LEARNING MADE EASY

2nd Cribl Special Edition

Observability Pipelines

for dummies®
A Wiley Brand

Route data in the optimal format

Improve performance, slash costs, be flexible

Control and shape your observability data

Brought to you by

Cribl®

Alexandra Gates

About Cribl

Cribl makes open observability a reality for today's tech professionals. The Cribl product suite defies data gravity with radical levels of choice and control. Wherever the data comes from, wherever it needs to go, Cribl delivers the freedom and flexibility to make choices, not compromises. It's enterprise software that doesn't suck, enables tech professionals to do what they need to do, and gives them the ability to say "Yes." With Cribl, companies have the power to control their data, get more out of existing investments, and shape the observability future.

Founded in 2017, Cribl is a remote-first company with an office in San Francisco, CA. For more information, visit www.cribl.io and cribl.io/community.

Connect with Cribl on social media:


www.linkedin.com/company/cribl


twitter.com/cribl_io

Observability Pipelines

2nd Cribl Special Edition

by Alexandra Gates

for dummies®
A Wiley Brand

Observability Pipelines For Dummies®, 2nd Cribl Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-98212-8 (pbk); ISBN: 978-1-119-98213-5 (ebk). Some blank pages in the print version may not be included in the ePDF version.

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Manager and Development Editor: Carrie Burchfield-Leighton

Sr. Managing Editor: Rev Mengle

Managing Editor: Camille Graves

Acquisitions Editor: Ashley Coffey

Business Development Representative: Molly Daugherty

Table of Contents

Introduction

How do you get data visibility from your infrastructure and applications in order to properly observe, monitor, and secure their running states while minimizing overlap, wasted resources, and cost?

Many business folks need a broad variety of tools, such as up and down monitoring, metrics, a time series database (TSDB), log analytics, event streaming, security information and event management (SIEM), user behavior analytics (UBA), and data lakes, and they need them in all their environments to solve challenges related to system and application performance and security monitoring and compliance. This complex mix of services is essential to run a growing organization. So how do you get visibility in the face of this complexity and ever-increasing data volumes? The answer is by using an observability pipeline.

About This Book

This book introduces you to the concept of observability pipelines, which help slash costs, improve performance, and get the right data, to the right destinations, in the right formats, at the right time. *Observability* is the practice of interrogating your environment without knowing in advance the questions you need to ask. That means making choices and living with the consequences. These choices almost always present tradeoffs. More data means more money. Standardizing on one platform limits flexibility, and dropping data may limit what you can learn from your environment.

Peruse the chapters in this book to see how implementing an observability pipeline can keep you learning limitlessly from your environment.

Foolish Assumptions

When writing this book, I made a couple of assumptions about you, the reader:

- You're reading this book because your organization is beginning to embrace the practice of observability, and it's your job to make sure everyone gets the data they need to achieve their goals.
- You're interested in observability and the challenges it can pose to the unprepared enterprise.

Either way, you're in the right place to discover what observability pipelines are and how proper implementation can drive improvements across your business.

Icons Used in This Book

Throughout this book, you see special icons in the margins of the chapters. These items alert you to important information. Here's what to expect:

TIP

This icon highlights information that may save you time, money, and more. Tips can help you do things quicker or easier.

REMEMBER

Content with this icon is important to remember on your observability pipeline journey.

WARNING

Information contained here points out struggles you want to avoid with observability pipelines.

TECHNICAL STUFF

If you like to know the technical details about a topic, pay attention to this icon. It provides you with all the techie, juicy details.

Beyond the Book

This book can help you discover more about observability pipelines, but if you want resources beyond what this book offers, here's some insight for you:

- **cribl.io/blog/the-observability-pipeline**: This blog, "The Observability Pipeline," shows you how implementing an observability pipeline can help you achieve increased visibility, easier administration, and reduced cost.
- **cribl.io/blog/observability-lake**: This blog, "The Observability Lake: Total Recall of an Organization's Observability and Security Data," shows you how to store data in open formats in a low-cost way, to avoid vendor lock-in and get the most from your data.
- **cribl.io/blog/why-log-systems-require-so-much-infrastructure**: Check out this blog, "Why Log Systems Require So Much Infrastructure," to get an explanation of the economics of storing data in analytics tools and suggestions of strategies for dramatically reducing storage and compute infrastructure requirements.
- **sandbox.cribl.io**: Cribl Stream Sandbox is a collection of interactive, self-guided courses that show you how to implement many of the concepts in this book.
- **cribl.io/resources/observability-pipeline-buyers-guide**: The Observability Pipeline Buyer's Guide is a downloadable PDF that walks you through the most important factors to consider when implementing an observability pipeline. It compares open-sourced tools to commercially available solutions and includes a checklist of features to ensure that your observability pipeline can evolve with your business.
- **cribl.io/roi-calculator**: Explore how much Cribl Stream could potentially reduce your infrastructure costs.

IN THIS CHAPTER

- Understanding observability
- Figuring out your needs
- Looking into how your environment operates
- Questioning your data
- Understanding why all data can't be structured
- Finding your observability solution

Chapter 1
Embracing the Practice of Observability

You may be reading this book to discover how an observability pipeline helps you get control over all the data you use in your observability and security projects. Before I get into the nitty gritty, in this chapter, I cover the basics: what observability is, how you should collect and structure your data, and how observability pipelines can tie all these topics together.

Defining Observability

Observability is a growing practice in the world of software development and operations. Observability gives you the opportunity to learn about the important aspects of your environment without knowing in advance the questions you need to ask. Put more simply, it seeks to answer how much you can understand about a system by looking at it from the outside.

For example, how much can you tell from outside an aircraft about whether the engine is working? You can probably tell if it's running because it's making noise and vibrating — and maybe that

it's running properly if there's not a lot of violent shaking inside the cabin, if the plane flies, and so on. But to learn more intricate details, you need sensors — different ones that measure things such as how much air is being pulled through the engine, how hot the engine is, the consistency of the RPM, and how efficiently the plane is consuming fuel.

REMEMBER

Observability is like an analog dial; the amount of observability you employ can fall anywhere along a wide spectrum, spanning from nothing to acutely tuned to every aspect of your environment that's measurable. You can have low or high levels of observability, but what matters is if you have the level that *your* business needs.

Determining Your Needs

If all your business cares about is if the engine (from the preceding section) is working to spec, then that's all the observability you need. But, what if you also wanted to know how the passengers in the plane are doing? For that, you need an entirely different set of sensors: air pressure and quality, temperature, and maybe even the output from onboard surveys. Or maybe you'd want to know how many times people complained to flight attendants.

Similarly, if your business depends on how happy your customers are, you need more and different kinds of sensors to determine this value. You need to generate, collect, and analyze the data from these sensors, as well as store it somewhere.

TIP

So you may be wondering how you determine the level of observability that's right for your business. Start with the goals of your organization. These may be things like sales targets, customer satisfaction, market share, reputation in the industry, and so on. Although these goals may not seem like they have a lot to do with how well your IT systems are working, I encourage you to look a little deeper.

For example, break down your sales targets a bit to see where observability can help. For instance, assume you're a clothing retailer. Sales are impacted by the demand for your clothes, your prices, store locations, and your online presence. The obvious place for observability to play a part in sales is web sales. Ask yourself the following questions:

- How quickly does our website load?
- Are there differences in site performance for mobile or desktop browsers?
- Can we guarantee the safety of sensitive customer information (do you think twice about buying from a company that just announced hackers compromised its customer data)?

If you look a little closer, observability can also address in-person sales:

- Is our inventory system giving fast, reliable answers to buyers so shelves are stocked with the most popular styles and brands?
- Does our loyalty application run so slowly in the store that customers forego signing up for a discount leading us to lose a new advertising channel?

REMEMBER

To optimize observability, you must

1. **Seek to understand the countless ways your IT systems impact the goals of the organization.**
2. **Start compiling a list of questions about how your systems, applications, networks, and so on are operating to have the impacts from Step 1.**
3. **Translate the questions from Step 2 into things you can measure.**
4. **Figure out what measurements are acceptable.**

   For example, you want to have sub-second response times for returning queries about available inventories for a specific item.

So, what kinds of sensors should you have in place to make the measurements that will help you understand how well things are running? You can collect this data in several ways:

- **Servers, cloud instances, virtual machines (VMs), or containers** often take snapshots of their operations on regular intervals and write them into logs (more on these in Chapter 2).
- **Logs** can also be created by log scrapers or forwarders — pieces of software that look at your systems, take measurements, and write them as log data.

- **Agents** are another type of software that sit on all the end points of your systems and collect metrics (see Chapter 2) that explain what's going on within your environment.

These measurements can be for time intervals or related to interactions and transactions within your applications and systems. For example, when a sale is made, you want to understand

- How much time it took to complete
- How quickly you return a confirmation message
- What percentage full is the database where you store information about the sale
- When a database with customer information is queried and what servers made this request
- If this server is trusted or a potential threat

Understanding How Your Environment Is Operating

At a minimum, your enterprise's business goals almost certainly include strong security, high-service stability, and increased customer happiness. To understand how you're meeting those goals, you must collect and analyze data correlated with your desired outcomes. But how do you do that, you ask?


TIP

Start by collecting and analyzing the three main components (also called *pillars*) of observability: metrics, logs, and traces — more on these in Chapter 2. To attain a comprehensive view of how your environment performs, most organizations collect data from hundreds to hundreds of thousands of sources. After you have that data, you need to get it into the tools you use for analysis in the right format. I cover delivering data to your analytical tools in Chapter 2.

Asking Questions of Your Data

In order to ask questions of your data, you have to structure it in a way that the analytics tools your organization uses can understand. Unfortunately, many data sources have unique structures that aren't easily readable by all analytics tools, and some data sources aren't structured at all. Some of the tools your organization uses to review and analyze data may expect the data to have already been written to log files in a particular format, known as *schema-on-write,* and some tools involve an indexing step to process the data into the required format as it arrives, known as *schema-on-read.*

For more information on how to ask these questions and what schema-on-write and schema-on-read entail, head to Chapter 3.

Explaining Why Observability Requires All Types of Data

Applications are typically instrumented by the developers who write them. The goals developers have for their instrumentation aren't often the same as the goals of the operators who run their applications or the end-users who interact with them. Developers tend to instrument to identify problems with the code and not so much with the consumer experience in mind. The goals of these operators, or systems supporting the applications, may change over time and well after the developer has moved on to other projects.

WARNING

Asking developers to consider all the ways in which their apps may emit useful data, and to whom it may be useful, in a structured way across many modules and applications and projects isn't generally successful.

True observability requires visibility into a broader spectrum of attributes about your environment. This means collecting a much wider range of data up front. Instead of trying to anticipate specific questions, strive to observe and gather data from all areas of your operating environment and have the flexibility of deciding later whether that data can help you answer any new questions

that may arise. If you were taking a test and knew which questions were going to be on it, you'd only have to study a narrow set of topics. Because you don't know what's on the test, it's best to master all aspects of your subject if you want that A grade. Of course, studying everything takes a lot of time, just as collecting more data can be expensive without the right approach. An observability pipeline can help you achieve that mastery over your environment without crippling your budget.

Finding Your Right Solution by Using an Observability Pipeline

To take advantage of the wide variety of options possible for structure, content, routing, and storage of your enterprise data, an observability pipeline allows you to ingest data and get value from that data in any format, from any source, and then direct it to any destination — without breaking the bank. An observability pipeline can result in better performance and reduced infrastructure costs.

As your goals evolve, you have the freedom to make new choices, including new tools and destinations as well as new data formats. The right observability pipeline helps you get the data you want, in the formats you need, to wherever you want it to go.

Data volumes are growing year over year, and at the same time, companies are trying to analyze new sources of data to get a complete picture of their IT and security environments. They need flexibility to get data into multiple tools from multiple sources but don't want to add a lot of new infrastructure and agents. These companies need a better strategy for retaining data long term that's also cost effective.

An observability pipeline helps you parse, restructure, and enrich data in flight — before you pay to store or analyze it. It gets the right data, where you want, in the formats you need. Observability pipelines unlock the value of observability data by giving you the power to make choices that best serve your business without the negative tradeoffs (I cover tradeoffs in Chapter 4). As your goals evolve, you have the freedom to make new choices, including new tools and destinations.

As an example, take a look at the observability pipeline in Figure 1-1. It's a universal receiver of data that can collect, reduce, and transform data and route it to a wide variety of observability and security tools and storage destinations.

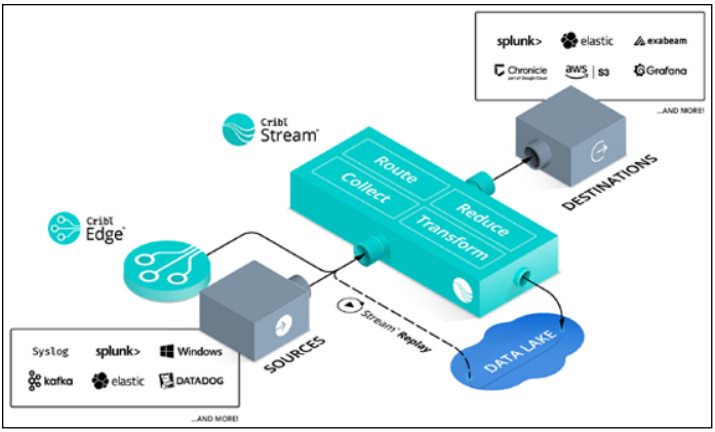

FIGURE 1-1: Cribl Stream — the observability pipeline with replay.

Check out Chapter 5 for more reasons why you'd want to use an observability pipeline as your data solution.

IN THIS CHAPTER

- Figuring out your observability goals
- Looking at the types of valuable data
- Getting your data to the right tools for analysis

Chapter 2
Evaluating Where You Stand in Meeting Your Goals

You can't get to where you're going if you don't know where you're starting. To be successful on a journey, you need to set a course to your final destination. So how do you do that for your IT environment? Data is how you evaluate how well you're meeting these goals. It can show you where you are on the map, how quickly you're traveling, and which roads get you to your destination most efficiently. In this chapter, I give you information on evaluating your goals both in security and IT operations. You also discover how to get information from your data and deliver that to your analytical tools.

Determining Your Enterprise's Observability Goals

If observability is about seeking answers to questions about how well your IT environment is running, then you need to know how to measure what's acceptable for meeting the observability goals of your business. Some of these goals revolve around the following:

- The cost of storage and compute infrastructure required to run your environment
- How secure your applications and data are from threats
- How well your systems are performing
- How stable your systems are

After you understand how your IT environment affects your observability goals, you need to come up with a list of metrics that indicates success or failure as well as values for each of these metrics so you know when you're succeeding or failing. (I cover what metrics are in the later section "Gleaning Information from Your Valuable Data.")

TECHNICAL STUFF

Choosing a list of what metrics to measure and what outcomes are good or bad is a subset of the practice of observability. For a long time, IT departments have been held to service level agreements (SLAs), which are essentially contracts that stipulate what level of service they provide. For example, one may be that your website needs to have 99.9 percent uptime, requiring scheduled or unplanned outages to comprise less than 0.1 percent of the time. An evolution of the SLA is the service level objective (SLO). SLOs are goals the IT department maintains to deliver better service. SLOs are often more stringent than SLAs because they're aspirational and help set the priorities of the organizations; whereas, SLAs are the bare minimum of what's acceptable.

TIP

To start your list of metrics, begin by breaking down all the ways your systems impact your observability goals. Then take higher level measurements and break them down into smaller components that impact that metric. For example, site uptime can be impacted by average response time, CPU utilization percentage, available storage percentage, and the number of concurrent query requests to an inventory database. Many of these items

will already be captured by standard log and metrics collection, but you can add new measurements as you understand how they impact larger goals.

After you have your list of metrics, try to understand which data sources can be queried to determine the values of your desired metrics. In many cases, the sources of data reside within the infrastructure itself. In other cases, you may need to add tools to measure other aspects of your environment that you aren't collecting today.

REMEMBER

One of the most direct tools for recording metrics about your systems and applications are agents. *Agents* are software programs deployed on your infrastructure that write out information about what's going on in that server, application, network device, and so on. Together with standard information coming from the infrastructure, data recorded by agents can give you a better picture of how well you're meeting your goals and which finer grained aspects need attention. Just like secret agents in a spy movie, agent software sneaks in and collects intel without anyone knowing it was there. You may also want to factor in third-party data sources that may impact how your systems are running. For example, how does external air temperature impact the performance of a server farm?

Gleaning Information from Your Valuable Data

After you establish your goals and you know what to measure — and what the most relevant data sources are — you need to figure out how to make sense of this data and turn it into insights. A few steps lie between getting your data and learning from it. Observability practitioners generally consider three main pillars of data as inputs for learning about an IT environment. These pillars are metrics, logs, and traces.

Metrics

Metrics are numeric representations of data measured over intervals of time. Metrics can harness the power of mathematical modeling and prediction to derive knowledge of the behavior of a system over intervals of time in the present and future.

For example, every time you go for a medical checkup, the nurse who takes you back to a room collects a set of metrics, such as your height, weight, blood pressure, temperature, and pulse. The nurse logs the time, as well as other "dimensions" such as your name, patient number, what doctor you're seeing, and the reason for your checkup. Think of this collection of multiple metrics, with one set of dimensions, as a metric event.

In the digital world, dimensions may include an application name, a host name, a payment type, or whatever else may have been of importance to the developer, who wrote the code, and the product manager. From cloud instances, servers, applications, and devices, you may get metric events that include

- A timestamp
- Metric values, such as
  - Percentage of CPU utilization
  - Percentage of memory in use
  - Load average
  - CPU temperature
- Dimensions, such as
  - Hostname
  - Location
  - Department
  - Business function

Each of these events can be used to analyze and report on what's measured. They consolidate important measurements for monitoring for quick point-in-time health checks. By aggregating, you get a sense of performance without having to store each unique log file.

Logs

A *log* is a system-generated record of data that occurs when an event (see the preceding section) has happened, and this log describes what's going on during the event. A log message contains the log data. *Log data* are the details about the event such as a resource that was accessed, who accessed it, and the time. Each event in a system is going to have different sets of data in the message.

Think about a ship's log from back in the old wooden sailing-ship days. Several times a day, the captain (or someone assigned to the task) noted standard things:

- Date and time
- Heading and speed
- Latitude and longitude
- Personal notes, such as
  - "Today, we ran out of rum."
  - "The cook burned his hand, and Dr. Ian bandaged it up."

For each time he wrote something down, he entered a *log entry,* and the book he wrote in was the *logbook.* A captain's *log* was the collection of all that captain's logbooks.

In the digital world, logs refer to information being written by the operating system and by the applications and processes running on that system. As with the captain's log example, each log entry usually includes a set of standard things to report, such as the following:

- The date and time of the event (also known as the *timestamp*)
- The name of the system logging the data
- The severity of the event (critical, warning, and so on)
- The user or location involved
- The application name

In reality, the log message format is generally a lot less wordy than this example. Log messages may be in key-value format, JSON format, CSV, or literally anything else. Translating one format to another, on the fly, can help get logs generated by one system into a completely different tool.


TECHNICAL STUFF

In the digital age, log entries are called *log events.* Log events of a particular type, or those from the same source, are written to a log file locally — or sent across the network to another system. There are different approaches one can use in the transmission of the log events, but generically you can refer to the whole process as "sending log events to a log server." And, just as you had a

captain's log (things logged by the captain), the digital equivalent of this includes Windows security logs, web server logs, email logs, and so on.

Traces

A *trace* marks the path taken by a transaction within an application to be completed. This may be a query of the database or execution of a purchase by a customer. Think of those Indiana Jones movies where they showed the red line traversing the globe to represent how he got from one adventure to the next. A single trace can provide visibility into both the route traveled as well as the structure of a request.

The path of a request allows software engineers and site reliability engineers (SREs) to understand the different services involved in the path of a request, and the structure of a request helps you understand what services, other applications, databases, and other system elements are involved in the transaction or request.

In application monitoring, a trace represents all the things an application transaction spent its time on — all ordered over time from beginning to end. Ant farms are a good way to think about how traces work. They visually display the path an ant takes to do its work. Traces, similarly, show how different parts of an application perform their jobs. Traces also show where potential problems may occur. If you're constantly seeing the biggest bottleneck at the point you query a database, you can see what else is happening at the same time:

- Are there conflicts?
- Are other applications trying to access the same data at the same time?
- Is there an inefficiency in the code that makes this request?

Traces show application developers how the application is doing its work so they can prioritize areas to be optimized. If something is broken in the overall application, traces can show what's happening before and after the problem to pinpoint what needs to be fixed.

Delivering Data to Your Analytical Tools

The data you need to collect can come from some or all the pillars of observability (see the earlier section "Gleaning Information from Your Valuable Data"), and believe me when I tell you that many strong opinions exist around which pillar is the most valuable, what kind of data to collect, and how to collect it. But in reality, it all depends entirely on your business and its goals.

Most organizations collect a wide array of data from multiple sources. After the data is collected, you have to get it to a tool to analyze it. How does this happen? Pipelines stream data from sources to their destinations. After collecting data, pipelines then stream it to an analytics tool in the format required by that tool. Whether you're looking at metrics, logs, or traces, it's likely that you're generating far too many to make sense of it all without using an analytics tool.

REMEMBER

Each tool your organization uses may have widely different formats for how data can be read and interpreted. Think of something as simple as a timestamp. Some tools may be formatted YYYY-MM-DD hh:mm:ss; others may include fractional seconds or the day of the week in abbreviated format such as *Thu.* Different tools may also have different names for field values or expect log data to be in a specific order. Regardless of the format of the tools you use, you have to deliver that data to the tools in a way that they can use.

Data usually streams in real time from your collectors to your analytical tools. Streams of data are produced in the forms of events. As events occur, metrics, logs, and traces are generated and enter the stream. These events may be regularly scheduled, such as time intervals to check various measurements of your environment. They can also be related to something that happens in your environment — an application notices that a customer has made an order, or you get an unauthorized access request to your inventory database.

Regardless of why the event enters the stream, you need to decide how to get that data to the right tool to be analyzed. In simple environments, you can create unique pipelines for each pair of data sources and destinations. For most organizations, however, that approach will quickly become cumbersome because you have multiple tools analyzing overlapping pieces of the same data, as illustrated in Figure 2-1.

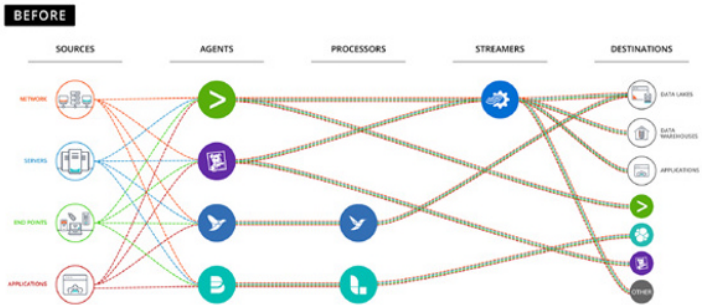

FIGURE 2-1: Creating unique pipelines for each pair of data sources and destinations.

REMEMBER

An observability pipeline, shown in Figure 2-2, can take multiple sources and translate that data for a variety of different tools. With an observability pipeline, you can unify collection of data in one tool and then process, filter, and reformat that data and route it to the destination(s) best suited for analyzing and taking action on that data.

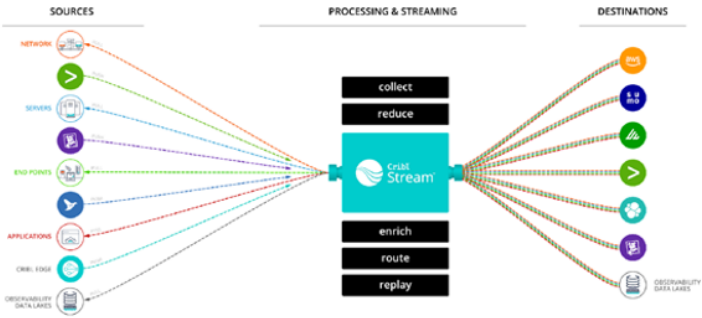

FIGURE 2-2: Unify collection and routing of data with an observability pipeline.

IN THIS CHAPTER

- Reviewing and analyzing data
- Defining structured logging
- Understanding the need to collect unstructured data

Chapter 3
Identifying Your Choices to Structure Data for Your Analytics Tools

An important consideration at this point is how you structure the data for your analytics tools. The format and methods of this structure impact storage volume, compute requirements, and analytical performance. This chapter covers several approaches and helps you decide which are best suited to meet your observability goals.

Reviewing and Analyzing Data with Your Current Tools

Your current tools are tasked with helping you find answers to various observability and security questions. To perform this analysis, the data needs to be structured and formatted so these tools can quickly find your answers. Because enterprises use a lot of different tools, there isn't always a consistent approach regarding structure. A lot of the data you ask these tools to analyze isn't structured at all or is contained in complex, nested structures that aren't easy to read.

The format for data in an analytics tool is often called a *schema,* and each tool has a unique schema. Getting data into these schemas generally falls into two approaches:

- **Schema-on-write** formats data before being stored in the tool. This process requires you to shape data into specifically organized log files.
- **Schema-on-read** typically involves indexing all the data before it's processed into the required format before it can be stored and analyzed.

To complicate matters further, security teams often need to use data from a large number of different sources, much of it in different formats. In order to correlate and analyze networks, file systems, applications, and threat feed data from differing sources, they have to be normalized — made to fit a standard structure that supports a security analyst searching and calculating across all the types of data.

Historically, this process was done via enforcement of schema-on-write, trying to know in advance what format the data needs to be in, and what it should contain, and enforcing that schema before anything gets logged for the most effective use in analysis, alerting, or troubleshooting.

This process used to be less difficult when there wasn't as much data, but those days are gone, and it's simply not possible to build and maintain a system that can handle the volumes, varieties, and velocity of modern enterprise data streams. Not only is there a lot more data that's valuable to look at, but also there's a lot more data that isn't valuable, and it's impossible to know which is which when so many different use cases exist for the data.

REMEMBER

While schema-on-read has certain advantages, the tool often dictates which approach you use, and some tools may be better on other factors that are important to observability efforts.

Looking at the advantages of schema-on-write

Schema-on-write is the more straightforward approach to ingesting and analyzing observability data. You define your schema up front, which makes it significantly faster to query. Requiring data to be in a pre-ascribed format makes it faster to store data as it's

ingested. With schema-on-write, you also require decisions to be made up front about which data may be useful in order to get the answers you need.

Seeing the advantages of schema-on-read

Among the reasons you would want to use schema-on-read is that this approach maximizes the types of data that can be stored and analyzed. Because many formats can be ingested, you aren't as restricted to the types of information from which you can draw insights. Unstructured or semi-structured data is also more easily used in schema-on-read tools because data streams aren't required to adhere to a set format as they're ingested.

Understanding Structured Logging

Structured logging is an approach that takes into account how data will be used by various tools and destinations. By structuring data — either as you collect it or before you route it via a pipeline to a log analytics tool — you make that data more useful for uncovering insights about your environment.

Log data is often an artifact of the application development process. Developers generate data from their systems to help them debug and optimize the applications themselves. They're generally not thinking about how their decisions will impact the ongoing maintenance and management of these systems. As a result, the data may be structured in a way that's optimized for the application development process and not for observability. Many times, no structure exists at all because observability may be an afterthought to the development process. This obviously doesn't help with your organization's overall observability efforts.

You may be asking yourself why developers can't just add the structure to logs as they're generated. This doesn't work for a few reasons:

- That isn't their job. That's the truth; once they have their applications working the way they want, they move on to the next project.

- A lot of log data is generated based on hardware or network protocols, and the structure (or lack thereof) is fixed.
- Developers can't always go back and re-instrument their systems to perfectly match the needs of how your tools analyze data.

REMEMBER

To understand structured logging, you have to work with developers to build observable systems. Consider capturing a wider array of information to future-proof changes in what you'll need to observe later. But how can you do that? Observability pipelines are part of the solution, of course. But the most important thing is to consider how data will be used and how and when to structure it for optimal use.

Why Does Observability Require All Types of Data?

Many types of data don't neatly fit into the schemas required by the tools you use. They may, however, have a lot of information that's critical to answering your organization's biggest questions. You don't always control how that data is generated, especially if it's third-party data sources, such as a database of known security threats or a table that converts IP addresses into geographical locations. Also, some logs are auto generated by operating systems, third-party applications, or servers. The short answer to the question, "Why does observability require all types of data?" is because they're filled with useful information.

Data points, such as the geolocation of an IP address, may be useful for balancing the workloads of regional teams. Perhaps you have regional IT teams that are better equipped to investigate the source of security threats in their regions. The data coming from servers may not be formatted in the schemas required for analysis, but it has several key pieces of information that can be placed into a structured log to provide more context and strengthen the insights you seek to draw from your environment.

You also can't change the fact that data comes in all shapes and sizes, so spending a lot of time forcing data into a structure that fits today's goals is likely to be upended when business goals evolve and you're asking new questions of your infrastructure.

REMEMBER

If you don't collect all types of data, regardless of how they're formatted, you may be missing the opportunity to analyze it all later. One of the advantages of schema-on-write (see the earlier section, "Looking at the advantages of schema-on-write") is the efficiency it gives you by putting all data into a schema that you need now. By collecting this data and storing it for later, you'll have a more complete picture of your environment.

IN THIS CHAPTER

- Dealing with multiple types of expenses
- Trying to maintain flexibility
- Maximizing observability with comprehensive data analysis
- Adding an index to your toolbox

Chapter 4
Looking at the Tradeoffs in Making IT Decisions

Making decisions in IT often presents tradeoffs between the following competing goals:

- Cost
- Flexibility
- Comprehensive data analysis
- Speed

REMEMBER

How much observability data you collect and how long you retain it can impact all four of these factors.

Being able to add new tools gives you a more flexible approach to observability but can bust your budget (check out Chapter 3 for more about analytical tools). An observability pipeline can help you balance these tradeoffs and eliminate the negative impacts from these choices. This chapter discusses these tradeoffs in more detail.

Tradeoff #1: Cost

Many factors influence the cost of analyzing data. Three of the biggest costs are software licensing fees, data storage costs, and infrastructure compute expenses.

Software licensing/consumption fees

Commercial software companies and cloud service providers typically charge licensing or consumption fees based on the amount of data ingested into the platform. The more data you send to the tool, the more you spend. Even if some of that data has no analytical value whatsoever, you still pay for it.

Data storage costs

You have to store data somewhere in order to analyze it. The cost of storing this data is significant — in some cases, making up more than half of the total cost of licensing fees. How long you retain this data is also a multiplier on the cost of storing it (or part of the consumption calculation). Keeping data in storage for multiple months is often required to meet compliance targets.

WARNING

Where you store data impacts the cost (for example, you can choose S3 or other low-cost storage), but your choices may be limited if you need access to this data for later analysis.

Infrastructure compute expenses

More data also drives the amount of compute infrastructure required. In order for data queries and analysis to be performant, you need to have enough computing power. As with storage costs (see the preceding section), low value data is just as taxing on your systems as valuable data. Your tools need to work harder to find the insights you need.

TIP

How much computing power is enough? You need computing power that's fast enough to perform queries and correlations of your data. Think about looking for a needle in a haystack — the bigger the haystack the longer it will take you to find the needle. Adding additional people that are looking can speed up the search. Similarly, to get a performant search of logs, you either need a smaller set of data to query or more servers performing the analysis.

Tradeoff #2: Flexibility

When companies started analyzing log data, they may have been satisfied with a single tool for the needs of different departments. As the types of analysis and the availability of tools have evolved, companies are increasingly employing multiple tools to get the answers they need. Different teams need to get different answers, so it makes sense that they have the flexibility they need to choose the best tool for the jobs they're performing. You can attain this flexibility and mitigate this tradeoff through an observability pipeline.

WARNING

Standardizing on one analytics platform limits innovation and stifles departmental independence. If you try to use a one-size-fits-all approach, you're limiting visibility into your environment.

Most of these analytics tools have their own agents and collectors that need to be placed on all the endpoints across the enterprise. This means you're collecting a lot of the same data multiple times, just in different formats. All this duplicative data adds to the amount you have to store, which, you guessed it, drives more costs.

Tradeoff #3: Comprehensive Data Analysis

The goal of observability is to learn about your environment to improve performance, availability, and security. Each new question you ask of your environment needs a different set of data to analyze (even though those sets of data contain a lot of the same data points). Different teams set out to answer unique questions that drive new requirements for data collection. You need comprehensive data analysis to maximize observability.

You'll likely find answers to most of your questions in real time in the data stream. Consider the opportunity to teleport into endpoints to determine which data you want to route to which destination to provide the most value to your organization.

Some questions, however, require the analysis of past data. One example is investigating a security breach. These breaches often occur long before they're discovered. If you can't easily access data from the time of the breach, your investigation may be incomplete. Consider adding the ability to route some data sets for immediate analysis and other data sets to long-term, cheap storage to meet data retention, compliance, and act as an insurance policy should you need additional context on a security breach.

Tradeoff #4: Adding an Index for Faster Searches

Log systems, by the standards of most data analytics systems, are easy to get data into and query. Compared to traditional databases or data warehouses, log systems generally require minimal planning. This success has led to a new problem where log systems have become the dumping ground for many use cases for which they're poorly suited.

Indexing log data was a huge innovation. Searching for a specific word or pattern used to be a time-consuming task. Indexes now map the data as it comes in so it can be searched through much faster. Think about reading through every piece of data to search for a specific pattern — using an index is obviously much faster.

An index reduces the size of the data you're searching by limiting the search to a subset of the data, so the speed of the search is based on how many columns of indexing you're adding. The more you index the faster the searches, but that also means the more data you need to keep and the more silos you have to manage.

REMEMBER

Therein lies the tradeoff — the size of data required for an index. Indexes require extra storage in addition to the raw text, but indexes can greatly reduce the amount of time and computing power required to go find rare terms in large data sets. The ability to rapidly find rare terms in terabytes or petabytes of raw data is a massive innovation. Unfortunately, as is often the case, this core innovation has been stretched to be a one-size-fits-all solution for all log data problems. For some workloads, indexes are a wasteful optimization.

IN THIS CHAPTER

- Discovering strategies for observability
- Introducing Cribl Stream

Chapter 5
Navigating a Successful Observability Journey

REMEMBER

To truly understand your environment and learn how to improve it, you need to study the data it generates about how it operates. In Chapter 3, I mention that observability requires you to collect and analyze all types of data, regardless of format or schema. This means gathering performance, health, and security measurements from all your applications, infrastructure, and other endpoints — that can add up to a lot of data and get expensive (check out Chapter 4 for more about cost). Luckily, an observability pipeline can help you manage massive amounts of data without swallowing up your budget.

An observability pipeline helps you properly structure and route data to the tools that you use to answer your most important questions. This chapter helps you pilot a successful observability journey by looking at observability strategies. Also, in the last section of this chapter, I detail one example of using an observability pipeline.

Looking at Proven Observability Strategies

Collecting, storing, and analyzing data in your pursuit of observability nirvana can take several paths. In this section, I outline some proven strategies using an observability pipeline.

Collecting your universal set of observability data

Historically, every analytics tool required its own unique agent or collector. Even though different tools use a lot of the same data, each tool collects data in its own format. That means you may be collecting overlapping sections of data multiple times for different tools. Adding agents for every tool across your environment is a labor-intensive process and can rob your applications of computing power. Collecting duplicate data sets can also strain the performance of your observability tools.


TIP

A better approach is to collect all the data that describes your environment from every application, server, device, endpoint, and third-party data source. Use this universal set of data to feed all your analytics systems.


REMEMBER

An observability pipeline can translate all your incoming data formats to fit any format your tools (or data destinations) use. Because observability pipelines can use data from each tool and transform it to be analyzed by multiple other tools, you won't need to deploy a new agent or collector for every tool. Collect the data once, and re-use it as often as you need to.

Separating your systems of analysis and systems of retention


REMEMBER

In order to analyze data, you need to put it somewhere. Many organizations retain data for several months, for compliance purposes, and some industries require multiple years of retention. The cost of retaining this data can be significant and depends largely on volume, the retention time, and where you store it. Indexed logging tools excel at fast searches, but tradeoffs of using

an index are a significantly larger data footprint and the need to use more specialized storage technology. These tradeoffs make retaining data long term in these tools much more expensive. For more about this tradeoff, check out Chapter 4.

Several systems for retaining observability data are much more cost-effective than an indexed analytics tool. These include file systems, data lakes, and low-cost cloud storage. One example is Amazon Simple Storage Service (Amazon S3).


TIP

Separate the systems you use to *analyze* data from the systems you use to *store* data for longer-term retention. S3-compliant storage has significant advantages as a system of retention, as compared to indexed analytics tools. If you choose infrequent access — an option that assumes you won't need access to the data often — S3 storage can cost a fraction of the cost of the block storage that an indexed tool uses.

The best practice is to send a full copy of your data to low-cost storage. Because the cost associated with this option is much lower than storing data in your system of analysis, you can keep more data for longer retention periods. You can still send data with high analytical value in real time to your analytics tools, but you don't need to keep that data for retention purposes; simply drop it from those systems after you're done analyzing it.

On rare occasions, you may need to investigate older data (the vast majority of analysis is done on data generated in the past few days). One example of analyzing older data is for data breach investigations. These often occur several months (and in some cases, several years) before they're discovered. Because you stored a copy of your data in low-cost storage, you can retrieve this data and send it to your security tools to properly scope and remediate the breach. This process is made much easier by using an observability pipeline to collect data and replay it to the right tool.

Transforming your data and routing to the best tools

After you have your universal set of observability data, you can maximize that data's value by transforming the data into any format and enriching it with relevant context, then sending it to the most effective tools to achieve your observability goals.

TIP

Choose the best tool for each team to answer their observability questions. Tools have become a lot more specialized over the years, which means that a security team may want to choose the best security information and event management (SIEM) tool to meet its goals, while the IT operations team may want a different tool to analyze data for its projects. Both of these teams are likely looking at a lot of the same data, from the same sources, but they may be asking different questions. And depending on your cloud and digital transformation goals, you may need to send similar data to premises-based and cloud-hosted tooling. By transforming your universal set of data to meet the needs of individual teams, or business priorities, you reduce the tradeoffs among flexibility, cost, and the complexity of adding extra agents (check out Chapter 4 for more on tradeoffs). An observability pipeline allows you to collect that data once, enhance and filter it, and then use it with whichever set of tools best fits your needs.

Getting to Know Cribl Stream and Cribl Edge

Cribl Stream is an enterprise-ready, out-of-the-box observability pipeline that's purpose-built to help you unlock the value of your observability and security data. Stream is a universal log-and-metrics router that helps you parse, restructure, filter, mask, and enrich data in flight, so you can ensure that you get the right data where you want it, and in the formats you need. A companion product, Cribl Edge, acts as an agent allowing you to teleport to endpoints and decide what to collect, connecting to Cribl Stream as necessary for appropriate transformation, routing, reduction, and enrichment.

TIP

With Cribl Stream and Cribl Edge, you can process observability data — logs, traces, metrics, and so on (I cover these in Chapter 2) — in real time and deliver them to your analysis platform of choice. It allows you to

- Add context to your data, by enriching it with information from external data sources.
- Help secure your data, by redacting, masking, or encrypting sensitive fields.
- Optimize your data, per your performance and cost requirements.

- Complement or update your existing agent structure, using one agent to feed multiple data streams and analytics tools.
- Collect and send only the data relevant to your observability, security, and analytics goals or compliance mandates.

In Figure 5-1, you see an illustration of the basic process that Cribl Stream uses to manage and store your data.

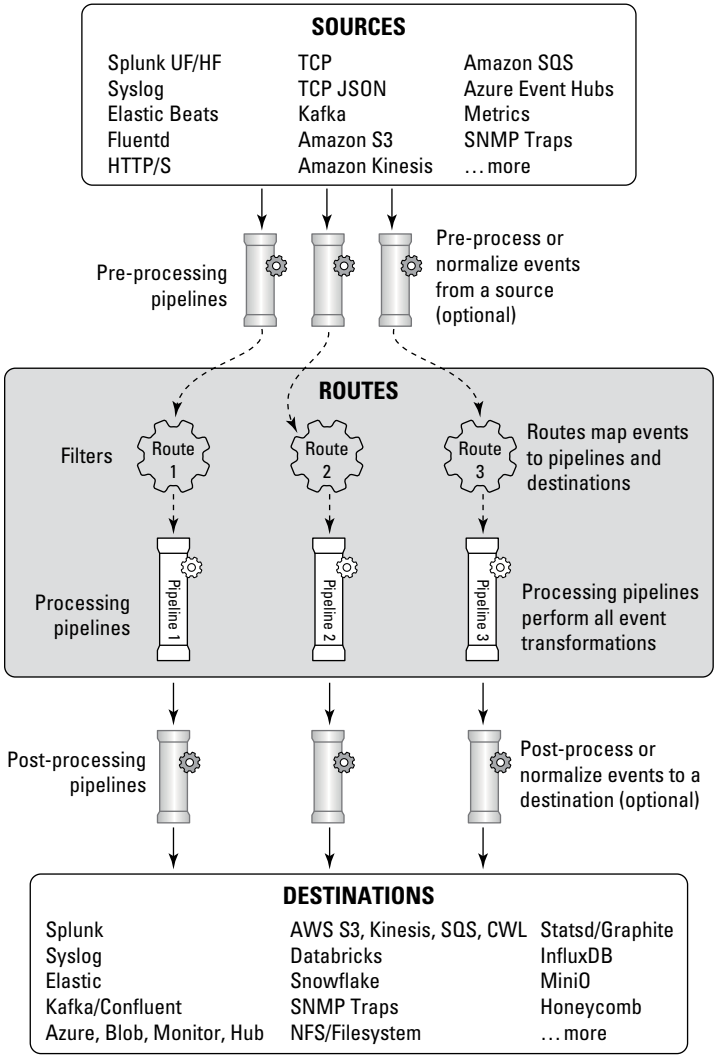

FIGURE 5-1: Looking at Cribl Stream in action.

Stream also has a special feature called Stream Replay that lets you land data in lower-cost storage and send it to an analytics system later if you need it. Analytics tools typically look at data that streams in real time — or very close to when it's actually being measured. Replay allows you to turn previously collected data into streaming data so it can be analyzed by these tools. The data you've collected from lower-cost storage or application programming interfaces (APIs) is processed and then routed to the right tool (replayed) through Stream whenever you want to analyze it.

For more information, visit `docs.cribl.io/docs/about`.

IN THIS CHAPTER

- Routing and reducing your data
- Replaying data from object storage to analytics tools
- Masking your customers' data
- Performing more comprehensive analysis

Chapter 6
Ten Reasons to Use a Highly Flexible, Performant Observability Pipeline

You have a wide variety of options for structure, content, routing, and storage of your data, but an observability pipeline allows you to ingest and get value from data in any format and from any source, and then you can direct it to your destination of choice — to keep up with the growth of data without bankrupting your company. The right approach to observability helps you find the balance between cost, performance, complexity, and comprehensiveness.

Enterprises need the flexibility to make decisions on a source-by-source basis. For each shape of data in each log file, the decision may be different. Making these types of data reshaping decisions often involves going back to the developers and asking them to log differently. For security professionals, your infrastructure software and cloud vendors dictate the formats of much of your data. Shuffling data off to cheap storage to have a cost-effective place to land full-fidelity data may be possible with your existing

tooling, but replaying the data is an intense manual effort of writing and running scripts and workarounds. Even deciding at ingestion time where to send data is impossible in most pipelines. Most solution ingestion pipelines are built only for that solution.

REMEMBER

An observability pipeline can

- Simplify decisions as the data is moving.
- Increase or decrease the number of fields based on simple, graphical no-code pipelines.
- Defer decision making until later.

  This process happens by spooling data to cheap object storage like S3 compatible storage or data lakes and replaying it later if you decide the data needs to be analyzed further or shaped differently.

In this chapter, I give you ten reasons to use an observability pipeline, such as Cribl Stream. I cover Cribl Stream in more detail in Chapter 5.

Route Your Data to Multiple Tools and Destinations

With an observability pipeline, you can take data from any source and route it to any tool (destination). Put data where it has the most value. Route data to the best tool for the job — or *all* the tools for the job. For more information on delivering data to your analytical tools, flip back to Chapter 2.

Reduce Data with Little Analytical Value to Control Costs or Make Room for More Interesting Data

An observability pipeline can help you reduce less-valuable data before you pay to analyze or store it. This solution can help you

- Dramatically slash costs.
- Eliminate null fields.

- Remove duplicate data.
- Drop fields you'll never analyze.
- Free up space for emerging data sources that add more value for the business.

Using an observability pipeline means you keep all the data you need and only pay to analyze and store what's important to you.

Transform Data into Any Format without Adding New Agents

Take the data you have and format it for any destination, without having to add new agents. By transforming the data you already have and sending it to the tools your teams use, you increase flexibility without incurring the cost, compute, and effort of recollecting and storing the same data multiple times in different formats.


REMEMBER

Agents are software that's deployed on your infrastructure that write out information about what's going on in your servers, applications, network devices, and so on. Check out Chapter 2 for a bit more about agents.

Retain More Data for Longer Periods

You never know when you may need a piece of data for later investigation, so you can hold on to your data longer by routing a copy to cost-effective storage. Send a copy of your data to cheap object storage such as data lakes, file systems, or infrequent-access cloud storage, and you'll always have what you need — without paying to keep the data in your systems of analysis.

Collect Data from Cheap Storage and Replay It

Most data flows in streams to analytics tools and is analyzed in real time, but some data may be interesting to analyze at a later point in time to learn about trends or to investigate a security breach that may have happened in the past. This is where Replay can help.

Replaying your data is collecting data from object storage, third-party data sources, or application programming interfaces (APIs) and restreaming them to an analytics tool. Replay allows you to take data at *rest* and stream it as if it was flowing in real time.


REMEMBER

Collecting a subset of data from low-cost object storage and restreaming it to any analytics tool as needed is a process that increases flexibility and comprehensive visibility, while minimizing the costs.

Enrich Data with Third-Party Sources for Deeper Context

An observability pipeline adds context to your data for more comprehensive analysis. Sometimes adding a small amount of data can unlock answers to critical questions. You can enrich your current data streams with key pieces of information to build a more comprehensive view.


TIP

Third-party sources, such as known threats databases, can enrich log data by alerting security systems to events that may require more scrutiny. They can also help you ignore log data from trusted domains. Similarly, another third-party source like a Geo-IP lookup table can provide geographical information for more context when performing investigations of past events.

Mask Sensitive Data to Protect Your Customers and Limit Liability

An observability pipeline configures data streams for maximum protection. Redaction and masking keep sensitive data private. You want to protect your customers and limit liability, and it's easy to do just that with an observability pipeline.


TECHNICAL STUFF

Redaction is the omitting of sensitive data elements such as credit card numbers from the view of people, applications, or processes accessing the data. *Masking* is the hiding of sensitive data so it can't be accessed by certain people, applications, or processes.

Manage Who Sees What with Role-Based Access Control

Role-based access control allows you to assign access policies — implementing restrictions or giving access — for teams and individuals. This security step gives your organization much more control over who can access particular types of data and what level of functionality they can use in your logging and pipeline tools.


TIP

Use role-based access control to manage your teams' access to data, so they can only see what they need to perform their jobs. Instead of each team being responsible for getting its own data into the tools of its choice, an observability pipeline centralizes the management of data and helps ensure people can only access what they need to do their jobs.

Collect Data from REST APIs for More Comprehensive Analysis

Getting a full view of your environment often means analyzing data that comes from sources other than traditional event streams. An observability pipeline can help you easily collect data from Representational State Transfer Application Programming Interfaces (REST APIs) and other sources in real time or for ad hoc, batch analysis — which formats this data for use by any analytics tool.

Better Understand Your Data with a Robust and Intuitive Management Interface

Reduce management overhead with a robust and easy-to-use Graphical User Interface (GUI)-based configuration and testing interface. Capture live data and monitor your observability pipeline in real time. Gain visibility into your data.

Your Data.
Your Choice.

Take control of **all** your observability data.

As Trusted By

Implement an observability pipeline

This book introduces you to observability pipelines, which help slash costs, improve performance, and get the right data, to the right destinations, in the right formats. Observability is the practice of interrogating your environment without knowing first the questions you need to ask. An observability pipeline helps you get the data out of your infrastructure and applications in order to properly observe, monitor, and secure their running states while minimizing overlap, wasted resources, and cost.

Inside...

- Embrace the practice of observability
- Determine your needs
- Understand your environment
- Ask questions of your data
- Learn why and how to structure data
- See how observability pipelines work
- Save money, analyze more data

Alexandra Gates is the Director of Product Marketing at Cribl. She is passionate about helping IT and security professionals understand and embrace the power of observability. Previously, she has led product and content marketing at Extreme Networks, Aerohive Networks, and Xirrus.

ISBN: 978-1-119-98212-8
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.