Observability first Kafka: Visibility at scale

Kafka is the backbone of modern data streaming. Issues like broker crashes must be detected early to prevent downtime.

We'll explore how the MELT stack (Metrics, Events, Logs, Traces) provides complete visibility into Kafka systems.

Abhishek Walia

Why Kafka Needs Observability

Critical Infrastructure

Kafka serves as the backbone for data streaming in modern systems.

Early Detection

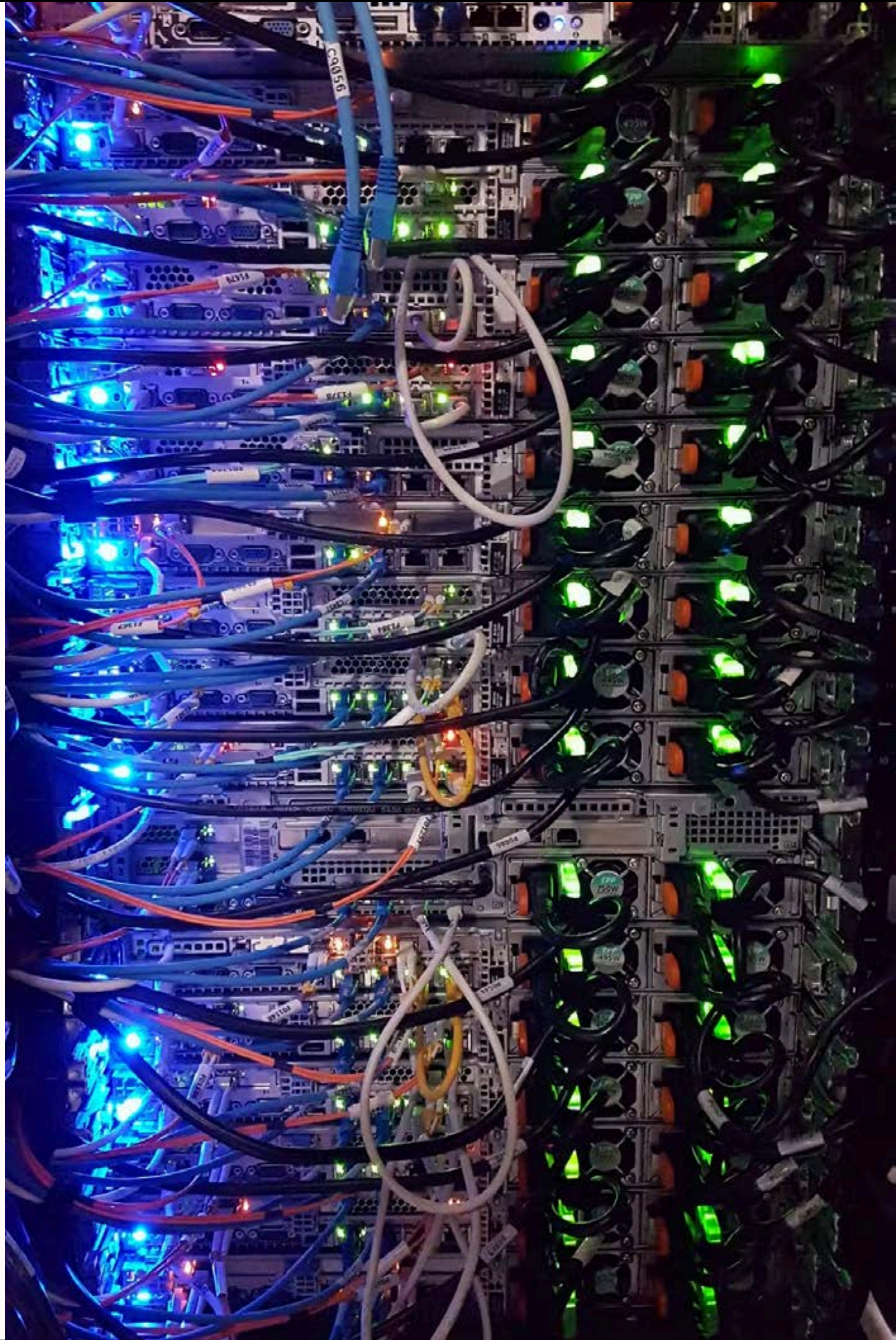
Issues like broker crashes and replication lag must be caught early.

Performance Insights

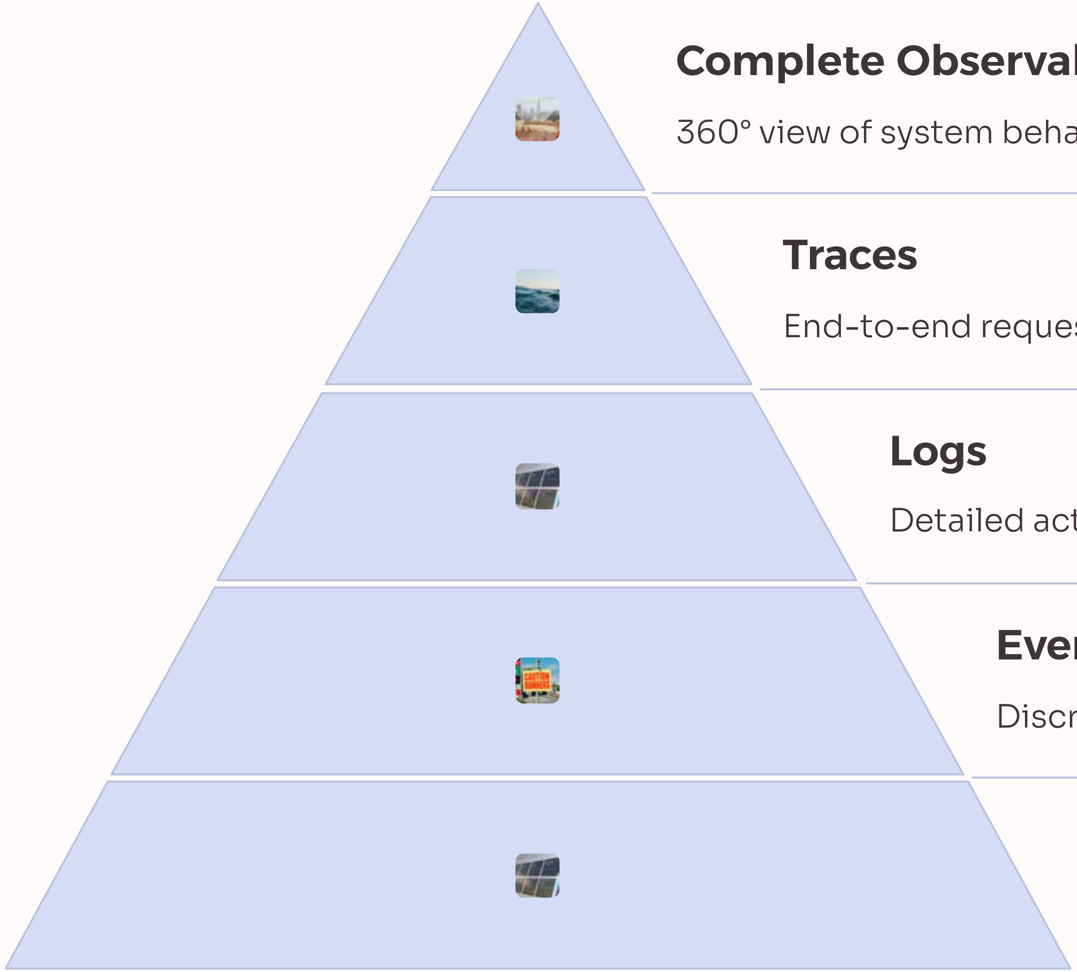
Engineers need visibility into reliability and performance metrics.

Fast Troubleshooting

Observability enables quick resolution when problems occur.



The MELT Stack Overview



Complete Observability

360° view of system behavior

Traces

End-to-end request flows

Logs

Detailed activity records

Events

Discrete state changes

Metrics

Numerical measurements

Open-Source Toolchain

Prometheus

Collects metrics and manages alerts

OpenTelemetry

Provides instrumentation across all signals

Grafana

Visualizes metrics, logs, and traces

Metrics: System Health Indicators

What Are Metrics?

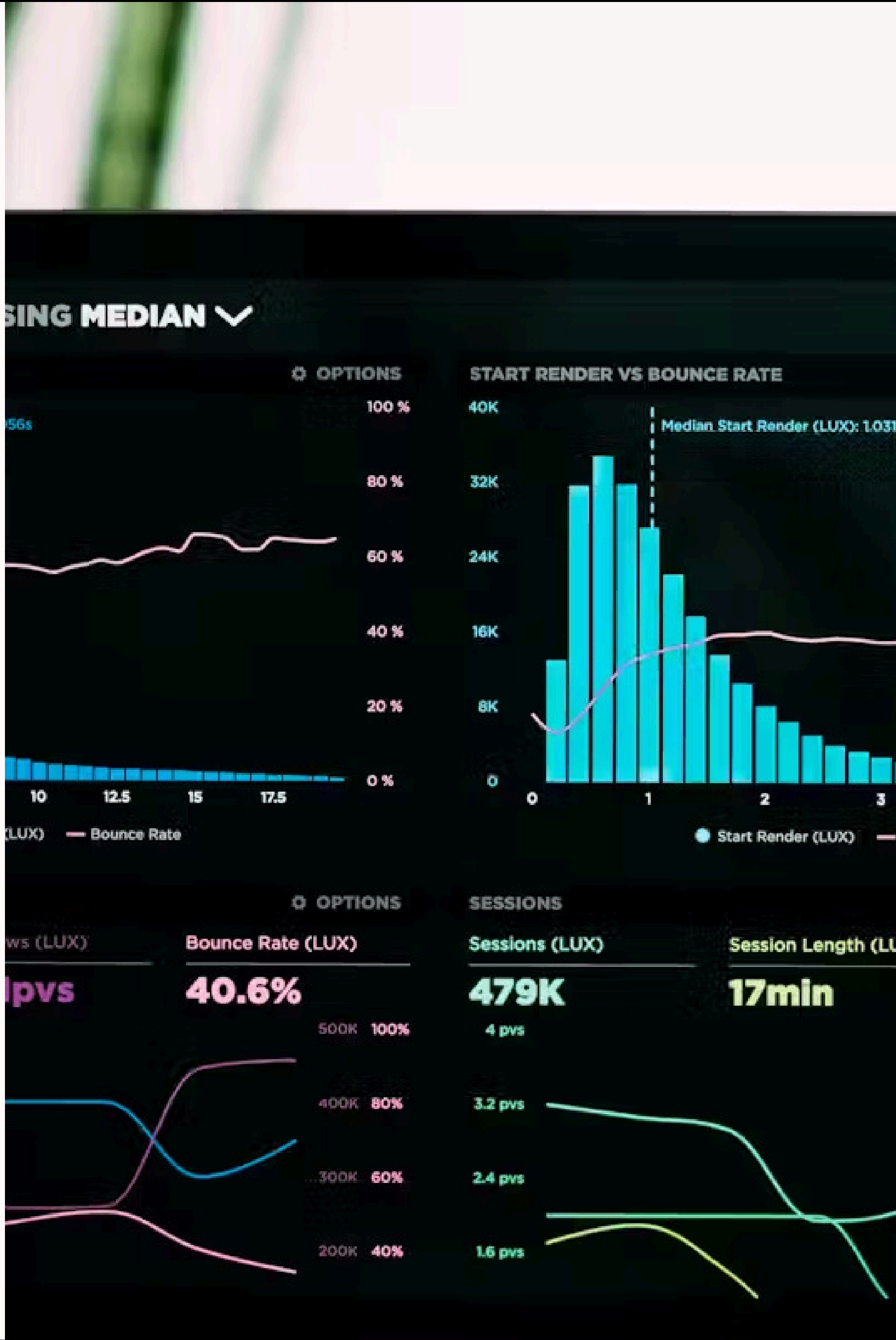
Numerical measurements indicating system health and performance. Examples include CPU utilization, request latency, and error rates.

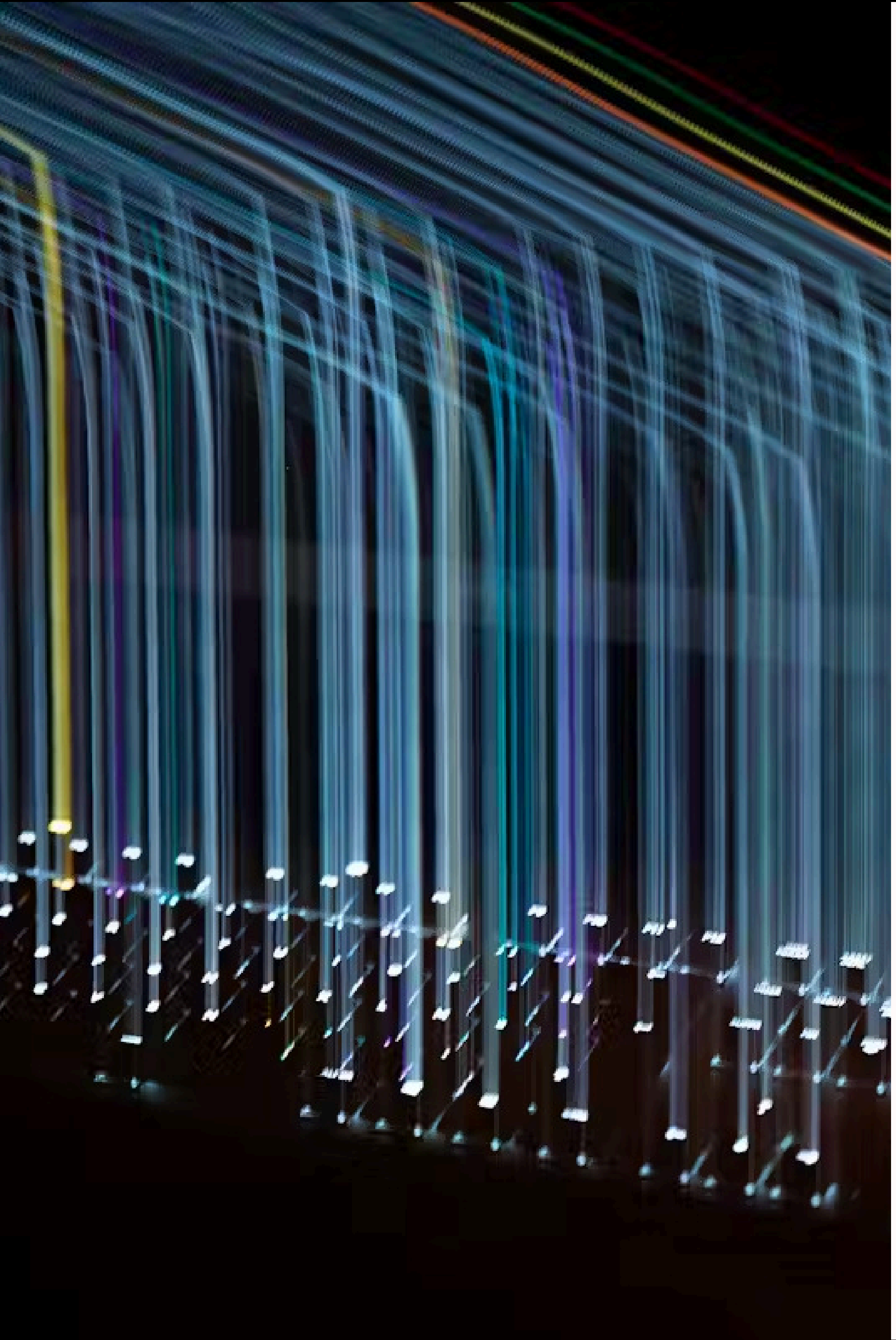
Why They Matter

Metrics provide a high-level overview of system state. They enable real-time anomaly detection through alerts.

For Kafka

Key metrics include throughput, consumer lag, broker resource usage, and replication health.





Events: Context for Changes

What Are Events?

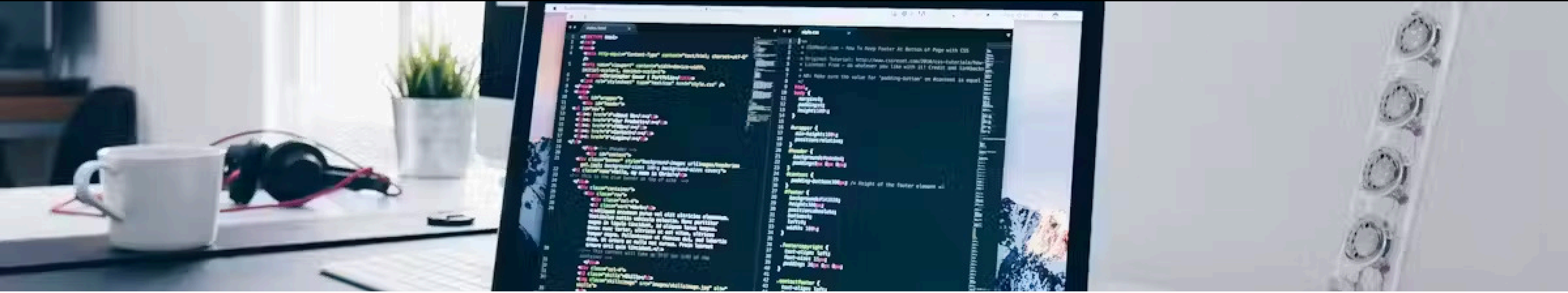
Discrete state changes or significant actions occurring at a point in time. Examples include deployments, crashes, and configuration changes.

Why They Matter

Events correlate system changes with observed issues. They provide context to metric spikes or errors.

For Kafka

Important events include broker deployments, scaling actions, and consumer group rebalances.



Logs: Detailed Activity Records

What Are Logs?

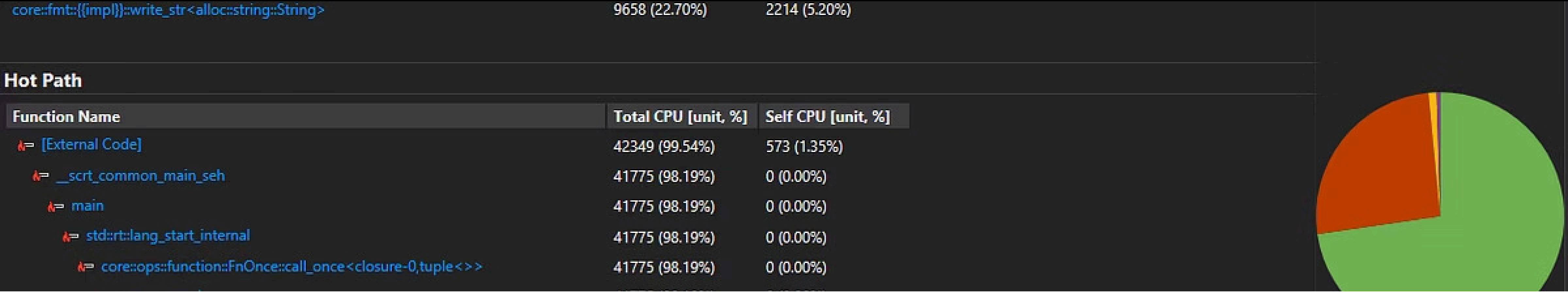
Timestamped, detailed records of system activities. Examples include server logs, error stack traces, and authentication failures.

Why They Matter

Logs offer granular insights into what happened and when. They're essential for debugging incidents.

For Kafka

Broker logs show failures, controller elections, client disconnections, and processing exceptions.



core::fmt::{{impl}}::write_str<alloc::string::String>	9658 (22.70%)	2214 (5.20%)

Hot Path

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
[External Code]	42349 (99.54%)	573 (1.35%)
__scrt_common_main_seh	41775 (98.19%)	0 (0.00%)
main	41775 (98.19%)	0 (0.00%)
std::rt::lang_start_internal	41775 (98.19%)	0 (0.00%)
core::ops::function::FnOnce::call_once<closure-0,tuple<>>	41775 (98.19%)	0 (0.00%)

Traces: End-to-End Request Flows

What Are Traces?

End-to-end records of operations as they propagate through distributed systems. Example: a request traveling through multiple microservices.

Why They Matter

Traces link components of a workflow. They help identify bottlenecks by measuring transaction flow.

For Kafka

Traces show message paths from producer through broker to consumer, revealing processing delays.

Why Use All Four MELT Signals

Complementary Perspectives

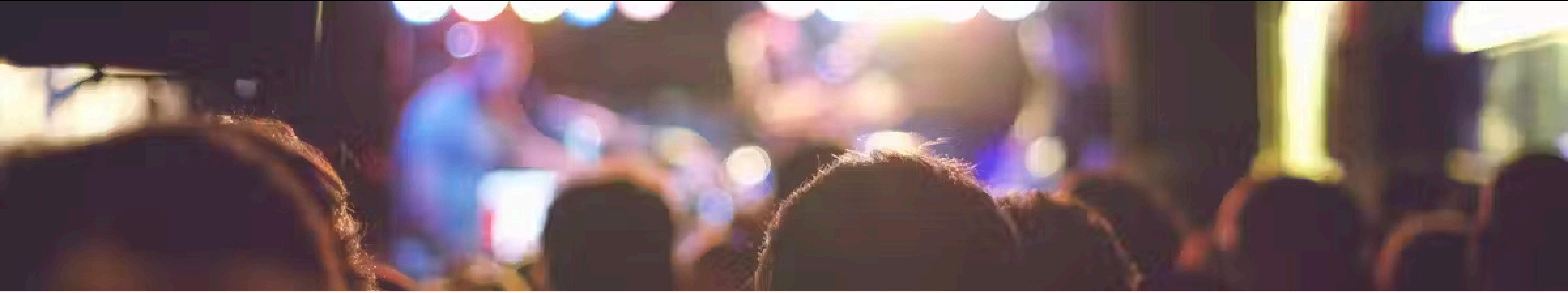
Each telemetry type provides a different view of system behavior.

- Metrics detect anomalies in real-time
- Events tie anomalies to specific changes
- Logs provide deep details for debugging
- Traces show end-to-end request paths

Complete Picture

Using all four signals together creates a 360° view that single-point monitoring can't achieve.

Example: A latency spike (metric) correlates with a broker restart (event), with details in logs and the exact slow component visible in traces.



Kafka Metrics: Observability Goals

Throughput

Messages per second published or consumed

Consumer Lag

Delay between production and consumption

Resource Usage

CPU, memory, and disk I/O on brokers

Request Latency

Time to serve produce/fetch requests

Kafka Metrics: More Key Indicators

Replication Health

Number of under-replicated partitions

Partition Distribution

Leader counts per broker

Error Rates

Failed requests and other error indicators

Network Utilization

Bytes in/out across the cluster

Kafka Metrics: Instrumentation

Enable JMX Metrics

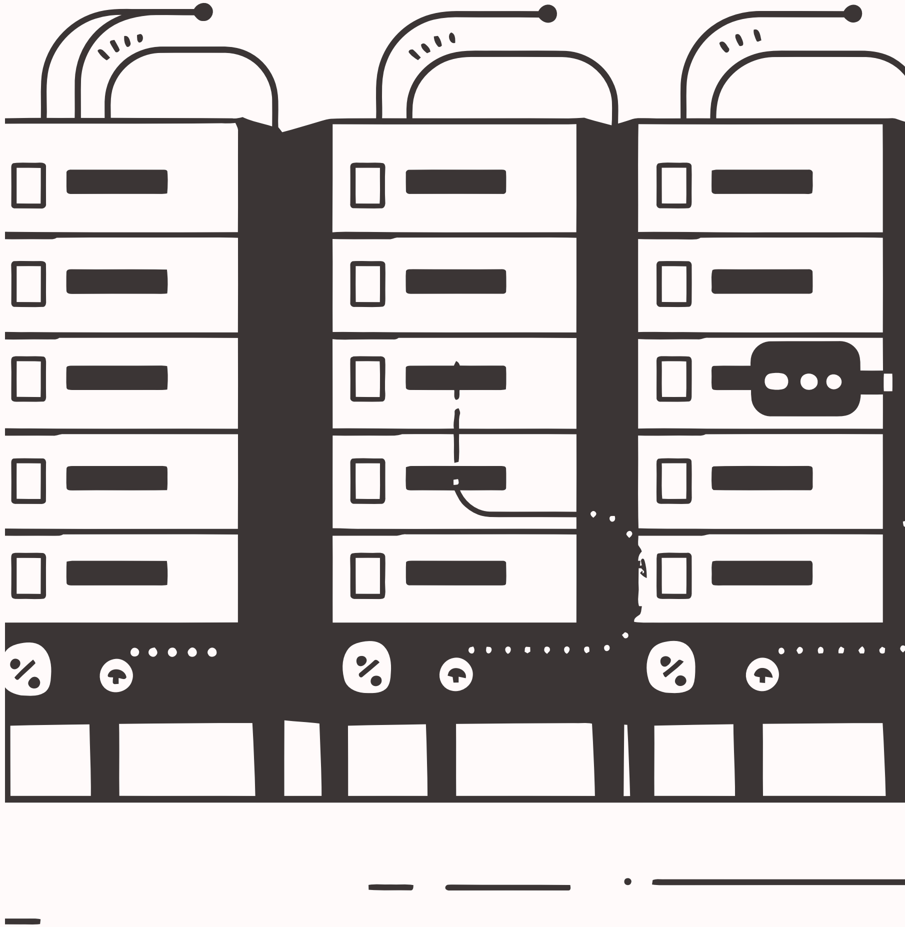
Apache Kafka exposes metrics via Java Management Extensions (JMX) on both brokers and clients.

Deploy Metrics Scraper

Use Prometheus JMX exporter or OpenTelemetry JMX collector to pull metrics from brokers.

Collect Client Metrics

With Kafka 3.7+ (KIP-714), brokers can centrally collect standardized client metrics.



Kafka Metrics: Implementation Example

Kafka

Brokers & clients expose metrics

OpenTelemetry

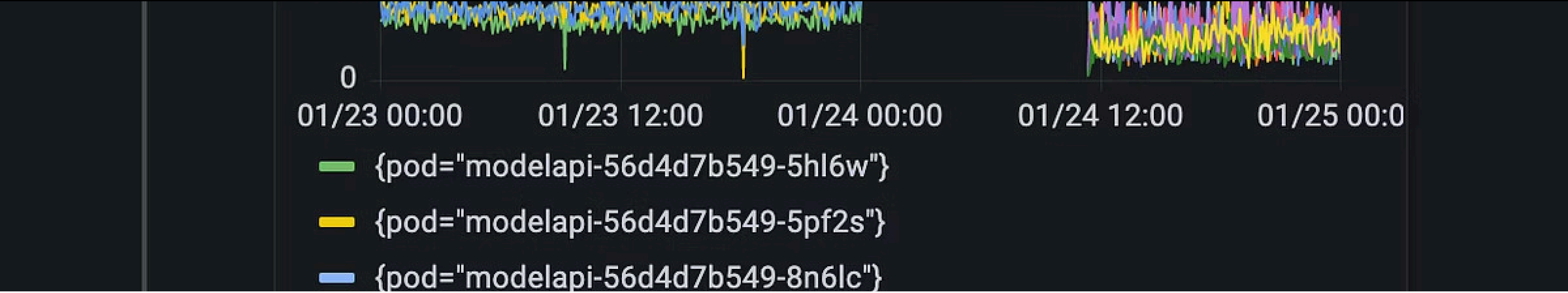
Collector receives and processes metrics

Prometheus

Stores time-series data

Grafana

Visualizes metrics on dashboards



Kafka Events: Observability Goals

Infrastructure Changes

Broker deployments, restarts, or failures

Scaling Actions

Adding/removing brokers, partition reassignments

Configuration Changes

Topic configuration updates, ACL changes

Consumer Group Events

Rebalances, new consumers joining

Kafka Events: Why They Matter

Context for Anomalies

Events explain why metrics might change suddenly.

Example: If throughput dips at 3:00 PM, an event log might show a broker was taken down for maintenance.

Correlation Benefits

- Faster root cause analysis
- Clear timeline of changes
- Reduced troubleshooting time
- Better understanding of cause-effect

Kafka Events: Instrumentation

Automate Event Logging

Add hooks in deployment scripts or Kubernetes operators to log events when actions occur.

Leverage Kafka's Signals

Parse Kafka logs for specific keywords indicating important state changes.

Include External Events

Capture OS/hardware events or network issues that might impact Kafka performance.

Kafka Events: Implementation Example

Event Sources

Deployment tools, Kafka logs, infrastructure changes

2

OpenTelemetry Collector

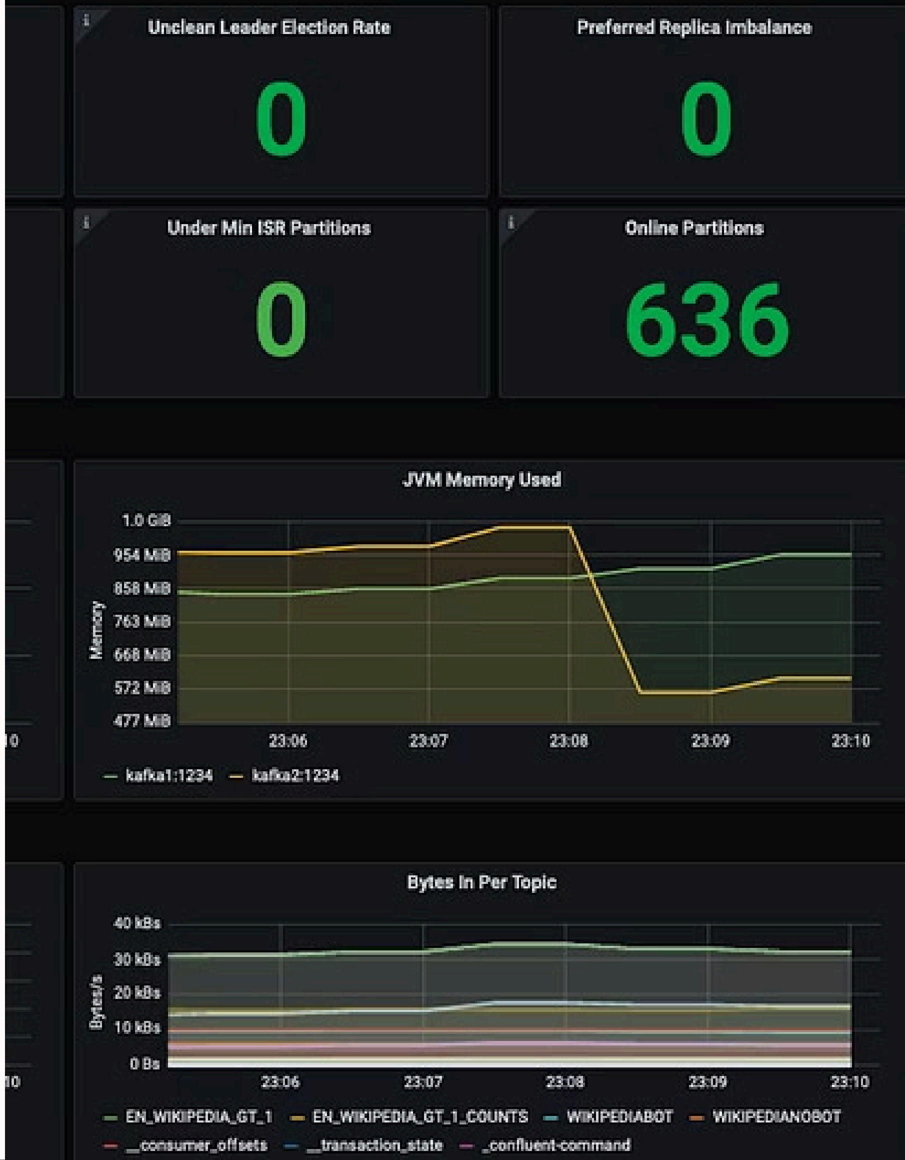
Captures and forwards event data

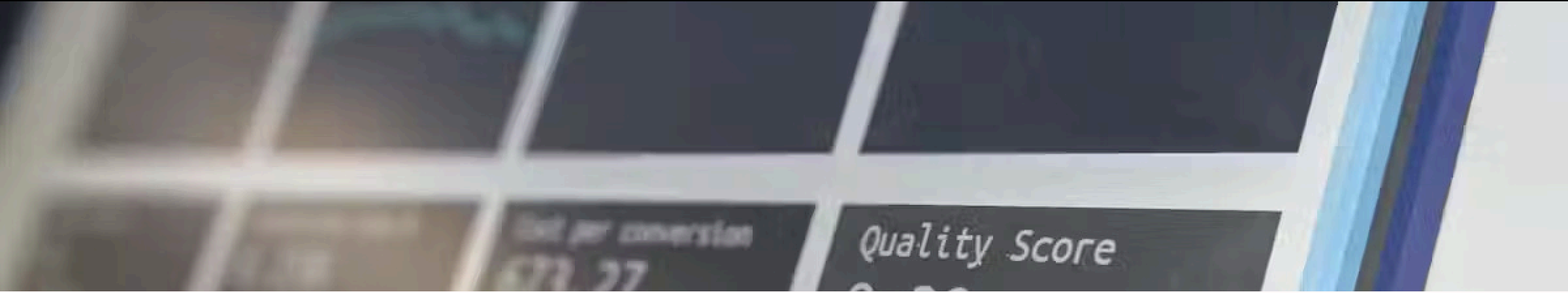
Grafana Loki

Stores event logs for querying

Grafana Annotations

Displays events as markers on metric graphs





Kafka Logs: Observability Goals

Detailed Troubleshooting

Logs contain rich details about internal state and errors that metrics alone can't explain.

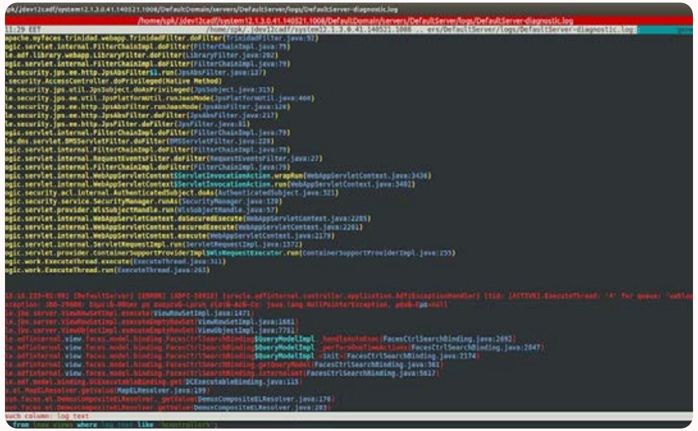
Error Investigation

Logs show when brokers experience failures, when exceptions occur, or when clients disconnect.

Root Cause Analysis

Centralized access to broker logs helps identify specific errors during failures.

Kafka Logs: What They Reveal



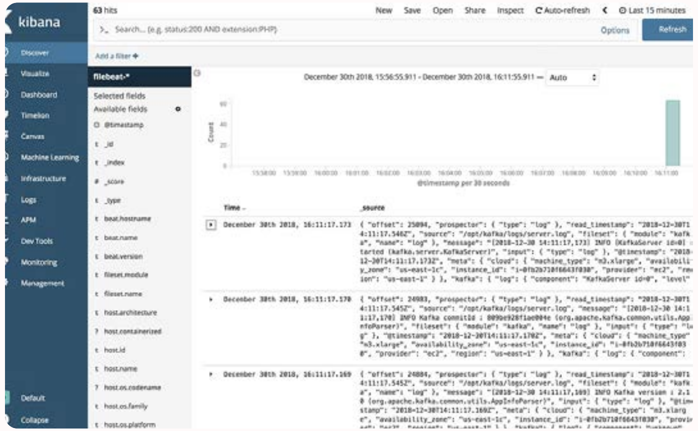
```
pk/jdev12cadf/system12.1.3.0.41.140521.1008/DefaultDomain/servers/DefaultServer/logs/DefaultServer-diagnostic.log
/home/spk/jdev12cadf/system12.1.3.0.41.140521.1008/DefaultDomain/servers/DefaultServer/logs/DefaultServer-diagnostic.log
11:29 EET                /home/spk/.jdev12cadf/system12.1.3.0.41.140521.1008 .. ers/DefaultServer/logs/DefaultServer-diagnostic.log
pache.myfaces.trinidad.webapp.TrinidadFilter.doFilter(TrinidadFilter.java:92)
ogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:79)
le.adf.library.webapp.LibraryFilter.doFilter(LibraryFilter.java:202)
ogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:79)
le.security.jps.ee.http.JpsAbsFilter$1.run(JpsAbsFilter.java:137)
.security.AccessController.doPrivileged(Native Method)
le.security.jps.util.JpsSubject.doAsPrivileged(JpsSubject.java:315)
le.security.jps.ee.util.JpsPlatformUtil.runJaasMode(JpsPlatformUtil.java:460)
le.security.jps.ee.http.JpsAbsFilter.runJaasMode(JpsAbsFilter.java:120)
le.security.jps.ee.http.JpsAbsFilter.doFilter(JpsAbsFilter.java:217)
le.security.jps.ee.http.JpsFilter.doFilter(JpsFilter.java:81)
ogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:79)
le.dms.servlet.DMSServletFilter.doFilter(DMSServletFilter.java:220)
ogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:79)
ogic.servlet.internal.RequestEventsFilter.doFilter(RequestEventsFilter.java:27)
ogic.servlet.internal.FilterChainImpl.doFilter(FilterChainImpl.java:79)
ogic.servlet.internal.WebAppServletContext$ServletInvocationAction.wrapRun(WebAppServletContext.java:3436)
ogic.servlet.internal.WebAppServletContext$ServletInvocationAction.run(WebAppServletContext.java:3402)
ogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:321)
ogic.security.service.SecurityManager.runAs(SecurityManager.java:120)
ogic.servlet.provider.WlsSubjectHandle.run(WlsSubjectHandle.java:57)
ogic.servlet.internal.WebAppServletContext.doSecuredExecute(WebAppServletContext.java:2285)
ogic.servlet.internal.WebAppServletContext.securedExecute(WebAppServletContext.java:2201)
ogic.servlet.internal.WebAppServletContext.execute(WebAppServletContext.java:2179)
ogic.servlet.internal.ServletRequestImpl.run(ServletRequestImpl.java:1572)
ogic.servlet.provider.ContainerSupportProviderImpl$WlsRequestExecutor.run(ContainerSupportProviderImpl.java:255)
ogic.work.ExecuteThread.execute(ExecuteThread.java:311)
ogic.work.ExecuteThread.run(ExecuteThread.java:263)

[illegible] [DefaultServer] [ERROR] [ADFC-50018] [oracle.adfinternal.controller.application.AdfcExceptionHandler] [tid: [ACTIVE].ExecuteThread: '4' for queue: 'weblo
xception: [illegible] java.lang.NullPointerException, [illegible]=null
le.jbo.server.ViewRowSetImpl.execute(ViewRowSetImpl.java:1471)
le.jbo.server.ViewRowSetImpl.executeEmptyRowSet(ViewRowSetImpl.java:1681)
le.jbo.server.ViewObjectImpl.executeEmptyRowSet(ViewObjectImpl.java:7791)
le.adfinternal.view.faces.model.binding.FacesCtrlSearchBinding$QueryModelImpl._handleAutoExec(FacesCtrlSearchBinding.java:2692)
le.adfinternal.view.faces.model.binding.FacesCtrlSearchBinding$QueryModelImpl._performOneTimeActions(FacesCtrlSearchBinding.java:2047)
le.adfinternal.view.faces.model.binding.FacesCtrlSearchBinding$QueryModelImpl.<init>(FacesCtrlSearchBinding.java:2174)
le.adfinternal.view.faces.model.binding.FacesCtrlSearchBinding.getQueryModel(FacesCtrlSearchBinding.java:361)
le.adfinternal.view.faces.model.binding.FacesCtrlSearchBinding.internalGet(FacesCtrlSearchBinding.java:5617)
le.adf.model.binding.DCExecutableBinding.get(DCExecutableBinding.java:115)
.el.MapELResolver.getValue(MapELResolver.java:199)
un.faces.el.DemuxCompositeELResolver._getValue(DemuxCompositeELResolver.java:176)
un.faces.el.DemuxCompositeELResolver.getValue(DemuxCompositeELResolver.java:203)
uch column: log text
 from [illegible] where [illegible] like '%controllers%';
```

Error Details

Stack traces and exception messages that explain failures

State Changes

Controller elections, partition movements, and leadership changes



```
December 30th 2018, 16:11:17.173  { "offset": 25094, "prospector": { "type": "log" }, "read_timestamp": "2018-12-30T1
                                  4:11:17.546Z", "source": "/opt/kafka/logs/server.log", "fileset": { "module": "kafk
                                  a", "name": "log" }, "message": "[2018-12-30 14:11:17,173] INFO [KafkaServer id=0] s
                                  tarted (kafka.server.KafkaServer)", "input": { "type": "log" }, "@timestamp": "2018-
                                  12-30T14:11:17.173Z", "meta": { "cloud": { "machine_type": "m3.xlarge", "availabili
                                  y_zone": "us-east-1c", "instance_id": "i-0fb2b710f6643f030", "provider": "ec2", "reg
                                  ion": "us-east-1" } }, "kafka": { "log": { "component": "KafkaServer id=0", "level"
December 30th 2018, 16:11:17.170  { "offset": 24983, "prospector": { "type": "log" }, "read_timestamp": "2018-12-30T1
                                  4:11:17.545Z", "source": "/opt/kafka/logs/server.log", "message": "[2018-12-30 14:1
                                  1:17,170] INFO Kafka commitId : 809be928f1ae004e (org.apache.kafka.common.utils.App
                                  nfoParser)", "fileset": { "module": "kafka", "name": "log" }, "input": { "type": "lo
                                  g" }, "@timestamp": "2018-12-30T14:11:17.170Z", "meta": { "cloud": { "machine_type"
                                  "m3.xlarge", "availability_zone": "us-east-1c", "instance_id": "i-0fb2b710f6643f03
                                  0", "provider": "ec2", "region": "us-east-1" } }, "kafka": { "log": { "component":
December 30th 2018, 16:11:17.169  { "offset": 24884, "prospector": { "type": "log" }, "read_timestamp": "2018-12-30T1
                                  4:11:17.545Z", "source": "/opt/kafka/logs/server.log", "fileset": { "module": "kafk
                                  a", "name": "log" }, "message": "[2018-12-30 14:11:17,169] INFO Kafka version : 2.1
                                  0 (org.apache.kafka.common.utils.AppInfoParser)", "input": { "type": "log" }, "@tim
                                  stamp": "2018-12-30T14:11:17.169Z", "meta": { "cloud": { "machine_type": "m3.xlarg
                                  e", "availability_zone": "us-east-1c", "instance_id": "i-0fb2b710f6643f030", "provi
```

Client Activity

Connection/disconnection events and client-specific issues

Kafka Logs: Instrumentation

Configure High-Quality Logs

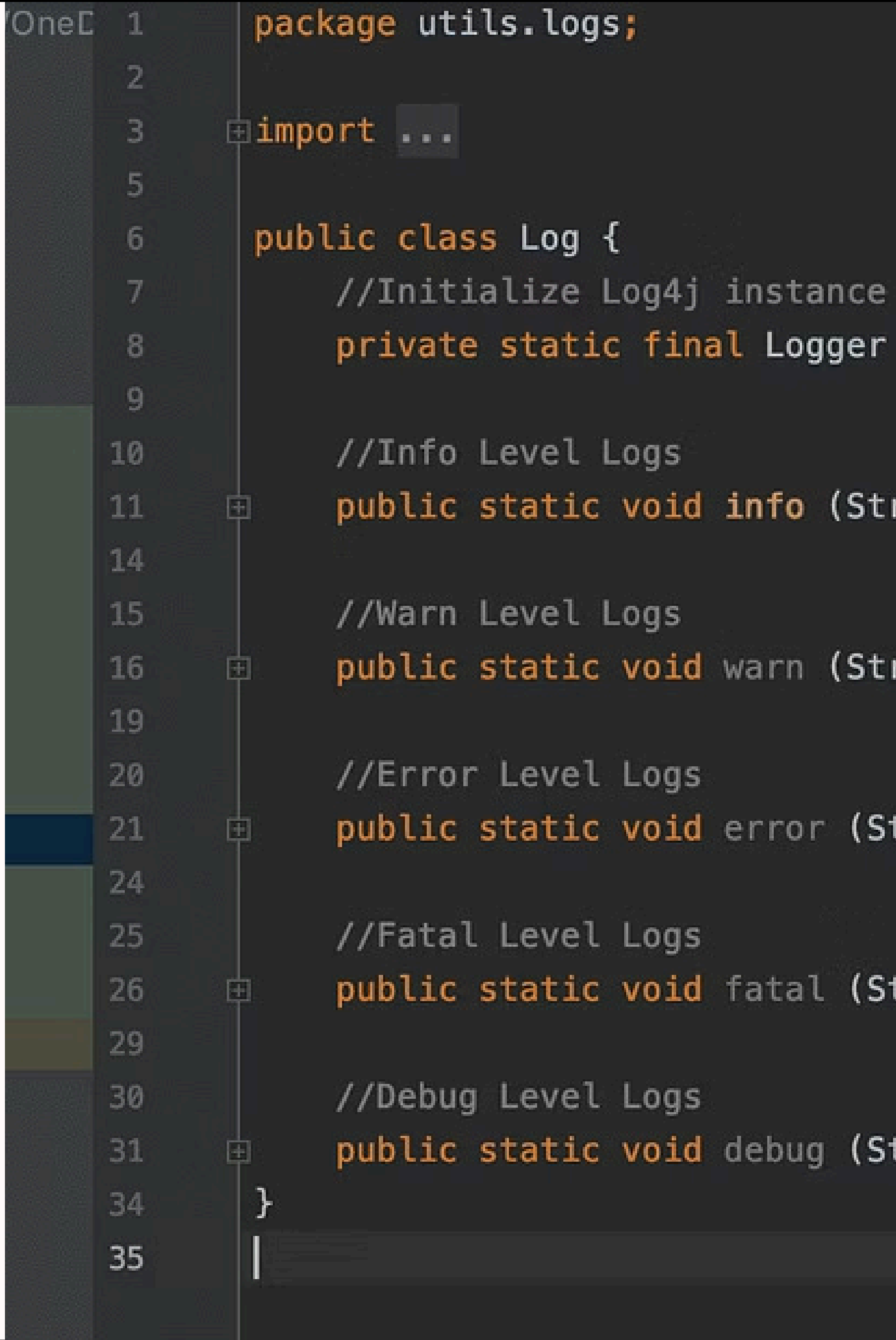
Update log4j.properties to set appropriate log levels and use structured formats like JSON.

Deploy Log Shipping

Use agents like Filebeat or OpenTelemetry Collector to stream logs to a central location.

Establish Retention Strategy

Define how long logs are kept and how they're indexed for efficient querying.



```
package utils.logs;

import ...

public class Log {
    //Initialize Log4j instance
    private static final Logger

    //Info Level Logs
    public static void info (Str

    //Warn Level Logs
    public static void warn (Str

    //Error Level Logs
    public static void error (St

    //Fatal Level Logs
    public static void fatal (St

    //Debug Level Logs
    public static void debug (St
}
```

Kafka Logs: Implementation Example

Kafka Log Files

Structured logs from brokers

2 OpenTelemetry Collector

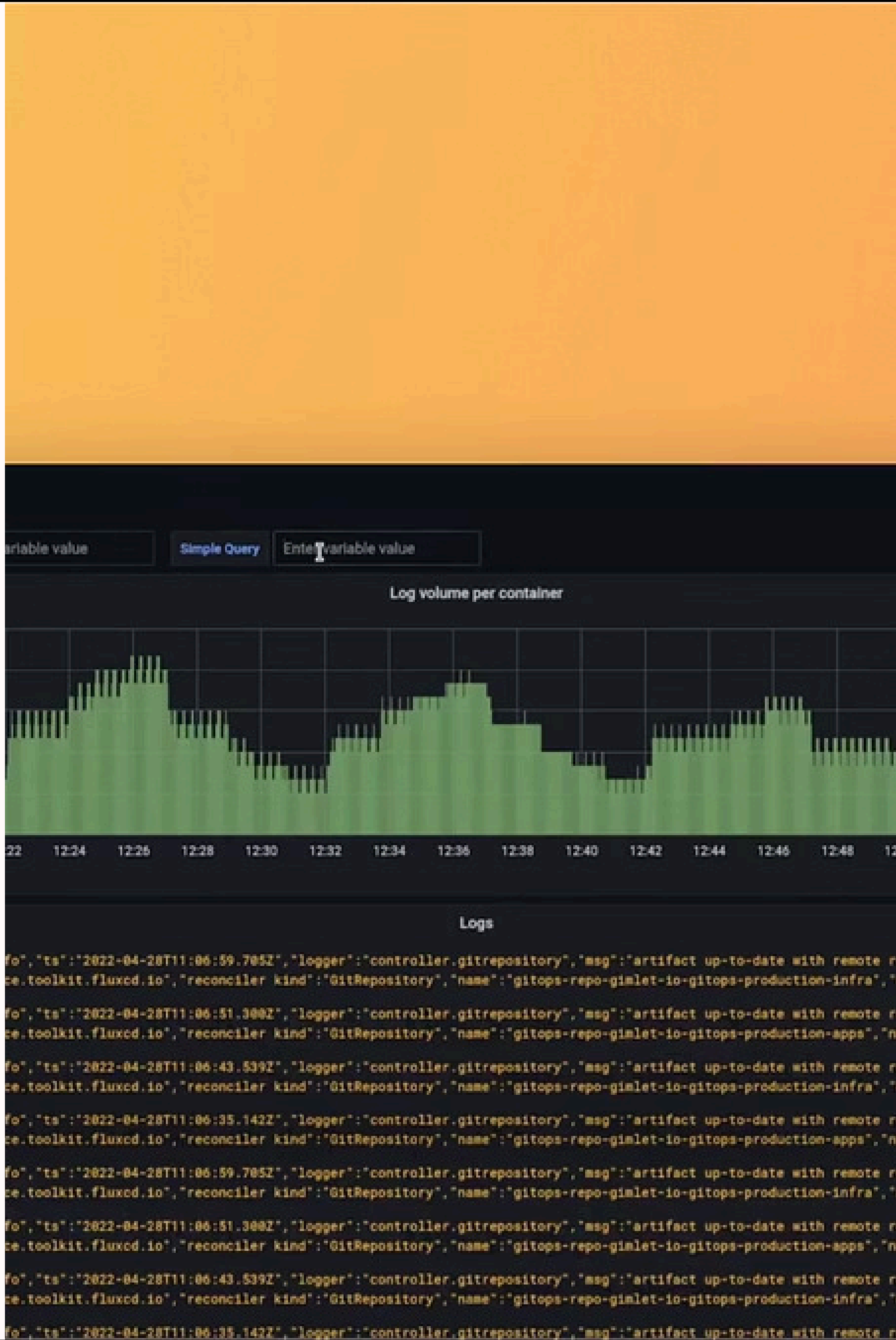
Tails files and adds metadata

3 Grafana Loki

Stores and indexes logs

Grafana UI

Search and filter logs



```
fo","ts":"2022-04-28T11:06:59.705Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-infra","

fo","ts":"2022-04-28T11:06:51.300Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-apps","n

fo","ts":"2022-04-28T11:06:43.539Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-infra","

fo","ts":"2022-04-28T11:06:35.142Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-apps","n

fo","ts":"2022-04-28T11:06:59.705Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-infra","

fo","ts":"2022-04-28T11:06:51.300Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-apps","n

fo","ts":"2022-04-28T11:06:43.539Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
ce.toolkit.fluxcd.io","reconciler kind":"GitRepository","name":"gitops-repo-gimlet-io-gitops-production-infra","

fo","ts":"2022-04-28T11:06:35.142Z","logger":"controller.gitrepository","msg":"artifact up-to-date with remote r
```

Kafka Traces: Observability Goals

End-to-End Visibility

Follow messages through the entire system from producer to consumer.

Latency Measurement

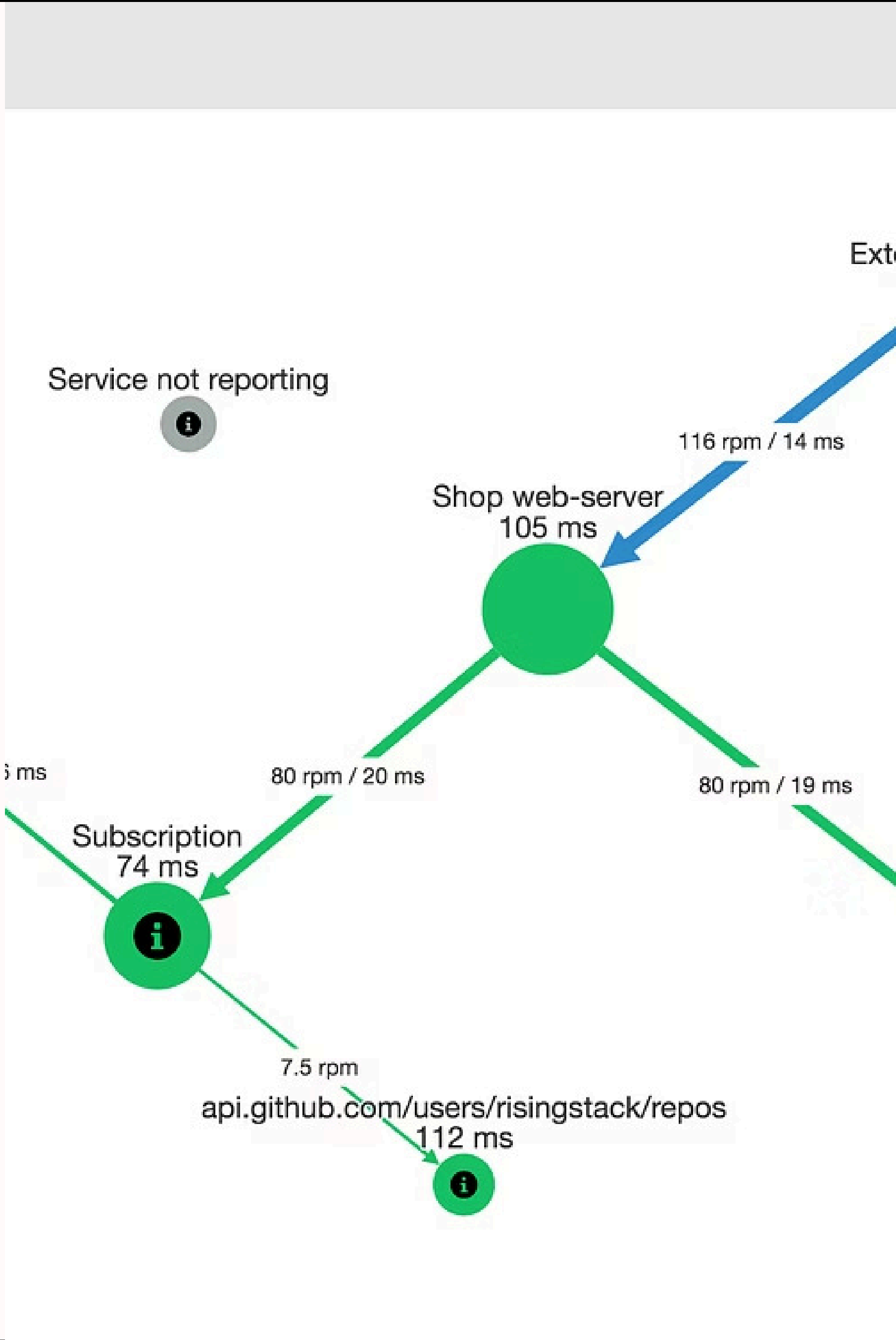
See how long each step takes in the message journey.

Bottleneck Identification

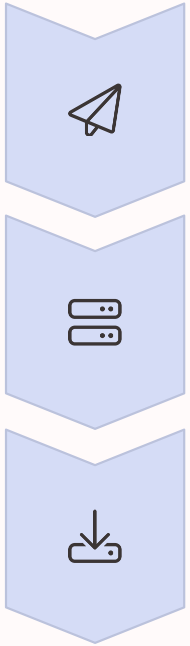
Pinpoint where delays occur in the processing pipeline.

Failure Localization

Determine exactly where messages get lost or errors happen.



Kafka Traces: Message Journey



Producer

Application sends message

Kafka Broker

Message queued and stored

Consumer

Message received and processed

Kafka Traces: Instrumentation

Trace Context Propagation

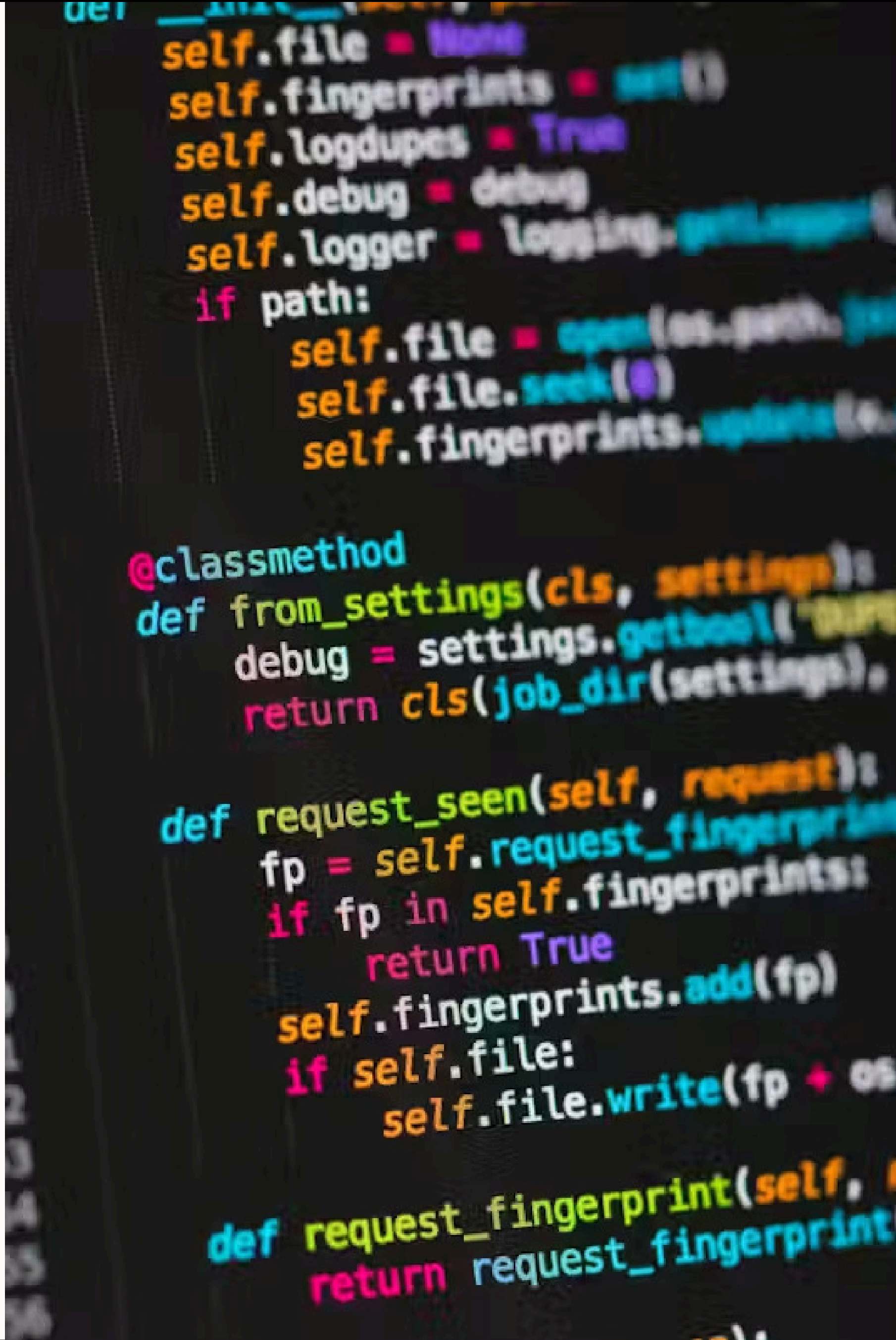
Producers attach trace identifiers to message headers for consumers to continue the trace.

Span Creation

Create spans for "send" and "receive/process" operations to model the message transfer.

Auto-Instrumentation

Use OpenTelemetry Agents to help generate spans.



```
def __init__(self, p
    self.file = None
    self.fingerprints = set()
    self.logdupes = True
    self.debug = debug
    self.logger = logging.getLogger(
    if path:
        self.file = open(os.path.join(
        self.file.seek(0)
        self.fingerprints.update(x.

@classmethod
def from_settings(cls, settings):
    debug = settings.getbool('DUPE
    return cls(job_dir(settings),

def request_seen(self, request):
    fp = self.request_fingerprint
    if fp in self.fingerprints:
        return True
    self.fingerprints.add(fp)
    if self.file:
        self.file.write(fp + os

def request_fingerprint(self,
    return request_fingerprint
```

Kafka Traces: Implementation Example

Instrumented Clients

Kafka producers and consumers with OpenTelemetry

OpenTelemetry Collector

Receives spans via OTLP protocol

Jaeger, ZipKin or similar

Stores and processes trace data

Grafana

Visualizes traces with span details

Trace Example: Message Processing

Producer Send

5ms - Message published to Kafka

Kafka Queue Time

50ms - Message waiting in broker

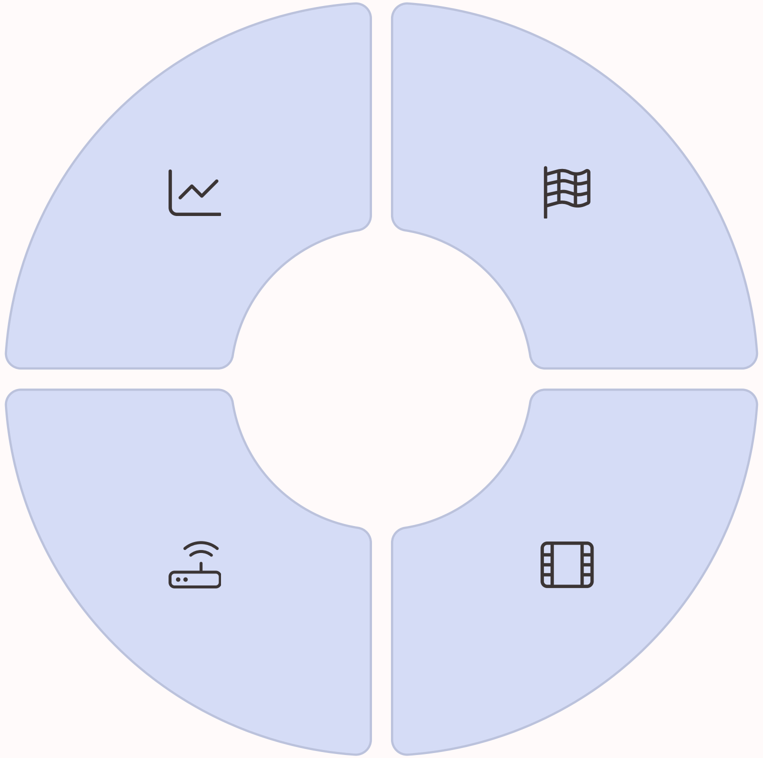
Consumer Receive

10ms - Message pulled by consumer

Processing

200ms - Business logic execution

Bringing MELT Together



Metrics

Show symptoms like rising latencies or dropping throughput

Events

Provide context for changes (e.g., "broker 2 restarted")

Traces

Follow data journeys to identify slow components

Logs

Offer detailed error messages and stack traces

Practical Implementation

OpenTelemetry

Unified way to instrument metrics, logs, and traces in your Kafka ecosystem.

- Vendor-neutral format
- Single agent for all signals
- Consistent instrumentation

Prometheus

Battle-tested metrics collection and alerting for Kafka's JMX metrics.

- Powerful query language
- Robust alerting
- Time-series database

Grafana

Visualization platform that ties everything together in one interface.

- Unified dashboards
- Cross-signal correlation
- Alert management

Benefits of MELT for Kafka

60%

Faster Detection

Reduction in time to detect issues

75%

Quicker Analysis

Reduction in root-cause analysis time

40%

Fewer Incidents

Reduction in production incidents

90%

More Confidence

Engineers report higher confidence in production systems

Key Takeaways

Synergy of Signals

The combination of MELT is far more powerful than any pillar alone.

Open-Source Tooling

Leverage existing tools like OpenTelemetry, Prometheus, and Grafana.

Complete Visibility

Gain deep insight into Kafka's behavior at runtime.

Production Confidence

Run Kafka at scale with greater reliability and performance.

Helpful Resources

Be sure to check out this great pre-built observability stack for Kafka:

- **https://github.com/confluentinc/jmx-monitoring-stacks**

Broker Metrics

Description	Metric
Controller Event queue time	kafka.controller:type=ControllerEventManager,name=EventQueueTimeMs
Byte in rate from clients	kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec,topic=([-.\w]+)
Byte in rate from other brokers	kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesInPerSec,topic=([-.\w]+)
Requests Error rate	kafka.network:type=RequestMetrics,name=ErrorsPerSec,request=([-.\w]+),error=([-.\w]+)
Log flush rate and time	kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs
Leader election rate	kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs
Is controller active on broker	kafka.controller:type=KafkaController,name=ActiveControllerCount
Num of under replicated partitions (|ISR| < |all replicas|)	kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions
Num of under minIsr partitions (|ISR| < min.insync.replicas)	kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount
Partition counts	kafka.server:type=ReplicaManager,name=PartitionCount
The average fraction of time the network processors are idle	kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent
Number of reassigning partitions	kafka.server:type=ReplicaManager,name=ReassigningPartitions
Size of a partition on disk (in bytes)	kafka.log:type=Log,name=Size,topic=([-.\w]+),partition=([0-9]+)

Common Client Metrics

Description	Metric
Total new connections established in the window.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,name=connection-creation-rate,client-id=([-.\w]+)
The average number of network operations (reads or writes) on all connections per second.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,name=network-io-rate,client-id=([-.\w]+)
The average number of outgoing bytes sent per second to all servers.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,name=outgoing-byte-rate,client-id=([-.\w]+)
Bytes/second read off all sockets.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,name=incoming-byte-rate,client-id=([-.\w]+)
The fraction of time the I/O thread spent waiting.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,name=io-wait-ratio,client-id=([-.\w]+)

Producer Metrics

Description	Metric
The total amount of buffer memory that is not being used.	kafka.producer:type=producer-metrics,name=buffer-available-bytes,client-id=([-.\w]+)
The fraction of time an appender waits for space allocation.	kafka.producer:type=producer-metrics,name=bufferpool-wait-time,client-id=([-.\w]+)
The average number of bytes sent per partition per-request.	kafka.producer:type=producer-metrics,name=batch-size-avg,client-id=([-.\w]+)
The average compression rate of record batches, defined as the average ratio of the compressed batch size over the uncompressed size.	kafka.producer:type=producer-metrics,name=compression-rate-avg,client-id=([-.\w]+)
The average time in ms a request was throttled by a broker	kafka.producer:type=producer-metrics,name=produce-throttle-time-avg,client-id=([-.\w]+)
The average time in ms record batches spent in the send buffer.	kafka.producer:type=producer-metrics,name=record-queue-time-avg,client-id=([-.\w]+)
The average number of records sent per second.	kafka.producer:type=producer-metrics,name=record-send-rate,client-id=([-.\w]+)

Consumer Metrics

Description	Metric
The average delay between invocations of poll().	kafka.consumer:type=consumer-metrics,name=time-between-poll-avg,client-id=([-.\w]+)
The average fraction of time the consumer's poll() is idle as opposed to waiting for the user code to process records.	kafka.consumer:type=consumer-metrics,name=poll-idle-ratio-avg,client-id=([-.\w]+)
The number of commit calls per second	kafka.consumer:type=consumer-coordinator-metrics,name=commit-rate,client-id=([-.\w]+)
The number of partitions currently assigned to this consumer	kafka.consumer:type=consumer-coordinator-metrics,name=assigned-partitions,client-id=([-.\w]+)
The average time taken for a group rejoin	kafka.consumer:type=consumer-coordinator-metrics,name=join-time-avg,client-id=([-.\w]+)
The number of group joins per second	kafka.consumer:type=consumer-coordinator-metrics,name=join-rate,client-id=([-.\w]+)
The average time taken for a group rebalance	kafka.consumer:type=consumer-coordinator-metrics,name=rebalance-latency-avg,client-id=([-.\w]+)
The number of group rebalance participated per hour	kafka.consumer:type=consumer-coordinator-metrics,name=rebalance-rate-per-hour,client-id=([-.\w]+)
The number of failed group rebalance event per hour	kafka.consumer:type=consumer-coordinator-metrics,name=failed-rebalance-rate-per-hour,client-id=([-.\w]+)
The number of seconds since the last rebalance event	kafka.consumer:type=consumer-coordinator-metrics,name=last-rebalance-seconds-ago,client-id=([-.\w]+)
The average number of bytes consumed per second	kafka.consumer:type=consumer-fetch-manager-metrics,name=bytes-consumed-rate,client-id=([-.\w]+)
The average number of bytes fetched per request	kafka.consumer:type=consumer-fetch-manager-metrics,name=fetch-size-avg,client-id=([-.\w]+)
The average lag of the partition	kafka.consumer:type=consumer-fetch-manager-metrics,name=records-lag-avg,partition=([-.\w]+),topic=([-.\w]+),client-id=([-.\w]+)

Thank you!