DataShelf Hub

CLOUD & RELIABILITY

Observability from Day One

Logs, metrics and traces you can actually debug an incident with

FREE SAMPLE CHAPTER

Omar El Alaoui
Data & systems engineer

PDF · CH. 01

ABOUT THIS SAMPLE

A free first chapter

What follows is Chapter 1 of *Observability from Day One*, reproduced in full and provided at no cost by DataShelf Hub. It is unabridged — nothing has been trimmed or summarised. If it's useful to you, the remaining seven chapters continue exactly where this one leaves off.

This sample is for your personal use while you decide whether the book is right for you. Please don't republish or redistribute it — share a link to the book's page instead.

TITLE	Observability from Day One
AUTHOR	Omar El Alaoui
SHELF	Cloud & Reliability
FORMAT	PDF, DRM-free, 236 pages
THIS SAMPLE	Chapter 1 of 8 — 15 pages
PUBLISHER	DataShelf Hub · datashelfhub.com

WHAT'S IN THE FULL BOOK

Table of contents

8 chapters · 24 sections

Continued on the next page →

TABLE OF CONTENTS, CONTINUED

Chapters 5–8

CHAPTER ONE

01

Debugging What You Can't See

When a system breaks at 3am, you don't rise to the occasion — you fall to the level of the instrumentation you built before you needed it.

The page arrives at 3:14am. Checkout error rate is up. You open the dashboard you built six months ago, the one that seemed thorough at the time, and it tells you CPU is fine, memory is fine, and the checkout service is up. None of that tells you why a customer just saw a 500. You open a terminal and start grepping logs across four services, by hand, trying to find the one request that failed, hoping whoever wrote that log line six months ago thought to include something useful.

This chapter is about the difference between the system you built and the system you can actually debug, because they are not the same thing, and the gap between them is invisible right up until the moment it costs you an hour at 3am. Observability isn't a dashboard you add later. It's a property of how a system is built, and like most such properties, it's dramatically cheaper to build in from day one than to retrofit onto something already running.

1.1 The 3am Problem

Every incident asks the same three questions, in the same order, whether anyone says them out loud or not: what's broken, why is it broken, and what do I do about it. The entire value of observability work — all of it, every log line and metric and trace you'll ever instrument — is measured by how fast your system lets you answer those three questions when you're tired, under pressure, and staring at something you didn't design to fail this way.

Most systems are optimised for the wrong moment. They're built, tested, and demoed when everything works, and the instrumentation that gets added along the way answers questions that made sense during development — is the service up, is it slow — without answering the question that actually matters during an incident: given that something is wrong, where specifically, and why.

> *"'Is the service up' is a question for a status page. 'Which of these four downstream calls is timing out, for which customers, and since when' is the question you actually need answered at 3am."*

That gap between the two kinds of question is the entire subject of this book. It isn't about adding more monitoring — most struggling systems already have plenty of dashboards. It's about the specific shift from monitoring, which tells you something is wrong, to observability, which tells you why, without requiring you to have anticipated the exact failure in advance.

That last clause is the whole point, and worth sitting with. A dashboard you built to watch for a specific failure mode only helps when that exact failure mode recurs. Real incidents are rarely repeats — they're new combinations of conditions nobody specifically monitored for. Observability is what lets you investigate a failure mode you've never seen before, using data collected before you knew you'd need it.

MONITORING VERSUS OBSERVABILITY, IN ONE INCIDENT

Monitoring tells you	checkout error rate is elevated
Observability tells you	it's specifically EU customers, specifically the payment-capture call, timing out since a deploy 40 minutes ago
The difference	whether you can ask a question you didn't anticipate, and get an answer

That capability — answering a question you didn't anticipate — doesn't come from any single tool. It comes from a small number of disciplined habits applied consistently across a system: logging structured, correlatable data rather than free text; choosing metrics that map to what a user actually experiences rather than what's easy to measure; and tracing requests across the service boundaries that make modern systems so hard to reason about by eye alone.

Those three — logs, metrics, traces — are the subject of Chapters 2 through 4, and each gets the depth it deserves. This chapter's job is narrower: to make the case for why they need to exist before the incident that needs them, and to give you a mental model — monitoring versus observability — sharp enough to tell, for any given piece of instrumentation, which one you're actually building.

1.2 Monitoring Versus Observability

The two terms get used interchangeably often enough that it's worth drawing the line precisely, because the line changes what you build. Monitoring is watching a predetermined set of signals for predetermined failure conditions — CPU above 90%, error rate above 1%, service unreachable. It answers "is something wrong" for the specific things you thought to watch.

Observability is a property of the system itself: whether its externally visible outputs — logs, metrics, traces — contain enough information to reconstruct its internal state after the fact, for a question you didn't know you'd need to ask when you built it. Monitoring is a set of checks. Observability is a capability the system either has or doesn't, baked into how it's instrumented.

A TEST FOR WHETHER YOU HAVE OBSERVABILITY, NOT JUST MONITORING

The question	could you answer a question about last week's incident that nobody anticipated asking, using only what's already recorded?
If yes	your instrumentation is genuinely observable
If no	you have monitoring for the failures you predicted, and blind spots for everything else

Most production systems fall firmly into the second category, not from negligence but from sequencing: monitoring gets built first, because it's simpler and answers the immediate question of "is it up." Observability requires a different kind of upfront investment — structured logs with consistent fields, trace context propagated through every service boundary, metrics chosen for what they'll let you ask later rather than what's convenient to emit now.

That investment is exactly what the rest of this book teaches, chapter by chapter. But the return on it compounds specifically during incidents, which is the one time engineering investment is hardest to justify in advance and easiest to regret not having made.

WORTH REMEMBERING

You cannot buy observability after an incident starts. Whatever data your system was recording when the failure occurred is the entire universe of evidence you'll have to work with — which is exactly why this book's title isn't a slogan.

1.3 Instrumenting from the Start

"From day one" in this book's title is a specific, practical claim: the cost of building observable systems is dramatically lower when it's part of how you build a service from its first commit, rather than retrofitted after the fifth incident that observability would have made trivial to diagnose. Retrofitting means auditing every log call across a codebase for consistency, threading trace context through call paths that were never designed to carry it, and redefining metrics whose meaning has quietly drifted for years.

Building it in from the start means none of that archaeology — a small, consistent set of conventions, applied as each new endpoint and service is written, that cost a few extra minutes per feature and pay for themselves completely the first time they turn a multi-hour incident into a five-minute diagnosis.

That's the trade this book is asking you to make: a small, consistent tax on every feature you ship, in exchange for a system that tells you the truth about itself when you most need it to. It's a trade every senior engineer eventually makes, usually after the incident that teaches it the expensive way. This book exists so you don't have to learn it that way.

Chapters 2 through 4 build the three pillars — logs, metrics, traces — in the detail each deserves. Chapters 5 and 6 turn that instrumentation into dashboards and alerts people actually trust. Chapter 7 puts it all to work on a live incident, start to finish, and Chapter 8 is about making the habits stick across a whole team, not just your own code.

None of it is complicated in isolation. It's the consistency — applied to every service, from day one — that turns a collection of logging tricks into an actual capability. Let's start with the first pillar: logs worth keeping.

§ Chapter Summary

Seven things worth carrying into Chapter 2.

BEFORE YOU MOVE ON

01 Every incident asks the same three questions — what's broken, why, and what to do — and observability is measured by how fast you can answer them.

02 Monitoring answers "is something wrong" for failures you predicted. Observability answers questions you never anticipated, using data recorded in advance.

03 You cannot buy observability after an incident starts — whatever your system was recording when it broke is all the evidence you'll get.

04 Instrumenting from day one costs minutes per feature. Retrofitting it later means auditing an entire codebase's worth of inconsistent logging.

05 The three pillars — structured logs, metrics that map to user pain, and distributed traces — only become a real capability when applied consistently.

BEFORE CHAPTER 2

Pick your last real incident. Ask honestly: could you have answered its root cause using only data your system was already recording, or did you end up adding a temporary log line and waiting for the problem to recur? If it's the latter, that gap is exactly what Chapters 2 through 4 are built to close.

END OF SAMPLE

Seven chapters still to go.

Chapter 1 draws the line between monitoring and real observability; the rest of the book builds the three pillars that make the difference real — structured logs, metrics that map to user pain, distributed tracing — and puts them to work on an actual live incident.