SAMPLE

Building the AiOps Infrastructure of Tomorrow

Building Intelligent IT Operations for Tomorrow

Aiman JoeHanz

Free sample: Introduction & Chapter 1

putrabooks.github.io

Buy the full book:

Carousell carousell.sg/p/building-the-aiops-infrastructure-of-tomorrow-ebook-pdf-1467067035

Gumroad

Ko-fi

ElevenReader

Payhip

Contents

Introduction

Chapter 1: The Complexity Crisis in Modern IT Operations

Chapter 2: Why Traditional Monitoring and Alerting Are Breaking

Chapter 3: The Cost of Inaction

Chapter 4: AIOps Foundations — Data, Observability, and Context

Chapter 5: Building the Data Pipeline — Logs, Metrics, Traces, Events

Chapter 6: Machine Learning for Anomaly Detection

Chapter 7: Intelligent Alerting and Noise Reduction

Chapter 8: Automated Root Cause Analysis and Diagnosis

Chapter 9: Automated Remediation and Self-Healing Systems

Chapter 10: Building a Continuous Learning AIOps Culture

Glossary

About this Series

Connect

This free sample contains the Introduction and Chapter 1. The complete book contains the full contents above.

Introduction

If you have spent any time managing modern IT infrastructure, you already know the feeling. It is three in the morning. Your phone is buzzing. An alert has fired — one of hundreds that have fired this week — and you are staring at a dashboard trying to figure out whether this one matters. The logs are voluminous, the metrics are ambiguous, and the cascade of dependent systems means that a single misconfigured load balancer can look like a database failure, a network outage, and a storage problem all at once. You are intelligent, experienced, and well-equipped. And yet, in this moment, you are drowning.

This is the reality of IT operations in the age of complexity. And it is the problem this book exists to solve.

AIOps — Artificial Intelligence for IT Operations — is not a product. It is not a vendor. It is not a buzzword to slap on a dashboard. AIOps is a fundamental rethinking of how we detect, diagnose, and resolve issues in increasingly complex, distributed, and dynamic technology environments. It uses machine learning, advanced analytics, and automation to do what human operators can no longer do alone: make sense of the firehose of telemetry data, separate signal from noise, and act before customers ever know something went wrong.

The need is urgent. In the last decade, the systems we run have grown beyond human capacity to manage manually. Microservices, containers, serverless functions, multi-cloud architectures, and event-driven designs have multiplied the number of moving parts by orders of magnitude. A single user request might traverse fifty services across three cloud providers before returning a response. When something breaks, the blast radius is enormous, the root cause is buried in layers of abstraction, and the time to resolution becomes the difference between a minor hiccup and a major incident.

Meanwhile, the expectations have only gotten higher. Customers demand five nines of availability. Businesses demand faster releases. Regulators demand audit trails for everything. And the teams responsible for keeping it all running are asked to do more with the same headcount, using tools that were designed for a world that no longer exists.

This book is for the people on the front lines of that world. Site reliability engineers, DevOps practitioners, platform engineers, IT operations leads, and technology decision-makers who know that the old ways of monitoring and alerting are no longer enough. If you have ever been buried under alert fatigue, spent hours tracing an outage across services you did not build, or wished that your monitoring system could just tell you what actually mattered

— this book is for you.

We will not sell you a vendor. We will not tell you that AIOps is a magic wand. What we will do is hand you a practical, end-to-end method for building AIOps capabilities in your organization: from the data foundations and observability pipelines, through machine learning for anomaly detection and noise reduction, to automated root cause analysis and self-healing remediation. We will cover the architecture, the team structures, the tooling landscape, the pitfalls, and the roadmap for getting from where you are today to a state where your operations are intelligent, proactive, and increasingly autonomous.

The future of IT operations is not more dashboards. It is not more alerts. It is systems that understand their own behavior, detect problems before they impact users, diagnose root causes with precision, and fix themselves. That future is not far away. It is something you can start building today.

The data is stark. Research from major industry analysts suggests that the average enterprise experiences dozens of significant IT incidents per year, with the average cost of a single hour of downtime ranging from tens of thousands to millions of dollars depending on the industry and scale. Operations teams in large organizations routinely drown in thousands of alerts a day, of which only a sliver deserve action. On-call engineers describe most of what they receive as noise, and burnout in operations roles ranks among the worst in technology.

These are not isolated problems — they are systemic symptoms of a fundamental mismatch between the complexity of modern systems and the maturity of the tools used to manage them. And they are getting worse, not better, as systems grow more distributed, more dynamic, and more interconnected.

AIOps is not a speculative technology. It is a practical, proven approach that is already being used by leading technology organizations to detect anomalies in real time, reduce alert noise by orders of magnitude, diagnose root causes in minutes instead of hours, and resolve common incidents automatically before any human needs to be involved. The results are measurable: detection that once took minutes collapses to seconds, alert volume falls by half or more, and a meaningful share of common incidents resolve themselves without a human in the loop — within the first two years of implementation.

The stakes could not be higher. In an era where digital experience is the primary interface between businesses and their customers, every second of downtime, every millisecond of latency, every failed transaction is not just a technical issue — it is a business issue. The companies that can operate their

complex systems reliably and efficiently will outcompete those that cannot. The companies that can detect and resolve problems before users notice will build trust that competitors cannot erode. The companies that can free their engineering talent from operational firefighting will innovate faster than those whose best people are perpetually putting out fires.

And yet, most organizations are operating with tools and practices designed for a world that no longer exists. They are using maps from a previous century to navigate a landscape that has been entirely reshaped. The gap between the complexity of their systems and the maturity of their operational practices is not just a technical debt — it is a strategic vulnerability.

This book aims to close that gap. Not by selling you a vision of a fully autonomous future that is years away, but by giving you a practical, step-by-step framework that you can start implementing today. Every chapter is designed to be actionable — you should be able to read a chapter and know what to do next, what to build, what to measure, and what to avoid. The examples are drawn from real-world implementations, the recommendations are battle-tested, and the roadmap is proven.

If you are an engineering leader, this book will help you understand why your operations need to change and how to build the case for investment. If you are an SRE or platform engineer, it will give you the technical blueprint for building AIOps capabilities. If you are a technology decision-maker, it will help you evaluate tools, set realistic expectations, and measure progress. Whatever your role, if you are responsible for keeping complex systems running in a world of increasing complexity, this book is for you.

A note on scope: this book focuses on the operational layer — detecting, diagnosing, and resolving issues in IT infrastructure and applications. It does not cover related but distinct domains such as security operations (SecOps), business process monitoring, or application performance management from a user-experience perspective. While AIOps principles and techniques are applicable to these domains, each has its own specific challenges, tools, and best practices that are beyond the scope of this book.

A note on technology: the AIOps landscape is evolving rapidly. New tools, platforms, and techniques are emerging constantly. This book is not tied to any specific vendor or product. The principles, architectures, and practices described here are vendor-agnostic and will remain relevant regardless of which tools you choose. This book deliberately names no products; where a tool class is described, it describes the category itself.

A note on expectations: AIOps is not a silver bullet. It will not eliminate all

incidents, automate all operations, or replace all human judgment. What it will do is make your operations dramatically more effective: faster detection, more accurate diagnosis, less noise, more automation, and a better experience for your team and your customers. The goal is not perfection — it is continuous, measurable improvement. If you commit to the journey, the results will follow.

Let us begin.

Chapter 1: The Complexity Crisis in Modern IT Operations

There was a time when IT operations was relatively straightforward. You had a set of servers, each running a known set of applications. You monitored CPU, memory, and disk space. When a threshold was crossed, you got an alert. You logged in, found the problem, fixed it, and moved on. The systems were finite, the dependencies were linear, and the relationship between cause and effect was discoverable with patience and a terminal.

That world is gone.

Consider the trajectory. In the monolithic era, a typical application might consist of a single deployable unit — a WAR file, a binary, a virtual machine image — running on a handful of servers. Monitoring meant watching a few dozen metrics. Troubleshooting meant looking at a handful of log files. The total surface area was small enough that a single experienced operator could hold the entire system in their head.

Then came virtualization, which multiplied the number of logical servers. Then cloud computing, which made provisioning trivial and encouraged sprawl. Then microservices, which shattered the monolith into dozens or hundreds of independently deployable services. Then container orchestration, which introduced constant churn — workloads being created and destroyed, rescheduled across nodes, autoscaling up and down by the minute. Then serverless, where the execution units are so ephemeral that traditional monitoring barely applies at all.

Today, a mid-sized technology company might run thousands of microservices across multiple cloud providers, with tens of thousands of containers in constant motion. Each of these generates metrics, logs, and traces. Each has dependencies on others. Each can fail, degrade, or misbehave in ways that propagate through the system in unpredictable patterns. The total number of observable signals — metrics, log lines, trace spans, events — easily reaches into the millions per minute for a moderately active system.

No human can process that volume of data. No team can manually correlate the signals across that many dimensions in real time. The complexity has outgrown the tools and the practices that were designed for a simpler age.

The other dimension of complexity is dependency. Modern systems are deeply interconnected. A frontend service calls an API gateway, which routes to a user service, which queries a database, which reads from a cache, which is populated by a background job that pulls from a message queue, which

receives events from an upstream data pipeline. Any link in this chain can fail, and the failure modes are diverse: a service can be down, slow, returning errors, returning wrong data, rate-limiting, or experiencing partial degradation.

When a user reports that "the app is slow," the problem could be anywhere in this chain. It could be a DNS resolution delay, a TLS handshake issue, a slow database query, a cache miss storm, a network bandwidth bottleneck, a garbage collection pause, or a cascading failure from a downstream service that is retrying requests it should not. The number of possible root causes is enormous, and the symptoms often look identical from the outside.

Traditional monitoring tools — which are designed around static thresholds and individual metrics — are fundamentally inadequate for this world. They can tell you that CPU is high on server 42, but they cannot tell you that the real problem is a connection pool exhaustion in a downstream service that is causing timeouts that are causing retries that are causing the CPU to spike. The causal chain is invisible to a tool that looks at metrics in isolation.

Complexity has also created a knowledge gap. In the monolithic era, a small team of operators might know the entire system. Today, systems are built and maintained by dozens or hundreds of teams, each owning a slice of the architecture. No single person or team understands the whole. When an incident occurs, the response requires coordinating across teams, each of which understands their own component but not how it interacts with the rest.

This means that incident resolution is no longer a technical problem alone — it is a coordination problem. It requires finding the right people, getting them in a room (virtual or physical), and facilitating a diagnosis that spans multiple domains of expertise. The mean time to resolve (MTTR) is often dominated not by the time it takes to implement a fix, but by the time it takes to figure out who needs to be involved and what is actually happening.

The volume of operational data has grown faster than our ability to make sense of it. A typical enterprise generates terabytes of logs per day. Metrics are collected at second-level granularity across thousands of dimensions. Distributed traces capture spans across dozens of services for every request. The data is there — in fact, there is too much of it. The challenge is no longer collection. It is comprehension.

Organizations have responded by building bigger dashboards, setting more alerts, and hiring more people to watch them. But this is a losing game. You cannot solve a complexity problem by adding more complexity. You cannot solve a data volume problem by throwing more human attention at it. The scale has exceeded what human cognition can handle, and the gap is widening

every year.

The complexity crisis is real, it is accelerating, and it is not going to resolve itself. As systems grow more distributed, more dynamic, and more interconnected, the gap between what we can observe and what we can understand will continue to widen. The old approach — human operators manually watching dashboards and responding to alerts — was never scalable, and it is becoming increasingly untenable.

The most fundamental issue with traditional monitoring is that it does not scale. Not in the technical sense — modern monitoring tools can ingest millions of metrics per second. The scaling problem is human. Traditional monitoring relies on human cognition for the most critical step: making sense of the data. A human must look at the metrics, correlate them with the logs, check the traces, consider the recent events, form a hypothesis about the root cause, and decide on a course of action.

This human-in-the-loop model worked when systems were small enough that a single person could hold the entire architecture in their head. It breaks down when the system has hundreds of services, thousands of instances, and millions of data points per minute. The cognitive load of making sense of that volume of data exceeds what any human can do in real time.

You cannot solve this by hiring more people. Throwing more operators at the problem creates coordination overhead that often exceeds the benefit. Five people staring at five different dashboards, each seeing a piece of the picture, trying to communicate their findings to each other in real time — this is not a solution. It is a different kind of problem.

You also cannot solve this by building more dashboards. A dashboard is a presentation tool, not an analysis tool. It shows you data, but it does not tell you what the data means. More dashboards mean more places to look, more tabs to switch between, and more cognitive overhead. The problem is not a shortage of visibility — it is a surplus of data and a deficit of meaning.

Traditional monitoring systems require extensive configuration: thresholds for every metric, alert rules for every condition, dashboard layouts for every team, routing rules for every alert. In a small, stable environment, this is manageable. In a large, dynamic environment, it is a full-time job — or several full-time jobs.

Consider the math. If you have 1,000 services, each with 50 metrics, and you set a high and low threshold for each, that is 100,000 threshold values to configure and maintain. Every time a service is updated, its performance characteristics may change, and the thresholds may need to be re-tuned.

Every time a new service is added, new thresholds need to be set. Every time a service is deprecated, its thresholds need to be removed. This is a never-ending configuration treadmill that consumes engineering time without producing lasting value.

And the thresholds themselves are rarely right. In practice, most thresholds are set once (often by copying from a template) and never adjusted. They are either too sensitive (generating noise) or too lax (missing problems). The team learns to work around them — muting noisy alerts, ignoring certain channels, developing informal heuristics about which alerts to take seriously. The formal monitoring system becomes increasingly disconnected from the actual operational practice, which is based on the team's collective experience and intuition rather than the configured rules. PRACTICAL INSIGHT

In most shops, a threshold is copy-pasted from a template on day one and never revisited. Tuned loose, they sleep through real failures; tuned tight, they cry wolf until nobody listens. The humans around them adapt — muting, ignoring whole channels, and building private folk wisdom about which pages are real. The formal monitoring system becomes increasingly disconnected from the actual op

Traditional monitoring creates a dangerous illusion of visibility. You have dashboards, so you must be able to see what is happening. You have alerts, so you must be able to detect problems. But visibility is not comprehension. A dashboard that shows you 500 metrics is not the same as understanding the relationships between those metrics. An alert that tells you CPU is high is not the same as understanding why it is high and what to do about it.

The visibility illusion is particularly dangerous because it masks the gaps. You believe you can see everything because you have tools that appear to show everything. But the tools show data, not insight. They show symptoms, not causes. They show the present, not the emerging future. The gaps — the things you cannot see, the relationships you cannot infer, the problems you cannot predict — are invisible precisely because the tools give you the impression that nothing is hidden.

AIOps is not about adding more visibility. It is about adding comprehension. It is about teaching the system to understand what the data means, to infer the relationships between metrics, to identify the causes behind the symptoms, and to predict the problems that are emerging.

To understand why the shift to AIOps is not optional but inevitable, it is worth examining the economics of manual operations in a complex environment. The cost of manual operations grows linearly (or worse) with the complexity of the

system, while the budget for operations is typically flat or growing slowly. This creates a widening gap that eventually becomes unsustainable.

Consider the components of operational cost in a traditional environment:

- Tooling costs. Monitoring tools, log management, APM, SIEM — each priced per host, per metric, per GB of logs. As the infrastructure grows, these costs grow proportionally. Many organizations find that tooling costs double every 2-3 years as their systems grow, even as the value per dollar of tooling stays flat or declines.

- Personnel costs. 24/7 coverage requires a minimum of 4-5 people per role (to cover shifts, vacations, and sick days). As the number of services and the complexity of incidents grow, the team needs more people, more specialized skills, and more experienced (i.e., more expensive) engineers. Personnel is the largest cost component, and it is the hardest to control.

- Incident costs. Each incident has a direct cost (downtime, lost revenue, SLA penalties) and an indirect cost (engineering time spent on resolution rather than development, customer support load, reputational damage). As incidents become more frequent and more complex, these costs grow.

- Opportunity costs. Every hour an engineer spends on operational toil is an hour not spent on engineering work — building new features, improving the architecture, reducing technical debt. The opportunity cost is the value of the engineering work that was not done, and it compounds over time as technical debt accumulates and innovation slows.

The total cost of manual operations in a complex environment is enormous and growing. AIOps does not eliminate these costs, but it changes the cost curve. Instead of growing linearly with complexity, the cost of AIOps-powered operations grows sub-linearly — the initial investment is higher, but the marginal cost of managing additional complexity is much lower, because the intelligence and automation scale without proportional increases in human effort.

This is the economic case for AIOps: it is not about spending less on operations, it is about changing the relationship between complexity and cost.

In a world where complexity is inevitably increasing, the ability to manage complexity without proportional cost increases is a decisive advantage.

This is the shift from observability to intelligence, and it is the shift that this book is about. To build systems that can process the volume of data, find the patterns, surface the anomalies, and present human operators with actionable insight rather than raw data. This is the promise of AIOps, and it is the

The migration to cloud computing was supposed to simplify operations. In many ways, it did — you no longer have to rack servers, wire network cables, or manage physical data centers. But the cloud also removed constraints that previously kept complexity in check. When provisioning a new server takes minutes instead of weeks, the natural tendency is to provision more. When each cloud provider offers dozens of managed services, the architecture tends to incorporate more of them. When auto-scaling can spin up hundreds of instances in response to traffic, the number of running instances at any given time can be an order of magnitude higher than in a fixed-infrastructure world.

The cloud did not reduce complexity. It shifted it. The physical complexity of data centers was replaced by the logical complexity of distributed systems. And logical complexity is, in many ways, harder to manage — it is invisible, it changes rapidly, and it is harder to reason about. You can walk into a data center and see the servers. You cannot walk into a cloud and see the microservices.

The multi-cloud trend has amplified this. Organizations that use two or three public clouds simultaneously are managing separate sets of APIs, separate monitoring consoles, separate best practices, and separate failure modes. The operational surface area multiplies with each cloud provider added. Cross-cloud dependencies — a service in one provider calling a service in another — introduce network variability, identity complexity, and observability gaps that single-cloud architectures do not have.

Modern DevOps practices have enabled organizations to deploy changes dozens or hundreds of times per day. This is a tremendous achievement, and it has real benefits: smaller changes are easier to reason about, faster feedback loops improve quality, and the ability to roll back quickly reduces the risk of any individual deployment.

But high deployment velocity also means that the system is in a constant state of flux. At any given moment, some service is being deployed, some configuration is being changed, some infrastructure is being provisioned or torn down. Each change is a potential source of instability. The more changes, the more potential sources. And because changes are deployed independently across services, the interactions between changes — service A's new version interacting with service B's new version — are combinatorial and impossible to fully test in advance.

This creates a paradox: the practices that enable agility (frequent deployments, independent services, rapid provisioning) also create the operational complexity that makes agility unsustainable without intelligent operations. You cannot deploy a hundred times a day and also manually

investigate every anomaly that results. The only way to sustain high velocity is to automate the operational response — which is precisely what AIOps is designed to do.

There is a cruel irony in the current state of observability. We have more tools, more data, and more visibility than ever before in the history of computing. And yet, we have never been less able to understand what is actually happening in our systems. The tools collect, but they do not comprehend. The dashboards display, but they do not explain. The alerts fire, but they do not prioritize.

The observability gap is not a data problem — we have more data than we know what to do with. It is a comprehension problem. The volume, velocity, and variety of operational data have exceeded the cognitive capacity of human operators. No amount of dashboard refinement or alert tuning will close this gap, because the fundamental limitation is not the presentation of data but the processing of it. And that is exactly the kind of problem that machine learning is designed to solve.

That is AIOps's actual promise: fewer screens, better questions, answers a human can act on. Fewer pages, sharper insight. Not more dashboards, but more decisions. The technology to deliver on this promise exists today. The question is not whether it can be done, but whether your organization is ready to do it. The rest of this book is a guide for getting ready.

KEY TAKEAWAYS - IT operations complexity has grown exponentially — microservices, cloud-native infrastructure, and distributed systems generate millions of telemetry points per minute. - Traditional monitoring tools cannot handle the scale, velocity, and diversity of modern infrastructure signals. - AIOps promises not more data, but more understanding; not more alerts, but more insight; not more dashboards, but more decisions.

End of free sample. The full book is available at putrabooks.github.io

ABOUT THE AUTHOR

Aiman JoeHanz writes the Building Infrastructure of Tomorrow series, taking the machinery of modern life apart page by page until its moving parts sit on the table, labeled. In this volume he follows the machinery that watches the machinery: the telemetry, baselines, and quiet automations standing between a healthy system and a 3 a.m. page.