collect **traces** from microservices and export to **Jaeger**.
- Collect **metrics** from applications and export to **Prometheus**.
- Process logs and send them to **Elasticsearch**.

Open Telemetry Collector

Microservices



Telegraf

Microservices

- Telegraf is a server agent for collecting and reporting metrics.
- It is an agent for collecting and sending metrics and events from a wide array of sources to databases like InfluxDB, Prometheus, or via OpenTelemetry.
- It is part of the TICK stack, which includes Telegraf, InfluxDB, Chronograf, and Kapacitor.
- Developed by InfluxData, Telegraf is designed to be easily extendable, with a wide variety of plugins available to gather metrics from different systems and services.

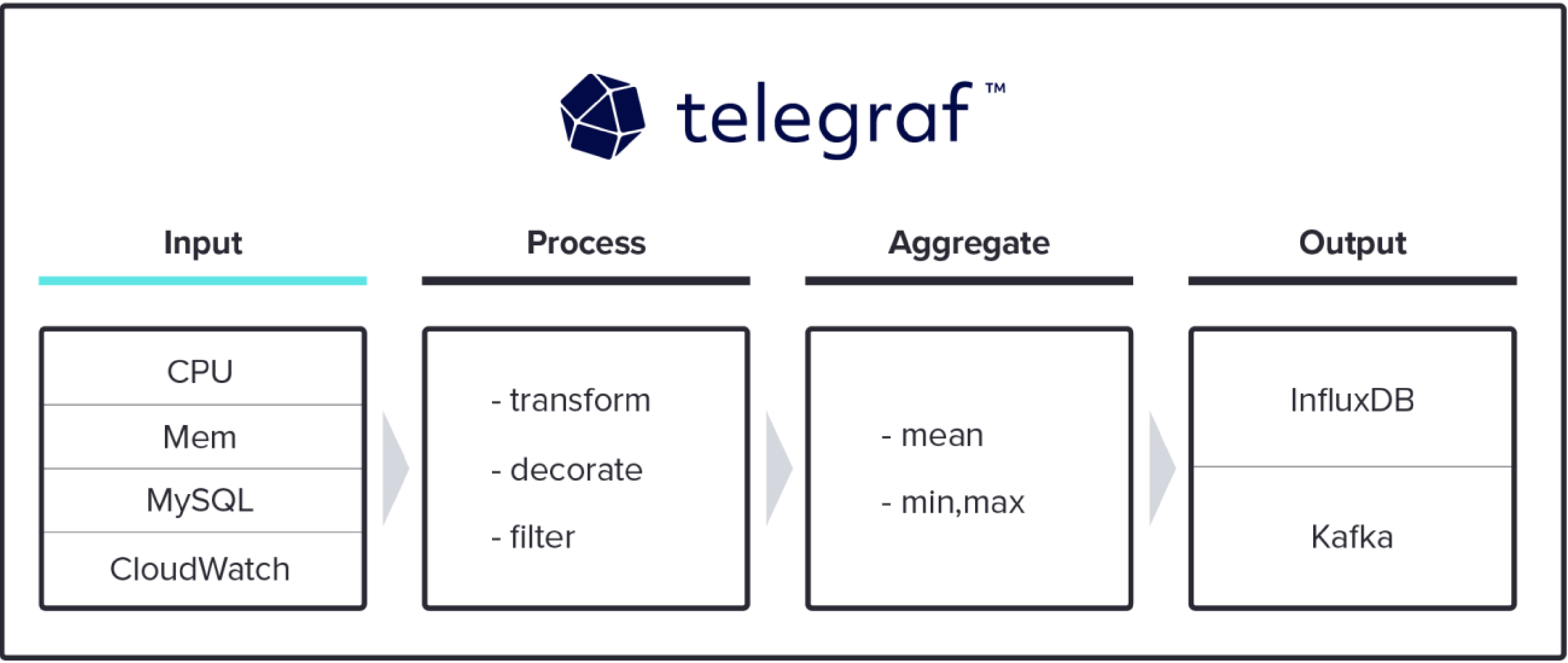
Why Use Telegraf?

Microservices

- Telegraf offers several key benefits:
- Extensive Plugin Ecosystem: Telegraf supports over 200 plugins for collecting data from various sources, including databases, systems, and applications.
- Ease of Use: It is simple to install and configure, making it accessible for beginners.
- Efficient: Telegraf is lightweight and designed to handle high-throughput metrics collection with minimal resource usage.
- Integration with InfluxDB: Seamless integration with InfluxDB for storing and querying time-series data.

Installing Telegraf

Microservices



Telegraf

Key Features:

- **Plugins:** 300+ input, output, and processor plugins.
- **Input Sources:** System stats, logs, databases, services (Docker, Kubernetes).
- **Output Options:** InfluxDB, Prometheus, Kafka, OpenTelemetry.
- **Lightweight & Efficient:** Suitable for edge and server environments.

Telegraf

Microservices

Use Case:

- Collect system-level metrics (CPU, memory, disk, network).
- Send custom metrics to monitoring backends.
- Acts as a **bridge** between services and monitoring tools.

Grafana Mimir

Microservices

Overview:

- Grafana Mimir is a **highly scalable, long-term storage backend for Prometheus metrics**, optimized for large-scale environments.

Grafana Mimir

Microservices

Key Features:

- **Horizontal Scalability:** Handle massive amounts of Prometheus metrics.
- **Multi-Tenancy:** Supports isolated metric storage for different users/teams.
- **High Availability:** Supports sharding and replication.
- **Integration:** Works seamlessly with Grafana for visualization.

Grafana Mimir

Microservices

Use Case:

- Store and query time-series metrics for long-term analysis.
- Backend for Prometheus setups needing high availability and scalability.
- Centralized metric storage across multiple Prometheus instances.

Integration Workflow Example

1. **Instrumentation:**
   1. Use **OpenTelemetry SDK** in applications to capture traces and metrics.
   2. Use **Telegraf** to gather system-level metrics.
2. **Data Collection:**
   1. Send telemetry data to **OpenTelemetry Collector**.
   2. Telegraf sends metrics directly to **Prometheus** or **OTel Collector**.
3. **Storage:**
   1. OpenTelemetry Collector exports metrics to **Prometheus Remote Write** or **Grafana Mimir**.
   2. Telegraf can also push to **Mimir** (via Prometheus output plugin).
4. **Visualization:**
   1. **Grafana** reads from **Mimir** to visualize metrics and traces.

Comparison of Roles

Microservices

Tool	Primary Role	Data Type
OpenTelemetry	Collects app-level telemetry (metrics/traces/logs)	Metrics, Traces, Logs
Telegraf	Collects system/infrastructure metrics	Metrics
Grafana Mimir	Stores Prometheus metrics at scale	Metrics

Real-World Scenario Example

Microservices

- Microservice App:
  - Uses Open Telemetry to generate traces and metrics.
- Host Server:
  - Runs Telegraf to collect CPU, memory, and disk metrics.
- Both push data to Open Telemetry Collector.
- Collector exports data to Grafana Mimir.
- Metrics and traces are visualized in **Grafana Dashboards**.