SCALEOUT

Configuration Management Tools -
Observability

Max Andersson

- The Need for observability
  - Not purely an operational concern.
  - Observability is not only about logs metrics and traces.

- Observatiblity is a property of and system that has been
  - Designed, build, tested, deployed, operated, monitored, maintained and evolved.

- Observability because
  - No complex system is ever fully healthy
  - Distributed systems are unpredictable
  - Impossible to predict all states of partial failures the system can end up in.
  - Failure needs to be be embraced at every phase, from design to operation.
  - Ease of debugging is a important for maintenance of an evolving system.

- The Three Pillars of Observability
  - Logs
  - Metrics (Telemetry)
  - Traces

- Monitoring and Observability
  - Alerting based on monitoring data
  - Blackbox vs Whitebox monitoring
  - Monitoring should provide an overview of health.
  - Ability to drill down on component level
  - Monitoring is a part of the Security process

- Alerting
  - Event Failures should immediately provide visibility of impact.
  - All alerts and signals that generate them should be actionable.

- Failure modes
  - Tolerated
  - Alleviated
    - Graceful degradation
      - Backpressure
      - Retries
      - Timeouts
      - Circuit braking
      - Rate limiting
  - Triggered

- Monitoring Signals
  - USE Metrics
    - Free Memory (**U**tilization)
    - CPU queue length (**S**aturation)
    - Device errors (**E**rrors)
  - RED Metrics
    - **R**equest rate
    - **E**rror rate
    - **D**uration of requests
  - Golden Signals (minimum viable signals)
    - Latency
    - Errors
    - Traffic
    - Saturation

- Coding for failure
  - Semantics
  - Characteristics
  - Debuggable

- Coding for failure
  - Semantics
  - Characteristics
  - Debuggable

- Semantics
  - How a service is deployed and what tools
  - Is the service binding to a standard port ?
  - How an application handles signals
  - How processes start on a given host
  - How it registers with service discovery
  - How graceful (or not) the restarts are.
  - How configuration is fed to the process
  - Concurrency model of the application
    - Multithreaded, single threaded, event driven, actor based. Etc.
  - How the Reverse proxy/ Ingress handels connections

- What are logs?
  - Historical Records of events.
  - Records events and status of systems in a time sequential format
  - Records Activity on a system/network
  - Provides an Audit trail of who, what, where, when and why.

- Why are they important?
  - Determining what happened - Audit trail
  - Intrusion Detection
  - Incident Containment
  - Forensic Analysis
  - Proactive Analysis
  - Real-time alerts
  - Determining the health of a network
    - Debugging/Trubbleshooting
    - Proactive maintenance

- Log Types
  - Event Logs
    - Immutable
    - Timestamped records
    - Discrete Events
  - Formats
    - Plaintext
    - Structured
    - Binary
      - Replication
      - Point-in-time recovery
      - Systemd journal logs
      - Pflog

- Type of Logs
  - Traffic Logs
  - Threat Logs
  - URL Filtering Logs
  - Data Filtering Logs
  - Correlation logs
  - Tunnel inspection logs
  - Config logs
  - System logs
  - User Logs
  - Alarm Logs
  - Authentication logs

- Where to find the logs?
  - Operating Systems
  - Applications
  - Device Logs
    - Routers
    - Firewalls
    - Switches
    - etc..

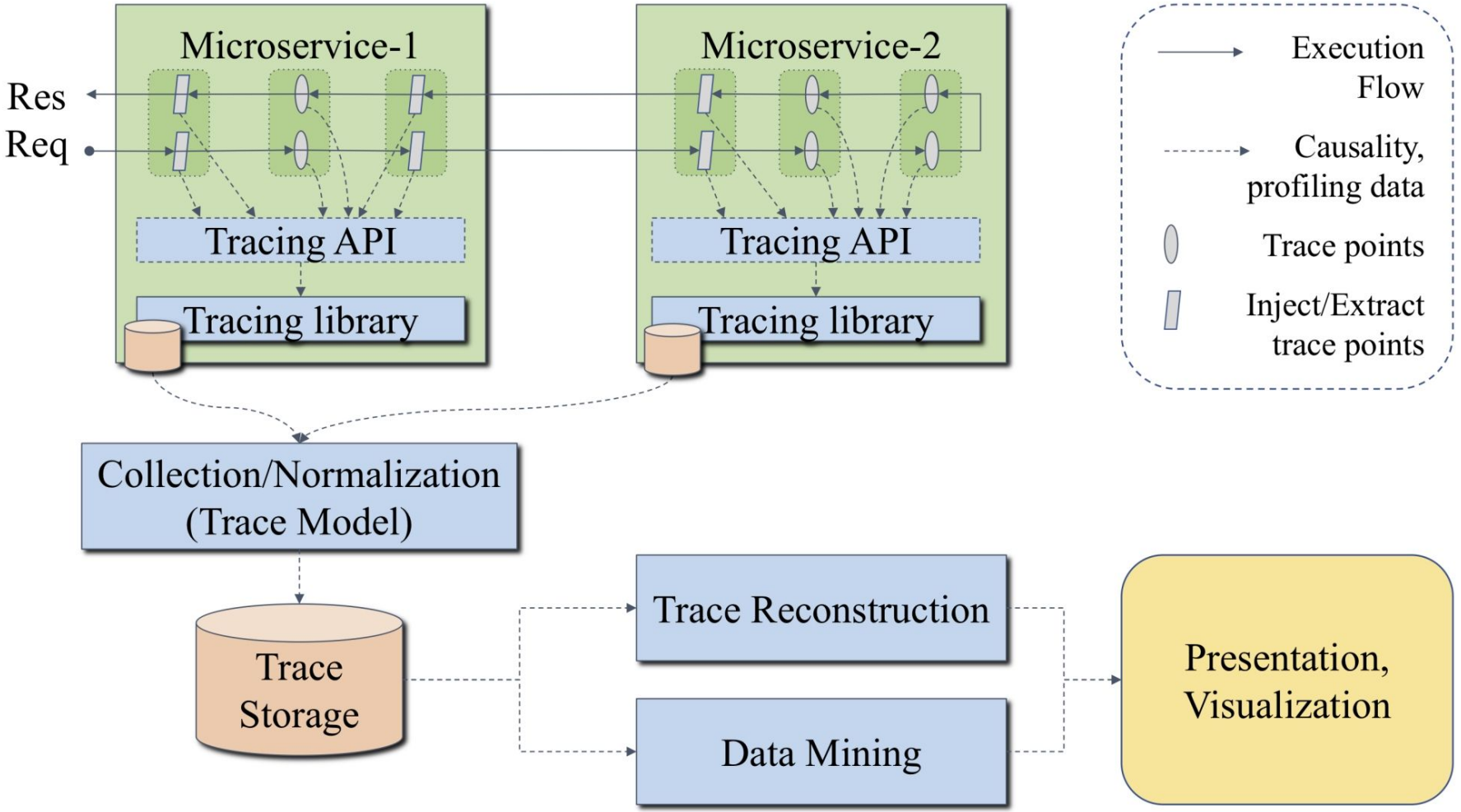
- Challenges
  - Different log formats
  - Logs are written by developers
    - Format and content can be hard to obscure and hard to understand.
  - Information Volume
  - Anomalies Detection is hard!

- Best Practices
  - Centralized log collection.
  - Secure transmission of logs
  - Normalize the data to same format
  - Review the logs
    - Manually and Automatically
  - Log Rotation
    - Time schedule
    - Naming Convention
  - Log Retention
    - Disk Space
    - Regulatory requirements?
    - Archive

- So….
  - What to log? **Logging Policy**
  - What to centralize? **Log collection**
  - What to save? **Log retention**
  - What to look at? **Log Review**
  - What to alert on? **Log Monitoring**

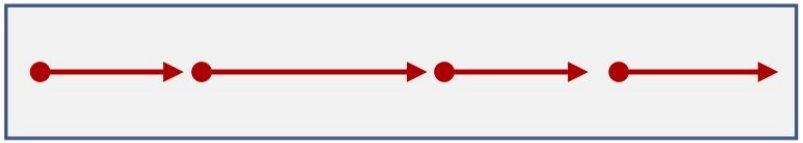
- Tracing
  - A Trace is a representation of a services of causally related distributed events that represents the end-to-end request flow.
  - Identify a specific point in request flow.
  - Zipkin & Jaeger are the most popular OpenTacing-compliant open source distributed tracing solutions

- Distributed Tracing
  - Tracing Infrastructure attaches Contextual metadata to each requests.
  - Trace points in code, records events to be annotated.
  - Events are tagged with context and causality references generated by previous events

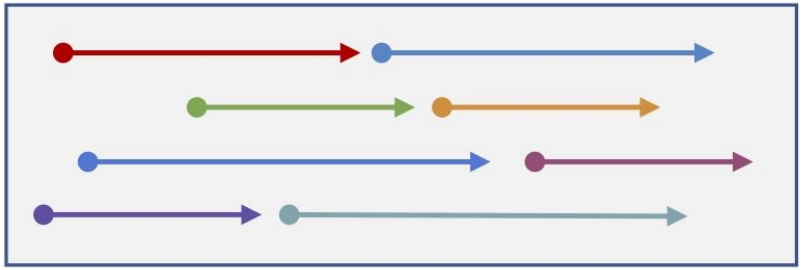

Microservice-1
Microservice-2
Res
Req
Tracing API
Tracing library
Tracing API
Tracing library
Execution Flow
Causality, profiling data
Trace points
Inject/Extract trace points
Collection/Normalization (Trace Model)
Trace Storage
Trace Reconstruction
Data Mining
Presentation, Visualization

- Service meshes
  - Makes implementing tracing easier
  - The data plane implements the tracing and collection of statistics from a proxy.
  - Getting support of all of its services can mean requirements of the application layer

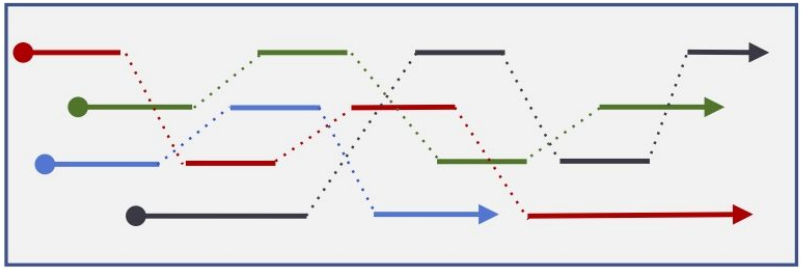
No Concurrency



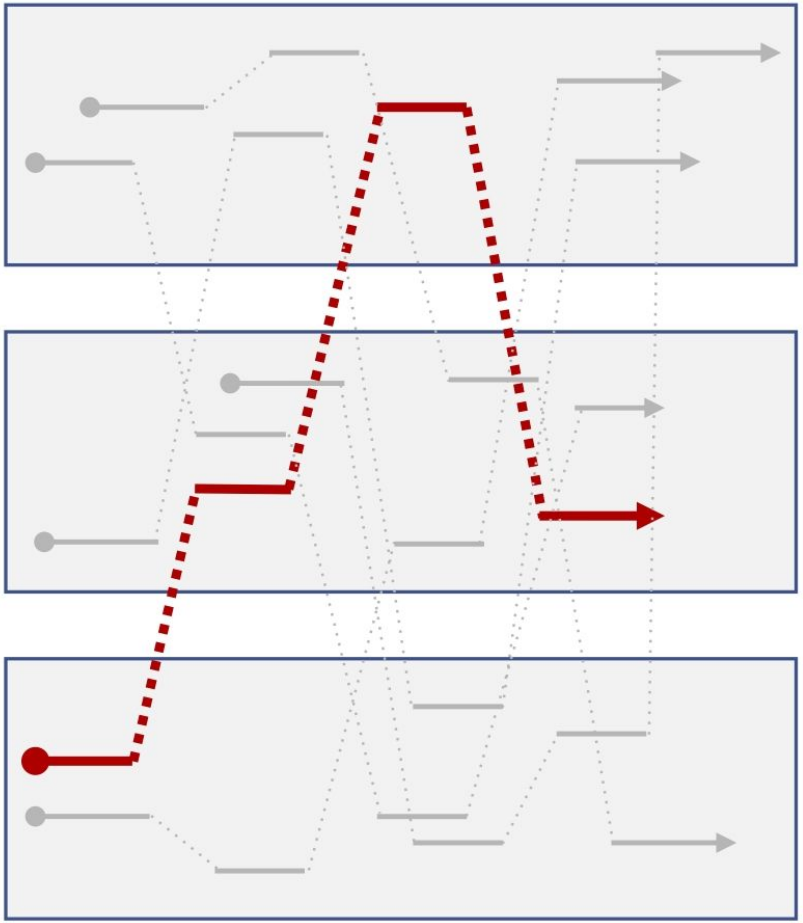
Basic Concurrency



Async Concurrency



Distributed Concurrency



Observability