Observability-Driven Software Engineering

Wahab Hamou-Lhadj
Concordia University
Montréal, QC, Canada

Naser Ezzati-Jivan
Brock University
St. Catharines, ON, Canada

Keynote Presentation

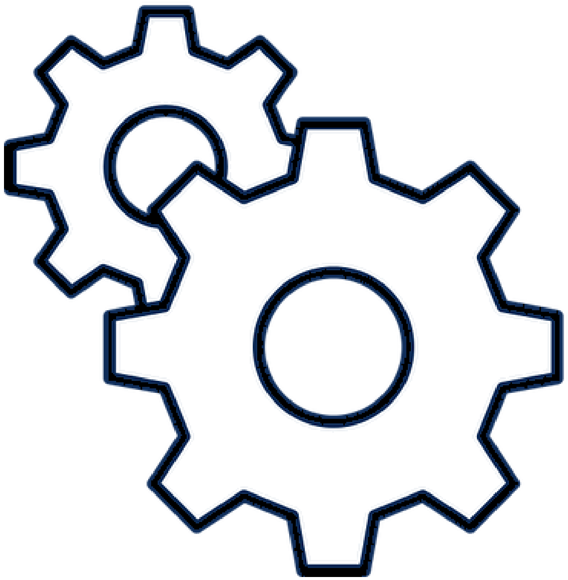
5th International Conference on Wireless, Intelligent, and Distributed Environment for Communication (WIDECOM)
Windsor, ON, Canada
October 12, 2022

User vs. Operational Data

- **User data** describes information about users.
  - E.g. social media data, user preferences, geo-location data, images, etc.
  - Applications include marketing campaigns, fraud detection, image recognition, etc.

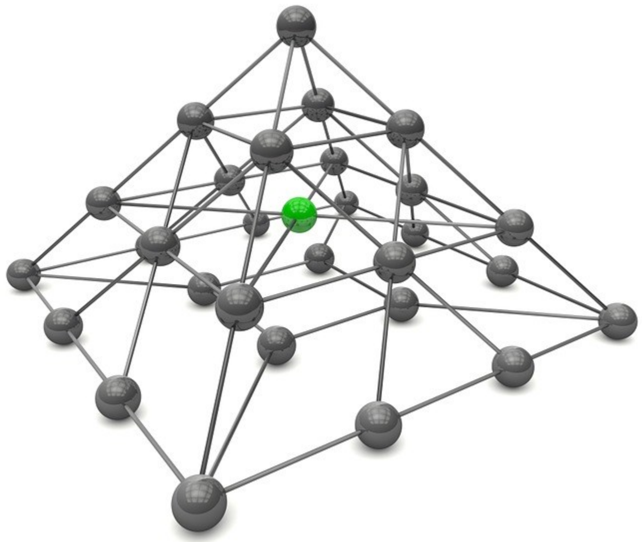
User vs. Operational Data

- **Operational (machine) data** describes information about a system (or a machine)
- It is collected automatically from devices, IT platforms, applications with no direct user intervention.
    - Useful for diagnosing service problems, ensuring reliability, detecting security threats, improving operations, and so on.



Operational Data for Software-Intensive Systems

- The proper functioning of software-intensive systems **relies heavily on operational data** to diagnose and prevent problems.
- New trends in SW dev. make this challenging:
  - Highly distributed and parallel systems
  - Micro-service architectures
  - Virtualisation and containerization
  - Device connectivity and IoT
  - Cyber physical systems
  - Intelligent and autonomous systems
  - Agile, DevOps, and continuous delivery processes

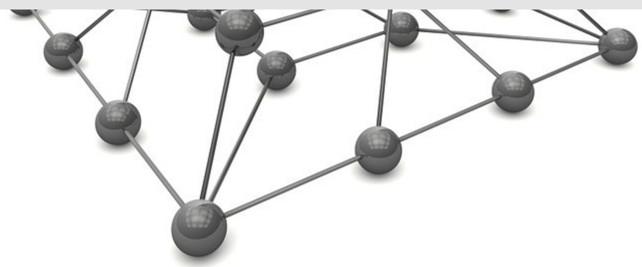


Operational Data for Software-Intensive Systems

- The proper functioning of software-intensive systems **<u>relies heavily on operational data</u>** to diagnose and prevent problems.

We need better runtime system analysis and fault diagnosis and prediction methods that provide full visibility of a system's internal states.

- Virtualisation and containerization
- Device connectivity and IoT
- Cyber physical systems
- Intelligent and autonomous systems
- Agile, DevOps, and continuous delivery processes



Software Observability

- In control theory:
  - **Observability** is "a measure of how well internal states of a system can be inferred from knowledge of its external outputs" [Wikipedia]

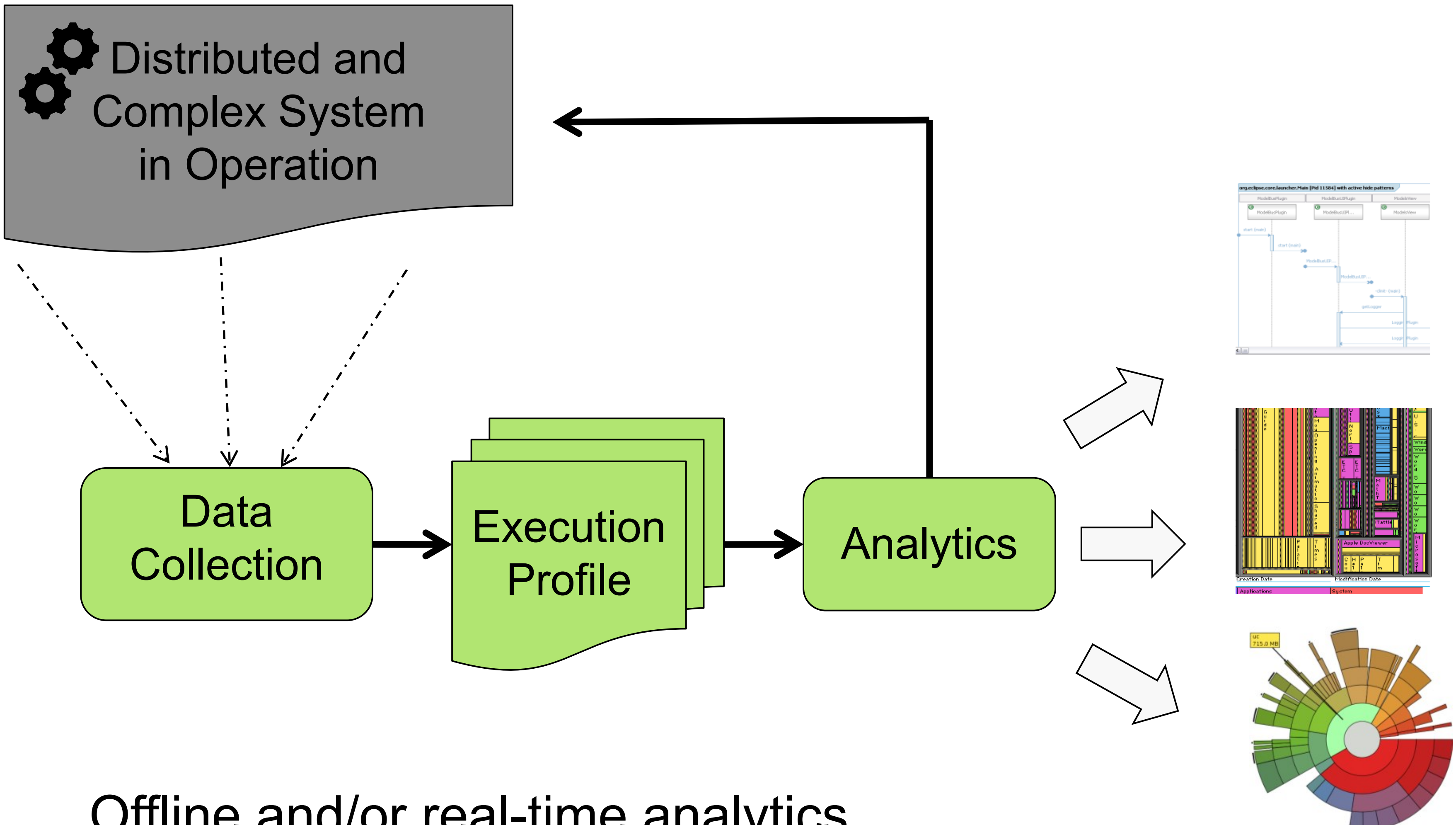
- Software Observability:
  - A set of end-to-end techniques and processes that allow us **to reason** about **what a software system is doing and why** by analyzing its external outputs.

Monitoring vs Observability

- **Monitoring:**
  - Tracks known metrics and raises alerts when thresholds are not met (e.g., 4 golden signals of Google SRE: latency, traffic, errors, and saturation)
  - Answers the question: “how is the system doing?”
  - Helps diagnose known problems

- **Observability:**
  - Answers the question: “what is the system doing and why?”
  - Enables to reason about the system by observing its outputs
  - Helps diagnose known and unknown problems

Building Blocks



Offline and/or real-time analytics

Operational Data

- **Logs:**
  - Records of events generated from logging statements inserted in the code to track system execution, errors, failures, etc.
  - Different types of logs: system logs, application logs, event logs, etc.
- **Traces:**
  - Records of events showing execution flow of a service or a (distributed) system with causal relationship
  - Require additional instrumentation mechanisms
- **Profiling Metrics:**
  - Aggregate measurements over a period of time (e.g., CPU usage, number of user requests, etc.)

Emergence of AI for IT Operations

- AIOps is the application of AI to enhance IT operations
- An important enabler for digital transformation
- Building Blocks:
  - Data collection and aggregation
  - Pattern recognition
  - Predictive analytics
  - Visualization
- Applications:
  - Fault detection and prediction
  - Root cause analysis
  - Security
  - Regulatory compliance
  - Operational intelligence

Characteristics of Logs and Traces

- **Velocity:** the data (in some cases) must be processed in real time
- **Volume:** mountain ranges of historical data
- **Variety:** captured data can be structured or unstructured
- **Veracity:** captured data must be cleaned
- **Value:** not all captured data is useful

Challenges

- **Standards and Best Practices:**
  - Lack of guidelines and best practices for logging, tracing, and profiling
  - Lack of standards for representing logs, traces, and metrics (not the OpenTelemetry initiative)
- **Data Characteristics**
  - Mainly unstructured data
  - Size is a problem
  - Not all data is useful
  - High velocity

Challenges

- **Analytics and Tools:**
  - Mainly descriptive analytics
  - Predictive analytics not fully explored
  - Mainly offline analysis techniques
  - Lack of usable end-to-end observability tools

- **Cost and Management Aspects**
  - Cost vs. benefits not well understood
  - No clear alignment of observability with other initiatives
  - Roles and responsibilities are not well defined

Challenges

- **Analytics and Tools:**
  - Mainly descriptive analytics

There is a need for systematic and engineering approaches to software observability that promote best practices throughout the entire software development lifecycle

- **Cost and Management Aspects**
  - Cost vs. benefits not well understood
  - No clear alignment of observability with other initiatives
  - Roles and responsibilities are not well defined

Observability By Design

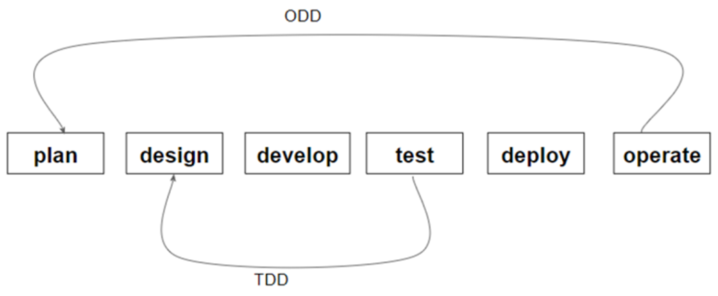
- Bringing observability **to early stages** of the software development lifecycle.

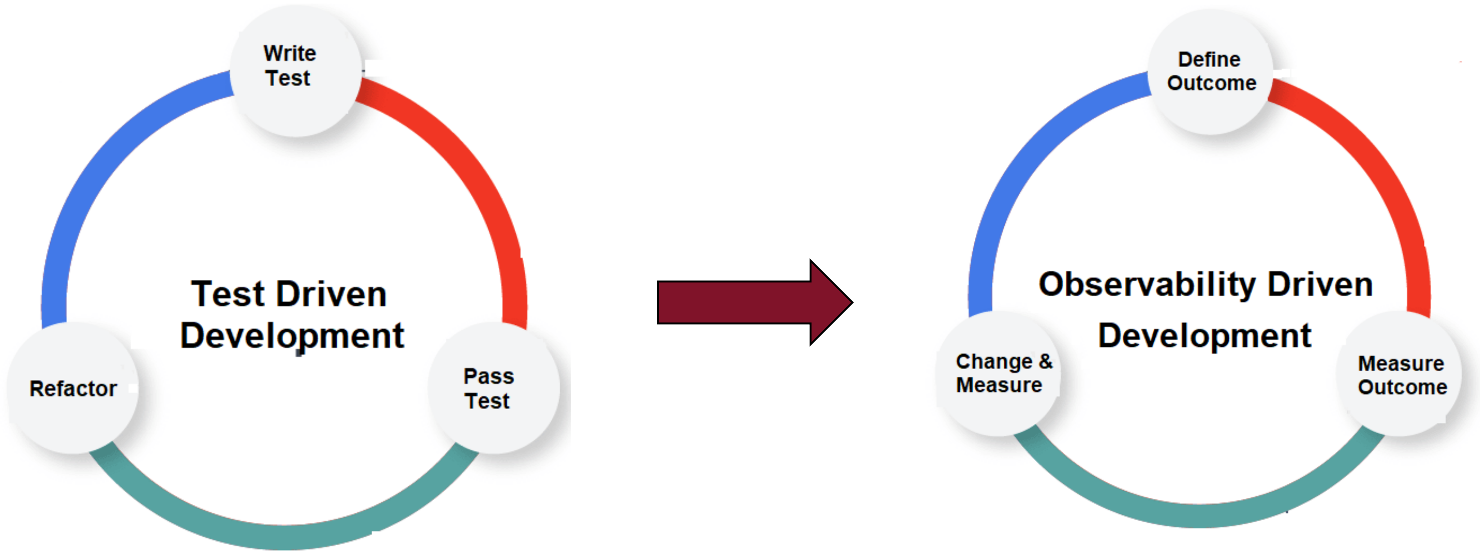
- Defining a set of **observability patterns, best practices, and reusable solutions** to be used as guiding principles for developers.

- A **systematic approach** to tracing, logging and profiling of software systems that considers different phases of the software process.

Observability-Driven Development (ODD)

- Leveraging tools and hands-on developers to observe system state and behavior
  - Interrogating the system, not just setting and measuring thresholds and metrics for it

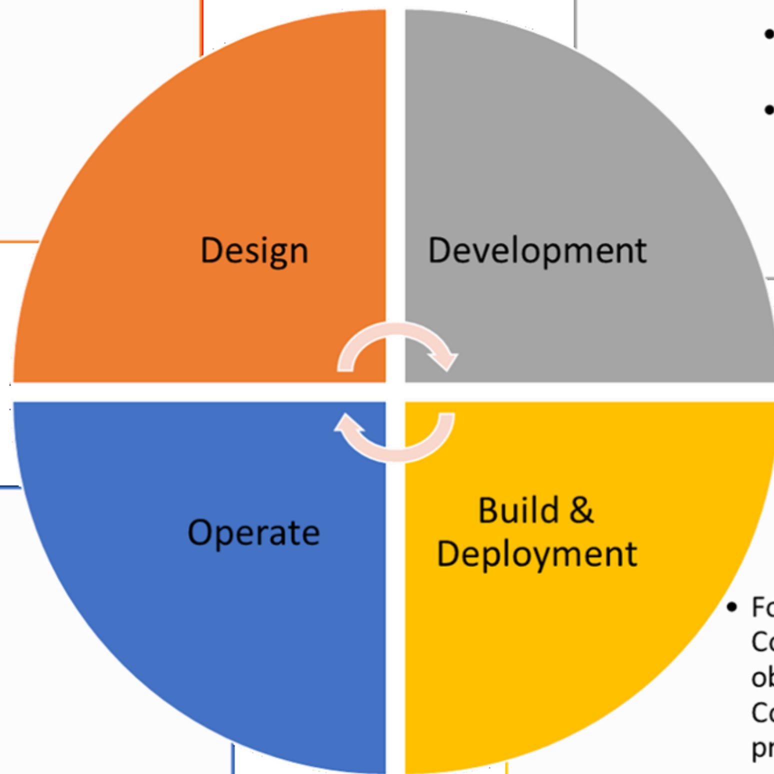




Observability-Driven Development (ODD)

- Determining what to observe based on Quality of Service and KPIs to be met
- Identifying where to observe and designing in such a way to make instrumentation easy

- Standardization of the instrumentation
- Adding sufficient context for getting better insight
- Implementing at framework level



- Proactive monitoring and querying
- Feedback loop from the observations to development team

- Following ‘Observability as Code’ practice to enforce observability as part of Continuous Deployment process.
- Observing for unusual behaviour at an early stage through automation

From Telemetry to OpenTelemetry

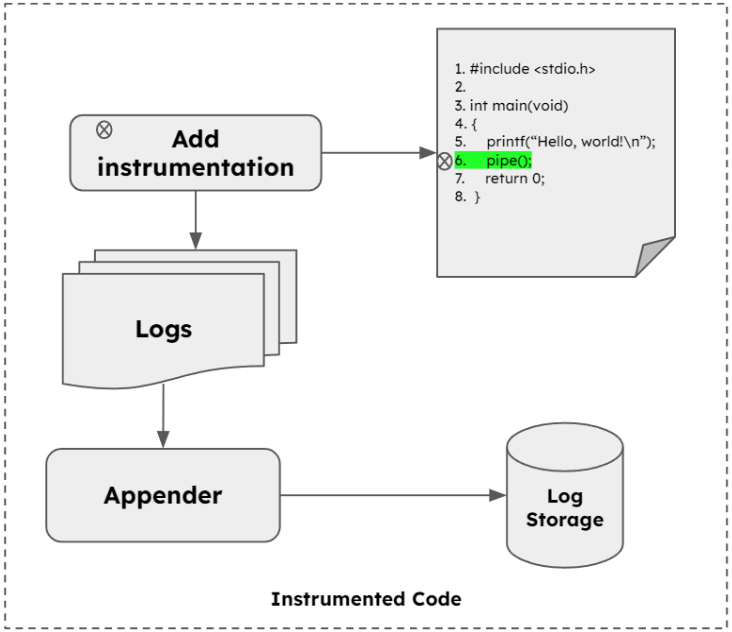
- Observability is often equated with telemetry
  - "If you have metrics, logs, and traces, then you have Observability"
- Observability, is the process of deriving value from telemetry
  - Telemetry is important but not sufficient
- We also need tools to analyze and visualize the telemetry
  - OpenTelemetry

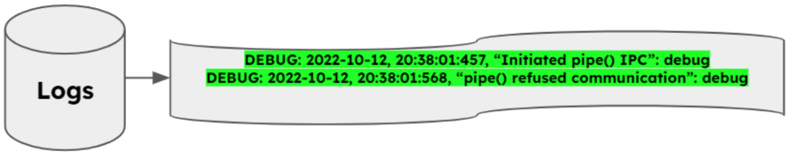
Instrumentation

- Definition
- Example
- Challenges
    - Level of details
    - Lots of noises
    - Cost (overhead)

Instrumentation (Logging)

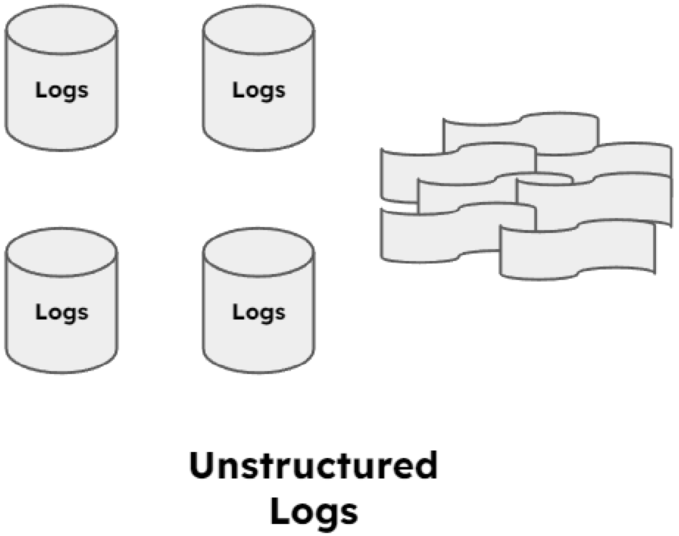
- Limited context of request origin.
  - Can be specific to a certain machine/group.
  - Failure due to other dependency
  - Causal information missing.
- Finding/locating logs for analysis is cumbersome.
- **NOT** an **automatic** process.





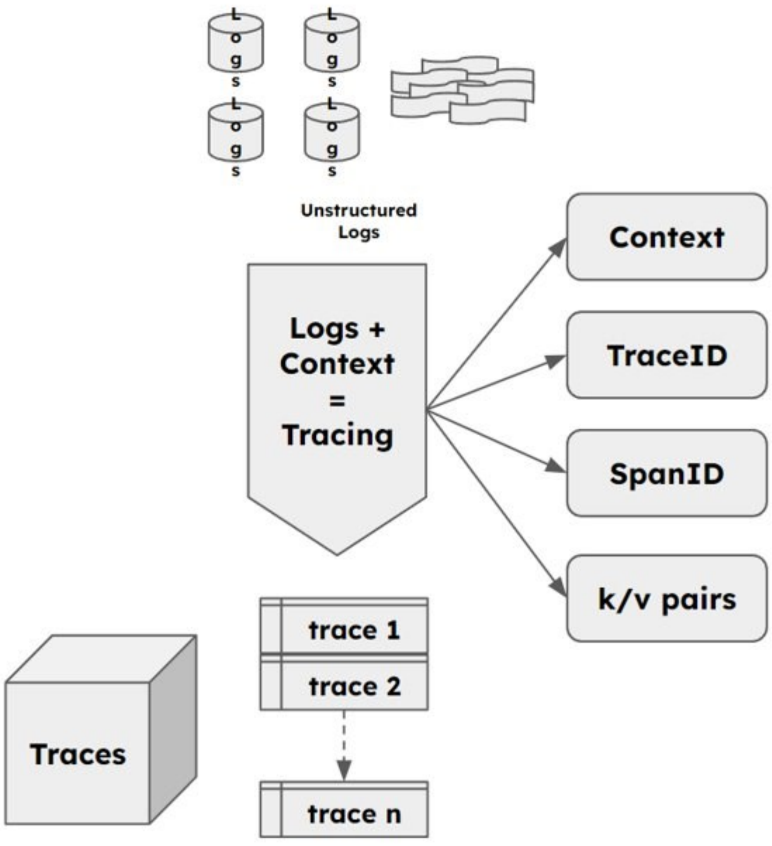
Instrumentation needs context

- Naïve logging is unstructured data.
- Prohibitive in gathering all information (costly).
- Time expensive to reconstruct a request/transaction.
- Sometimes even impossible!



Logging + Context = Tracing

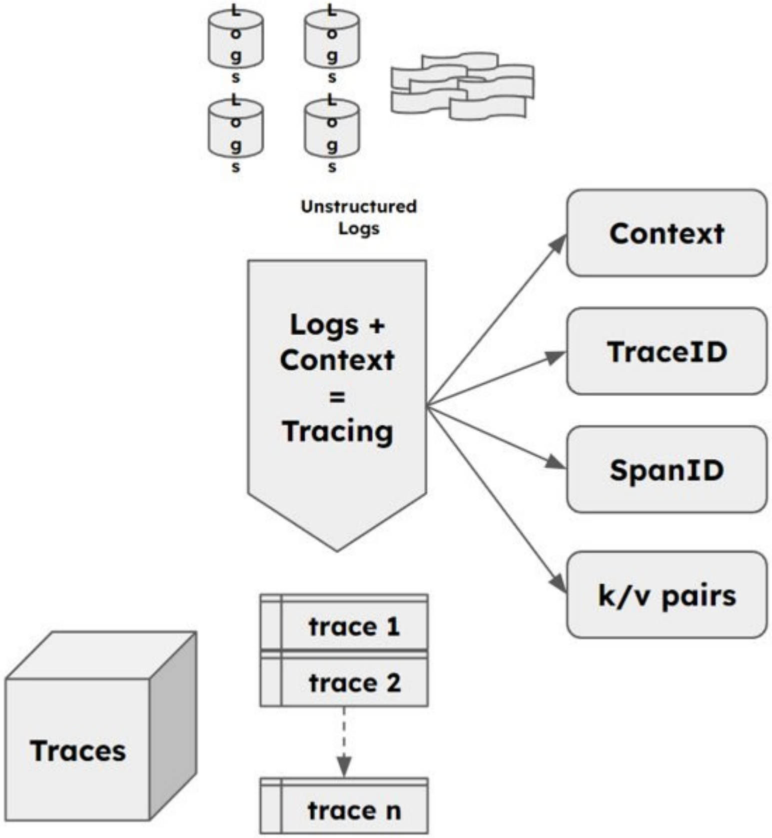
- OpenTelemetry standardizes this transformation.
- Context allows for causal relationship construction.
- Scattered events can be mapped to distributed nodes.
- Unique identifier for each trace allows fast lookups.
- End result = structured data.



Converting logs to traces

Logging + Context = Tracing

- OpenTelemetry standardizes this transformation.
- Context allows for causal relationship construction.
- Scattered events can be mapped to distributed nodes.
- Unique identifier for each trace allows fast lookups.
- End result = structured data.



Converting logs to traces

Logs to Events

- Every span has an event
- Every event has a name/message + timestamp + optionally if log has structered data (k/v pairs), add to span
- Switching from Logs to Traces:
  - Context (what)
  - Resources (where)
  - Logs (events attached to traces)

Example Event

- HTTP event
- GET something/somewhere
- Attributes: (unique to the event + generic to the event, but very important for locating this event- static resources), dynamic context: values change from req to req (duration, starttime, error or not, app specific attributes, account id or project id)

TRACING -> GRAPH -> CAUSALITY

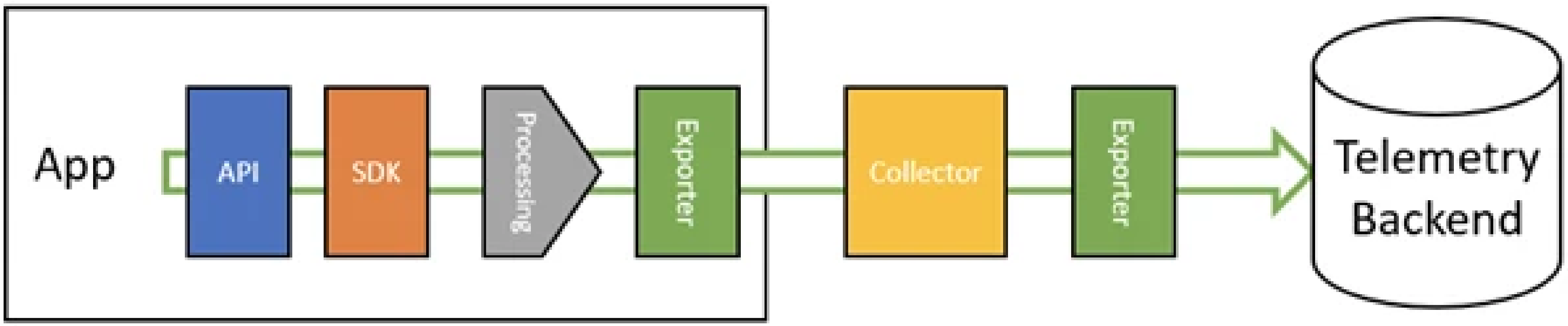
- Needs more contextualization:
  - More attributes
  - Trace-id: identifies transaction
  - Span-id: identifies operation
  - Parent span-id: causality
  - Operation name: compare across different runs of the same operation

Trace Analysis

- Not trying to look at individual transactions
- Correlation across many runs of the same transaction
- Identify that correlation to find the causation (root cause) of the problem.

OpenTelemetry

- Vendor-neutral telemetry



- Instrumentation
  - Changes to the application (source code or configuration)
  - "With great instrumentation comes great observability."
- Data pipeline
- Visualization & Analytics

Client-Server Java Spring Boot Configuration (https://spring.io/)

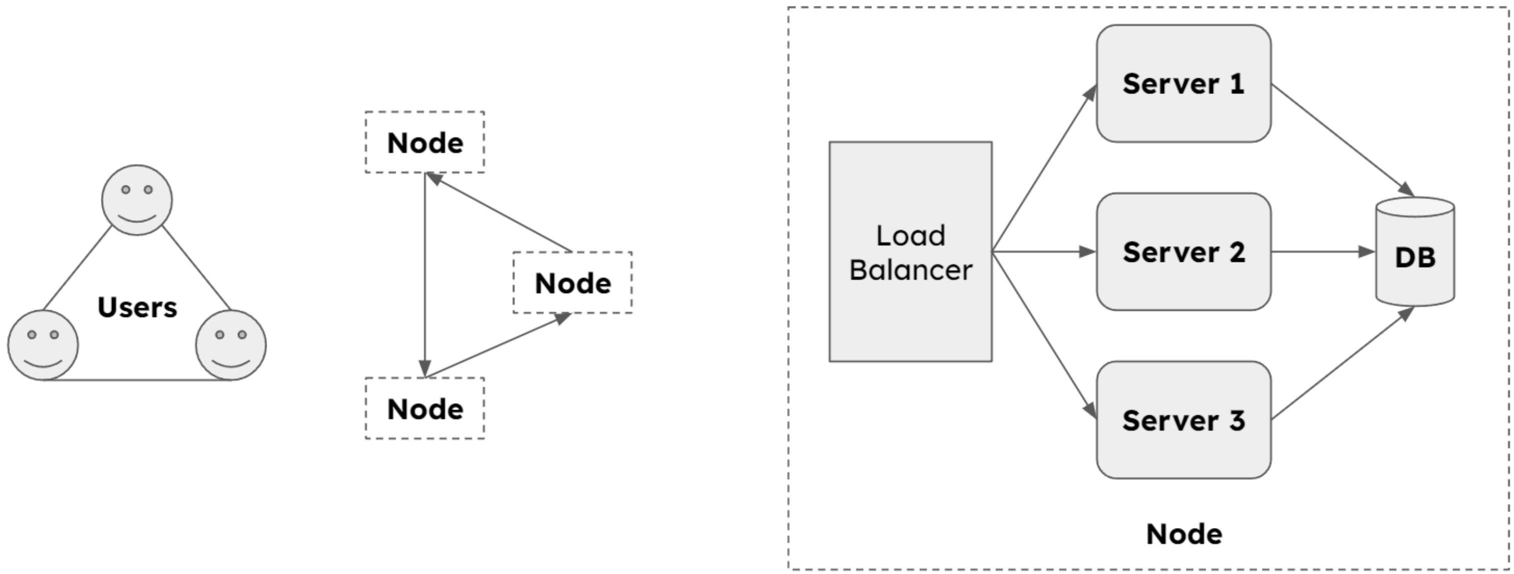


Figure: Microservice Spring Boot Distributed Configuration

Log Providers in Spring

- Nodes serves different endpoints.
- Each endpoint has a logging facility provided by Spring.
- Endpoint activation is internally tracked by the Spring engine.
- Log providers enables live swap in/out.

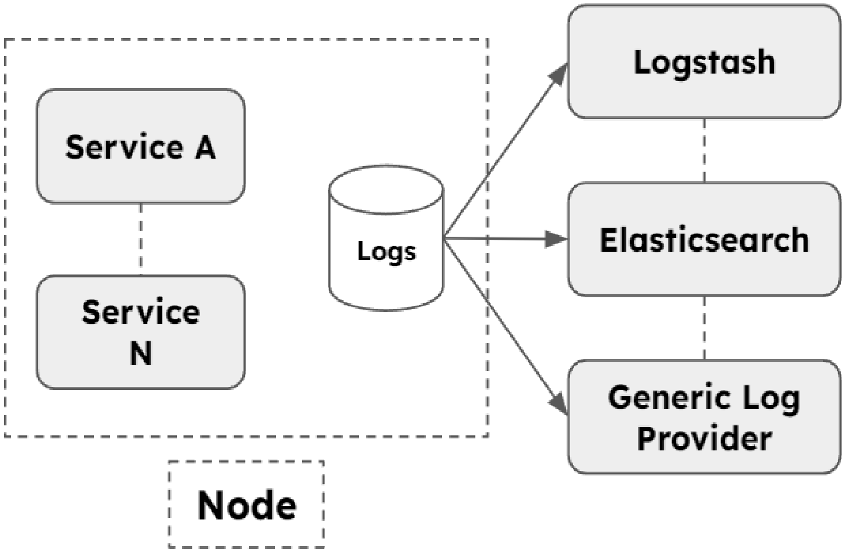


Figure: Log Mechanism Provider within Node

OpenTracing Concept in Spring

- Each unique service can be instrumented.
- OpenTracing API is an interface that Spring Boot provides (Spring Boot Actuator).
- Exposes various metrics (Health, Events, Prometheus, HeapDump)

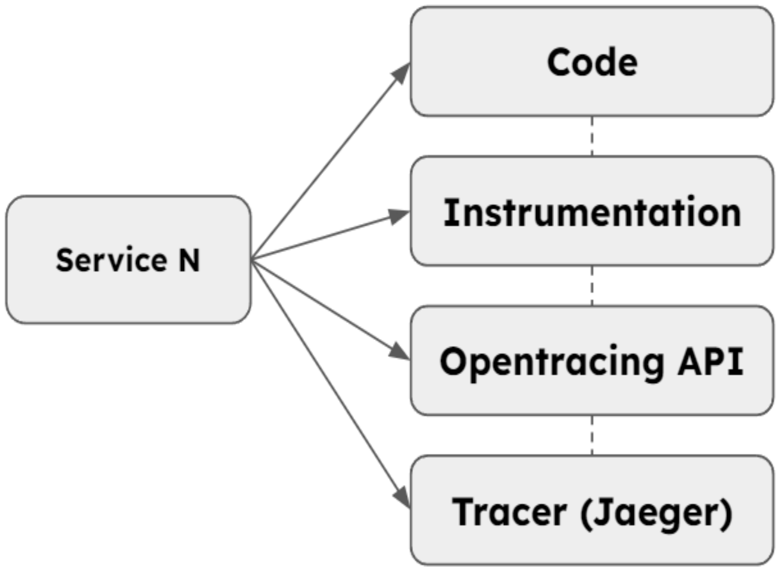


Figure: Trace Mechanism within Service

System Trace Lifecycle (Spans)

- Each Span context is unique storage facility defined from developer's point of view.
- High **cardinality** of data (e.g user_id)
- Context can be augmented with additional information.
- Spans can be analyzed during fault diagnosis without overwhelming trace size.

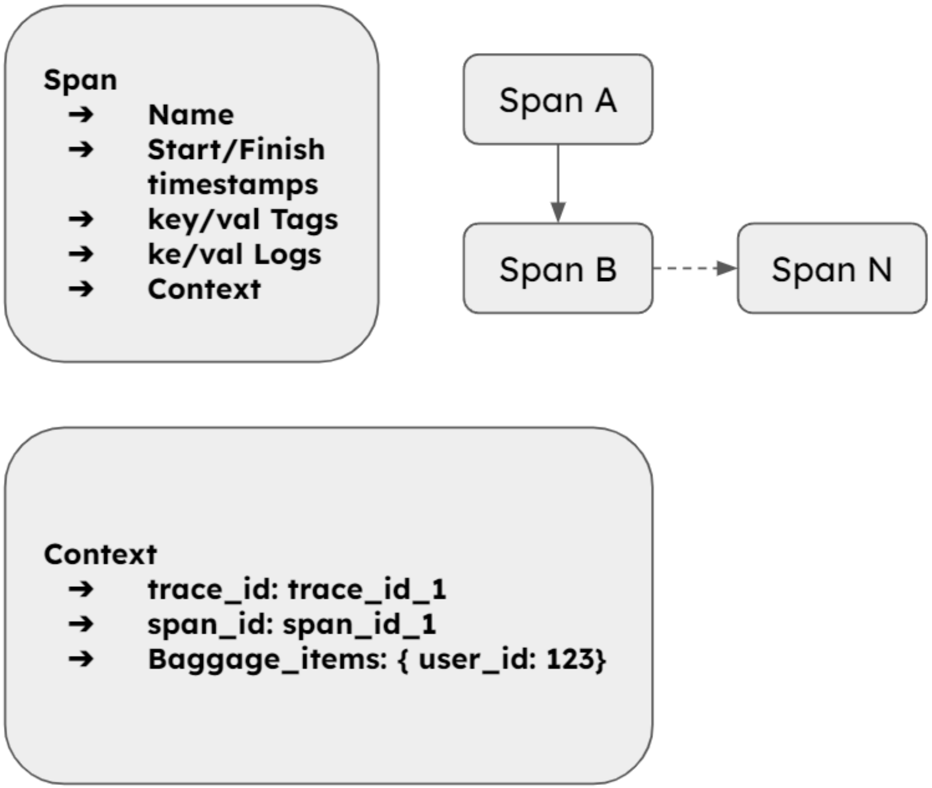
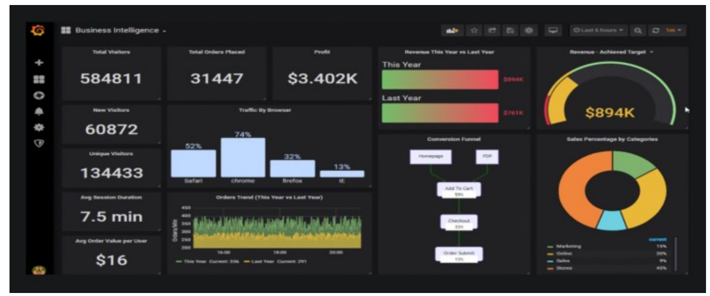


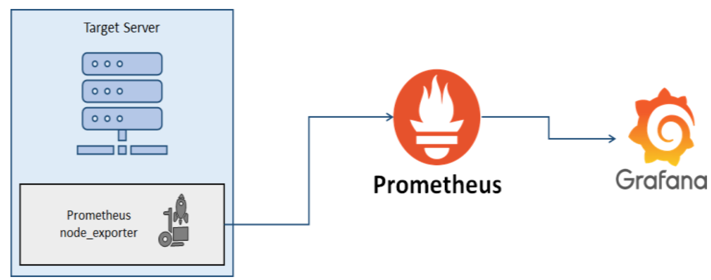
Figure: Custom Span Context Configuration

Metric Analysis & Visualization

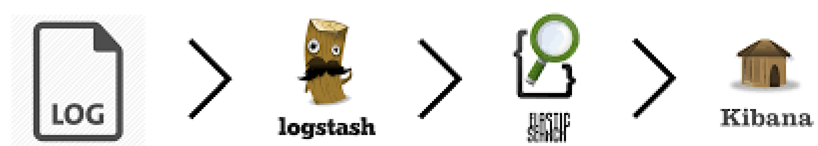
- Grafana
- Prometheus
- Kibana



https://grafana.com/



https://prometheus.io/docs/visualization/grafana/



https://www.elastic.co/guide/en/kibana

Observability Culture

- Observability in action!
- Before and after a problem,
- Data-driven decision making
- Educate team
- Encourage standard tools/techniques
  - Log formatting
  - Metric conventions
- Practice, share success stories, and feedback
- Measure your progress and observer your observability culture!

Contact Information

Wahab Hamou-Lhadj, PhD, ing.

Concordia University

wahab.hamou-lhadj@concordia.ca

http://www.ece.concordia.ca/~abdelw

Naser Ezzati-Jivan , PhD

Brock University

nezzatijivan@brocku.ca

http://www.cosc.brocku.ca/~nezzatijivan/