THE SCALABLE SAAS ARCHITECTURE HANDBOOK

DESIGNING SYSTEMS THAT GROW WITHOUT BREAKING

ARCHITECTURE STRATEGY
MONOLITHS TO MICROSERVICES

EVENT-DRIVEN SYSTEMS
BUILD FOR DECOUPLING AND SCALE

DATA AT SCALE
SHARDING, PARTITIONING, AND BEYOND

CLOUD-NATIVE INFRASTRUCTURE
DESIGN FOR ELASTICITY AND EFFICIENCY

RESILIENCE BY DESIGN
AVAILABILITY, FAULT TOLERANCE, AND CHAOS

OPERATIONAL EXCELLENCE
OBSERVABILITY, AUTOMATION, AND CONTINUOUS IMPROVEMENT

CONNOR BRENDON

The Scalable SaaS Architecture Handbook

Designing Systems That Grow Without Breaking

Steve Publications

This book is available at
https://leanpub.com/thescalablesaasarchitecturehandbook

This version was published on 2026-07-18

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Designing Systems That Grow Without Breaking

This book is a practical, authoritative guide to architecting Software-as-a-Service applications that scale gracefully under real-world load. It covers the full spectrum of architectural decisions: from choosing between monoliths and microservices, through event-driven design and database sharding, to cloud-native infrastructure, multi-tenancy patterns, resilience engineering, and operational excellence. Every chapter is grounded in real systems, real numbers, and hard-won lessons from companies that have lived through the transition from startup-scale to enterprise-scale. The reader will finish with a clear mental model for evaluating any architectural decision through the lens of long-term scalability.

[FIGURE: Book cover illustration showing layered architecture diagram – from infrastructure layer at bottom through data, services, and API layers, with scaling arrows pointing outward. Style: clean technical line art.]

Introduction: The Scalability Imperative

In the summer of 2008, a single database corruption incident took Netflix offline for three days. Customers could not manage their DVD queues. The company could not ship physical discs. Every function of the business was paralyzed because everything depended on one monolithic Oracle database running in a private data center [1]. That outage was not a product failure or a marketing misstep. It was an architecture problem, and it would become the catalyst for one of the most consequential system transformations in modern software history.

Over the next seven years, Netflix migrated from that monolith to more than 700 independently deployable microservices running on Amazon Web Services [2]. The transformation was not easy, and it was not universally successful at every step. But by the time it completed in January 2016, Netflix had built a system capable of serving hundreds of millions of concurrent streams across dozens of countries without the kind of catastrophic single-point-of-failure that nearly destroyed the company a few years earlier. Subscribers grew from 9.4 million at the time of the outage to approximately 89 million by migration's end [3].

Netflix is an extreme case. Most SaaS companies will never need 700 microservices. But the lesson from their journey applies to every software team that plans to grow: your architecture will either enable your growth or become its bottleneck, and the decisions you make today compound in ways that are expensive, sometimes prohibitively so, to reverse later.

[FIGURE: Netflix migration timeline – horizontal timeline from 2008 to 2016 showing key milestones: Aug 2008 DB corruption incident, 2009 cloud pathfinders begin, 2010-2012 stateless decomposition, 2012-2014 data tier migration to Cassandra/S3, 2015 multi-region active-active and chaos engineering formalized, Jan 2016 final service (Billing) migrated and private data centers shut down. Include subscriber growth curve overlaid.]

Why SaaS Is Different

Scaling a SaaS application is fundamentally different from scaling any other kind of software. There are three reasons for this.

First, SaaS systems serve multiple customers simultaneously on shared infrastructure. This multi-tenancy requirement means that performance problems for one customer can cascade to every other customer. A runaway query from a single large tenant can exhaust database connections and degrade the platform for everyone. Your architecture must include isolation mechanisms that prevent any one tenant from bringing down the whole system.

Second, SaaS products run continuously. Unlike a batch processing system or an internal tool that operates during business hours, a SaaS platform is always on. Downtime is not just inconvenient; it is a direct revenue loss and a trust violation. Your architecture must support continuous deployment without downtime, graceful degradation under failure conditions, and recovery procedures that work when you are sleep-deprived at 3 a.m.

Third, SaaS economics tie architecture directly to unit economics. Every server instance, every database connection, every gigabyte of egress bandwidth has a cost that scales with your user base. An architecture that works for 100 customers may become economically unviable at 10,000 customers if its resource consumption does not scale linearly with revenue. Cloud cost optimization is not a separate concern from architecture; it is an architectural constraint. McKinsey estimates the potential value from "FinOps as code" at approximately $120 billion, based on expected global cloud IaaS and PaaS spending of roughly $440 billion [18].

What This Book Covers

This book takes you through the full lifecycle of architectural decisions for a growing SaaS platform. We start with foundations: the metrics that actually matter, the frameworks for thinking about scalability, and the economics of architectural trade-offs. We then move through system design choices in the order you would typically encounter them as a company grows.

Chapters 1 through 3 cover the fundamental structural decisions: when to keep a monolith, when to decompose into microservices, and how to do it

correctly using domain-driven design principles. Chapters 4 through 5 focus on the data layer: event-driven architecture patterns for decoupling services, and database strategies including sharding, replication, and polyglot persistence. Chapter 6 tackles multi-tenancy specifically, since it is the defining characteristic of SaaS architecture.

Chapters 7 through 8 address infrastructure and resilience: cloud-native deployment patterns, Kubernetes best practices, serverless trade-offs, and the resilience patterns that keep systems running when things go wrong (and things always go wrong). Chapters 9 through 10 cover performance tuning and security, two domains where architectural decisions made early save enormous amounts of rework later. Chapter 11 focuses on the operational playbook: CI/CD pipelines, deployment strategies, and SRE practices adapted for SaaS. The final chapter catalogs the most common architectural anti-patterns and how to avoid or fix them.

Every chapter includes real-world case studies with concrete metrics, implementation artifacts (configuration snippets, architecture decision records, cost comparison tables), and step-by-step guidance you can apply immediately.

How to Read This Book

Read the chapters in order if you are building a new system or planning a major architectural evolution. Each chapter builds on concepts introduced earlier, and the progression mirrors the typical growth trajectory of a SaaS company. If you are looking for specific guidance on a current problem, use the chapter headings as a navigation guide. The book is designed so that each chapter can stand alone as a reference while contributing to the larger narrative.

Every claim in this book is grounded in real systems, published research, or well-established engineering principles. Where there are disagreements in the community, I surface them and explain my reasoning for a particular recommendation. Scalability is not about following a checklist; it is about understanding trade-offs deeply enough to make the right choice for your specific context.

Let us begin.

Chapter 1: Foundations of SaaS Architecture

Before you choose a database, decompose a service, or configure an auto-scaling group, you need to understand the foundations that make SaaS architecture distinct from every other type of software engineering. These foundations are not abstract principles; they are concrete constraints that shape every decision you will make in this book.

What Makes SaaS Different

SaaS applications share three characteristics that do not apply to most other software systems. Understanding these characteristics is the first step in making architectural decisions that actually serve your business model.

Multi-tenancy as a core constraint. In SaaS, multiple customers (tenants) share the same application instance and infrastructure. This is what makes SaaS economically viable: you spread fixed costs across thousands of tenants rather than provisioning dedicated resources for each one. But multi-tenancy introduces a class of problems that simply does not exist in single-tenant software. A runaway query from one tenant can exhaust database connections for everyone. A data isolation bug can expose Customer A's records to Customer B, which is not just a technical problem but a legal and reputational disaster. Your architecture must enforce tenant boundaries at every layer: the application logic, the database, the cache, and the network.

Continuous delivery as a product feature. SaaS products are updated continuously, often multiple times per day. Unlike desktop software where updates happen on a user's schedule, SaaS updates affect all tenants simultaneously. This means your deployment architecture must support zero-downtime releases, instant rollbacks when something goes wrong, and gradual rollouts that limit blast radius. The ability to ship changes safely and frequently is not an operational concern; it is a competitive advantage. Companies that can deploy daily outpace companies that deploy monthly in every measurable business metric.

SLAs as product features. In SaaS, your service level agreements are part of your product offering. Enterprise customers evaluate your uptime guarantees, latency commitments, and data durability promises before they sign a contract. These are not aspirational targets; they are contractual obligations with financial penalties for non-compliance. Your architecture must be designed to meet these commitments consistently, not just on average. A 99.9% availability SLA means you can afford 43.8 minutes of downtime per month. A 99.99% SLA reduces that to 4.4 minutes. The difference in architectural complexity between these two targets is enormous.

The Four Pillars Framework

Every SaaS architecture decision can be evaluated against four pillars. These are not independent objectives; they are competing priorities that you must balance deliberately.

Scalability is the ability of your system to handle growing load without proportional increases in cost or complexity. Horizontal scaling (adding more instances) is generally preferable to vertical scaling (buying bigger machines) because it provides elasticity and fault tolerance. But horizontal scaling introduces its own challenges: state management, data distribution, and network coordination. A truly scalable architecture minimizes shared state, routes requests efficiently, and allows individual components to scale independently.

Reliability is the ability of your system to continue functioning correctly in the face of failure. This means designing for the reality that every component will fail: servers crash, networks partition, databases corrupt data, and humans make mistakes. Reliable systems use redundancy (multiple copies of critical components), isolation (preventing failures from cascading), and automated recovery (detecting and fixing problems without human intervention). The Google Site Reliability Engineering team codifies this through the concept of error budgets: the amount of unreliability your system is allowed to accumulate before reliability work takes priority over feature development [2].

Performance is the speed at which your system responds to user requests. Performance matters because it directly affects user experience, conversion rates, and infrastructure costs (faster responses mean fewer concurrent connections needed). The Google SRE book identifies four "golden signals" for monitoring distributed systems: latency (time to serve a request), traffic

(demand on the system), errors (rate of failures), and saturation (how full your resource pool is) [1]. These signals form the basis of every performance measurement in this book.

Critically, latency must be measured in percentiles, not averages. If you run a web service with an average latency of 100 milliseconds at 1,000 requests per second, 1% of requests might easily take 5 seconds. Those tail latencies are what your users remember and what determine whether you meet your SLA commitments. The P50 (median), P95, and P99 percentiles each tell a different story about system behavior, and all three should be monitored continuously.

Cost efficiency is the relationship between your infrastructure spend and the business value it delivers. In SaaS, infrastructure cost scales with revenue, so unit economics are everything. A SaaS company spending $0.50 in infrastructure to deliver $10.00 in monthly recurring revenue has a dramatically different growth trajectory than one spending $3.00 to deliver the same revenue. Cost efficiency is not about cutting corners; it is about right-sizing resources, eliminating waste through automation, and choosing architectural patterns that scale economically. The FinOps Foundation identifies that organizations with proactive budget controls experience approximately 35% better cost outcomes compared to those without [17].

These four pillars are in constant tension. Adding redundancy improves reliability but increases cost. Caching improves performance but introduces consistency challenges. Microservices improve scalability but add operational overhead. The art of SaaS architecture is finding the right balance for your specific context at each stage of growth.

[FIGURE: Four pillars trade-off diagram – a four-quadrant matrix showing Scalability, Reliability, Performance, and Cost Efficiency as axes. Each quadrant shows examples of trade-offs: e.g., “Adding replicas improves reliability but increases cost” in the Reliability/Cost quadrant. Include arrows showing directional tensions between pillars.]

Architecture Decision Records: Capturing Your Reasoning

One of the most undervalued practices in architectural work is documenting your decisions. Architecture Decision Records (ADRs) are lightweight documents that capture the context, options considered, and rationale behind

each significant architectural choice. They prevent teams from re-litigating old decisions and provide onboarding material for new engineers.

ADR Template:

```
# ADR-001: Choose PostgreSQL as Primary Database

**Status:** Accepted
**Date:** 2024-03-15
**Context:** We need a primary database for our SaaS platform. Requirements include:
  - ACID transactions for financial data
  - JSON support for flexible tenant configurations
  - Full-text search capability
  - Active open-source community and hiring pool

**Options Considered:**
1. PostgreSQL -- strong on all requirements, mature ecosystem
2. MySQL -- simpler but weaker JSON and full-text support
3. MongoDB -- flexible schema but weaker transaction guarantees
4. CockroachDB -- distributed by design but newer, smaller talent pool

**Decision:** PostgreSQL 15+ with pgvector extension for future AI features.
**Consequences:**
  - Positive: Best-in-class ACID compliance, JSONB performance rivals document stores
  - Negative: Sharding requires third-party tools (Citus) or custom implementation
  - Mitigation: Plan for Citus migration at 50M rows per largest table
```

Maintain ADRs in your codebase (e.g., `docs/adr/` directory) and treat them as living documents. Reference them when discussing architectural changes, and update their status when decisions evolve.

Measuring What Matters

You cannot manage what you do not measure, and most SaaS teams measure the wrong things. Here are the metrics that actually drive architectural decisions:

Latency percentiles. Track P50, P95, and P99 for every critical user-facing operation. Set SLOs (Service Level Objectives) on each percentile. The P99 tells you about your worst-case user experience; if it is unacceptable, your architecture has a problem regardless of how good the average looks.

Error rates. Track the percentage of requests that fail, broken down by error type (5xx server errors, timeouts, circuit breaker activations). A rising error rate is often the first sign of an architectural problem, appearing days or weeks before it becomes a user-visible issue.

Throughput and capacity headroom. Know your current request volume and your maximum sustainable throughput. The gap between them is your capacity headroom, and it determines how much growth you can absorb without architectural changes.

Error budgets. Derived from your SLOs, error budgets quantify how much unreliability you can tolerate before reliability work becomes mandatory. If your SLO is 99.9% availability, your monthly error budget is 43.8 minutes of downtime. When you burn through that budget, feature development pauses until reliability is restored. This creates a disciplined trade-off between velocity and stability.

Unit economics. Track infrastructure cost per tenant, per active user, or per API call. This metric tells you whether your architecture scales economically. If cost per unit increases as you grow, you have an architectural problem that will worsen over time.

The Cost of Premature Optimization vs the Cost of Delayed Architecture

Every SaaS team faces two opposing risks: spending too much on architecture before you need it, and spending too little until it is too late to fix affordably.

Premature optimization is the tendency to build complex distributed systems for workloads that a simple monolith could handle. The classic example is a startup with 100 users deploying 50 microservices on Kubernetes. The operational overhead of such a system dwarfs any performance benefit, and the team spends its time managing infrastructure instead of building product features. Martin Fowler puts it bluntly: "Don't even consider microservices unless you have a system that's too complex to manage as a monolith."

Delayed architecture is the opposite failure: ignoring architectural debt until the system becomes so entangled that any change requires a complete rewrite. This typically happens when teams prioritize feature velocity for too long without investing in modularity, observability, or automated testing. The

result is a system where every deployment is risky, performance degrades unpredictably, and new engineers take months to become productive.

The sweet spot lies between these extremes: build the simplest architecture that can handle your current load with reasonable headroom, but invest continuously in architectural hygiene. Modularize your codebase even while it runs as a single deployable unit. Implement observability from day one. Write automated tests that cover critical paths. These investments are cheap when the system is small and become exponentially more expensive as it grows.

A useful rule of thumb: plan your architecture for 10x your current load, but build only for 2x. This gives you enough headroom to grow without constant architectural changes while avoiding the waste of provisioning for scenarios that may never materialize. The key is knowing when 2x becomes insufficient and triggering architectural evolution proactively rather than reactively.

The chapters that follow will give you the specific patterns, tools, and decision frameworks to navigate this balance throughout your SaaS platform's lifecycle.

Chapter 2: Monoliths and When to Break Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

The Case for Starting Monolithic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Signs Your Monolith Has Outgrown Its Purpose

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Practical Decomposition Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Domain-Driven Design as a Decomposition Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Strangler Fig Pattern in Practice

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Case Study: Netflix's Seven-Year Migration (2008-2016)

Chapter 3: Microservices Architecture at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Service Boundaries and the Bounded Context Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Synchronous vs Asynchronous Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

API Gateway Patterns and Edge Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Service Discovery and Configuration Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Observability as a Non-Negotiable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 4: Event-Driven Architecture Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

The Event Publishing Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Message Brokers Compared

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Guaranteeing Delivery Without Losing Sanity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Handling Out-of-Order Events and Idempotency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Saga Pattern for Distributed Transactions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 5: Database Architecture for Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Relational vs NoSQL: The Real Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Read Replicas and Write-through Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Database Sharding: Consistent Hashing, Range-Based, and Directory-Based

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Polyglot Persistence Done Right

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Migration Strategies for Evolving Schema Under Load

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 6: Multi-Tenancy Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Tenant Isolation Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Data Partitioning by Tenant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Quota Management and Noisy Neighbor Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Tenant Onboarding as a Scalability Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Compliance and Data Sovereignty Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 7: Cloud-Native Infrastructure Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Container Orchestration at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Serverless for SaaS: When It Works, When It Does Not

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Infrastructure as Code and GitOps Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Multi-Region Deployment Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Cost Optimization Without Sacrificing Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 8: Resilience Patterns and Failure Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Circuit Breakers, Bulkheads, and Rate Limiting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Retry Strategies with Exponential Backoff and Jitter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chaos Engineering as a Discipline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Disaster Recovery: RTO, RPO, and Actual Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Post-Incident Reviews That Actually Improve Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 9: Performance Tuning at Every Layer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Profiling Before Optimizing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Connection Pooling and Resource Exhaustion Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Database Query Optimization at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Load Testing That Actually Reveals Bottlenecks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 10: Security Architecture for SaaS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Zero Trust in Multi-Tenant Environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Authentication and Authorization at Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Secrets Management Across Distributed Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Data Encryption Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Compliance Automation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 11: The Operational Playbook

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

CI/CD Pipelines for Microservices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Blue-Green and Canary Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

On-Call Culture and Alert Fatigue Prevention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

SRE Practices Adapted for SaaS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Chapter 12: Common Pitfalls and Anti-Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Distributed Monoliths (Microservices Done Wrong)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Overengineering Before You Have Users

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

The Configuration Sprawl Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Ignoring Data Migration Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Vendor Lock-in Without a Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

The Observability Gap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Summary: The Anti-Pattern Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Conclusion: Architecting for the Next Decade

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

The Four Pillars: A Synthesis Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

Emerging Trends Through the Four-Pillar Lens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

A Forward-Looking Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/thescalablesaasarchitecturehandbook.