Finding and Fixing What's Hurting Your Observability

Juraci Paixão Kröhling

Software Engineer, CEO

ollygarden.com

OllyGarden

- Founder @ OllyGarden
- OTel Governance Committee
- Telemetry Drops (YouTube, LinkedIn)
- CNCF Ambassador (OTel Night)
- Emeritus: Jaeger, Collector, Operator, ...

I want to make your telemetry pipeline **EFFICIENT.**

OllyGarden

Agenda

01 The holy trinity telemetry

02 Bad telemetry in production

03 Fixing it

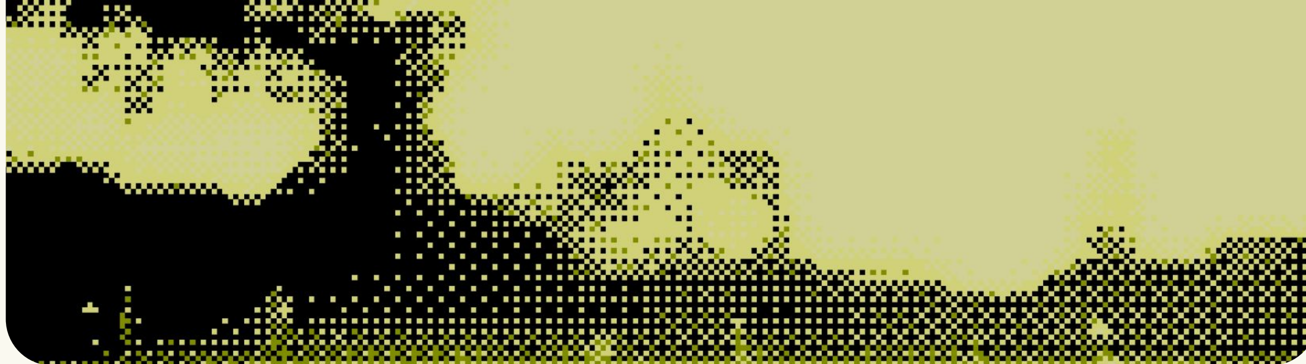
04 Takeaways

05 Q&A

01.

You don’t have too much telemetry

Sampling garbage gives you a *smaller* pile of garbage.



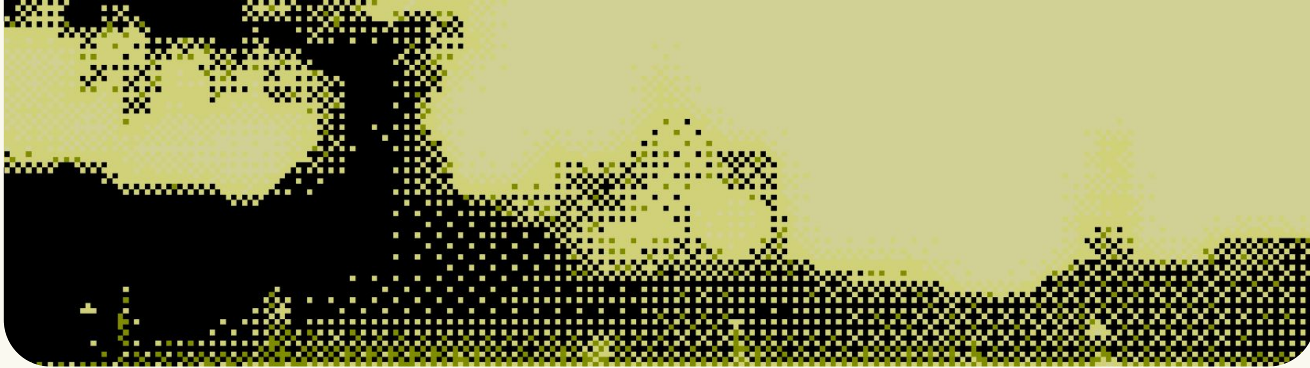
OllyGarden

Can you answer these 3 questions?

What are we collecting?

Who owns it?

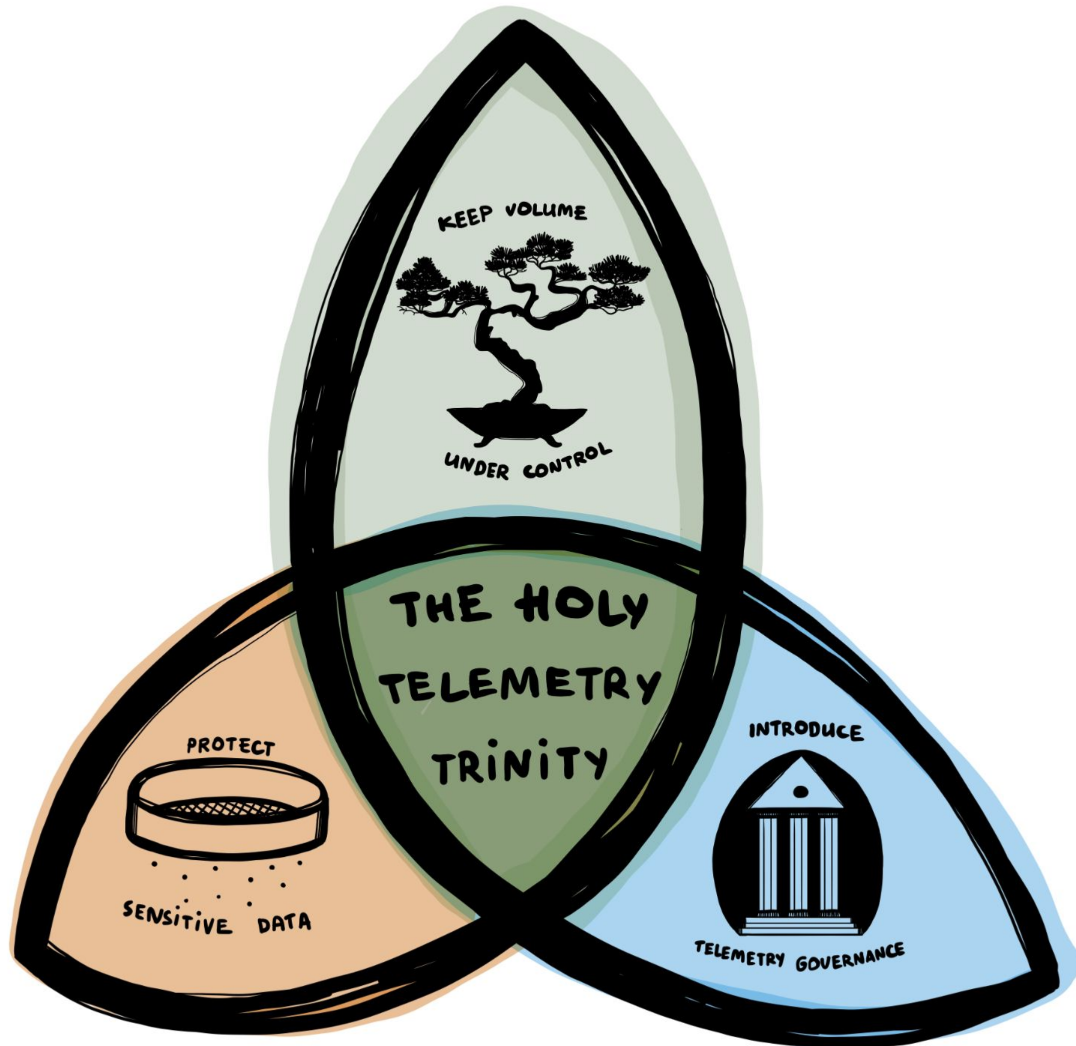
Is it valuable?



OllyGarden

The holy telemetry trinity

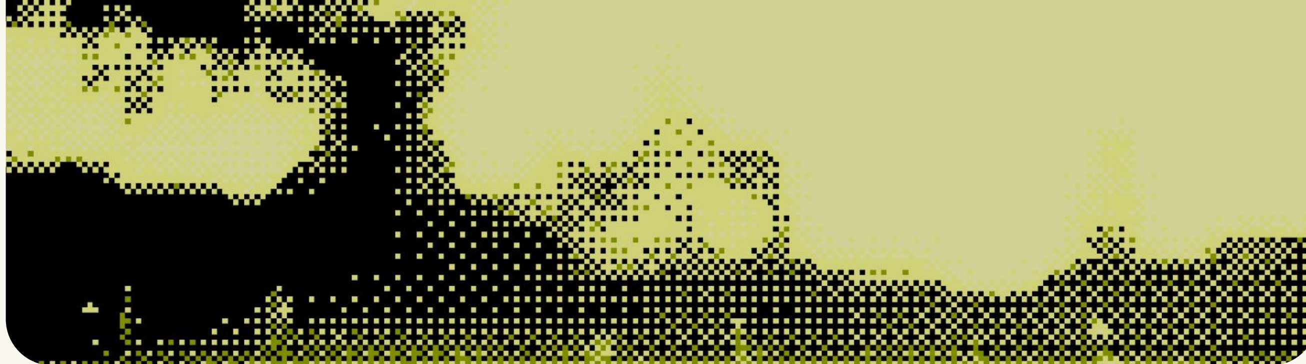
OllyGarden



Source:

youtu.be/cpB5NTtUdwQ

Elena Kovalenko



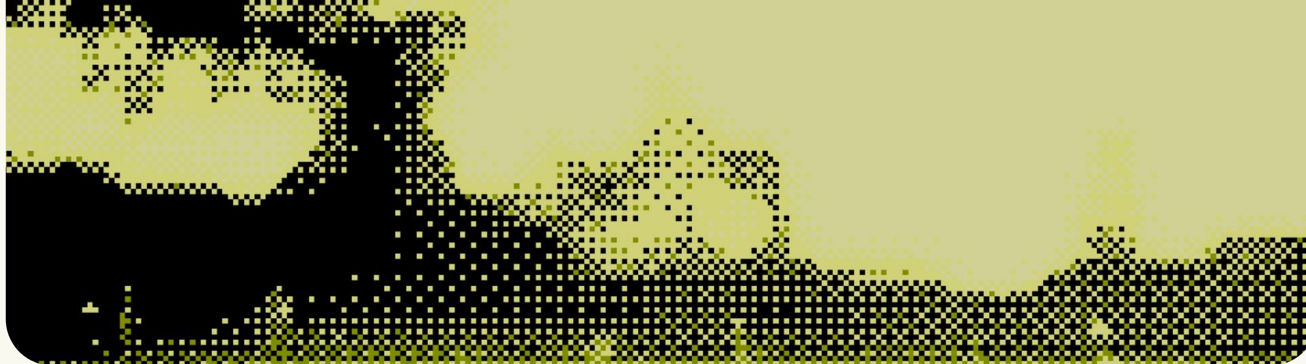
OllyGarden

The holy telemetry trinity

Volume: excessive spans, logs, and metrics that provide no debugging value.

Governance: inconsistent naming, missing conventions, unused attributes.

Sensitive data: PII landing in observability backends where it does not belong.



OllyGarden

How did we get here???

Service mesh instrumentation

Zero code instrumentation (eBPF, Java Agent, ...)

Learning curve: observability *is* hard to get right

02.

What bad telemetry looks like in production

Patterns from analysis of billions of telemetry points.

OllyGarden

Resource attribute bloat

- Every tool that writes a pod annotation or label creates a new resource attribute on every telemetry record
- Resource attributes multiply: they appear on every span, log, and metric from that resource
- The difference between 85 and 236 keys was not app complexity; it was tool sprawl

85
resource attr keys (minimal tooling)

236
resource attr keys (with tool sprawl)

1GB
from process.command_args alone in 5 min

OllyGarden

Resource attribute bloat

Attribute Size Analysis

Top Resource Attributes by Size

Attribute	Avg Chars	Max Chars	Occurrences
k8s.pod.annotation.cnpg.io/podSpec	2,048	2,048	30,985
process.command_args	710	1,129	56,107
k8s.pod.annotation.ad.VENDOR.com/postgres.checks	376	389	28,808
k8s.pod.annotation.ad.VENDOR.com/rabbitmq.checks	364	364	35,376
k8s.pod.annotation.ad.VENDOR.com/proxy.checks	307	307	11,058
k8s.pod.annotation.ad.VENDOR.com/elasticsearch.checks	209	304	19,492
k8s.pod.annotation.ad.VENDOR.com/rabbitmq.logs	207	207	35,376
k8s.pod.annotation.ad.VENDOR.com/keda-admission-webhooks.checks	147	147	896
k8s.pod.annotation.ad.VENDOR.com/keda-operator.checks	147	147	2,288
k8s.pod.annotation.ad.VENDOR.com/keda-operator-metrics-apiserver.checks	147	147	936

OllyGarden

Health check floods

- Kubernetes probes, load balancer checks, and monitoring systems generate millions of traces daily
- Each one: GET /health, status 200, sub-millisecond
- Nobody has ever used these to debug an incident

99%
droppable with tail sampling

1%
kept for monitoring

0
incidents debugged using health check traces

OllyGarden

Health check floods

Service	Span Name	Kind	Namespace	Occurrences
service-trigger	PING	CLIENT	prod	2,707
service-trigger	PING	CLIENT	test	2,576
service-controller	PING	CLIENT	test	2,023
service-controller	PING	CLIENT	prod	993
service-controller	SENTINEL	CLIENT	test	756
service-trigger	EVAL	CLIENT	test	734
service-controller	SENTINEL	CLIENT	prod	706
service-trigger	GET	SERVER	prod	116
service-trigger	GET /health/readiness	SERVER	prod	110

OllyGarden

Not only that...

#	Trace ID	Span Count	Service	Root Span Name	Cluster
1	3c9d9989cc3ba8562e6913fab4119e14	11,915	(unknown)	(unknown)	(unknown)
2	309c725a2ebf95d1de622d8009d0a7f2	5,442	(unknown)	(unknown)	(unknown)
3	514a221d542a0b5781ccc3b05195e83a	2,568	(unknown)	(unknown)	(unknown)
4	0000000000000000c75b1ce5ac7ceb79	2,244	scheduler	scheduler.UxVpGHGd2UKqMWrqWY62Nj	cluster-01-stage

Oversized span attributes

Attribute	Avg chars
mcp.response.value	49,648
mcp.request.argument	5,548
exception.stacktrace	3,889
db.statement	326
http.url	158

50KB
per span for MCP responses

152MB
from db.statement in 5 min

~1KB
what a normal span should be

OllyGarden

Metrics and cardinality

High cardinality in traces is essential for debugging. In metrics, it is destructive. 5 methods x 10 status codes x 3 regions = 150 time series. Add user.id with 500K users and you get 75 million time series.

150
time series (bounded)

75M
time series (+ user.id)

$18K
cost increase

OllyGarden

Are these being used?

Metric	Service	Datapoints	Distinct Values	Value Diversity %	Usefulness
kube_pod_status_phase	kube-state-metrics	60,870	2	0.0%	Likely Useless - only 0/1 toggle, extremely high volume
kube_pod_container_resource_requests	kube-state-metrics	36,568	169	0.5%	Borderline - resource config rarely changes, but useful for capacity planning
node_filesystem_device_error	prometheus-node-exporter	26,184	2	0.0%	Likely Useless - binary 0/1, high volume

OllyGarden

Are these being used?

Metric	Service	Namespace	Datapoints	Distinct Values	Value Diversity %	Usefulness
k8s.container.status.reason	unknown	logging	69,498	2	0.0%	Likely Useless - binary status across high cardinality
k8s.container.status.reason	unknown	system	40,824	2	0.0%	Likely Useless - same pattern
k8s.container.status.reason	unknown	cert-manager	40,338	1	0.0%	Likely Useless - constant value
k8s.container.status.reason	unknown	test	27,945	1	0.0%	Likely Useless - constant value

Internal span proliferation

- Auto-instrumentation captures every method call within a service
- Framework internals like Temporal persistence, Netty dispatchers, and hierarchy polling produce orphaned spans with no parent and no debugging value
- Instrument at boundaries: HTTP, database, queue, external API. Skip internal calls under 1ms.

13,001
spans in one trace

70,236
orphaned Temporal spans

39%
of all spans were droppable

OllyGarden

Not only that...

#	Service	Span Name	Kind	Cluster	Namespace	Occurrences
1	io.temporal.history	persistence.ExecutionStore/GetHistoryTasks	INTERNAL	cluster-02-stage	--	6,873
2	io.temporal.history	persistence.ExecutionStore/GetHistoryTasks	INTERNAL	cluster-03-stage	--	4,561
3	io.temporal.history	persistence.ShardStore/GetOrCreateShard	INTERNAL	cluster-04-stage	--	2,096
4	EventStore	NettyDispatcher.channelRead	INTERNAL	cluster-05-stage	--	1,343
5	EventStore	NettyDispatcher.channelRead	INTERNAL	cluster-06-stage	--	872

OllyGarden

How about logs?

Service	Severity	Body Preview	Occurrences	Usefulness
mongodb	INFO	client metadata	2,338	Likely Useless - connection metadata logged on every connection
unknown	Warning	Unknown cluster, will retry in 30 seconds	1,569	Actionable - persistent config issue, but highly repetitive
mongodb	INFO	Received first command on ingress connection	613	Likely Useless - routine connection event
unknown	DEBUG	Collecting data	397	Likely Useless - generic debug noise

OllyGarden

How about logs?

Metric	Value
Total logs	17,589
Unique messages	5,256
Duplicate logs	15,601 (88.7%)
Reduction potential	**70.1% (12,333 logs saved)**

OllyGarden

01.

Fixing it

Fix at source, not at pipe.

OllyGarden

Configure k8sattributesprocessor with an allow-list

```yaml
processors:
  k8sattributes:
    extract:
      metadata:
        - k8s.namespace.name
        - k8s.deployment.name
        - k8s.statefulset.name
        - k8s.daemonset.name
        - k8s.cronjob.name
        - k8s.job.name
        - k8s.pod.name
        - k8s.node.name
        - k8s.container.name
      labels:
        - tag_name: app
          key: app.kubernetes.io/name
          from: pod
```

OllyGarden

Transform processor for attribute cleanup

```yaml
processors:
  transform:
    error_mode: ignore
    trace_statements:
      - context: resource
        statements:
          # Remove JSON blob annotations from operators
          - delete_key(attributes, "k8s.pod.annotation.cnpg.io/podSpec")
          - delete_key(attributes, "k8s.pod.annotation.scaleops.sh/last-applied-resources")
          - delete_key(attributes, "k8s.pod.annotation.komodor.com/original-resources")
```

OllyGarden

Drop health check traces with tail sampling

```yaml
processors:
  tail_sampling:
    decision_wait: 2s
    num_traces: 1_000_000
    policies:
      - name: drop-99-percent-health-checks
        type: drop
        drop:
          drop_sub_policy:
            - name: health-url-path
              type: string_attribute
              string_attribute:
                key: url.path
                values:
                  - "^/health.*"
                  - "^/.*/health.*"
                  - "^/actuator/health.*"
                enabled_regex_matching: true
            - name: http-method-get
              type: string_attribute
              string_attribute:
                key: http.request.method
                values: ["GET"]
            - name: http-status-200
              type: numeric_attribute
              numeric_attribute:
                key: http.response.status_code
                min_value: 200
                max_value: 200
            - name: drop-99-percent
              type: probabilistic
              probabilistic:
                sampling_percentage: 99.0
      - name: sample-everything-else
        type: always_sample
```

OllyGarden

Drop orphaned and noisy span patterns with filter

```yaml
processors:
  filter/drop_orphans:
    error_mode: ignore
    traces:
      span:
        # Temporal persistence internals (orphaned, avg 1.8 spans/trace)
        - 'resource.attributes["service.name"] == "io.temporal.history"
          and IsMatch(name, "^persistence\\.(ExecutionStore|ShardStore)/.+")'

        # Netty low-level internals (avg 1.9 spans/trace)
        - 'resource.attributes["service.name"] == "ProtoEventStore"
          and name == "NettyDispatcher.channelRead"'

        # Synthetic smoke-test traffic
        - 'resource.attributes["service.name"] == "telemetrygen"'
```

OllyGarden

Log deduplication

```yaml
processors:
  logdedup:
    interval: 1s
    conditions:
      - severity_number >= SEVERITY_NUMBER_ERROR
      - attributes["log.type"] == "connection"
    exclude_fields:
      - attributes.request_id
      - attributes.timestamp
```

OllyGarden

Right signal for right data

```go
// Bad: adding user.id to a metric creates 500K time series
requestCounter.Add(ctx, 1, metric.WithAttributes(
    attribute.String("http.request.method", "GET"),
    attribute.String("user.id", userID), // creates a new time series per user
))

// Good: record user.id on the span, keep the metric clean
span := trace.SpanFromContext(ctx)
span.SetAttributes(attribute.String("user.id", userID))

requestCounter.Add(ctx, 1, metric.WithAttributes(
    attribute.String("http.request.method", "GET"),
    attribute.String("http.route", "/checkout"),
))
```

Truncate oversized span attributes

```yaml
processors:
  transform/truncate:
    error_mode: ignore
    trace_statements:
      - context: span
        statements:
          - truncate_all(attributes, 1024)
...
```

OllyGarden

OllyGarden

From pipeline controls to continuous governance

Pipeline controls are your immediate lever. Schema enforcement catches drift before it becomes expensive. Continuous review through AI maintains alignment across hundreds of services without competing with feature delivery.

Now
Collector processors (filter, transform, tail_sampling)

Medium
Schema enforcement with OTel Weaver

Long-term
Continuous review

AI changes the economics

Manual-quality instrumentation, at auto-instrumentation scale

Base model

Code understanding across Go, Java, Python, .NET. Off-the-shelf.

OTel expertise

Current API, SDK, semconv. Signal selection, naming conventions.

Your org's rules

Required attributes, SDK versions, naming standards, security policies.

Result

Review every PR. Detect PII at scale. Enforce naming across services. Audit telemetry freshness continuously

OllyGarden

Before you sample, clean up

- Inventory what you collect (top volume contributors)
- Categorize by type (health checks, internal spans, debug logs, high-cardinality metrics)
- Assess value: when did this last help resolve an incident?
- Fix at source where possible (agent config, log level policies, code review)
- Use the Collector as a safety net for what you cannot fix at source
- If volume remains a concern after cleanup, sample with intention

We know what we collect, and it is worth keeping.