universidade de aveiro

deti departamento de eletrónica, telecomunicações e informática

Software Engineering

Monitoring and observability

2025/2026

Outline

PlayStation Network's 24 days of downtime | 10 Years Ago This Month

Sony's 2011 hack was covered as one of the biggest security breaches of all time, but how significant was it in the long run?

Bank and building society users hit by 33 days of outages in two years, say MPs

Treasury committee finds disruption a result of 158 IT incidents affecting millions of customers

Maintenance error caused Facebook's 6-hour outage, company says

By Sheila Dang

October 6, 2021 1:49 AM GMT+1 · Updated October 6, 2021

Oct 5 (Reuters) - An error during routine maintenance on Facebook's network of data centers caused Monday's collapse of its global system for more than six hours, leading to a torrent of problems that delayed the repairs, the company said on Tuesday.

The outage was the largest that Downdetector, a web monitoring firm, said it had ever seen. It blocked access to apps for billions of users of Facebook (FB.O), Instagram and WhatsApp, further intensifying weeks of scrutiny for the nearly $1 trillion company.

Google Cloud, Snap, Spotify back up after brief outage

By Reuters

November 16, 2021 11:32 PM GMT · Updated November 16, 2021

[1/2] Google app is seen on a smartphone in this illustration taken, July 13, 2021. REUTERS/Dado Ruvic Purchase Licensing Rights

On 19 July 2024, the American cybersecurity company CrowdStrike distributed a faulty update to its Falcon Sensor security software that caused widespread problems with Microsoft Windows computers running the software. As a result, roughly 8.5 million systems crashed and were unable to properly restart[1] in what has been called the largest outage in the history of information technology[2] and "historic in scale".[3]

The outage disrupted daily life, businesses, and governments around the world. Many industries were affected—airlines, airports, banks, hotels, hospitals, manufacturing, stock markets, broadcasting, gas stations, retail stores, and governmental services, such as emergency services and websites.[4][5] The worldwide financial damage has been estimated to be at least US$10 billion.[6]

2024 CrowdStrike-related IT outages

Multiple blue screens of death caused by a faulty software update on baggage carousels at LaGuardia Airport, New York City

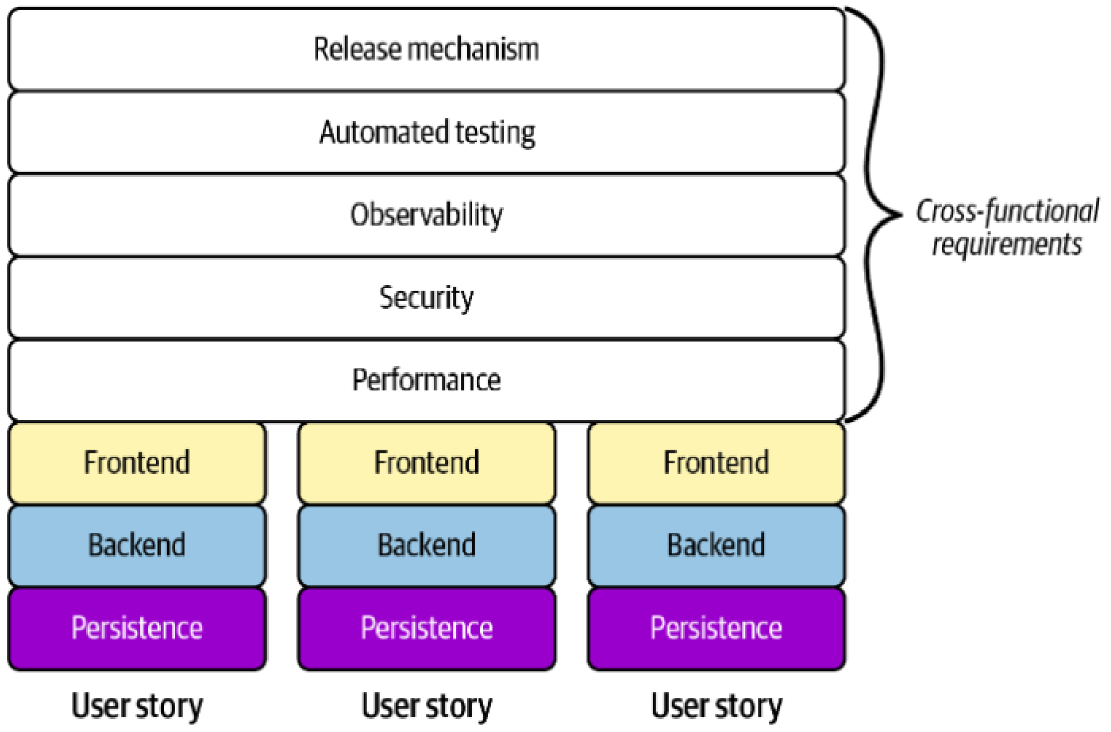
Date	19 July 2024; 13 months ago
Location	Worldwide
Type	IT outage, computer crash
Cause	Faulty CrowdStrike software update

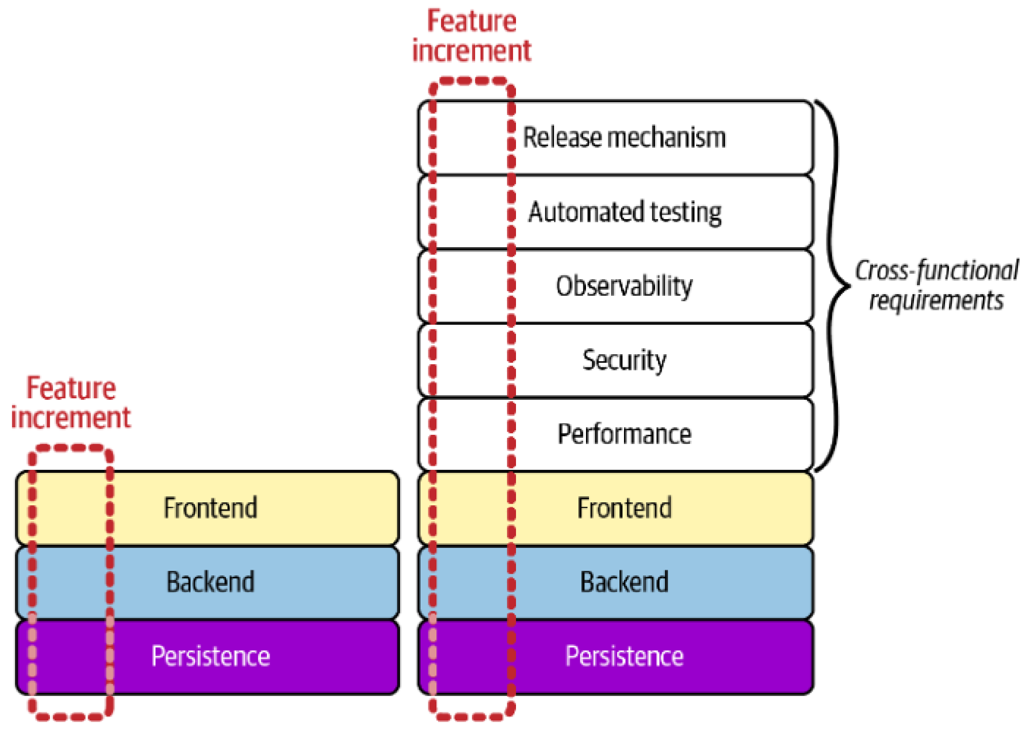
BRIAN BARRETT SECURITY JUN 8, 2021 1:28 PM

How an Obscure Company Took Down Big Chunks of the Internet

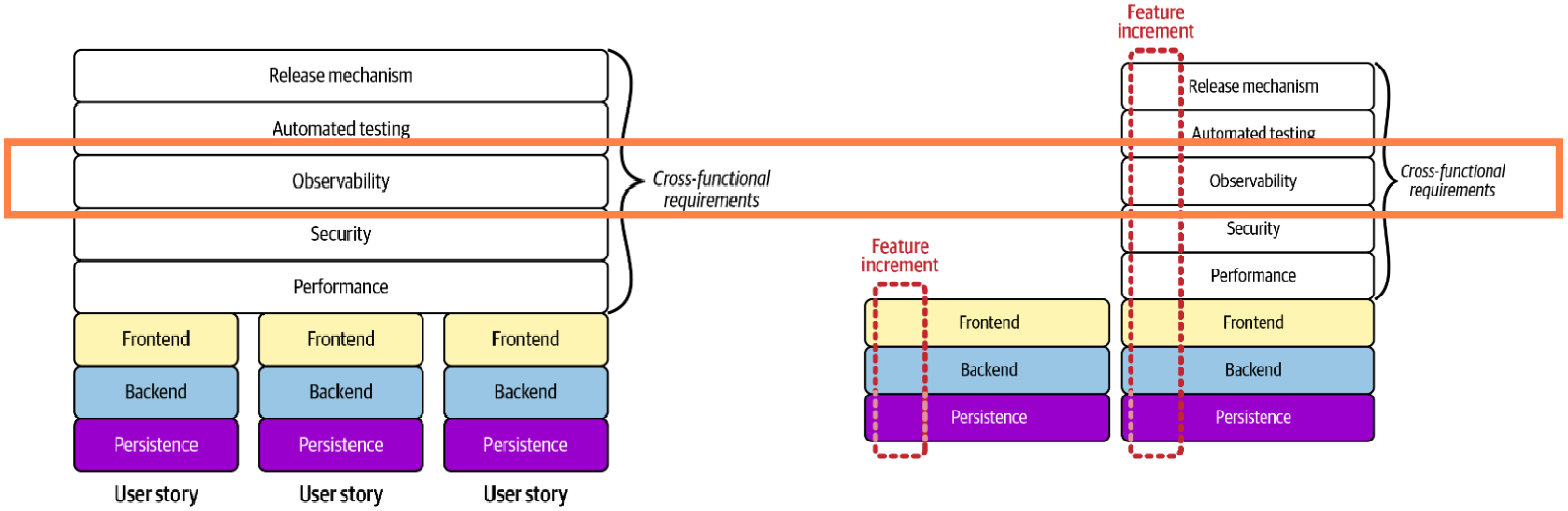
You may not have heard of Fastly, but you felt its impact when sites didn't load around the world Tuesday morning.

Looking back to previous lectures..





Looking back to previous lectures..



Introduction

Monitoring and observability

- What are the main reason to perform that?
- Service Level Agreement - a contract!
  - **Definition**: A formal contract between a service provider and a customer. It usually includes SLOs, and adds consequences (like penalties or compensation) if the objectives are not met.
  - Example: If availability drops below 99.9% in a given month, the customer receives a service credit.
- This highlights the importance of maintaining continuous monitoring and ensuring that the system remains observable."

Definitions

- **Service level quantifiers**: SLIs, SLOs, and SLAs; the core topic of this book.
- **Error budgets**: A measurement of the allowable amount of downtime or outage of some component within the system.
- **Fault tolerance**: The ability of a system to operate without failure when a single or multiple components fail.
- **Redundancy**: The duplication of systems or components as a backup mechanism.
- **Failure analysis**: The process of collecting and analyzing data to determine the cause of a failure.
- **Incident response**: The set of processes in place for responding to and managing incidents.
- **Capacity planning**: The set practices in place to help determine the current and future demand(s).
- **Monitoring**: The set of processes and tooling used to monitor, and later alert, events within the system.

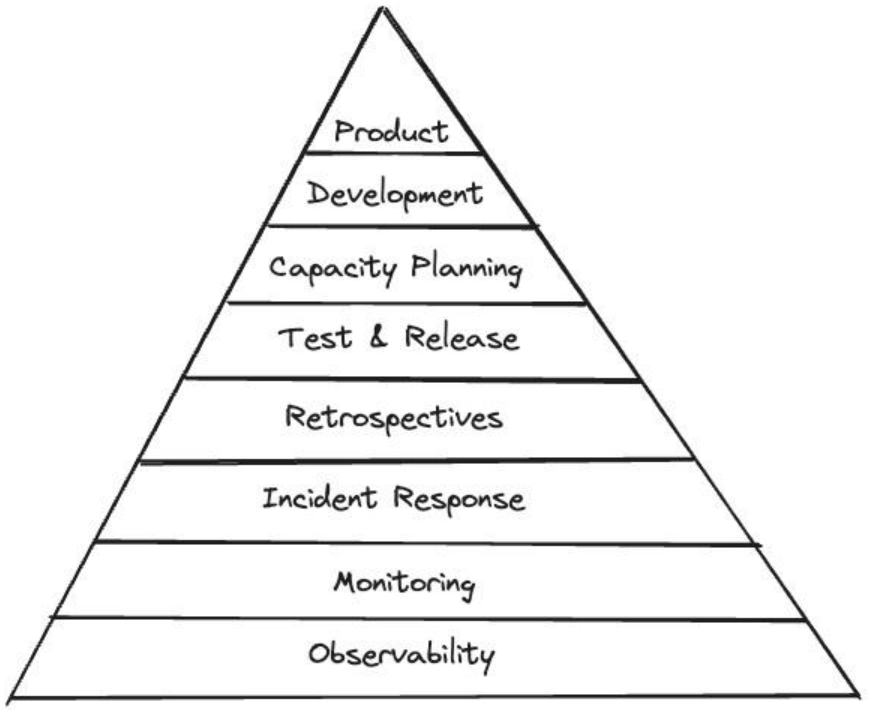
Service Level Indicator (SLI)

- **Definition**: A quantitative measure of some aspect of a service’s performance or reliability. It should provide the raw data that tells us how the system is behaving. The SLIs are measurements, not goals or contracts.
- **Examples**:
  - Availability: % of successful HTTP requests.
  - Latency: % of requests served within 300 ms.
  - Error Rate: % of failed transactions.

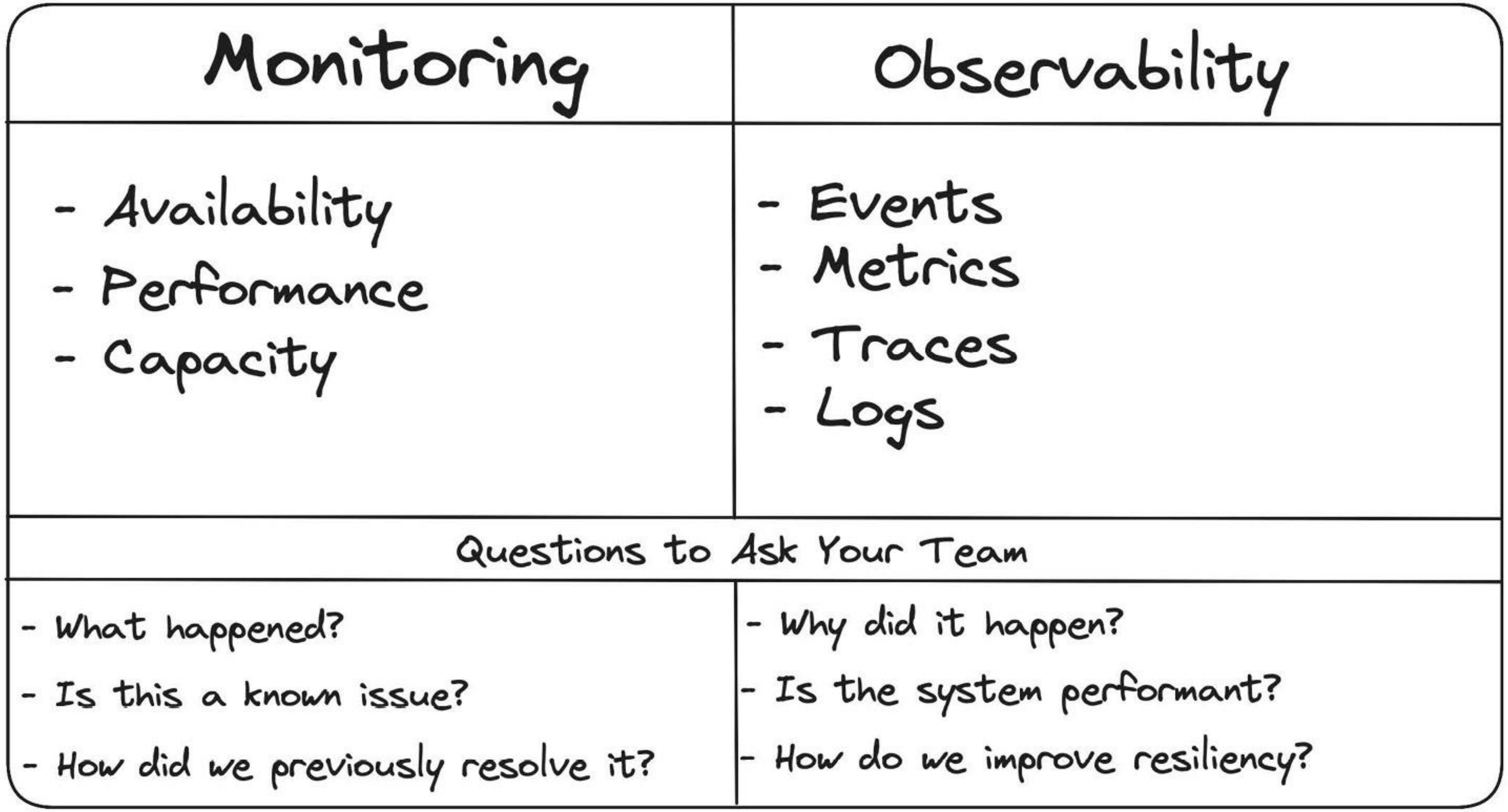
Service Level Objective (SLO)

- **Definition**: A target or threshold for a Service Level Indicator (SLI). The purpose is to translate raw metrics into reliability goals that guide engineering decisions.
- Key Idea: SLOs are commitments teams set internally to balance reliability with innovation speed.
- **Examples**:
  - Availability: 99.9% of requests succeed over 30 days.
  - Latency: 95% of requests served within 300 ms.
  - Error Rate: <0.1% failed logins per week.
- Relation:
  - SLI = Metric
  - SLO = Target for the metric
  - SLA = Contract based on the target

Hierarchy of reliability engineering



Monitoring vs Observability (1/2)



Monitoring	Observability
- Availability - Performance - Capacity	- Events - Metrics - Traces - Logs
Questions to Ask Your Team	
- What happened? - Is this a known issue? - How did we previously resolve it?	- Why did it happen? - Is the system performant? - How do we improve resiliency?

Monitoring vs Observability (2/2)

- **Observability**: Ability to understand and diagnose system behaviors through telemetry (logs, metrics, traces). Provides visibility, traceability, and insights into latency, errors, and performance.
- **Monitoring**: Systematic collection and analysis of observability data to track SLIs, detect anomalies, identify trends, and trigger proactive responses.
- Complementary roles: **Observability** explains **why something happens**, while monitoring alerts when it happens.
- Impact: Together, they provide visibility across system components, strengthen incident management, improve development quality, and enhance reliability.
- Connection to SLIs/SLOs: Essential for setting measurable objectives, aligning technical metrics with business goals, and ensuring customer-centric reliability.

End-users' expectations

- **Availability**: Is the system online?
- **Latency**: Does it take to long to access and use the application?
- **Reliability**: Can the user rely on this application?

Monitoring

Monitoring Anti-Patterns

- **#1 Tool Obsession:** Chasing “single pane of glass” or new tools without defined outcomes.
  - Do instead: Start from user/business outcomes → capabilities → minimal toolset; prototype, integrate, and consolidate.
- **#2 Monitoring-as-a-Job**: One team “owns” all monitoring; product teams uninvolved.
  - Do instead: Make monitoring a shared engineering skill; platform/observability team provides self-service tooling, not all alerts.
- **#3 Checkbox Monitoring**: Superficial metrics/alerts to tick compliance boxes; noisy, low trust.
  - Do instead: Define “what does working mean?”; create high-signal checks tied to user journeys and SLIs; tune/retire noisy alerts.
- **#4 Monitoring as a Crutch**: Adding more alerts to compensate for brittle systems.
  - Do instead: Fix root causes; invest in reliability engineering (SLOs, error budgets, chaos tests, resilience patterns).
- **#5 Manual Configuration**: Hand-crafted checks/dashboards; slow, inconsistent, stale.
  - Do instead: Automate discovery/registration; config-as-code; templates; golden dashboards; enforce review and versioning.

Pattern #1: Composable Monitoring

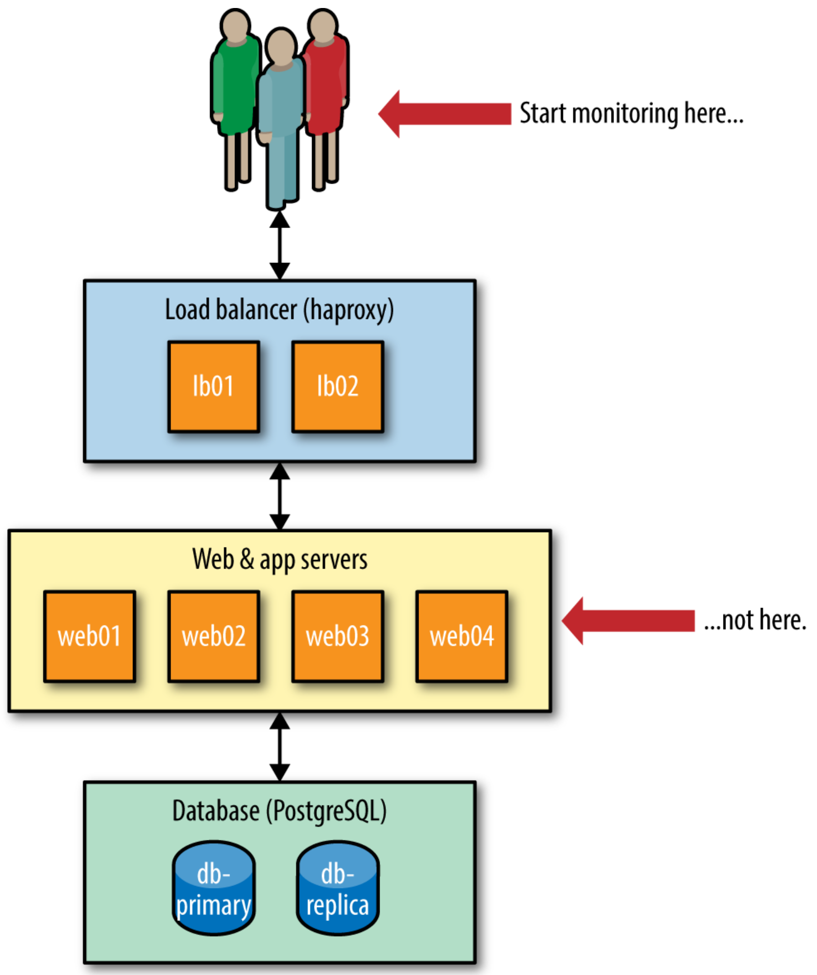
- Concept: Instead of monolithic tools, build a platform by combining specialized tools, loosely coupled.

"Write programs that do one thing and do it well. Write programs to work together.", Doug McIlroy

- Every monitoring system includes collection, storage, visualization, analytics, and alerting.
- Push vs pull, structured vs unstructured logs, TSDBs (Time Series Databases), and visualization principles define design choices.
- SLAs and alerts are secondary outcomes; the main goal is to understand and reason about systems.

Pattern #2: Monitor from the User Perspective

- Starting point: Begin monitoring at the point where users interact with the app.
- Key metrics:
  - HTTP response codes (especially 5xx errors).
  - Request times/latency.
- Benefit: Focuses attention on actual user impact, not just internal system health.



Pattern #3: Buy, Not Build

- Progression of Monitoring Maturity
- Organizations/Companies often follow a path:
  - Start with SaaS (quick setup), focus on product.
  - Move to FOSS tools (Graphite, Prometheus, InfluxDB, etc.) when SaaS no longer fits needs.
  - Custom platforms (Netflix, Dropbox, ..) only when scale/requirements exceed existing tools.Stages can overlap (e.g., SaaS logging + in-house metrics).
- Advice: Do not jump to custom too early, master fundamentals first.
- SaaS monitoring is typically cheaper, faster, and better for most companies. Only very large or specialized organizations truly need custom, in-house solutions.

Pattern #4: Continual Improvement

- Leading companies (Google, Netflix, etc.) reached advanced monitoring through years of iteration, not overnight success.
- Tools evolve: old ones get retired, new ones are built as needs change.
- Monitoring maturity requires consistent effort over time, not a single big leap.
- Expect to re-architect every 2-3 years as systems grow and industry practices shift.
- Key insight: Monitoring excellence is a long-term journey, built on steady, continual improvement.

Statistics Primer

- Undervalued & misunderstood: Stats won’t create “magic,” but basic methods are highly useful for monitoring.
- Threshold pitfalls (Nagios era): Single-point checks vs fixed warn/crit thresholds will create noise & flapping; flap detection only masks bad alert design.
- Modern approach: Decouple collection and evaluation, collect to a TSDB (e.g., via agents) at intervals, then evaluate over windows of data.
- Key shift: keep historical metrics to enables trends, smarter detection, and fewer false positives.

Core Tools

- Mean (average) & moving average: Smooths spikes; improves readability but hides extremes, choose smoothing windows carefully.
- Median: Better than mean for skewed datasets (robust to outliers).
- Seasonality: Repeating patterns (hourly/daily/weekly).
  - Compare now vs same time last week/day to detect anomalies.
- Quantiles/percentiles (e.g., p95, p99): Summarize "most users'" experience (latency, bandwidth).
- Design better alerts:
  - Alert on "is it working?" user-journey checks, not raw OS metrics.
  - Evaluate over time windows (rates/percentiles), not single samples.

Observability

Observability Requirements

Maintaining Logs and Metrics

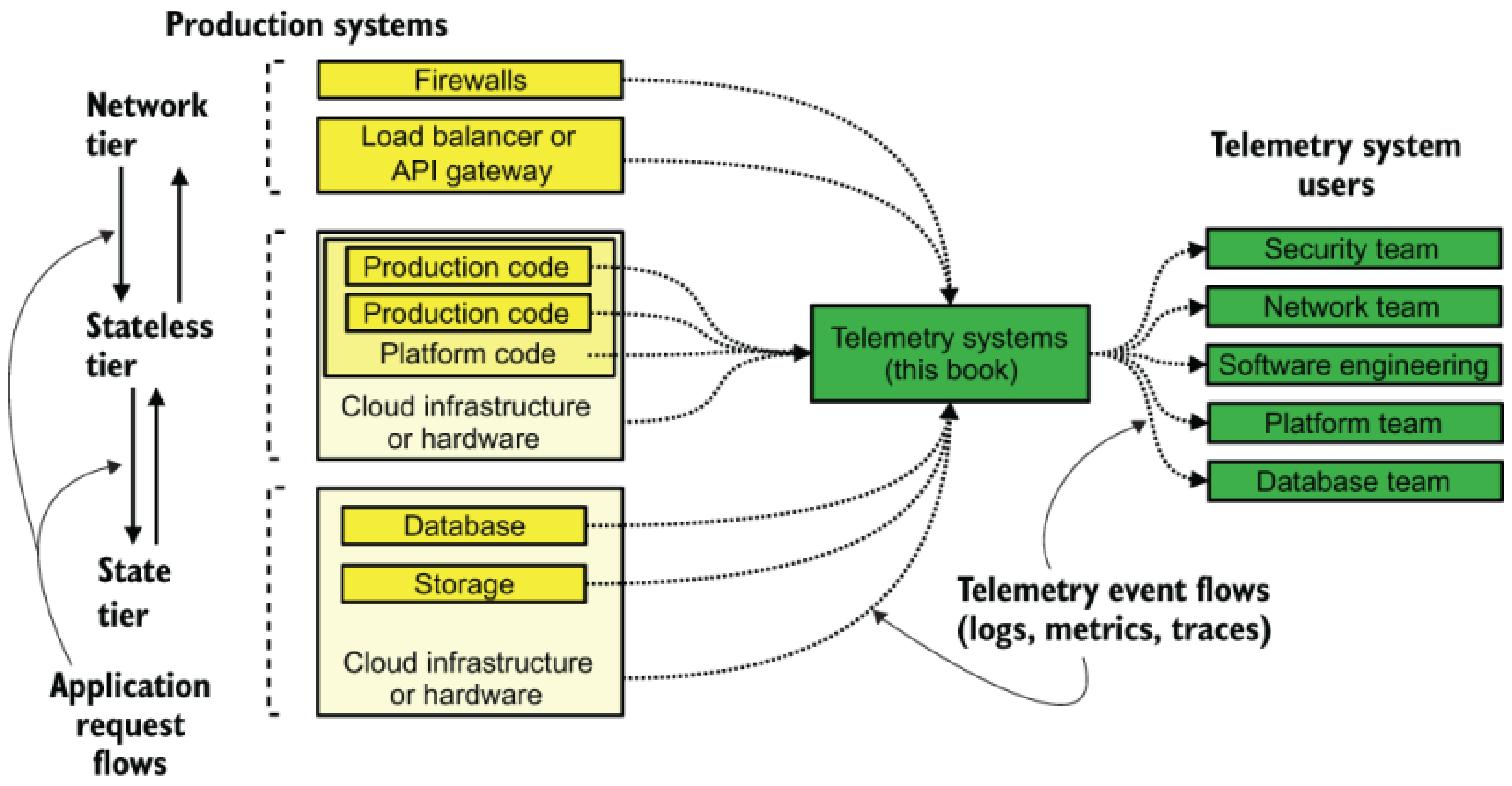
- Check if new functionality needs new or updated logs/metrics
- Clarify edge cases (e.g., WARN vs ERROR) → better discussions
- Encourage observability as a first-class concern
- Avoid last-minute, poorly tested logging
- Enable richer monitoring & testing in production (incl. canary releases)

Observability Requirements

Maintaining Dashboards and Alerts

- Dashboards/alerts often overlooked → include in “definition of done”
- Updates vary: none, minor tweaks, or full restructuring
- Ensure observability keeps pace with deployments
- Testing in production should check dashboards, not just user behavior
- Guarantees correct logs/metrics and timely notifications when failures occur

Software Telemetry - example



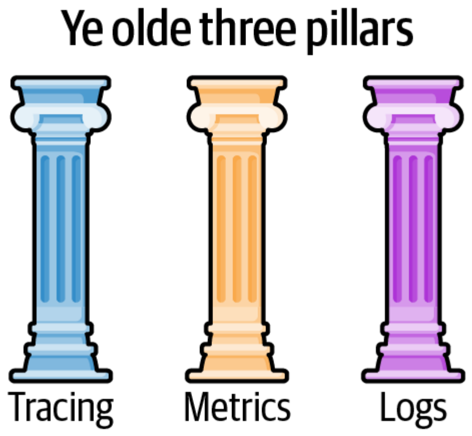
Unifying the three features – event based

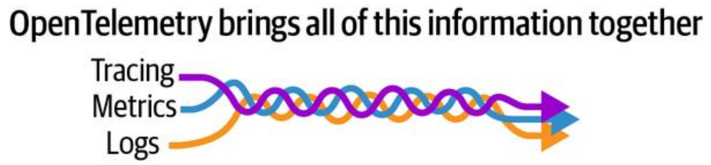
All of these is based on the events.

- **Logging**: recording events
- **Metrics**: data combined from measuring events
- **Tracing**: recording events with causality order

Use latency to compare a few tools:

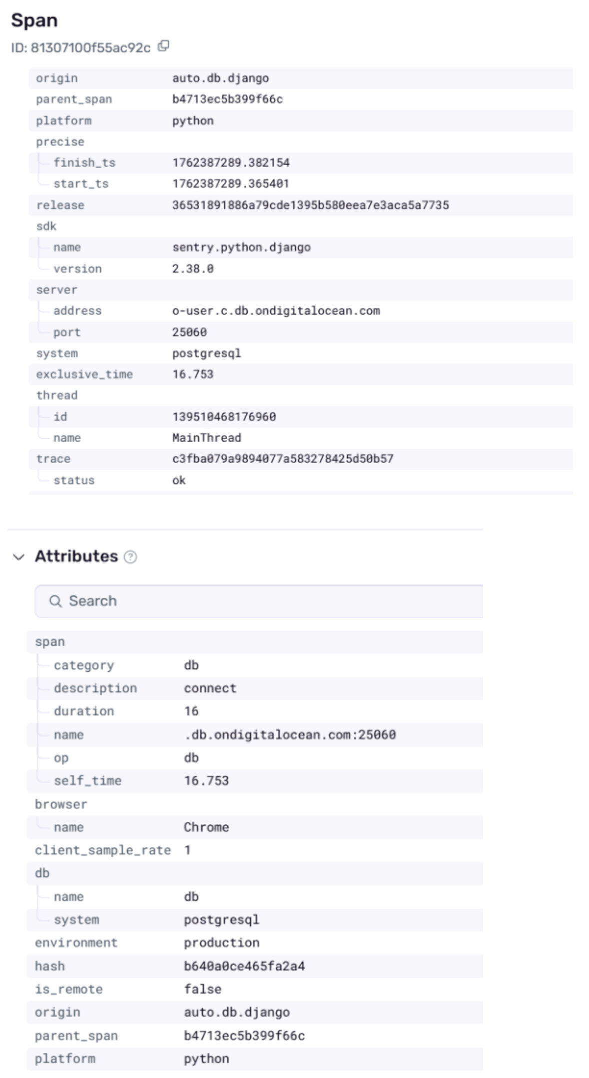
- Log - event (response time)
- Metric - value (response time)
- Trace - tree (response time)





Spans

- Span = operation within a transaction.
- Span contains
  - Parent’s Span identifier
  - An operation name
  - A start and finish timestamp
  - Attributes will be key-value pairs.
  - A set of zero or more Events, each of which it itself a tuple (timestamp, name, attributes)



Span

ID: 81307100f55ac92c

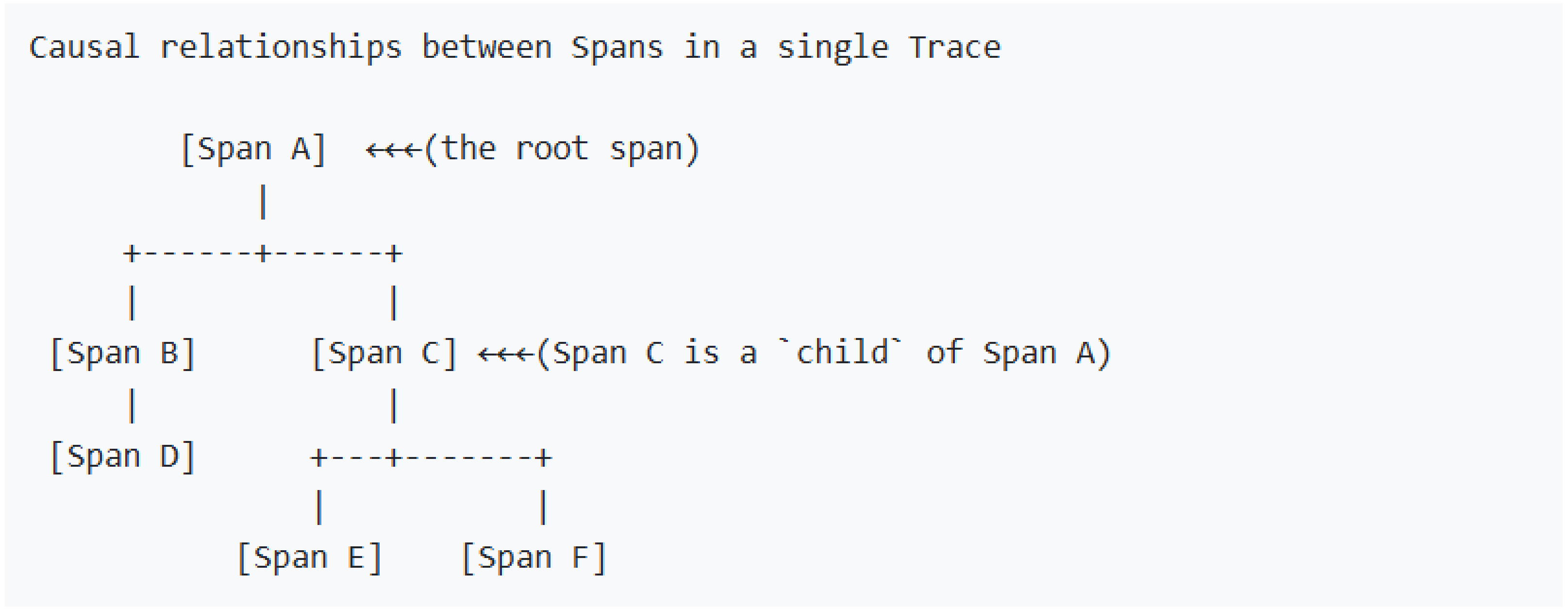
```
origin              auto.db.django
parent_span         b4713ec5b399f66c
platform            python
precise
  ├ finish_ts       1762387289.382154
  └ start_ts        1762387289.365401
release             36531891886a79cde1395b580eea7e3aca5a7735
sdk
  ├ name            sentry.python.django
  └ version         2.38.0
server
  ├ address         o-user.c.db.ondigitalocean.com
  └ port            25060
system              postgresql
exclusive_time      16.753
thread
  ├ id              139510468176960
  └ name            MainThread
trace               c3fba079a9894077a583278425d50b57
  └ status          ok
```

Attributes

```
span
  ├ category          db
  ├ description       connect
  ├ duration          16
  ├ name              .db.ondigitalocean.com:25060
  ├ op                db
  └ self_time         16.753
browser
  └ name              Chrome
client_sample_rate    1
db
  ├ name              db
  └ system            postgresql
environment           production
hash                  b640a0ce465fa2a4
is_remote             false
origin                auto.db.django
parent_span           b4713ec5b399f66c
platform              python
```

Tracing | Graph view

➢ Traces can be seen as **directed acyclic graph** of **Spans**



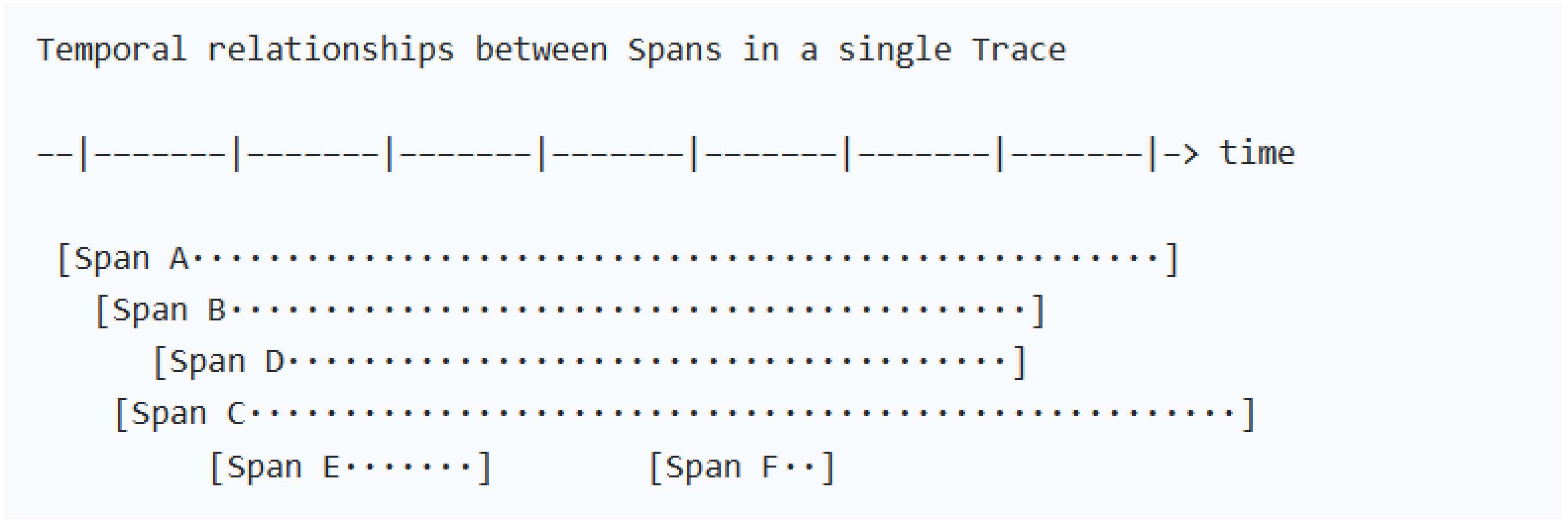
```
Causal relationships between Spans in a single Trace

        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
 [Span B]      [Span C] ←←←(Span C is a `child` of Span A)
     |             |
 [Span D]      +---+-------+
               |           |
           [Span E]    [Span F]
```

Source: https://opentelemetry.io/docs/specs/otel/overview/

Tracing | Temporal View

➢ Traces can be seen as **directed acyclic graph** of **Spans**



```
Temporal relationships between Spans in a single Trace

--|-------|-------|-------|-------|-------|-------|-------|-> time

 [Span A···················································]
   [Span B··········································]
      [Span D······································]
    [Span C····················································]
         [Span E·······]        [Span F··]
```

Source: https://opentelemetry.io/docs/specs/otel/overview/

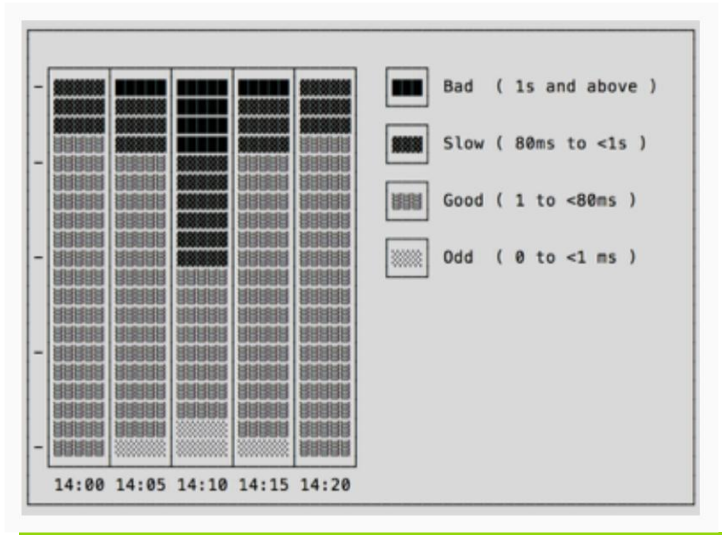
Latency comparison

Logging

127.0.0.1 - - [12/Sep/2025:12:44:32 +0100] "GET /api/v1/client/123 HTTP/1.1" 200 512 "https://example.com/dashboard" "curl/7.88.1" ***0.092***

The response time is 92 ms.

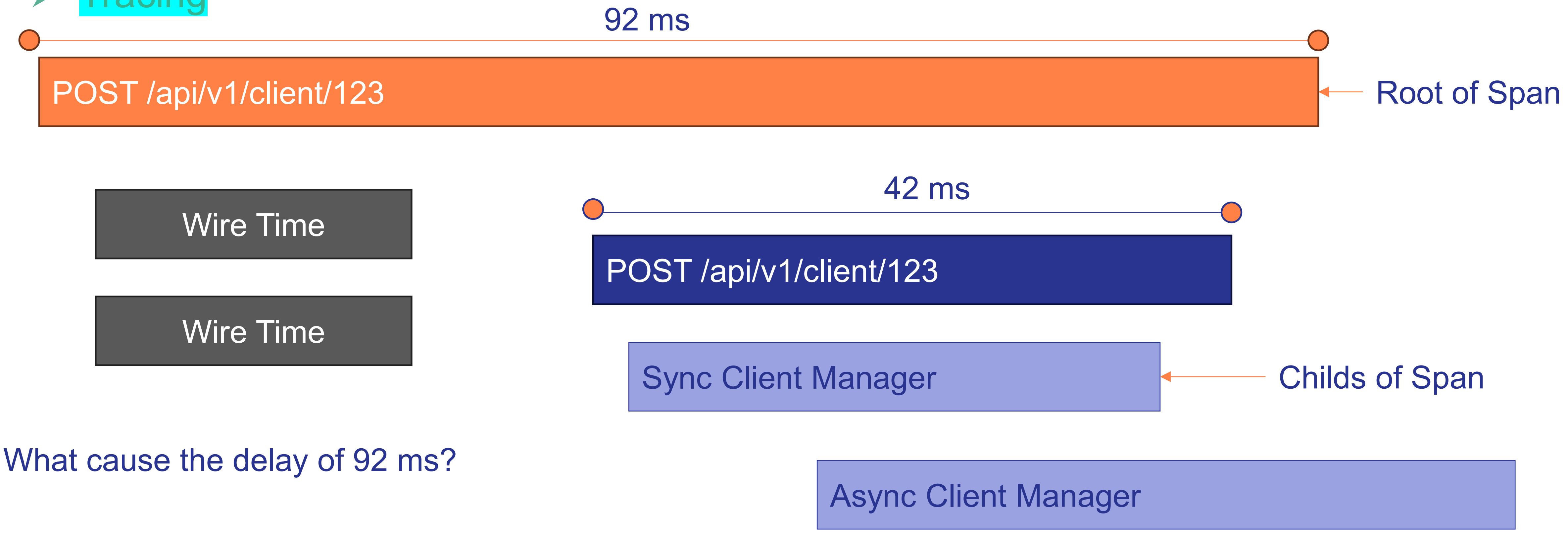
Metrics



Is the system slow? How fast were the requests at 14:19?

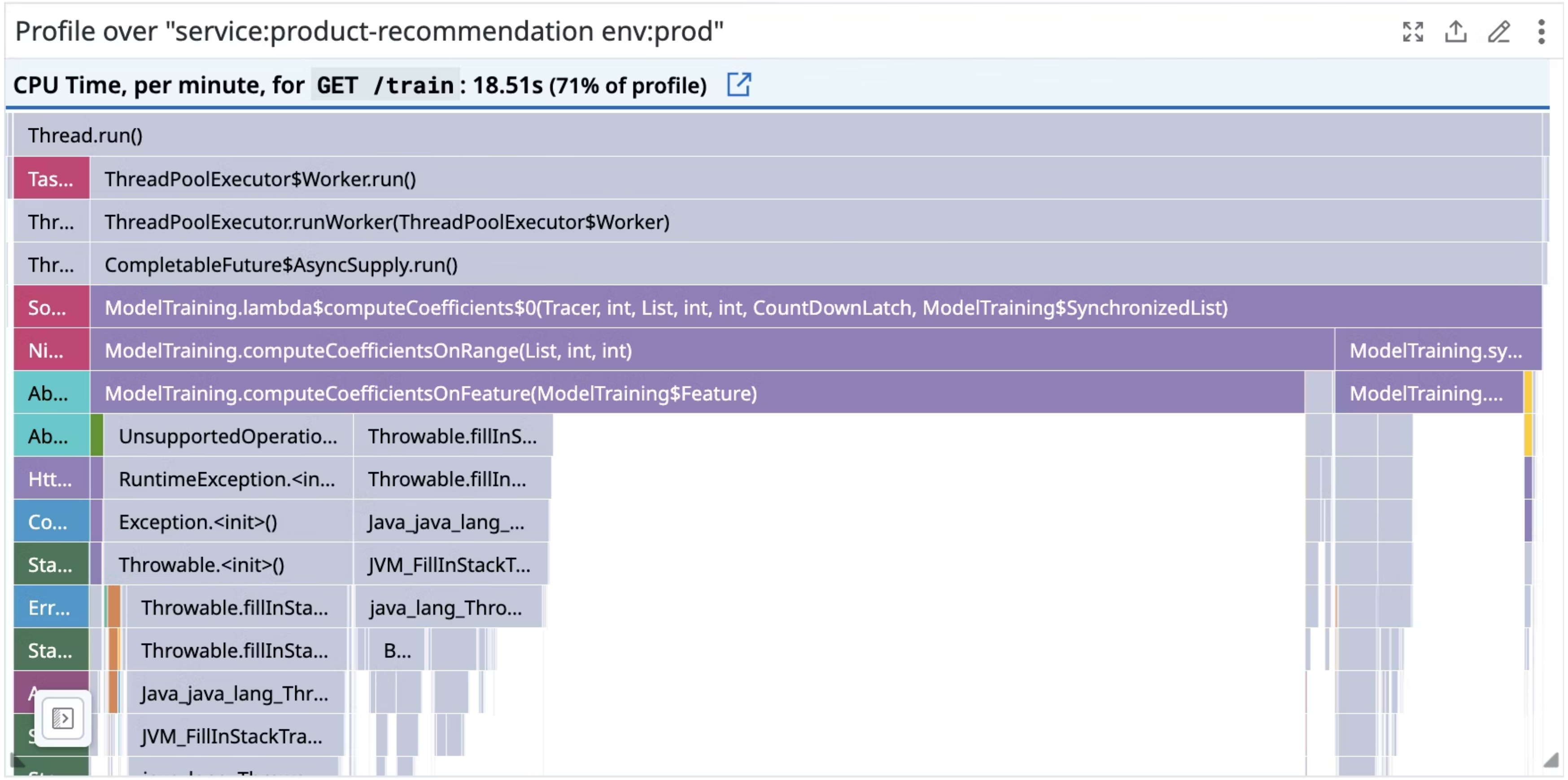
Trace | Latency comparison

- Tracing



What cause the delay of 92 ms?

Trace – another example


Profile over "service:product-recommendation env:prod"

CPU Time, per minute, for `GET /train` : 18.51s (71% of profile)

```
Thread.run()
Tas...  ThreadPoolExecutor$Worker.run()
Thr...  ThreadPoolExecutor.runWorker(ThreadPoolExecutor$Worker)
Thr...  CompletableFuture$AsyncSupply.run()
So...   ModelTraining.lambda$computeCoefficients$0(Tracer, int, List, int, int, CountDownLatch, ModelTraining$SynchronizedList)
Ni...   ModelTraining.computeCoefficientsOnRange(List, int, int)                ModelTraining.sy...
Ab...   ModelTraining.computeCoefficientsOnFeature(ModelTraining$Feature)       ModelTraining....
Ab...     UnsupportedOperatio...   Throwable.fillInS...
Htt...    RuntimeException.<in...  Throwable.fillIn...
Co...     Exception.<init>()       Java_java_lang_...
Sta...    Throwable.<init>()       JVM_FillInStackT...
Err...      Throwable.fillInSta... java_lang_Thro...
Sta...      Throwable.fillInSta...   B...
            Java_java_lang_Thr...
            JVM_FillInStackTra...
```

Source: https://docs.datadoghq.com/fr/dashboards/widgets/profiling_flame_graph/

Observability tools

- Prometheus: Metrics collection & time-series storage; strong in cloud-native/Kubernetes.
- Grafana: Visualization and dashboards; integrates with multiple data sources.
- Elastic Stack (ELK/Opensearch): Elasticsearch (storage/search), Logstash (processing), Kibana (visualization) / EDOT (Elastic Observability).
- OpenTelemetry (OTel): Vendor-neutral framework for generating and exporting traces, metrics, logs.
- Jaeger: Distributed tracing, often paired with OpenTelemetry.
- Fluentd / Fluent Bit: Log forwarding and aggregation; lightweight and extensible.
- VictoriaMetrics / Thanos / Cortex: Scalable TSDB solutions, often Prometheus-compatible.

Feature Usage Tracking

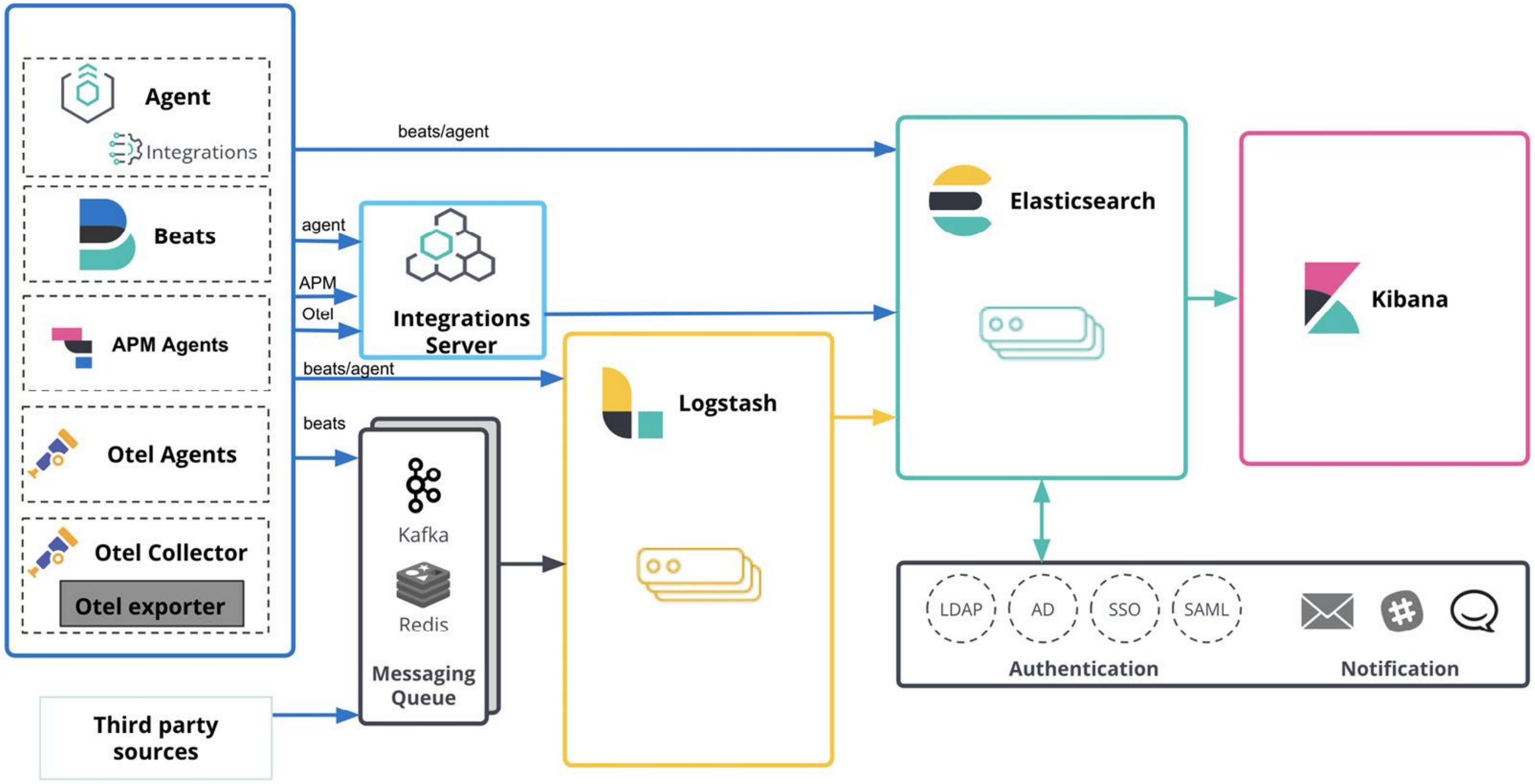
…is related, but it's not exactly the same as monitoring and observability in the strict reliability engineering sense.

- **Definition:** Collecting data on how users interact with the system, which features are used, how often, in what sequence, and by whom. The end goal is to understand user behavior and product adoption, not just system reliability.
- **Techniques/Tools**: Event logging, product analytics (Mixpanel, Amplitude, Google Analytics, Matomo, Elastic, Sentry, custom telemetry pipelines).

mixpanel

Google Analytics

Elastic Observability Solution



OpenTelemetry SpringBoot starter

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.opentelemetry.instrumentation</groupId>
      <artifactId>opentelemetry-instrumentation-bom</artifactId>
      <version>2.10.0</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
...
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
  <groupId>io.opentelemetry.instrumentation</groupId>
  <artifactId>opentelemetry-spring-boot-starter</artifactId>
</dependency>
```

OpenTelemetry SpringBoot starter

```
management.otlp.tracing.endpoint=http://localhost:4318/v1/traces
management.otlp.tracing.timeout=10s
management.tracing.sampling.probability=1

management.otlp.metrics.export.url=http://localhost:4318/v1/metrics
management.otlp.metrics.export.step=5s
management.opentelemetry.resource-attributes."service-name"=${spring.application.name}
```

OpenTelemetry SpringBoot starter

```
import io.opentelemetry.api.OpenTelemetry;
import io.opentelemetry.api.trace.Tracer;

@Controller
public class MyController {
  private final Tracer tracer;

  public MyController(OpenTelemetry openTelemetry) {
    this.tracer = openTelemetry.getTracer("application");
  }
}
```

Practice Guide

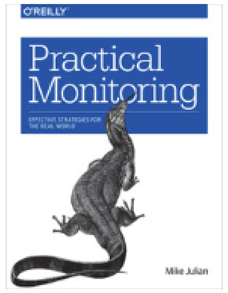
➢ In the project:

- Implementation of health checks in the project's containers.
- Implementation of persistent structured logs, following a structured logging methodology that allows easy understanding of system behavior.
- Definition of the SLOs/SLIs that your system aims to meet (must be documented in docs/observability.md).
- Implementation of observability in the backend component (e.g., via OTLP / OpenTelemetry).

Use the following label: category:observability

Tools: http://cloud.elastic.co/ | You can setup a trial of 14 days for this Sprint (only a member of the team should enable the trial).

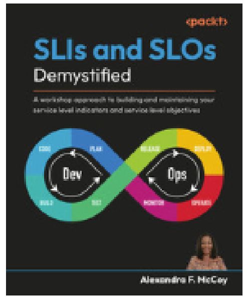
Bibliography



Practical Monitoring

Mike Julian

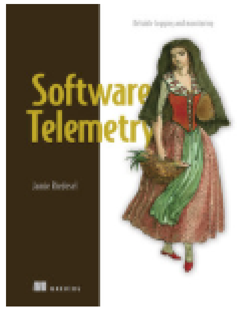
Published by O'Reilly Media, Inc.



SLIs and SLOs Demystified

Alexandra F. McCoy

Published by Packt Publishing



Software Telemetry

Jamie Riedesel

Published by Manning Publications