O'REILLY®

Compliments of mezmo

REVISED EDITION

The Fundamentals of Telemetry Pipelines

And How They Are Used to Maintain Application Performance

Russ Miles
with Kai Alvason

REPORT

MEZMO.COM/OREILLY

UNDERSTAND, OPTIMIZE, AND RESPOND WITH MEZMO TELEMETRY PIPELINE

CONTROL TELEMETRY DATA WITH CONFIDENCE

GET STARTED FOR FREE

MEZMO.COM/OREILLY

mezmo

The Fundamentals of Telemetry Pipelines, Revised Edition

And How They Are Used to Maintain Application Performance

Russ Miles

with Kai Alvason

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

The Fundamentals of Telemetry Pipelines, Revised Edition
by Russ Miles with Kai Alvason

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisition Editor: John Devins
Development Editor: Virginia Wilson
Production Editor: Kristen Brown
Copyeditor: Miah Sandvik
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

September 2023:	First Edition
September 2024:	Revised Edition

Revision History for the Revised Edition

2024-09-18: First Release

978-1-098-17832-1

[LSI]

Table of Contents

Preface

This report is for DevOps, site reliability engineers, and security engineers struggling under a promising deluge of telemetry data. High-quality metrics, traces, and logs are all essential to observing and working with modern systems but, when they're applied to a modern, complex cloud-based system, the result can be a confusing mess.

Telemetry pipelines give you the tools to navigate this mess, providing the ways and means to extract so much more from your data. In this report, you'll learn what the promise of telemetry pipelines is and how it feels to be able to use telemetry pipeline tools to master your flood of telemetry data. From there, we dive into the key concepts you'll need when you build your own telemetry pipelines, and we look at some real-world examples. By the end of this report, you'll understand how investing in your own telemetry pipelines can help you go even further, implementing valuable business cases, such as compliance and cost management.

CHAPTER 1

The Need for Telemetry Pipelines

Data is the new oil.

—Clive Humby

While the concept of observability and using observability tools for gaining insight into telemetry data has been around for a while, using telemetry pipelines to preprocess data before sending it to observability tools or storage is very recent. In this chapter you'll learn some fundamental concepts for understanding what a telemetry pipeline is and how it can help add value to the information provided through your observability tool. Plus, you'll see an example of a pipeline designed to engineer data from three different sources into useful information for downstream tools.

Taming the Data Flood

Contemporary cloud systems and applications provide a torrent of data through their logs, but the sheer volume of the data is so overwhelming as to render it almost meaningless. The development of observability tools has provided a means to find nuggets of information within the data flood, but at the very high cost of taking in the entirety of the data to glean a few meaningful insights. Like a dam against a flood, an observability tool is a means to derive some value from log data, at the cost of having to construct a monolithic structure that, at best, keeps downstream systems from being overwhelmed.

Unlike a dam that can only hold back the flood, a telemetry pipeline can channel your telemetry data, optimizing and adding value to it along the way, so that what arrives at the end is already information-rich and actionable. Instead of being a potentially destructive force that has to be controlled, your telemetry data becomes a resource that you can carefully and intentionally refine to meet your observability and data storage needs. Like oil that starts as a geyser from the ground and is then sent through refining processes to become a variety of products, your telemetry data can be refined from crude output to useful information that can power your enterprise.

The Incremental Value Chain

At its most basic, a telemetry pipeline is a series of operations that transform data from a source through a step-by-step process before delivering it to a destination. Often this will be an observability tool or a storage solution, and in many cases the purpose of the telemetry pipeline is to make the source data more easily ingested by the tool. In this most basic form, a telemetry pipeline is little more than a means to reformat data from point to point. A more advanced conception of a telemetry pipeline understands it as a means to increase the incremental value of your data as it passes through the pipeline. For example, consider these steps in the lifecycle of data as it becomes information:

1. Raw data from a single source has minimal value and requires extensive manual intervention to yield even basic insights.
2. Sending raw data from multiple sources into a telemetry pipeline centralizes the data and enables automated processing. This significantly reduces the toil required by individual teams to begin extracting information from the data and decreases the time to realize value from that information.
3. As data enters the pipeline and is analyzed, it becomes possible to begin understanding the data by surfacing common patterns, identifying useful versus redundant data, and recommending ways to optimize it. The value of the raw data is significantly increased simply by being able to separate what is valuable from what is not, and then pass the valuable data further down the pipeline.

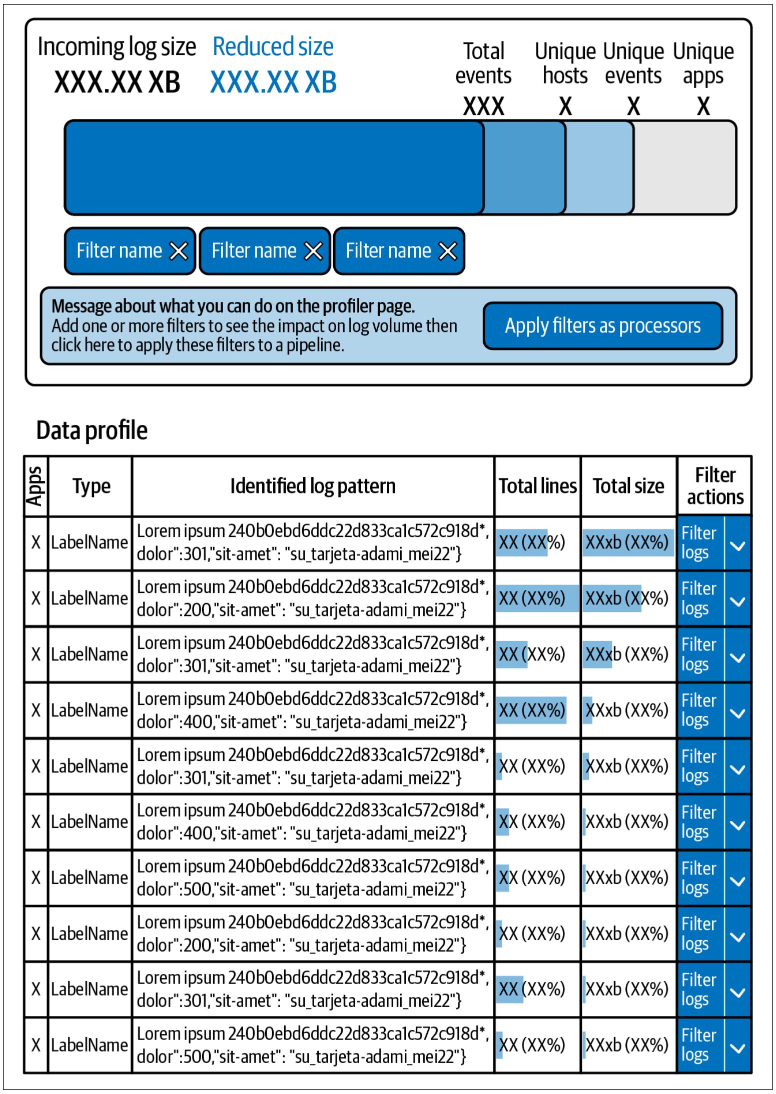
4. As it passes through the pipeline, the data is transformed and optimized by processor chains that are purpose-built to derive the most meaningful information for use by downstream teams.
5. At the end of the pipeline, the raw data becomes actionable information that can enable teams to rapidly respond to incidents, gain business insights, and operate in a more secure environment.

Understand, Optimize, Respond

All refining processes proceed in phases that begin with understanding the crude material, devising the processes that it will pass through during refinement, and then distributing and delivering it in the form that makes it most useful. For telemetry data, there are three phases for refining the raw data into useful information: *understand*, *optimize*, and *respond*. Each of these are reflected in the functional aspects of a telemetry pipeline.

An analytical tool can provide you with a basic understanding of your data (Figure 1-1). Understanding the data means being able to sort the potentially valuable data out from the general flow of events, metrics, and tracks that are generated by your sources. Functionally, this is usually handled by pipeline components like parsers, which can separate elements from the flow in the same way that a miner panning for gold in a stream is able to separate small nuggets from the gravel in his pan.

When used in conjunction with an analytics tool that can provide you with a profile of your data, you can identify redundant or irrelevant data that you may want to send to storage for a full-fidelity copy of the stream, but can route away from your observability tool saving both the cost and the toil of trying to process this information within your tool.



Message about what you can do on the profiler page.
Add one or more filters to see the impact on log volume then click here to apply these filters to a pipeline.

Apps	Type	Identified log pattern	Total lines	Total size	Filter actions
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":301,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":200,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":301,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":400,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":301,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":400,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":500,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":200,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":301,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs
X	LabelName	Lorem ipsum 240b0ebd6ddc22d833ca1c572c918d*, dolor":500,"sit-amet": "su_tarjeta-adami_mei22"}	XX (XX%)	XXxb (XX%)	Filter logs

Figure 1-1. An example of an analytics tool that presents a data profile of telemetry data as it enters a pipeline

Once you have an understanding of the data, you can then optimize it by sending it along processing chains to transform and engineer the data to meet your requirements. This is like taking crude oil and refining it into different products like gasoline, kerosene, and diesel fuel. Each of these end uses requires a different processing approach. Requirements like encrypting personally identifying information (PII), setting up metrics for observability tools, and sending different data components to different destinations also require the construction of specific processor chains. The later sections in this report go into detail about many of the typical processors used in a pipeline, and how to use them for purposes like controlling cost and assuring adherence to data compliance regulations.

The end goal of any telemetry pipeline is to provide downstream users with actionable information that enables them to respond to outages, incidents, and real-time business information. While observability tools can provide some of this functionality, it is largely after the fact, and there is often significant lag time between when an incident occurs, when the data is indexed by the tool, and when the tell-tale signals are transmitted. A telemetry pipeline, on the other hand, can have detection tools built into the processing chains that will not only send alerts, but can also change the functioning of the pipeline itself when an incident or outage is detected within the data. Rather than having to wait for an observability tool to catch up with a sudden surge in `500 - Internal Server Error` messages, for example, a responsive telemetry pipeline could send an immediate alert and begin to process those specific messages to aid in diagnosing the problem within the observability tool.

An Example Pipeline

In the most basic structural definition, a telemetry pipeline is built using data sources, processors, and downstream destinations. The design of the processing chains is determined by the content and format of the source data, as well as the format and information requirements of the tool and storage destinations. This seems simple enough, but the question naturally arises: how do I know which processors (or groups of processors) to use to achieve my data engineering requirements?

For example, imagine a situation in which your sources include transaction data in JSON format, Apache error messages as raw strings, and system events in JSON. In planning your telemetry pipeline architecture, there are several aspects of the source data that you need to consider:

- Sending the entire data stream to your storage and observability tools, when only some of the data is useful, will result in egregious costs.
- The financial data contains PII that must be encrypted before being stored or used in an observability tool.
- The raw strings of the Apache errors must be converted to JSON before they can be processed through the rest of the pipeline.
- Different components of each data type need to be routed to separate destinations.
- Events need to be converted to metrics to create visualizations and dashboards in observability and analytics tools.

With this understanding of your data, and the analysis of what is useful within it and what is not, you can develop the architecture and processing chains to optimize it for your requirements:

- Routing and dropping events that contain little useful data, like `Status - 200` responses and Apache `INFO` messages, can result in a significant decrease in the volume of data sent to downstream destinations. You can accomplish this by using a route processor that uses conditional statements to identify the information you want to drop and sends it to a drop destination. You can also set the processor to detect whenever there is a rise or drop in the expected number of messages beyond a set threshold, and to send an alert or trigger an incident response when the change is detected.
- The PII should be encrypted, or redacted entirely, before reaching its downstream destinations. You can accomplish this by using a redact processor to completely obfuscate the information, or an encrypt processor if you need access to the information later, for example if you need to be able to investigate fraudulent charges against a specific credit card number. In this case, you could send the encrypted information to a specific

storage location with limited access, and then use that storage location as the source for a decryption pipeline with an analytics tool as the destination.

- The Apache errors must be reformatted at the source so they can be processed through the pipeline. You can accomplish this by using a script processor to run a formatting script for each of the error messages before it is sent to the route processor.
- Large numbers of identical event messages have little informational value on their own, but when aggregated into metrics measured over time they can provide detailed insight into system health. You can accomplish this with an event-to-metrics processor that is configured to provide counts of specific metrics over specific time intervals. Again, in a responsive pipeline, this processor could also be set to send alerts and change the pipeline functionality when an in-stream change in the data is detected.

The final structure of a pipeline that is designed with these source and information requirements in mind would resemble the one shown in Figure 1-2.

Here is a list of the components shown in Figure 1-2:

1. Sources
2. Script execution processor to reformat Apache error messages
3. Route processor
4. Encrypt credit card number
5. Destinations representing a drop destination, a storage location, and an observability tool

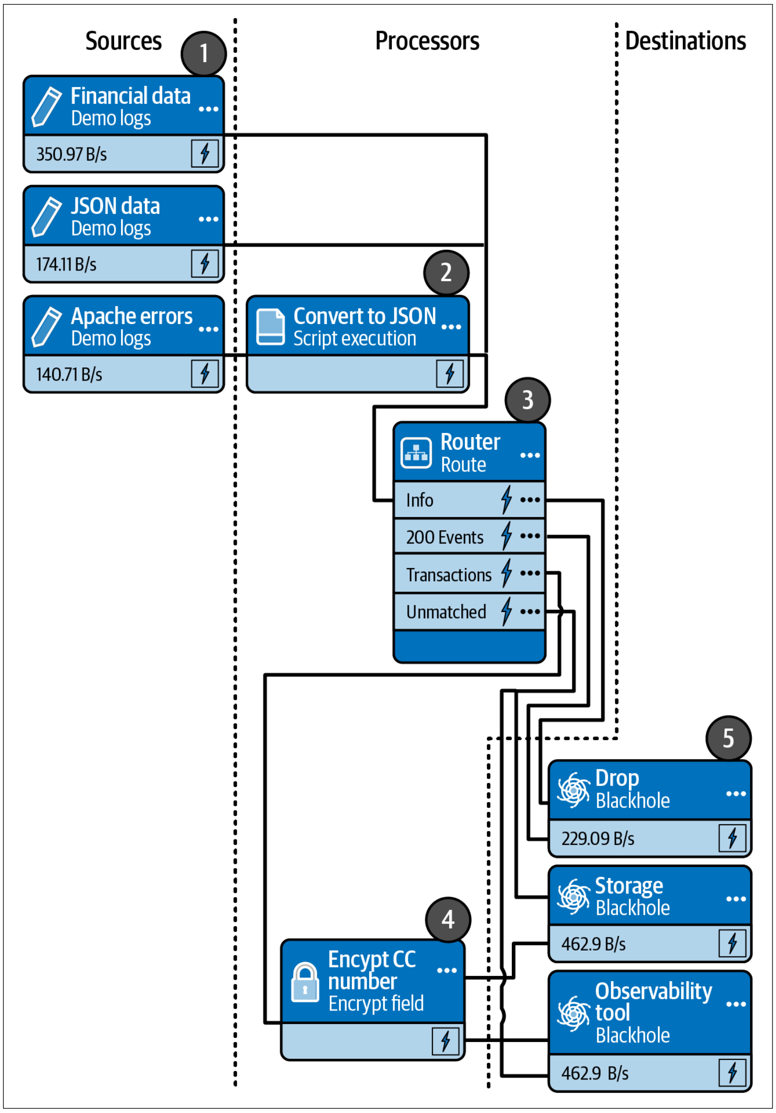


Figure 1-2. A schematic representation of a pipeline to meet basic data engineering requirements

In the remainder of this report you'll learn more about how to set up pipelines to meet specific use cases like cost reduction and data compliance, and about the most common processors used to meet these use cases.

CHAPTER 2

The Domain Language of Telemetry Pipelines

There is a tide in the affairs of men
Which, taken at the flood, leads on to fortune;
Omitted, all the voyage of their life
Is bound in shallows and in miseries.
On such a full sea are we now afloat.
And we must take the current when it serves,
Or lose our ventures.
—William Shakespeare, *Julius Caesar*

In Chapter 1, you saw how telemetry data has become a flood: a flood where the properties of your telemetry data need to be considered so that you can get the most value out of all the data that is available. In this chapter, you're going to explore the foundational concepts of telemetry pipelines so that you can begin to do exactly that.

The Building Blocks of Telemetry Pipelines

Telemetry pipelines are made up of five concepts: sources, streams, processors, destinations, and, of course, the pipeline itself. *Sources* are your faucets for the flood of telemetry data being emitted from your systems. Each source produces a *stream* of structured telemetry data. *Processors* are the workhorses that do what it takes to manipulate those streams of telemetry data you've tapped into.

Destinations are your channels to bring this refined telemetry data to downstream tools and other systems. Finally, the *pipeline* packages all these sources, streams, processors, and destinations into a unit you can work with and manage. Let's take a closer look.

Starting at the Source

Most data pipelines start by answering the question, "Where are we going to get our data from?" That is also true of telemetry pipelines. The difference is that your sources will be the various parts of your runtime systems emitting telemetry data: the logs, metrics, and events. The sources on your telemetry pipelines are your taps into the flood of telemetry data that is readily available. Depending on the telemetry pipeline tools you use, you may be presented with a very wide range of possible sources, as shown in Figure 2-1.

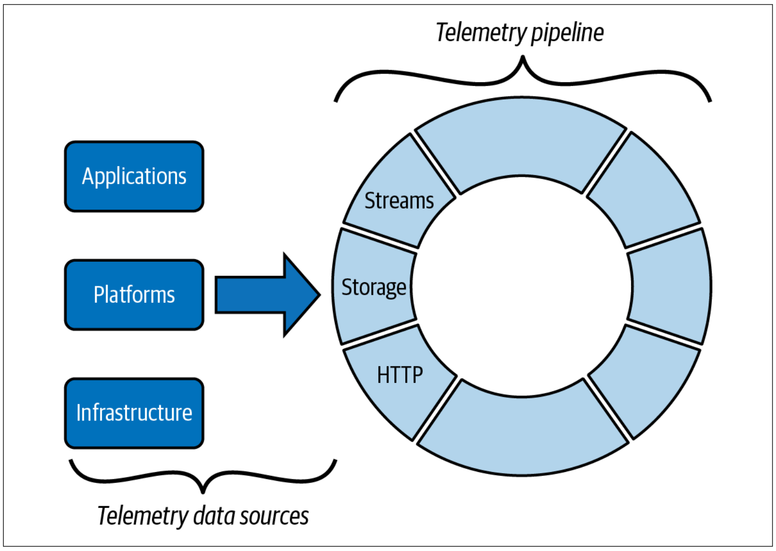


Figure 2-1. Telemetry pipeline sources

Sources can be extremely varied, including (but not limited to!) the following:

Applications
: Telemetry data emitted from your applications and services

Logs
: Recorded entries of events within the system

Metrics
: Snapshots or trends of important system characteristics, such as disk space, speed of response, and so on

Traces
: Information that combines events to give a picture of an important flow within the application or service, sometimes end to end

Events
: A discrete action happening at a moment in time; often recorded in log data

Infrastructure
: Telemetry data on the state of low-level infrastructure, including virtual machines, networks, gateways, and so on

Networks
: Traffic flow information like Netflow or sFlow and networking device metrics like Simple Network Management Protocol (SNMP) data

Platforms
: Telemetry data emitted from platforms like Kubernetes

Say you have a Kubernetes cluster that is already pushing its logging out to the Splunk HTTP Event Collector (HEC), but you're not yet doing anything useful with the other telemetry data, metrics, and system events in particular that Kubernetes makes available. Your first step is to bring those collections of data into your telemetry pipeline as new sources to make them candidates to be worked on before they can be surfaced as a richer, correlated picture.

What's the Destination?

Destinations can be very similar to sources. They are places that you want to push your conditioned telemetry data to.

A destination could manifest as a simple write to a storage location, such as Amazon S3, or could move your conditioned telemetry data into your favorite observability tool, as shown in Figure 2-2.

Your destinations provide the places to surface your newly conditioned telemetry data. You can add as many destinations as you need, from observability to Cloud Infrastructure Entitlement Management tools, to get your data where it is most useful.

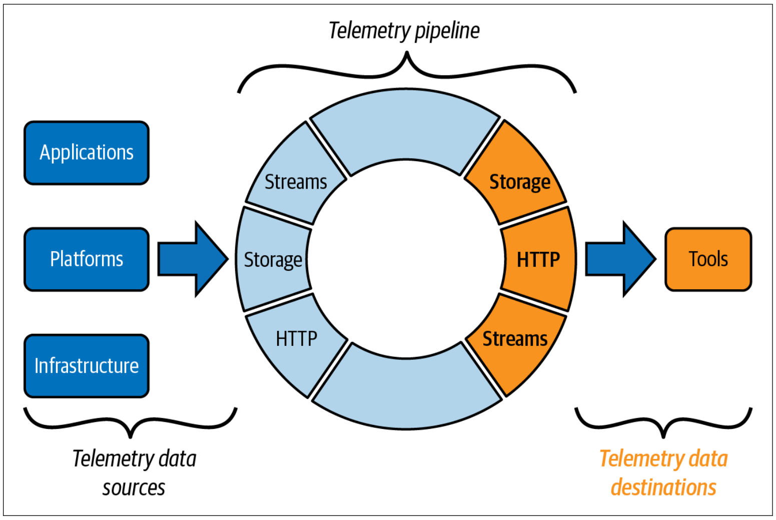


Figure 2-2. Telemetry pipeline destinations

Do Cross the Streams: From Sources to Destinations

A source and a destination establish a connection—a stream—between them. Sources parcel up the input data into structured events that can then flow to one or more destinations. Destinations take that stream of telemetry events and package them to surface them in a useful way, as shown in Figure 2-3.

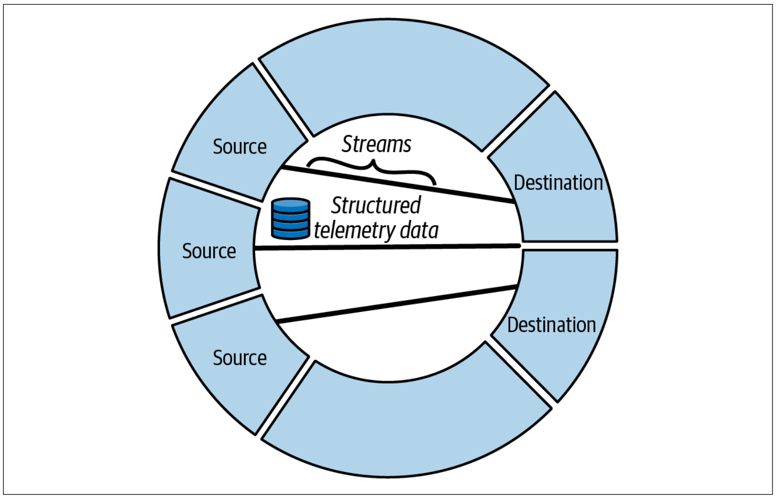


Figure 2-3. Streams of structured telemetry data

These streams are what you work with when harnessing the power of telemetry pipelines. Each stream is an opportunity to enrich, correlate, transform, and route structured telemetry events into new forms and to new destinations. Sources establish your streams, destinations take streams to useful external places, and the streams are what you focus and work on. The streams you establish open up a wealth of possibilities for the next telemetry pipeline concept: the processor.

Adding Processors

With a source and a destination, you have a channel through which telemetry data can flow but no additional work is being done. That work is the job of the telemetry pipeline *processor*. Processors connect to sources, to other processors, and, eventually, to destinations to perform functions like transforming and routing your telemetry data as it flows through your system, as shown in Figure 2-4.

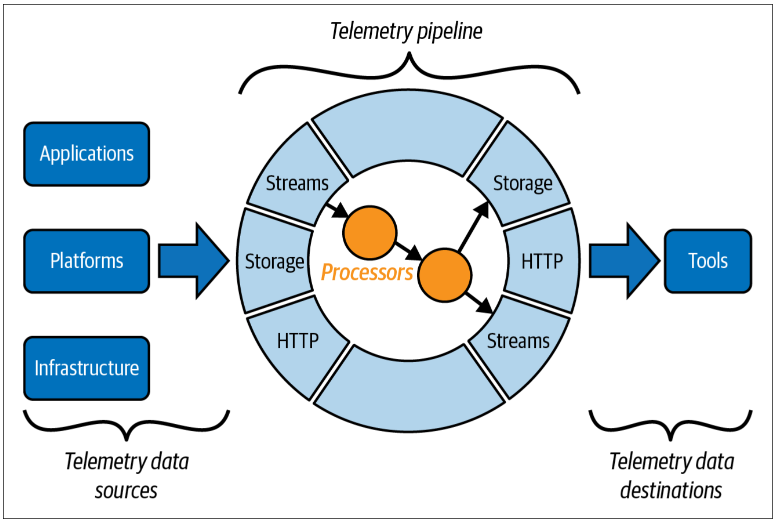


Figure 2-4. Telemetry pipeline processors connect to sources, other processors, and, eventually, destinations

The types of processors you'll see throughout this report include:

Route
: A route processor takes the content of an incoming event and, according to a set of rules, channels that event to usually more than one output stream. An example might be where personally identifiable information is detected in an event and is then routed to a compliant stream and destination. (See Chapter 5 for more on this case.)

Deduplicate
: A deduplicate (or *dedupe*, for short) processor looks for duplicate events and drops them, reducing the overall number of events in the stream, possibly for cost reasons.

Encrypt
: An encrypt processor will encrypt one or more fields within a telemetry event before passing it on. This is a great processor for when you want to hide personal or financial data from an audience at a particular destination.

Sample
: A sample processor will downsample the events in a stream, producing fewer events where specific events being dropped

doesn't significantly reduce the usefulness of the telemetry stream. This processor is frequently used for saving space and, therefore, saving money by reducing the data transmitted to and retained by a particular destination. (For more, see Chapter 4.)

Filter
: A filter processor can selectively drop or pass on an event based on its contents.

Reduce
: A reduce processor takes multiple log input events and combines them into a single log event based on specified criteria.

Parse
: A parse processor takes incoming data of a known format and converts it into a parsed set of values prior to subsequent processing.

Event to metric
: The event-to-metric processor provides an easy way to create a new metric event within the pipeline, typically from an existing log message. The new metric event can use data from the log to generate the metric, including the value if desired.

This is just a small sample[1] of the processors that you could potentially use for your telemetry pipelines.[2] The more powerful your processors, the more control you can exert over your telemetry data streams, and the more you can turn your flood into something valuable to your business.

Bringing It All Together

Enough with the concepts, let's look at a real example of all these telemetry pipeline building blocks in action. Let's take a situation where we have archived logs in an Amazon Web Services (AWS) S3 bucket that we want to bring into a telemetry pipeline so we can bring that data into Splunk for some further exploration and analysis. Maybe the auditors are knocking gently on the door to get

1 Please forgive the bad pun!

2 A good resource is the growing list of processors available in the Mezmo docs.

a peek at some historical records of how your systems performed or how they were changed.[3]

The first thing you do is bring that data into your telemetry pipeline by configuring an AWS S3 source. This can be done by configuring an Amazon Simple Queue Service (SQS) queue in your telemetry pipeline toolset. The AWS S3 bucket's data will be streamed into the Amazon SQS queue, and that can then be picked up by the AWS S3 source before being channeled to a destination, such as Splunk HEC, as shown in Figure 2-5.

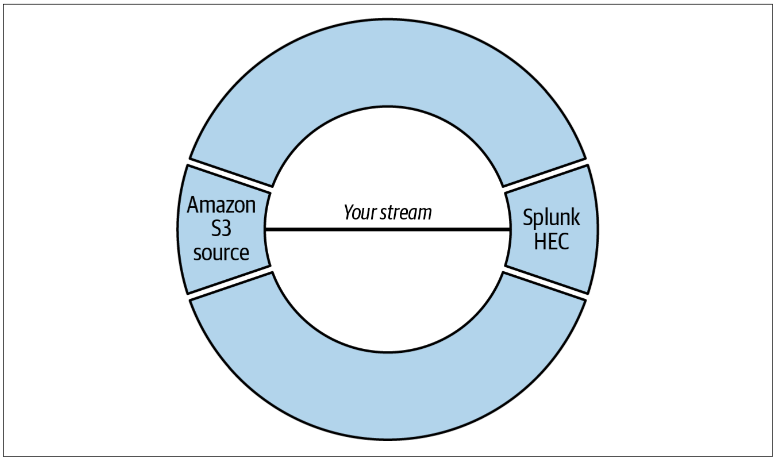


Figure 2-5. Establishing a stream to work with between your archived AWS S3 source (via SQS) and your Splunk HEC destination

The plan is to bring those logs into Splunk so that you can do some further analysis, but the cost could be high if all the data is brought in as is. Fortunately, the auditors are interested only in where interactions with your system failed at this point, so you can focus the data on just the 500 status entries in your logs. This is where the power of the pipeline comes in, as shown in Figure 2-6.

3 More on those types of scenarios in Chapter 5 when we discover how telemetry pipelines can help meet the needs of risk and compliance .

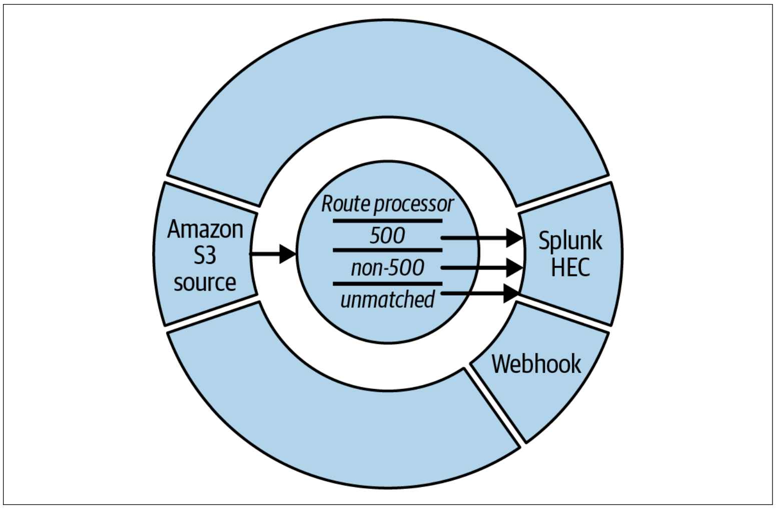


Figure 2-6. Adding a route processor to route only log messages that include a status of 500 to the Splunk HEC destination

First, you can configure a Splunk HEC destination, and then, to surface only what you need, you can add a route processor between the AWS S3 source and the Splunk HEC destination. You then route only the log traffic that contains a status of 500 to the Splunk HEC destination, trimming the data to only what you need for the auditors. However, when you look at the data that is flowing to your Splunk HEC destination, there are a *lot* of duplicate events. Those duplicates are making it harder for the auditors to make sense of what they are seeing in Splunk, and will cost you money to retain to boot.

The power of pipelines saves the day again, as shown in Figure 2-7.

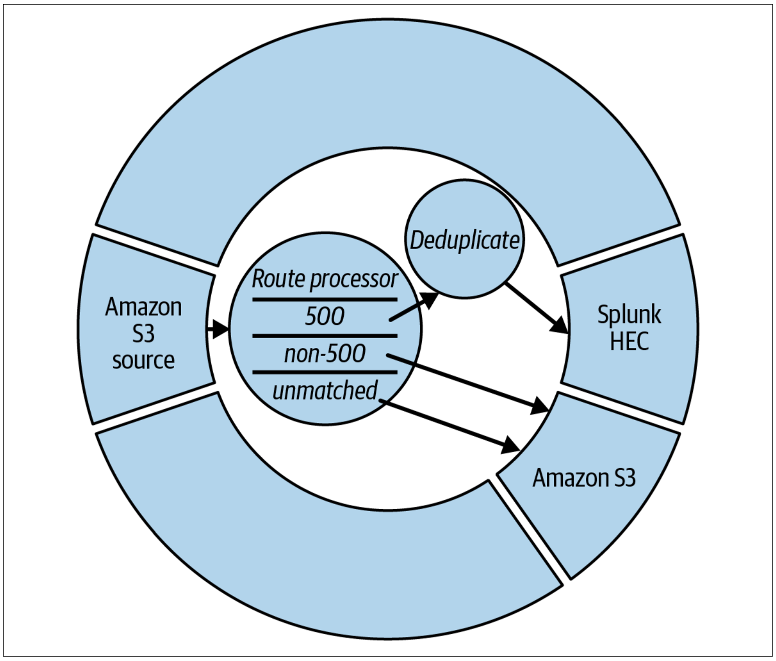


Figure 2-7. Deduplicating the stream of 500 status log events to save space, cost, and the patience of the auditors

A dedupe processor matches any events that meet a certain pattern within a specified number of sequential events, removing those events that match as duplicated. By adding a dedupe processor to your stream between your router and the Splunk HEC destination, you now surface in Splunk only the necessary deduplicated and conditioned 500 status events for the auditors to do their work. For good measure, you can add an Amazon S3 destination to hoover up, persist, and archive all the unmatched and non-500 status log messages, ready for the auditors to take away with them for further analysis if they decide they need to do so.[4]

4 And, after all, anyone who has worked in a regulated industry will tell you: happy auditors equal a happy life!

CHAPTER 3
Managing Your Telemetry Pipelines

Quis custodiet ipsos custodes?

("Who will guard the guards themselves?" or "Who watches the watchers?")

—Juvenal

Telemetry pipelines establish control of your flood of telemetry data. Sources bring your telemetry data into the pipeline, destinations define where your data should surface, and processors transform, enrich, and augment that data in all the ways you need to turn the flood into a useful trickle.

In this chapter, you're going to explore the different forms that your control can take, from the processors that are your building blocks to a real use case of pulling together a telemetry pipeline to condition and route a rich stream of data to your observability tools.

Managing and Debugging Your Pipelines

Developers, engineers, and other system builders are naturally optimists. We have to be; otherwise, we might be paralyzed by the ramifications of what we build if, or when, it goes wrong. But things *do* go wrong. Murphy's law is real. That's one of the reasons we want great telemetry underpinning powerful observability tools in the first place.

But Murphy's law extends to your telemetry pipelines as well. As you take full advantage of the multifarious sources at your disposal and grapple with them using the many processors in your toolkit, things can and will go wrong, and those slips can be costly. This is why you need similar techniques when working with your telemetry pipelines as you do when working with your systems: great debuggability through observability.

Getting a Grip: Ingress and Egress Data Volumes

Raw data in. Conditioned data out. That's the starting point for the metrics of interest when it comes to telemetry pipelines. Raw amounts of data by ingress and egress. Amount in and amount out, over time.

This blunt instrument provides a starting point for you to build confidence that your pipeline is working as expected, and confidence is exactly what metrics from your pipeline are trying to exude. Are you seeing an amount coming into your pipeline that feels comfortably expected? Is the amount going out unsurprising? Given the costs of data transfer and residency in various observability tools, that last question is often crucial. In your telemetry pipelines, your metrics of interest begin with comparing what comes in to what goes out at the pipeline level.

Exploring data volumes and metrics at the level of your pipelines is useful, but what about the broader picture across all your pipelines? That global view is where you are likely to see any worrying trends where your processors are perhaps not doing quite what you expected; for example, when you are using downsampling processors where you'd expect there to be less globally in egress than in ingress. Your telemetry pipelines can provide a view across all your telemetry data ingress and egress, from and to all your types of sources and destinations, to help you gain confidence that you have a grip on your data in and data out.

While data volumes may highlight a faux pas of sufficient magnitude, they can't help you debug what is actually going on in your streams as they are conditioned by your processors. For that you need a sharper tool: pipeline taps.

Taking a Peek: Debugging with Pipeline Taps

Figuring out what streams to manipulate and what those manipulations are producing through the many types of processors at your disposal is no mean feat.[1] This is where pipeline taps can help.

Pipeline taps are an implementation of the Wire Tap pattern, and they do exactly what they imply, which is intercept and observe messages on the fly so that you can get a glimpse of what is going right or wrong. They provide a window on the data flowing through your streams so you can simulate, inspect, and debug what your telemetry pipeline is doing to your telemetry data.

With a couple of taps, you can look at a sample of the data going in and coming out from one or more of your telemetry pipeline's processors. As shown in Figure 3-1, one example is to tap before and after a route processor to see if your routing logic is pushing the right traffic toward your destination.

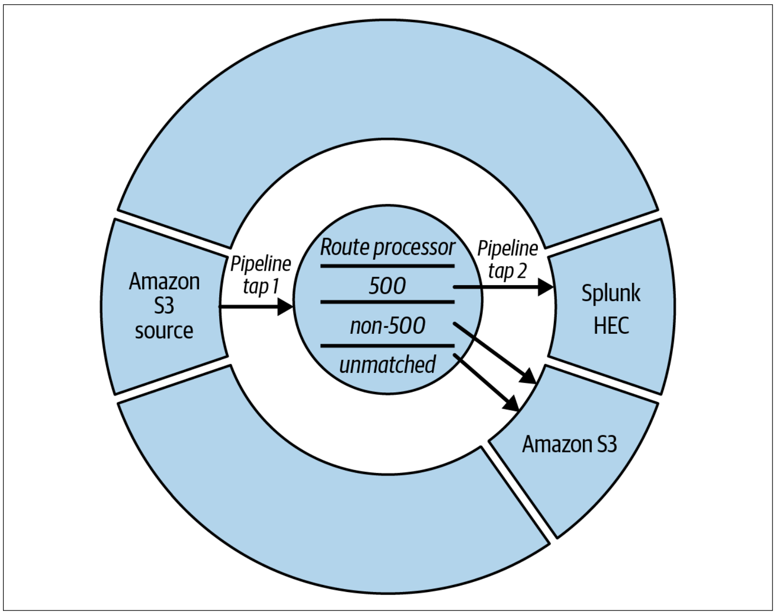


Figure 3-1. Using pipeline taps to inspect the telemetry events flowing into and out of the streams from a route processor

1 See Chapter 4 for a sample of the typical processors you can expect to use in your telemetry pipelines.

Debugging Your Pipeline

The need for pipeline taps becomes even clearer when you consider the example of debugging a pipeline that not only pushes data to an observability tool, but also filters, encrypts, and archives that data to a persistent storage location. Let's use the same example earlier where your 500 status log entries are being routed and deduplicated on the way to a Splunk HEC destination, ready for immediate perusal by some auditors. At the same time, you're storing all the log data in a new location in S3, as shown in Figure 3-2.

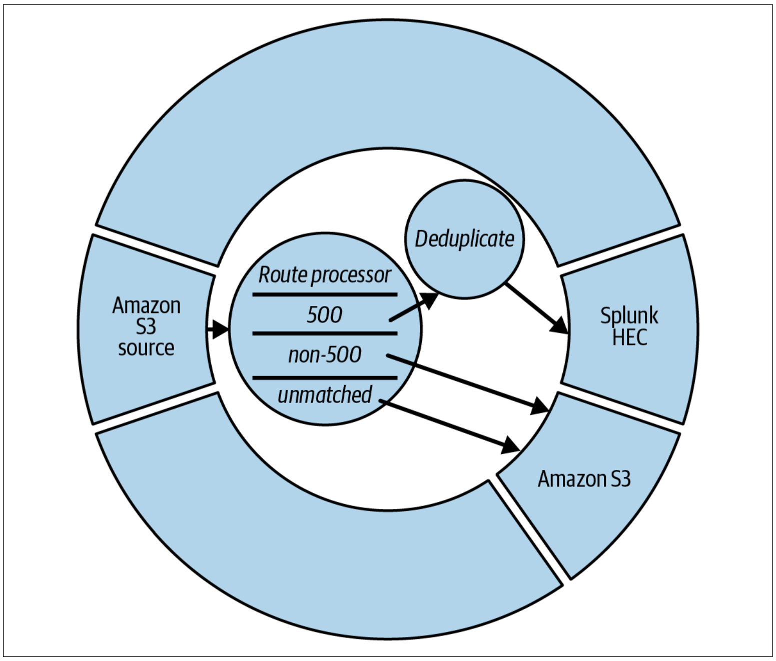


Figure 3-2. Sourcing log data from S3, routing through only the 500 status log events, and ensuring no duplicates are sent to save space before ferrying what remains to your Splunk HEC destination

A quick glance at the pipeline dashboard ingress and egress should confirm that the volumes of data being pushed to Splunk HEC are well within tolerable levels. That's a relief, since it is not cheap to push vast quantities of data to observability tools that can charge by the byte. (See Chapter 4 for more on cost control.)

The volume looks OK, but is the data of the right form? Adding a pipeline tap to the stream exiting the deduplicate processor can help you build confidence that the right data, and only the right data, is being popped over to Splunk, as shown in Figure 3-3.

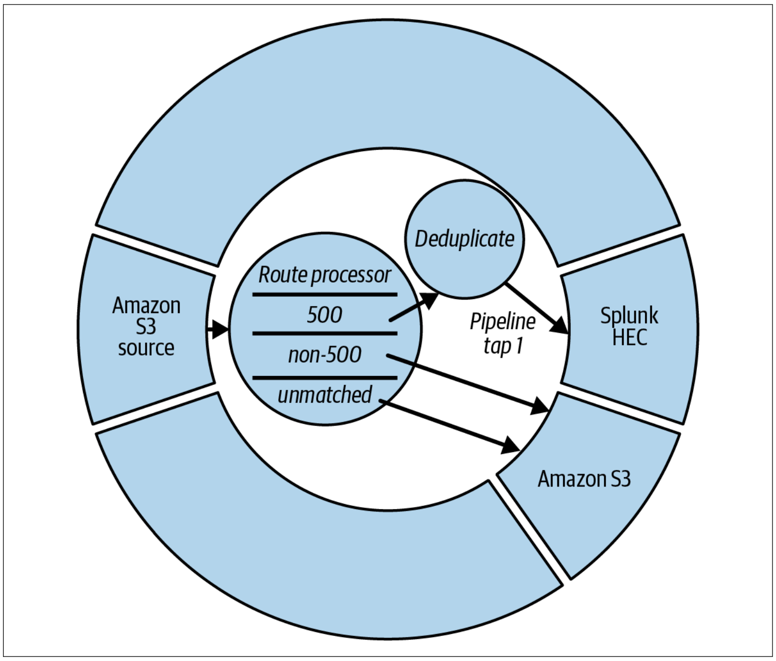


Figure 3-3. Tapping the stream of output data to build confidence that only 500 status log events are present and that any duplicates are being removed before flowing to Splunk HEC

You feel good; everything seems to be working fine. And then the universe decides to remind you that it doesn't have your best interests at heart, as the auditors raise a worrying question: is the data that's being directed by the route processor being stored encrypted at rest? Especially any personally identifiable information (PII)?[2]

A quick tap on the two streams heading to S3 confirms your fears. There *is* personal information, and it *is not* being encrypted. At least with your pipeline taps you can see that PII data is flowing through the pipeline, but how are you going to fix that unintended exposure?

It is exactly this sort of case that the encrypt processor is built for. Adding an encrypt processor to both of those streams can ensure

2 More on this sort of awkward question in Chapter 5, where you'll learn how to ensure that your telemetry is complying with a host of important rules.

that any data item that matches your criteria is effectively encrypted before storage, as shown in Figure 3-4.

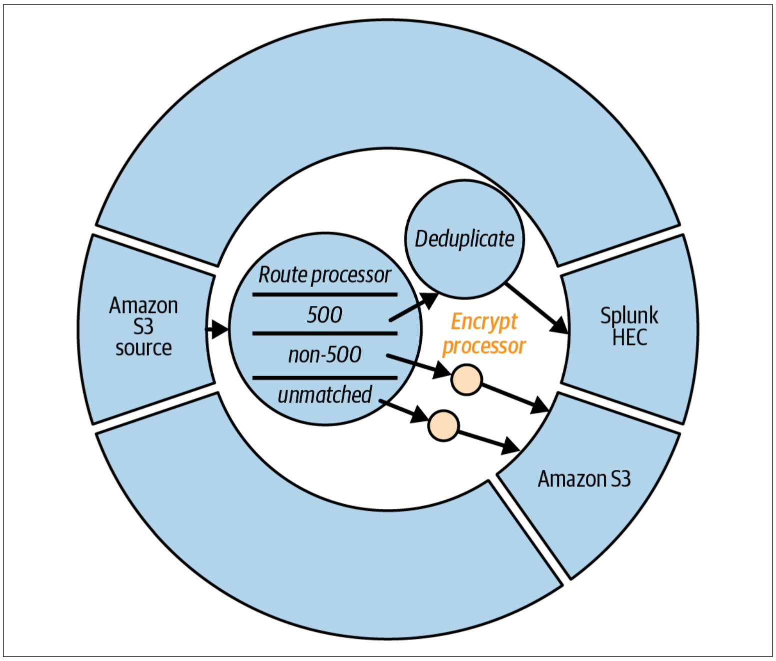


Figure 3-4. Adding an encrypt processor to the streams of telemetry data heading to archive on S3, ensuring that any fields that contain personal information are encrypted before transfer and storage

Nasty problem averted. Thanks to telemetry pipelines, and especially pipeline taps, you remain on friendly terms with both the auditors and the regulators!

From Control and Management to Business Value

In this chapter, you've begun to see the value of telemetry pipelines as well as how you can then use the pipeline metrics and pipeline taps to observe, improve, and even debug your telemetry streams before they get anywhere they shouldn't. But this is only the beginning. The benefits of telemetry pipelines go beyond data control and into saving money.

CHAPTER 4

Containing the Cost

"Show me the money!"

—Jerry Maguire

Volume is the problem, but not just because it is hard to navigate and work with. Data, especially in the cloud, costs money. Sometimes *lots* of money. If a potential $65 million bill doesn't scare you, then your organization is doing exceptionally well. For the rest of us, cost *really* matters.

Processors Are the Key

In Chapters 2 and 3, you got a glimpse of how telemetry pipelines can help with cost. Some key processors that can help you control cost are deduplicate, route, reduce, sample, filter, and conversion processors.

Deduplicate Where You Can

When it comes to cost, the deduplicate processor is your brutally simple friend. By applying some simple logic, the deduplicate processor can reduce your telemetry data streams significantly without losing any data. This is why the first step for designing a telemetry pipeline must include first getting an understanding of your data, so you can effectively determine which processor to use to target your data components.

Choose Your Route Carefully

At the simplest end of the scale, you can merely choose where your telemetry data goes. If you want to optimize your spend on Splunk, you can ensure that only the data necessary for Splunk is routed to it. The remaining data could be routed to low-cost storage, such as S3, so that nothing is lost just in case. It's that simple, sort of.

The art here is to ensure that you are still routing something useful to your destinations. A router might not give you the right level of intelligence to create a stream that is ultimately useful to your tooling destinations. You could end up paying for a first-class tool that is utterly hamstrung by third-class data.

As always, and with all of the cost-strategy processors, it's going to be a trade-off, but at least you get to *make* that trade-off in your telemetry pipelines.

Reduce, Sample, or Filter It Down (Carefully)

Sampling and filtering loses data.... Say it with me: sampling and filtering *by definition* loses data. That sounds bad, and it is; however, sampling and filtering can still be used with care when you really have to. The good news is that a telemetry pipeline gives you the choice to sample or filter your data if you need to *before* the stream hits an expensive destination.

Downsampling or selectively filtering your telemetry data using a sample or filter processor will reduce the number of events you see in the resulting stream.[1] For example, a simple sampling strategy is to ignore every second event; this is called 1/2 sampling. Only sending every third event is 1/3 sampling. This type of sampler is very common and is, for perhaps obvious reasons, called a *1/n sample processor*.

You can also configure a 1/n sample processor to ignore the sampling if an event matches a particular pattern. To prevent dropping those crucial events amid the noise, you can configure a sample processor to notice the events and let them through with no downsampling applied.

1 See "Adding Processors" on page 13 for a short description of these processors.

Sampling can be extremely helpful with keeping control of the flood of data, particularly where specific events are less useful and a broad brush is all that is needed. But you have to use it carefully and get those exception cases configured so that you don't accidentally sample out the one important piece of information that you need.

Filtering is more controlled. With a filter processor, you are looking for events that match a specific set of criteria so that, when they match, they can be dropped. You can filter out whole events or just drop single fields, reducing the data in the stream and lowering costs at any of your destinations.

In an ideal world, which none of us live in, we wouldn't sample or filter at all other than to improve the condition of the events using something gentle like deduplication. There is, however, a get-out-of-jail-free card available with telemetry pipelines. If you route your telemetry data *before* your sampler to an archive—something much cheaper than a full observability tool destination, such as S3—then you can persist a record of the raw data. That repository becomes an asset should you ever need to suck up the cost and bring that data back into your observability tools.[2] It's not perfect, but it gives you options you would not have had without a telemetry pipeline at your disposal.

Converting Events or Logs to Metrics

Another useful technique to reduce data volumes and increase insights is converting logs to metrics. Logs are inherently unstructured and voluminous, but you can derive metric data from logs by parsing the log data to extract specific information and then use that information to create metrics. Or you can count specific events within the log data and use that count to create a metric.

As an example, by identifying a specific value within a log message, such as the time it took to serve a request, you can create a new metric. Now you can use this distilled information for your analysis in security information and event management (SIEM) or for visualization in tools like Grafana. This not only reduces the log volume, but also helps extract business insights.

2 Retention management is often built into cloud storage, such as S3 lifecycle policies.

Cost, Controlled

By carefully using route, deduplicate, reduce, sample, filter, and conversion processors, you can at least control how much of your telemetry data ends up where. You can also choose to park your data in locations, such as S3, that can be cheap options for storage and, through telemetry pipelines, can be easily reborn into streams for processing and channeling in the future. It's with this choice and flexibility that telemetry pipelines really help you shine when managing the cost of your observability.

But in addition to cost, there is one other challenge with telemetry data that we've only hinted at so far. The elephant in the room. Data isn't just about cost, it's about control. Serious control. Serious go-to-jail-if-you-get-it-wrong consequences. We are, of course, talking about compliance.

By extracting metrics from raw events and log data, you can discern more value from your telemetry data assets and see where your money is being spent on effective ingress and egress of that data. And this is only the beginning.

The visibility and control your telemetry pipelines give you can help you deliver better business insights and even improve your security posture by making sure the right data is sent to your security tools at the right time. You can accelerate resolution times and improve customer experience as well as help ensure that you meet all required data compliance regulations. Next, it's time to explore that specific case. Risk and compliance, anyone?

CHAPTER 5

Embracing Compliance

Finance is about the money you make. Compliance is about the money you keep.

—Anonymous

In Chapter 4, I imagine we all universally agreed that $65 million is a lot of money. As the amount of one single observability bill, it feels a bit much. Well, in this chapter the stakes are raised just a touch higher. Millions are for lightweights. Let's talk about billions: $30.6 billion, to be in the approximate ballpark. Give or take a few million.

Fines for financial services noncompliance have reached as high as $30.6 billion in some extreme cases. Admittedly, this was for a host of irregularities ranging from misdemeanors to out-and-out fraud and enabling money laundering on an industrial scale, and so not necessarily what you can expect from mishandling your telemetry data.

But that's just financial noncompliance. What about health care data compliance, like HIPAA? For those hoping to operate in Europe, what about General Data Protection Regulation (GDPR)?

Whatever business you're in, there is likely a reasonable expectation that you will comply with various rules and regulations. Even if you're a small service provider just trying to obey local restrictions around how you safely handle PII, there are rules.

And observability tools hate rules. Limiting data access to specific individuals, ensuring that data is anonymized, ensuring that data can be processed and dropped if necessary—all of these things give creators of observability tools nightmares. Great observability thrives on breaking down barriers, aiming to establish a free environment where open questions can be asked about any current and historical condition within your systems. Asking that your observability tools and practices can audit all access, limit the proliferation of information, and ensure that all PII is removed is not just tricky—it can be anathema to your observability goals.

Luckily, you have telemetry pipelines. Observability tools hate rules, but telemetry pipelines *love* them.

The Key Is Processors, Once Again!

By now, you'll likely be seeing the form. Bringing your telemetry data flood into your telemetry pipelines gives you the visibility and control to work with that data and condition it to the restrictions of any sets of compliance rules you are looking to comply with.

The tools remain the same, even if their importance escalates. The key processors for helping you ensure that your telemetry data is obeying compliance rules are the route, redact, and encrypt processors. Let's look at how these come into play for a specific type of compliance: GDPR.

The Case of Conformance to GDPR

GDPR is an acronym that, when it came into European law, sparked fear in providers around the globe. It applied to anyone trying to operate a service in Europe; you didn't have to exist there to be subjected to it—you just had to trade in Europe. If your service's fingers were on European Union citizen data without following GDPR, you could expect a fine of up to 20 million Euros or 4% of your annual global turnover. Not $30-plus billion, but still a dent in the reported quarterly earnings.

The problem you face is that your logs probably contain the kind of personal data that GDPR tries to protect European citizens' rights to. Even if you just run a simple web service, you likely log user's IP addresses, the URLs they tried to request, their usernames, perhaps even their complete records on your system in the event of a failure (for debugging purposes, of course). This data is *helpful*, but it's also covered under the rights protected by GDPR.

While observability tools may shrug in the face of this problem, telemetry pipelines can have your back. Using route, encrypt, and filter processors, you can reduce your telemetry data in transit and at rest, potentially filtering out or pseudonymizing personal information.

What Happens When You Pseudonymize User Data?

Pseudonymization replaces PII with nonidentifying data that can still be used to recognize a person. As an example, a set of data in your telemetry might contain the details of John Doe, including his address and credit card information. After pseudonymization, the system will still know that the data pertains to a person, but it won't be able to tell the person is John Doe specifically.

Helping with Compliance: Another Telemetry Pipeline Payoff

Earlier, I told a white lie. While processors are the keys to actually implementing the steps necessary to be compliant with various regulations, it is the pipeline itself that is the real magic.

The fact that you can explore, condition, and control your streams of telemetry data makes it much more possible to enforce compliance rules on that data. While it's achievable to be compliant without telemetry pipelines, it's far from as easy or as clearly audited.

CHAPTER 6
Conclusion

Railroad iron is a magician's rod in its power to evoke the sleeping energies of land and water.

—Ralph Waldo Emerson

On the face of it, telemetry pipelines don't sound like they will change your world, but like the railroads in Ralph Waldo Emerson's quote, there's an enormous amount of power in their simplicity. From just the four building blocks of sources, streams, processors, and destinations, you can build pipelines that can help you solve problems more quickly, save your organization money, or even avoid huge compliance fines. And those are just the cases we've had time to cover here.

When you invest the time necessary to look at and think differently about your telemetry data, you can start to treat it as the valuable resource it really is. No longer is it merely a peripheral side effect; your telemetry data—metrics, traces, logs, and other events like alerts—is transformed into something visualizable, manageable, and, most importantly, valuable.

You get to streamline what gets through to your various live observability downstream tools, hedging your bets by storing anything you don't *think* you'll need today in a cheap backup location. Then the world opens up to deliver better business insights, improve your security posture, accelerate incident resolution times, improve customer experience, and, as described in the previous chapter, even ensure that you meet all required data compliance regulations. And all of that is likely just scratching the surface.

When you shift your thinking through using telemetry pipelines, the flood of telemetry data morphs from an overwhelming problem into an unmined opportunity. You shift from “Can I handle all this?” to “I can make this my own.” Telemetry pipelines put you in control of your telemetry. Use that power wisely.

About the Authors

Russ Miles is a people, team, and organizational developer; writer; psychologist; speaker; and humanistic engineering lead/manager.

Kai Alvason holds a PhD in comparative literature and has been a member of the faculty at the University of Virginia and Towson State University. They have held roles in technical writing and editing, information architecture, content strategy, and technical education at major enterprises like Dell and Amazon, and currently lead technical communications at Mezmo.