

Event-Driven Systems at the Edge

Streams, Queues and
Distributed State



DAVID ASHCROFT

Event-Driven Systems at the Edge

Streams, Queues and Distributed State

Steve Publications

This book is available at

<https://leanpub.com/event-drivensystemsattheedge>

This version was published on 2026-10-13



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.

© 2026 Steve Publications

Contents

Streams, Queues and Distributed State	1
Introduction	2
What You Will Learn	2
The Reference Application: EdgeFleet	3
Technology Stack	4
How to Use This Book	6
What This Book Does Not Cover	6
Chapter 1: Why Event-Driven? The Promise and the Peril	8
The Connected World at Scale	8
From Monolith to Events	9
The Cost of Getting It Wrong	11
What This Book Will Build	13
How to Read This Book	14
Chapter 2: Event-Driven Architecture Foundations	16
Events, Messages and Commands	16
The Producer-Consumer Model	19
Point-to-Point Versus Publish-Subscribe	21
Coupling, Cohesion and Autonomy	23
Anti-Patterns and Common Mistakes	24
Chapter 3: Message Brokers, Queues and Streams	28
Queues: Load Leveling and Work Distribution	28
Topic-Based Pub/Sub: Broadcasting and Filtering	28
Event Streams: Ordered, Persistent, Replayable	29
Broker Architectures: From RabbitMQ to Kafka to NATS	29
Choosing a Messaging Primitive	30
Chapter 4: Kafka Internals and Architecture	31

CONTENTS

Topics, Partitions and Offsets	31
The Broker and Controller	31
Replication and ISR	32
Producer Semantics and Batched Writes	33
Consumer Groups and Offset Management	33
Log Compaction and Retention	34
Summary	35
Chapter 5: NATS, Pulsar and Cloud-Native Alternatives	36
NATS and JetStream: Simplicity and Speed	36
Apache Pulsar: Multi-Tenant Cloud Scale	36
Managed Cloud Services: Kinesis, Pub/Sub, Event Hubs	37
Brokerless and Mesh Architectures	38
Platform Selection Matrix	38
Summary	38
Chapter 6: Events, Schemas and Serialization	39
Designing Domain Events	39
Serialization Formats: JSON, Protobuf, Avro	40
Schema Registry and Evolution Rules	40
Backward and Forward Compatibility	41
Summary	42
Chapter 7: Producers - Publishing Events Reliably	43
Producer Design Patterns	43
Kafka Producer Configuration	43
Idempotent Producers	44
Ordering Guarantees and Key-Based Routing	45
Reference application: EdgeFleet producers	46
Error Handling and Retries	46
Summary	47
Chapter 8: Consumers - Processing Events Correctly	48
Consumer Group Dynamics	48
At-Least-Once and At-Most-Once Consumption	48
Idempotent Consumers and Deduplication	49
Backpressure and Graceful Degradation	49
Reference Application: EdgeFleet Consumers	50
Summary	50

Chapter 9: Stream Processing - Transformations and Joins	51
Map, Filter and Aggregate	51
Tumbling, Sliding and Session Windows	51
Stream-Stream and Stream-Table Joins	52
Stateful Processing and Checkpoints	53
Reference Application: The Stream Processor	53
Chapter 10: Kafka Streams and Flink	54
Kafka Streams: Embedded and Lightweight	54
Apache Flink: Batch-and-Stream Unified	54
State Stores and RocksDB	55
Fault Tolerance and Recovery	56
When to Use Which	56
Summary	56
Chapter 11: Event Sourcing and CQRS	58
Command Query Responsibility Segregation	58
Event Sourcing: Appending the Truth	58
Sagas and Distributed Transactions	59
Read Models and Projections	60
Reference Application: Event-Sourced Domain	60
Chapter 12: Distributed State Management	62
Stateful Services in a Stateless World	62
Distributed Caches and KV Stores	62
Eventual Consistency and Conflict Resolution	63
Partitioning State and Hot Keys	63
Reference Application: Distributed State Store	64
Summary	64
Chapter 13: Edge Computing Fundamentals	65
Cloud, Fog and Edge: A Continuum	65
Latency, Bandwidth and Autonomy	65
Edge Runtimes: Containers, Functions and WASM	66
Edge Deployment Models	66
Reference Application: The Edge Topology	67
Summary	67
Chapter 14: Serverless Event Processing	68
Serverless Execution Models	68

CONTENTS

Cloud Functions: AWS Lambda, Cloudflare Workers, Azure Functions	68
Edge Functions and CDNs	69
Cold Starts, Limits and Workarounds	69
Reference Application: Serverless Handlers	70
Wrapping up	70
Chapter 15: Cloud-to-Edge Communication	71
Edge-to-Cloud Patterns	71
Store-and-Forward for Intermittent Connectivity	71
Delta Sync and Operational Transformations	72
Bi-Directional Event Routing	73
Reference Application: Edge-to-Cloud Pipeline	73
Summary	74
Chapter 16: Delivery Guarantees, Deep Dive	75
At-Most-Once, At-Least-Once, Exactly-Once	75
The Two-Phase Commit Problem	75
Transactional Outbox Pattern	76
Idempotent Operations: Making At-Least-Once Safe	77
Reference Application: Transactional Messaging	77
Summary	77
Chapter 17: Consistency, Ordering and Partitioning	78
CAP and the PACELC Theorem	78
Linearizability Versus Sequential Consistency	78
Partition Key Design and Hot Spots	79
Compensating for Disordered Events	80
Reference Application: Consistency Choices	80
Summary	80
Chapter 18: Fault Tolerance and Recovery	81
Failure Modes in Distributed Systems	81
Retries, Dead Letter Queues and Alerting	81
Circuit Breakers and Bulkheads	82
Leader Election and Raft	83
Reference Application: Failure Handling	83
Summary	83
Chapter 19: Observability and Operations	84
Logging: Structure, Correlation and Volume	84

Metrics: Red Signals and SLOs	84
Distributed Tracing End to End	85
Capacity Planning and Cost Control	86
Reference Application: Observability Stack	86
Summary	86
Chapter 20: Security, Secrets and Multi-Tenancy	87
Authentication and Authorization Models	87
TLS, mTLS and Encryption at Rest	87
Secret Management Patterns	88
Multi-Tenant Isolation and Quotas	88
Reference Application: Security Layer	89
Summary	89
Chapter 21: CI/CD, IaC and Deployment Automation	90
Infrastructure as Code with Terraform	90
Containerization and Kubernetes	90
GitOps and Progressive Delivery	91
Schema and Migration Pipelines	91
Reference Application: Complete Deployment	92
Summary	92
Chapter 22: Production Hardening and the Full System	93
The Complete Reference Architecture	93
Load Testing and Performance Validation	94
Runbooks and On-Call	94
Migrating from Monolith to Event-Driven	95
What Comes Next	95
Conclusion	96
References	97

Streams, Queues and Distributed State

This book shows you how to design, build, deploy and operate production-grade event-driven applications that span cloud data centers and edge nodes. You will learn the architecture and internals of modern streaming platforms, how to guarantee message delivery under failure conditions, how to manage distributed state correctly and how to stitch all of this into a single coherent system. A complete reference application runs throughout the chapters, with working code, infrastructure definitions and deployment instructions you can adapt to your own projects.

Introduction

A logistics company runs a fleet of delivery trucks across three continents. Every vehicle streams telemetry events every second: GPS coordinates, fuel levels, engine diagnostics, driver behavior metrics. The operations team in the cloud must correlate this data in real time, detect anomalies, route drivers around traffic, maintain an authoritative view of each truck state and generate alerts when something goes wrong. At the same time, each truck operates in environments with intermittent connectivity, where the network may drop for minutes or hours at a time. The system must tolerate all of this without losing data, duplicating events, or stalling operations.

This is what modern event-driven systems must do. They must span from constrained edge devices to powerful cloud data centers, handle millions of events per second, maintain consistency across distributed services and survive failures gracefully. They are the backbone of everything from autonomous vehicles to fintech platforms to real-time analytics engines.

Building them correctly is hard. The hardest part is not choosing the right tools: Kafka or Pulsar, Flink or Kafka Streams, Kubernetes or a managed service. The hardest part is understanding the distributed systems principles underneath: what exactly at-least-once delivery means when a consumer crashes mid-processing, why ordering guarantees are always partial, how to design idempotent operations that survive retries, how to manage state when every node might fail at any moment. Without this understanding, teams build systems that appear to work in development and then break in production under load, during network partitions, or when someone forgets to handle one edge case.

This book exists to bridge that gap. It treats event-driven architecture as a serious engineering discipline, not a buzzword. Every chapter connects architectural concepts to concrete implementation details, every pattern comes with working code and every recommendation explains the trade-offs behind it.

What You Will Learn

By the end of this book, you will understand how to:

- Design event-driven architectures using the right messaging primitives for each use case: queues, topics, streams and request-reply patterns.
- Select and configure streaming platforms based on your throughput, latency, durability and operational requirements. This includes Apache Kafka, NATS JetStream, Apache Pulsar and managed cloud services.
- Implement reliable event producers that never lose events and never create duplicates beyond what your system can tolerate.
- Build idempotent consumers that handle retries, duplicate messages and out-of-order delivery correctly.
- Design stream processing pipelines that perform windowed aggregations, joins and stateful computations with exactly-once semantics where possible and effectively-once semantics where that is the practical reality.
- Apply event sourcing and CQRS patterns to maintain an authoritative event log as the source of truth for complex domains.
- Manage distributed state across services using consistent designs: local databases with outbox patterns, replicated key-value stores and event-driven state machines.
- Deploy event-driven systems at the edge, handling intermittent connectivity, local buffering, store-and-forward patterns and bi-directional synchronization.
- Run and operate production systems with proper observability: structured logging, metrics, distributed tracing, alerting and runbooks.
- Secure the entire stack with authentication, authorization, encryption in transit and at rest and proper secret management.
- Automate infrastructure with Terraform, Kubernetes and CI/CD pipelines that deploy both the streaming platform and the applications that depend on it.

The Reference Application: EdgeFleet

A book about event-driven systems needs more than abstract examples. Throughout this book, we will build a complete reference application called

EdgeFleet, an IoT fleet telemetry and analytics platform. EdgeFleet models a realistic, non-trivial system that touches every topic we will cover.

EdgeFleet consists of the following components:

- **Edge sensors:** Simulated devices running on edge nodes that generate telemetry events (vehicle position, engine diagnostics, fuel levels) and publish them via MQTT to a local NATS JetStream instance. The edge nodes handle buffering when the connection to the cloud is lost.
- **Edge-to-cloud bridge:** A service running at the edge that subscribes to local events, batches them and forwards them to Kafka in the cloud. It handles reconnection, backpressure and dead lettering.
- **Cloud streaming platform:** An Apache Kafka cluster running on Kubernetes (deployed with Strimzi). Kafka topics store raw telemetry, enriched events and domain events. A Schema Registry validates Protobuf schemas.
- **Stream processor:** A Kafka Streams application that enriches raw telemetry with vehicle metadata, computes rolling statistics (average speed, fuel efficiency), detects anomalies and publishes domain events.
- **Fleet service:** An event-sourced microservice that maintains the current state of each vehicle by consuming domain events from Kafka. It uses a PostgreSQL database for the event store and read models, publishes domain events via a transactional outbox and handles saga-based distributed transactions for operations like vehicle assignment.
- **Alert service:** A serverless function (AWS Lambda or Cloudflare Worker) triggered by anomaly events that evaluates alerting rules and sends notifications.
- **Query API:** A REST and gRPC service that serves vehicle state, historical telemetry and analytics from read models optimized for queries.
- **Observability stack:** OpenTelemetry instrumentation across all services, metrics exported to Prometheus, traces collected in Jaeger, logs aggregated with Loki and dashboards in Grafana.

All code, configuration files and deployment manifests for EdgeFleet are provided in this book. You can run it locally with Docker Compose, deploy it to a Kubernetes cluster and extend it to edge nodes. The code is designed to be readable and educational, not production-ready copy-paste, but every pattern it demonstrates is production-proven.

Technology Stack

EdgeFleet uses a modern, coherent technology stack. Here is the rationale for each choice:

- **Apache Kafka** for the cloud streaming backbone. Kafka is the most mature and widely adopted event streaming platform, with excellent tooling, strong exactly-once semantics support and a vast ecosystem. We use Kafka 3.9+ with KRaft mode (ZooKeeper-less).
- **NATS JetStream** for edge messaging. NATS is lighter than Kafka, simpler to operate and designed for cloud/edge environments. JetStream adds persistence and at-least-once delivery on top of NATS. We use NATS for edge-to-edge communication and for bridging to Kafka.
- **Kafka Streams** for stream processing. Kafka Streams is embedded, scales linearly with Kafka partitions, supports exactly-once processing and avoids the operational overhead of a separate Flink cluster for many use cases. We cover Flink as an alternative for more complex stream processing needs.
- **Protocol Buffers** for event serialization. Protobuf provides a binary format that is small, fast and strongly typed, with robust schema evolution support via the Protobuf 3 language and Schema Registry.
- **PostgreSQL** for relational data and event storage. PostgreSQL is reliable, widely supported and integrates well with Debezium for change data capture. We use it for event sourcing, read models and application state.
- **gRPC** for service-to-service communication. gRPC provides strongly-typed APIs with low latency and integrates naturally with Protobuf schemas.
- **Go and TypeScript** for implementation. We use Go for core services that need high performance (producers, consumers, stream processors) and TypeScript for serverless functions and edge code. All code examples are complete and runnable.
- **Terraform** for infrastructure as code. Terraform provisions cloud resources, Kubernetes clusters and Kafka operators.
- **Kubernetes** for deployment and orchestration. Kubernetes manages containerized services, auto-scaling and rolling updates. Strimzi operator manages Kafka.

This stack is opinionated but defensible. Every choice reflects real production trade-offs, not theoretical preferences. Where alternatives exist, we explain why and when they might be better.

How to Use This Book

This book is organized from fundamentals to production deployment. Chapter 1 explains why event-driven architectures matter and where they fail. Chapters 2 through 5 cover the messaging primitives and platforms you can use. Chapters 6 through 12 focus on building correct, reliable event-driven systems: serialization, producers, consumers, stream processing, event sourcing and distributed state. Chapters 13 through 15 extend the architecture to edge environments. Chapters 16 through 18 dive into the distributed systems theory that governs delivery guarantees, consistency and fault tolerance. Chapters 19 through 21 address production concerns: observability, security and deployment automation. Chapter 22 ties everything together with the complete EdgeFleet architecture, operational practices and migration strategies.

You can read this book sequentially if you want a comprehensive foundation, or jump to chapters relevant to your immediate needs. The reference application code builds progressively; each chapter adds or modifies components as new concepts are introduced.

What This Book Does Not Cover

This book is narrowly focused on event-driven systems and their surrounding ecosystem. It does not cover:

- Microservices architecture in general, beyond what is necessary for event-driven design.
- Domain-driven design as a methodology, though we apply DDD concepts where they inform event design.
- Big data analytics platforms like Spark or Hadoop, except where they interface with streaming systems.
- Machine learning and AI/ML pipelines, except as consumers of streaming data.
- Specific cloud provider quirks beyond the patterns that apply universally.

If you need to understand event-driven architecture deeply, with working code and real engineering trade-offs, this book is for you. Let us begin.

Chapter 1: Why Event-Driven? The Promise and the Peril

The Connected World at Scale

In 2023, the average enterprise generated approximately 300 terabytes of new data per day. By 2026, that number has only grown. More sensors are deployed, more devices are online, more user interactions are tracked in real time. Traditional request-response architectures, where a client sends a request and waits for a response, cannot handle this volume without becoming unbearably slow, tightly coupled, or both.

Event-driven architecture emerged not as a fashion but as a necessity. It decouples producers of information from consumers of that information, allowing each to scale independently. Instead of waiting for a response, a producer publishes an event and moves on. Consumers subscribe to the events they care about and process them at their own pace. This simple shift unlocks scalability, resilience and flexibility that are impossible to achieve with synchronous communication alone.

Consider a payment processing system. When a user submits a payment, many things need to happen: charge the card, update the order status, send a confirmation email, record the transaction in analytics, trigger fraud detection, update inventory. In a monolithic request-response design, the payment service must call each of these operations synchronously or queue them in a single shared database. The entire chain is slowest link: if the email service is down, the payment times out. If the analytics service is slow, the user waits.

In an event-driven design, the payment service processes the charge, publishes a `PaymentCompleted` event and returns success to the user within milliseconds. The email service, analytics pipeline, fraud detection and inventory update each consume that event independently. If one of them is slow or down, the others keep working. The user experience does not degrade because a downstream analytics pipeline is behind.

This is the promise of event-driven architecture: systems that are scalable because components grow independently, resilient because failures are isolated and flexible because new consumers can be added without changing existing producers.

But there is also a peril. Event-driven systems are inherently distributed and distributed systems are fundamentally hard to reason about. Messages can be delivered out of order, duplicates can appear, consumers can crash mid-processing, network partitions can split your system into inconsistent views. The complexity does not disappear; it moves from the business logic into the plumbing. Teams that adopt event-driven architecture without understanding this plumbing build systems that appear correct in development and then fail in production in ways that are difficult to diagnose.

The rest of this book exists to ensure you understand the plumbing well enough to build systems that work. But first, let us look at how we got here.

From Monolith to Events

The history of event-driven architecture is intertwined with the history of software architecture itself. Early enterprise applications were monoliths: a single process that handled all business logic, stored all data in a single database and communicated through synchronous procedure calls. This design is simple to reason about. A change to the business logic affects one place. Transactions span the entire database. Debugging means looking at one codebase.

But monoliths do not scale well when the system grows beyond a single team or a single workload profile. When one part of the system needs to scale differently from another, when teams want to deploy independently, when different components have different reliability requirements, the monolith becomes a bottleneck.

The microservices movement addressed this by splitting the monolith into smaller, independently deployable services, each owning its own data. This solved the organizational and scalability problems but introduced a new challenge: how do services communicate?

Early microservice architectures relied heavily on REST APIs and synchronous HTTP calls. This approach works for simple request-response interactions but breaks down for complex workflows that span multiple services. Consider an e-commerce checkout: the order service calls the payment

service, which calls the fraud service, which calls the risk engine. If any call fails or times out, the entire chain fails. Under load, cascading failures spread through the system. The synchronous graph becomes a dependency graph that determines system reliability: the whole is only as reliable as its least reliable component.

Event-driven architecture offers a different model. Instead of service A calling service B directly, service A publishes an event describing what happened and service B consumes that event when it is ready. The services are no longer directly coupled. Service B does not need to know how to reach service A, or even that service A exists. It only needs to understand the events it consumes.

This decoupling is not new. Message-oriented middleware like IBM MQ and TIBCO Rendezvous existed in the 1990s. Modern event-driven architecture differs in three important ways:

1. **Event streams instead of transient messages:** Older message systems were fire-and-forget. A message was delivered once and then gone. Modern streaming platforms like Kafka retain events in a durable log, allowing consumers to replay history, rebuild state and process events at different speeds. This turns events from a communication mechanism into a primary data store.
2. **Many consumers, independent consumption:** Traditional pub/sub systems often treated subscribers as peers, delivering the same message to each. Modern systems support consumer groups where work is partitioned among consumers and each group maintains its own independent view of the stream. This allows you to have one group of consumers processing payments for operational purposes, another building analytics models and a third archiving data for compliance, all reading from the same event stream without interfering with each other.
3. **Tighter integration with stream processing:** Frameworks like Kafka Streams, Flink and Spark Structured Streaming make it possible to perform complex computations directly on event streams: aggregations, joins, windowed computations, pattern detection. Events are not just stored and forwarded; they are transformed, enriched and analyzed in real time.

The transition from monolith to microservices to event-driven architecture is not a straight line. Many organizations run hybrid systems: mono-

lithic cores with microservice extensions, REST APIs alongside event streams, synchronous calls for user-facing interactions and asynchronous events for backend processing. The goal is not purity but correctness: choosing the right communication model for each interaction based on latency requirements, reliability requirements and coupling tolerance.

The Cost of Getting It Wrong

Event-driven architecture is powerful precisely because it is complex. The distributed nature of the system introduces failure modes that simply do not exist in a monolith. Understanding these failure modes is the difference between a system that works reliably and one that appears to work until it catastrophically fails.

Let us look at some concrete examples of how event-driven systems go wrong, based on real-world incidents and common anti-patterns.

Lost events

A production system publishes an `OrderCreated` event to Kafka and then crashes before the message is acknowledged by the broker. The consumer never sees the event. The order is placed in the database but never shipped. The customer complains, the operation team investigates, finds the missing event and manually replays it. This happens again and again.

The root cause: the producer used `acks=0` configuration, which means the producer does not wait for broker acknowledgment before considering the send successful. When the broker loses power or crashes before persisting the message, the event is gone. This is a straightforward configuration error but surprisingly common in production systems where latency concerns override durability.

Duplicate processing

A payment service consumes a `PaymentAuthorized` event and charges the customer. Due to a network glitch, the consumer does not send its acknowledgment back to Kafka in time. The broker reassigns the partition to another consumer instance, which reprocesses the same event and charges the

customer a second time. The first consumer eventually recovers and processes the event as well, charging a third time. The customer is charged three times for a single purchase.

The root cause: the consumer is not idempotent. It assumes each event is processed exactly once, but at-least-once delivery means events may be delivered multiple times. The fix requires the consumer to check whether it has already processed an event (using a deduplication table or unique constraint) before applying its effects.

Out-of-order processing

A telemetry pipeline processes vehicle events by partitioning on vehicle ID. Under normal conditions, events for each vehicle arrive in order. But during a broker failover, the new leader serves some events from a replica that was slightly behind. Events arrive at the consumer in a different order than they were produced. The consumer updates vehicle state sequentially, resulting in an incorrect current position because an old event overwrote a newer one.

The root cause: the system assumed total ordering or stable ordering, but in a distributed system, ordering guarantees are always partial. The fix requires the consumer to handle out-of-order events, for example by checking event timestamps and rejecting events older than the current processed event for a given entity.

Consumer lag spiral

An analytics consumer falls behind during a traffic spike. Its processing time per event increases because each event requires a database query that becomes slower under load. The consumer lag grows. More events accumulate. The database becomes more strained. The consumer lag grows faster. Eventually the system must be restarted with more consumer instances, but by then days of data have been delayed.

The root cause: backpressure was not handled correctly. The consumer scaled vertically (more instances) only after the problem became severe, instead of scaling proactively based on lag metrics. Additionally, the consumer did not implement any form of adaptive rate limiting or batching that could slow consumption when the downstream system was overwhelmed.

Unbounded growth

A streaming pipeline retains data for “forever” because someone wanted to be able to replay history at any time. Six months in, the Kafka cluster runs out of disk space. The team increases storage, but the cost grows linearly with retention. Two years in, the storage bill exceeds the entire engineering team budget.

The root cause: retention policies were not designed with cost in mind. Kafka tiered storage (moving old segments to object storage) or finite retention with export to a data lake would have solved this, but the team did not plan for it.

Each of these failures is preventable. Each requires understanding a specific aspect of distributed systems engineering. The purpose of this book is to teach you that understanding, so you do not make the same mistakes.

What This Book Will Build

Throughout this book, we will construct a complete reference application called EdgeFleet. EdgeFleet is not a toy example with one producer and one consumer. It is a realistic, multi-component system designed to demonstrate the full range of event-driven patterns and production concerns.

EdgeFleet models a logistics fleet management platform where vehicles equipped with IoT sensors stream telemetry data from the edge to a cloud-based analytics and operations center. Here is what the system does:

- Edge devices on vehicles generate telemetry events every second: position, speed, fuel level, engine diagnostics.
- Edge nodes buffer events locally and forward them to the cloud, surviving network outages without losing data.
- A cloud-based Kafka cluster receives and stores all events durably.
- Stream processors enrich events with metadata, compute rolling statistics, detect anomalies and publish domain events.
- An event-sourced fleet service maintains the authoritative state of each vehicle by consuming domain events.
- A query API serves dashboards and operations tools with optimized read models.

- A serverless alerting system triggers notifications when anomalies are detected.
- The entire system is instrumented with metrics, logs and traces for observability.

The code for EdgeFleet is written in Go and TypeScript, deployed to Kubernetes with Terraform and designed to be runnable locally with Docker Compose. Every chapter adds or modifies components as new concepts are introduced. By the end of the book, you will have a complete, working system that demonstrates every pattern we cover.

How to Read This Book

This book is organized in layers, from foundations to production mastery. Here is the structure:

Part 1: Foundations (Chapters 1-5) - Understand what event-driven architecture is, what problems it solves and what tools exist. These chapters cover events versus commands, messaging patterns and the internal architecture of the major streaming platforms: Kafka, NATS JetStream, Pulsar and cloud-managed services. By the end of this part, you will know which platform to choose for which use case.

Part 2: Building Correct Systems (Chapters 6-12) - Learn how to implement reliable event producers and consumers, design event schemas that evolve over time, build stream processing pipelines and apply event sourcing and CQRS patterns. These chapters focus on getting the fundamentals right: idempotency, exactly-once semantics, ordering guarantees and state management. The EdgeFleet application takes shape here.

Part 3: Edge and Serverless (Chapters 13-15) - Extend the architecture to edge environments. Learn about edge-to-cloud communication patterns, store-and-forward for intermittent connectivity and serverless event processing. These chapters show you how to build systems that work not just in well-connected cloud data centers but also in the messy reality of edge deployments.

Part 4: Distributed Systems Deep Dive (Chapters 16-18) - Understand the theory that governs all of the above. Delivery guarantees, consistency models, the CAP theorem and PACELC, fault tolerance, consensus algorithms. These

chapters explain why certain patterns work, why others do not and how to make reasoned trade-offs.

Part 5: Production Engineering (Chapters 19-22) - Secure the system, observe it, deploy it and operate it. Topics include TLS and authentication, metrics and tracing, Terraform and Kubernetes deployment, capacity planning and runbooks. These chapters turn a working prototype into a production-ready platform.

You can read this book sequentially for the most complete understanding. If you are already familiar with event-driven basics, you can jump to the chapters most relevant to your current challenges. The reference application code builds progressively, so you should read the chapters in order if you plan to follow along with the implementation.

The goal of this book is not to give you a checklist of tools to install but to give you the engineering judgment to design systems that work. That judgment comes from understanding trade-offs, from knowing what can fail and how to handle it when it does. Let us start building that understanding, one chapter at a time.

Chapter 2: Event-Driven Architecture

Foundations

Before choosing tools or writing code, you must understand the vocabulary and fundamental concepts of event-driven architecture. Many teams struggle with event-driven systems not because the technology is hard but because they use the wrong mental model. An event is not a message. A message is not a command. A topic is not a queue. Getting these distinctions right determines whether your architecture is maintainable or a tangled mess.

This chapter establishes the core vocabulary, the essential patterns and the fundamental decisions that shape every event-driven system.

Events, Messages and Commands

In casual conversation, the terms event, message and command are often used interchangeably. In a well-designed system, they are distinct concepts with different semantics, different lifetimes and different responsibilities.

Events

An event describes something that has already happened. It is a factual statement about the past, expressed in the language of the domain. The event occurred independently of whether anyone is listening. Typical events include:

- OrderPlaced
- PaymentCompleted
- VehiclePositionUpdated
- EngineFaultDetected
- UserRegistered

Events have three key properties:

1. **They are immutable facts.** Once an event occurred, it cannot be undone. You can publish a compensating event like `OrderCancelled`, but the original `OrderPlaced` event remains true. This immutability is what makes events suitable as a source of truth.
2. **They are decoupled from their consumers.** The producer publishes an event without knowing who will consume it or how. New consumers can be added at any time without changing the producer. This is the core decoupling benefit of event-driven architecture.
3. **They are domain-centric.** Good event names and structures use the ubiquitous language of the domain, not technical terms. An event called `PaymentCompleted` is clear to both engineers and domain experts. An event called `TransactionStatusChangedToThree` is not.

Commands

A command is an instruction to perform an action. Unlike an event, a command has not yet occurred. It expresses an intent: “place this order,” “cancel this payment,” “update this configuration.” Commands are typically directed at a specific service or aggregate and may fail: the order might not be placed because inventory is exhausted, the payment might be rejected.

Commands have different properties than events:

1. **They are future-oriented.** A command describes what should happen, not what did happen. The success or failure of the command is unknown at the time it is issued.
2. **They are addressed.** A command is typically sent to a specific recipient that is responsible for executing it. This introduces coupling: the command sender must know how to reach the command handler.
3. **They can be rejected.** Because they describe a proposed action, commands can fail validation, conflict with business rules, or encounter resource constraints. The sender must handle rejection.

Messages

A message is a transport-level concept. It is the envelope that carries data across a communication boundary. An event or a command is carried inside

a message. The message includes metadata: headers, routing information, timestamps, correlation IDs. When people say “Kafka messages” or “NATS messages,” they are talking about the transport envelope, not the semantic content.

Distinguishing between the semantic content (event or command) and the transport (message) is important because the same semantic content might be transported in different ways: an event might be published to a Kafka topic, a NATS subject, or written to an outbox table to be relayed later. The event itself does not change; only the transport mechanism does.

The Event-Command Cycle

In a well-designed event-driven system, events and commands form a cycle:

1. A user or external system issues a command to a service.
2. The service executes the command, applies domain logic and updates its state.
3. If the command succeeds, the service publishes one or more events describing what happened.
4. Other services consume those events and update their own state or trigger their own commands.

For example in EdgeFleet:

1. The operations team issues an AssignVehicle command to the FleetService, requesting that vehicle V-1234 be assigned to route R-567.
2. The FleetService validates the assignment, updates its internal state and publishes a VehicleAssigned event.
3. The RoutingService consumes the VehicleAssigned event and updates its routing plan.
4. The NotificationService consumes the VehicleAssigned event and sends a message to the driver.

The command is directed (to FleetService) and may fail. The events are broadcast (to all subscribers) and represent facts. This separation keeps the system decoupled: the operations team does not need to know about RoutingService or NotificationService. They only need to know how to send the command to FleetService.

A common mistake is to blur this distinction. Teams sometimes publish “commands as events” – events like `ProcessPayment` or `SendEmail` that are really instructions disguised as facts. This undermines the event semantics: if the event is delivered twice, the consumer must figure out whether it should execute the action again. If it is truly an event, like `PaymentCompleted`, duplicate delivery is harmless because completing an already-completed payment has no effect (idempotency).

The Producer-Consumer Model

At its core, event-driven architecture is built on the producer-consumer model. A producer generates data (events) and sends it to an intermediate component. A consumer receives data from that intermediate component and processes it. The intermediate component – typically a message broker, queue, or streaming platform – decouples the two.

Without the intermediate component, the producer and consumer must run at the same time and communicate directly. This is synchronous coupling. With the intermediate component, they can run independently, scale independently and tolerate each other’s failures.

The role of the broker

The broker is the central piece of infrastructure in event-driven systems. Its responsibilities include:

- **Accepting messages from producers:** The broker receives events and acknowledges them to the producer according to its durability guarantees.
- **Storing messages:** The broker persists events so they survive broker restarts and are available for consumers.
- **Delivering messages to consumers:** The broker routes events to consumers according to the subscription model: point-to-point, publish-subscribe, or streaming.
- **Managing ordering:** The broker preserves ordering within its logical units (partitions, queues, streams) according to its design.
- **Handling failures:** The broker replicates data across nodes, elects leaders when nodes fail and recovers from crashes.

Different brokers emphasize different responsibilities. A lightweight broker like NATS prioritizes low-latency delivery with minimal persistence. A streaming platform like Kafka prioritizes durability, ordering and replayability. The choice of broker shapes what kinds of applications you can build on top of it.

Producer responsibilities

A producer is not just a code snippet that calls a `send()` function. A production-grade producer must handle several concerns:

- **Serialization:** Converting domain objects into a byte format suitable for transmission.
- **Routing:** Deciding which topic, queue, or subject to publish to.
- **Key selection:** Choosing a partitioning key that determines which partition the event goes to, which affects ordering and consumer assignment.
- **Acknowledgments:** Configuring the broker acknowledgment level to balance durability and latency.
- **Retries:** Handling transient failures by retrying sends with exponential backoff.
- **Idempotency:** Ensuring retries do not create duplicate events in the broker.
- **Batching:** Grouping multiple events into a single send for efficiency.
- **Error handling:** Deciding what to do when sending fails permanently (log, dead letter, alert).

We will cover each of these in detail in Chapter 7, but understanding them as producer responsibilities upfront helps you appreciate why producers are more complex than they appear.

Consumer responsibilities

Similarly, a consumer is not just code that calls `receive()` and processes the result. A production-grade consumer must handle:

- **Deserialization:** Converting bytes back into domain objects, including schema evolution handling.
- **Acknowledgment strategy:** Deciding when to acknowledge a message as processed (before, after, or conditionally).

- **Idempotency:** Handling duplicate messages without creating duplicate effects.
- **Error handling:** Deciding how many times to retry processing, when to send a message to a dead letter queue, when to alert.
- **Concurrency:** Managing multiple consumer instances, partition assignment and parallel processing.
- **Backpressure:** Slowing consumption when the downstream system cannot keep up.
- **Offset management:** Committing progress in a way that balances durability and throughput.

These responsibilities make consumers the most complex component in most event-driven systems. We will cover consumer design in Chapter 8.

Point-to-Point Versus Publish-Subscribe

Event-driven systems support two fundamental communication patterns: point-to-point and publish-subscribe. Understanding the difference is essential because each pattern is appropriate for different scenarios and confusing them leads to architectural mistakes.

Point-to-point (queuing)

In point-to-point communication, a message is sent to a queue and consumed by exactly one consumer. Once a consumer processes the message and acknowledges it, the message is removed from the queue. If multiple consumer instances are subscribed to the same queue, the broker distributes messages among them.

This pattern is appropriate for:

- **Work distribution:** Distributing tasks among workers, where each task needs to be processed exactly once. For example, sending confirmation emails: each email should be sent once, so you place email tasks in a queue and have multiple email workers consume from it.
- **Load leveling:** Absorbing bursts of work by queuing requests and processing them at a steady rate. For example, an API that receives more requests than it can handle synchronously can place work items in a queue and process them asynchronously.

- **Competing consumers:** When you need multiple instances of the same service to share a workload, each taking items from a shared queue.

Publish-subscribe (broadcasting)

In publish-subscribe communication, a publisher sends a message to a topic and all subscribed consumers receive a copy of the message. Each subscriber maintains an independent view of the topic: one consumer falling behind does not affect others.

This pattern is appropriate for:

- **Event notification:** Broadcasting facts about what happened so that multiple services can react. For example, when an `OrderPlaced` event occurs, the inventory service, shipping service, analytics service and notification service all need to react, independently.
- **Data replication:** Distributing data changes to multiple consumers that each build their own view of the data. For example, a change data capture pipeline publishes database changes to a topic and different consumers build search indexes, materialized views and analytics datasets.
- **Loose coupling:** Enabling producers to publish events without knowing who will consume them. New consumers can be added at any time.

The key distinction

The distinction is simple but crucial: in point-to-point, each message is consumed by exactly one consumer (the work is shared). In publish-subscribe, each message is consumed by all consumers (the work is duplicated).

Many streaming platforms support both patterns. In Kafka, for example:

- A consumer group with multiple instances implements point-to-point semantics: each partition is consumed by exactly one consumer instance in the group.
- Multiple consumer groups subscribing to the same topic implement publish-subscribe semantics: each group receives all messages.

Understanding this distinction helps you model your system correctly. If you need to send one email per order, use a queue (point-to-point). If you need to notify five different services that an order was placed, use a topic with publish-subscribe semantics.

Confusing these patterns is a common mistake. A team might put order events in a single queue, thinking it is a topic and discover that only one service processes each order while others miss events. Or they might use a topic for email sending, thinking it is a queue and discover that every email worker sends every email, resulting in duplicate notifications.

Coupling, Cohesion and Autonomy

Event-driven architecture promises loose coupling between services. Loose coupling means that a change to one service does not require changes to other services. This promise is powerful because it enables independent development, independent deployment and independent scaling. But loose coupling is not automatic. It must be designed.

Temporal decoupling

Temporal decoupling means that the producer and consumer do not need to be running at the same time. The producer publishes an event and moves on. The consumer processes the event when it is ready, even if that is hours later. The broker stores events until consumers are ready to process them.

This is a fundamental advantage over synchronous communication. In a REST API, if the consumer service is down, the producer's request fails. With events, the producer can publish and the consumer can process later when it recovers. Temporal decoupling is what makes event-driven systems resilient to transient failures.

Spatial decoupling

Spatial decoupling means that the producer and consumer do not need to know each other's network addresses. The producer publishes to a topic or subject and the broker routes messages to consumers. Consumers subscribe to topics or subjects and the broker delivers messages. Neither side needs to know about the other's identity or location.

Spatial decoupling enables independent deployment. You can add a new consumer without changing the producer. You can move the consumer to a different data center without changing the producer. You can have multiple consumers in multiple regions consuming from the same event stream.

Data coupling

Data coupling is more subtle. Even if producer and consumer are temporally and spatially decoupled, they are still coupled through the event schema. If the producer changes the event format, consumers that depend on the old format may break. This is the schema evolution problem, which we will cover in Chapter 6.

Managing data coupling requires discipline: versioning schemas, maintaining backward compatibility, using schema registries, testing consumers against schema changes. Teams that ignore data coupling end up with tightly coupled systems despite using events.

Service autonomy

The ultimate goal of decoupling is service autonomy: each service should be able to evolve independently, deploy independently and fail independently. Autonomy requires:

- **Independent data:** Each service owns its own database and does not share databases with other services. This is the “database per service” pattern.
- **Independent deployment:** Each service can be deployed without coordinating with other services. This requires backward-compatible event schemas.
- **Independent failure:** If one service fails, other services continue to operate. Events are buffered in the broker and processed when the failing service recovers.

Autonomy is not free. It requires more complexity in the system: event schemas, consumer groups, distributed transactions via sagas. But the trade-off is worth it for large systems where multiple teams own different parts of the system and must move independently.

Anti-Patterns and Common Mistakes

Experience with event-driven systems has revealed a recurring set of anti-patterns: design choices that seem reasonable upfront but create problems in production. Avoiding these pitfalls saves countless hours of debugging and rework.

Events as commands

Publishing events that are really instructions disguised as facts. For example, publishing an event called `ProcessInvoice` instead of `InvoiceReadyForProcessing`. The problem is semantic: the consumer interprets the event as something that must be done, not something that already happened. If the event is delivered twice, the consumer may process the invoice twice. If the event is replayed from history, old invoices are reprocessed.

The fix is to name events for what happened, not what should be done. Instead of `ProcessInvoice`, use `InvoiceReceived` (meaning: the invoice has arrived and is now available for processing). Consumers that need to process invoices subscribe to `InvoiceReceived` and decide whether and how to process each one.

Polling instead of pushing

Using events for communication but polling databases or APIs to detect changes. For example, a service publishes events but also has another service that polls its database every 10 seconds to check for new records. This defeats the purpose of events: the polling service does not trust the events or does not receive all the events it needs.

Polling is acceptable as a temporary workaround or for systems that cannot emit events, but it should not be a core architectural pattern. If you find yourself polling frequently, investigate whether the polled system can publish events instead.

Overusing events

Using events for every interaction, including simple request-response scenarios where synchronous communication would be clearer. For example, having

a service publish a `GetCurrentInventory` command as an event and wait for a response event instead of calling an API. This adds latency, complexity and debugging difficulty without any decoupling benefit, because the two services are tightly coupled anyway (one depends on the other's response).

Events are best for one-way notifications and data distribution. For request-response interactions where the caller needs an immediate answer, use synchronous APIs (REST or gRPC). You can combine both: use events for notifications and use APIs for queries. This is the essence of CQRS.

Implicit ordering assumptions

Assuming events will be processed in the exact order they were produced, across partitions or consumers. Kafka guarantees ordering only within a single partition. If events for the same entity are spread across multiple partitions, they may be processed in any order. If a consumer group has multiple instances, each instance processes its assigned partitions, but there is no guarantee that partition A finishes processing before partition B starts.

The fix is to design consumers that handle out-of-order events. Use entity IDs as partition keys to keep events for the same entity on the same partition. Check event timestamps and reject stale events. Design state updates to be idempotent and commutative where possible.

God topics

Creating a single topic for all events in the system instead of organizing events into logically related topics. A god topic called "events" with thousands of different event types is impossible to manage, impossible to secure (everyone who subscribes gets everything) and impossible to scale (one slow consumer affects all event types).

The fix is to organize events into topics by domain, bounded context, or aggregate. For example: orders-topic for order-related events, vehicles-topic for vehicle-related events, payments-topic for payment-related events. This improves security (only relevant consumers subscribe to each topic), scalability (topics can be scaled independently) and maintainability (clear boundaries between domains).

Synchronous calls disguised as events

Having a producer publish an event and then poll or wait for a response event before continuing. For example: publish a `PaymentRequested` event, then poll the database every second to check if `PaymentCompleted` appeared. This gives you none of the benefits of asynchronous communication (the producer is blocked anyway) while adding all the complexity (event publishing, event consuming, polling).

If you need an immediate response, use a synchronous API. If you need to correlate a response event with a request, use a correlation ID and handle the response asynchronously. Do not mix the two patterns.

Understanding these anti-patterns helps you recognize them before they become entrenched in your architecture. Event-driven systems are powerful, but they require discipline to use correctly. The next chapters will show you how to apply that discipline, starting with the messaging primitives and platforms that form the foundation.

Chapter 3: Message Brokers, Queues and Streams

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Queues: Load Leveling and Work Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Queue characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

When to use queues

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Queue implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Topic-Based Pub/Sub: Broadcasting and Filtering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Topic characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

When to use topics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Topic implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Event Streams: Ordered, Persistent, Replayable

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stream characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

When to use streams

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stream implementations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Broker Architectures: From RabbitMQ to Kafka to NATS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Message broker model (RabbitMQ)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log-based streaming model (Kafka)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Lightweight broker model (NATS)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Comparing the architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Choosing a Messaging Primitive

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 4: Kafka Internals and Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Topics, Partitions and Offsets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Topic partitioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Partition assignment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Offsets and consumer position

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The Broker and Controller

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Brokers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Controllers and KRaft mode

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Node types

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Replication and ISR

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Leader and follower replicas

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

In-Sync Replicas (ISR)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Replication factor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

min.insync.replicas

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Unclean leader election

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Producer Semantics and Batched Writes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The producer API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Batching and compression

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Acknowledgments

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Retries and idempotency

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumer Groups and Offset Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumer groups

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Rebalancing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Offset commits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log Compaction and Retention

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Time-based retention

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log compaction

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Tiered storage

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 5: NATS, Pulsar and Cloud-Native Alternatives

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

NATS and JetStream: Simplicity and Speed

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Core NATS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

JetStream

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

NATS cluster architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

NATS versus Kafka: practical comparison

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Apache Pulsar: Multi-Tenant Cloud Scale

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Pulsar architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Pulsar topics and subscriptions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Pulsar strengths and weaknesses

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Managed Cloud Services: Kinesis, Pub/Sub, Event Hubs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Amazon Kinesis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Google Cloud Pub/Sub

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Azure Event Hubs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

When to choose managed services

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Brokerless and Mesh Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Service mesh integration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

NATS as a mesh

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Platform Selection Matrix

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 6: Events, Schemas and Serialization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Designing Domain Events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Naming events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Event content

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Event granularity

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Avoiding commands in events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference application: first events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Serialization Formats: JSON, Protobuf, Avro

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

JSON

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Protocol Buffers (Protobuf)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Avro

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Schema Registry and Evolution Rules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

What a Schema Registry does

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Compatibility rules

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Subject naming strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Backward and Forward Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Producers upgrading before consumers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumers upgrading before producers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The two-phase upgrade pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Practical rules for schema evolution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 7: Producers - Publishing Events Reliably

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Producer Design Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Direct publishing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Publishing via outbox

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Publishing via CDC (Change Data Capture)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Choosing the pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka Producer Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Core settings

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Batching and compression

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Buffer and timeouts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Serialization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Partitioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotent Producers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The duplicate problem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

How idempotent producers work

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Ordering Guarantees and Key-Based Routing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Per-partition ordering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Key-based ordering

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Trade-offs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference application: EdgeFleet producers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Error Handling and Retries

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Transient failures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Metadata errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Serialization errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Schema Registry errors

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Dead letter handling for producers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-driversystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-driversystemsattheedge>.

Chapter 8: Consumers - Processing Events Correctly

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumer Group Dynamics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Partition assignment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Rebalancing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumer group ID

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-Least-Once and At-Most-Once Consumption

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-least-once (commit after processing)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-most-once (commit before processing)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Exactly-once consumption

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotent Consumers and Deduplication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotency strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Deduplication window

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotency versus deduplication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Backpressure and Graceful Degradation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Detecting backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Handling backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Consumer configuration for backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: EdgeFleet Consumers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 9: Stream Processing - Transformations and Joins

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Map, Filter and Aggregate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Map

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Filter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Aggregate

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Tumbling, Sliding and Session Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Tumbling windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Sliding windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Session windows

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Window semantics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stream-Stream and Stream-Table Joins

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stream-stream join

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stream-table join

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Table-table join

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Join semantics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stateful Processing and Checkpoints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State stores

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Checkpoints and fault tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Exactly-once processing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: The Stream Processor

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 10: Kafka Streams and Flink

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka Streams: Embedded and Lightweight

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The DSL and Processor API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Fault tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Strengths and limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Apache Flink: Batch-and-Stream Unified

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The DataStream API

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State and stateful operators

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Fault tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Strengths and limitations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State Stores and RocksDB

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Why RocksDB?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State size considerations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State TTL

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Fault Tolerance and Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka Streams fault tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Flink fault tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

When to Use Which

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 11: Event Sourcing and CQRS

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Command Query Responsibility Segregation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The unified model problem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

CQRS basics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

When CQRS makes sense

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Event Sourcing: Appending the Truth

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

How event sourcing works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Event store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Aggregates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Rebuilding state

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Benefits of event sourcing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Drawbacks of event sourcing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Sagas and Distributed Transactions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The saga pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Choreography versus orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Read Models and Projections

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Projection design

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Read model storage

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Eventual consistency

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Event-Sourced Domain

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Event store schema (PostgreSQL)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Aggregates

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Projection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Command handler

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 12: Distributed State Management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Stateful Services in a Stateless World

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Where to store state

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Event-driven state machines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Distributed Caches and KV Stores

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Redis

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

DynamoDB and similar

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Eventual Consistency and Conflict Resolution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Why eventual consistency is acceptable

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Handling inconsistencies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Conflict resolution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Partitioning State and Hot Keys

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Partitioning strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Hot keys

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Distributed State Store

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Vehicle state

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

State flow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Code example: cache invalidation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Optimistic concurrency control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 13: Edge Computing Fundamentals

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Cloud, Fog and Edge: A Continuum

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Why edge matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Latency, Bandwidth and Autonomy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Latency requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Bandwidth constraints

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Autonomy requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge Runtimes: Containers, Functions and WASM

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Containers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Serverless and edge functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

WebAssembly (WASM)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge Deployment Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Managed edge clusters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Self-managed Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Standalone edge devices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: The Edge Topology

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge node components

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge topology diagram

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Deployment (K3s + Docker Compose for simplicity)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Handling disconnection

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 14: Serverless Event Processing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Serverless Execution Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Cloud Functions: AWS Lambda, Cloudflare Workers, Azure Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

AWS Lambda

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Azure Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Cloudflare Workers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Google Cloud Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge Functions and CDNs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Why edge functions matter

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Cloudflare Workers example

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Cold Starts, Limits and Workarounds

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Cold starts

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Timeout limits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Memory and compute limits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Invocation limits

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: Serverless Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Lambda alert processor (Go)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Wrapping up

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 15: Cloud-to-Edge Communication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Edge-to-Cloud Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Push pattern (edge initiates)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Pull pattern (cloud initiates)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Bi-directional pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Store-and-Forward for Intermittent Connectivity

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

How store-and-forward works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Storage requirements

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Retry strategy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Ordering across reconnections

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Delta Sync and Operational Transformations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Delta sync basics

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Conflict resolution for delta sync

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Implementation for EdgeFleet

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Bi-Directional Event Routing

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Addressing edge nodes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Persistent connections

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Command delivery guarantees

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: Edge-to-Cloud Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Telemetry pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Command pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge-to-cloud bridge (simplified)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 16: Delivery Guarantees, Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-Most-Once, At-Least-Once, Exactly-Once

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-most-once delivery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

At-least-once delivery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Exactly-once delivery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Choosing a guarantee

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The Two-Phase Commit Problem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

How 2PC works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Problems with 2PC

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Transactional Outbox Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The dual-write problem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The outbox solution

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

The relay process

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Making the outbox idempotent

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotent Operations: Making At-Least-Once Safe

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Idempotency principles

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Making non-idempotent operations idempotent

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Transactional Messaging

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 17: Consistency, Ordering and Partitioning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

CAP and the PACELC Theorem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The CAP theorem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

What CAP really means

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

CAP in practice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The PACELC theorem

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Linearizability Versus Sequential Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Linearizability

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Sequential consistency

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Eventual consistency

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Partition Key Design and Hot Spots

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Good partition key properties

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Examples

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Hot spots

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Compensating for Disordered Events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

When ordering matters

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Causes of disorder

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Handling disordered events

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Consistency Choices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 18: Fault Tolerance and Recovery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Failure Modes in Distributed Systems

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Node failures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Network failures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Data corruption

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Cascading failures

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Retries, Dead Letter Queues and Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Retry strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Dead letter queues

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Circuit Breakers and Bulkheads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Circuit breaker

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Bulkhead

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Combining circuit breakers and bulkheads

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Leader Election and Raft

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Why leader election is hard

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Raft consensus

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Raft in practice

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Failure Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 19: Observability and Operations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Logging: Structure, Correlation and Volume

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Structured logging

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log correlation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log volume and cost

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Log aggregation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Metrics: Red Signals and SLOs

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Key metrics for event-driven systems

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Metrics collection with Prometheus

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Consumer lag monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

SLOs and alerting

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Distributed Tracing End to End

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

How tracing works

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Tracing Kafka events with OpenTelemetry

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Tracing across edge and cloud

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Capacity Planning and Cost Control

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Capacity planning

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Cost optimization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: Observability Stack

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 20: Security, Secrets and Multi-Tenancy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Authentication and Authorization Models

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka authorization

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Service-to-service authentication

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

TLS, mTLS and Encryption at Rest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Encryption in transit (TLS)

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Encryption at rest

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Certificate management

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Secret Management Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Secret management options

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Dynamic secrets

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Multi-Tenant Isolation and Quotas

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kafka multi-tenancy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Edge multi-tenancy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Reference Application: Security Layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Chapter 21: CI/CD, IaC and Deployment Automation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Infrastructure as Code with Terraform

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Why Terraform?

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Terraform for event-driven infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Terraform best practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Containerization and Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

Dockerfile best practices

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Kubernetes deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Deploying Kafka with Strimzi

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

GitOps and Progressive Delivery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

GitOps workflow

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Progressive delivery

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Schema and Migration Pipelines

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Schema evolution pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Database migration pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Reference Application: Complete Deployment

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Chapter 22: Production Hardening and the Full System

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

The Complete Reference Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Edge layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Cloud streaming layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Cloud processing layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Data layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Observability layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Security layer

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Load Testing and Performance Validation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Load testing strategy

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Performance expectations

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Load testing tools

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Runbooks and On-Call

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Key runbooks for EdgeFleet

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

On-call rotation

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Migrating from Monolith to Event-Driven

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Migration strategies

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Migration steps for EdgeFleet-like system

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

What Comes Next

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsatttheedge>.

Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.

References

This content is not available in the sample book. The book can be purchased on Leanpub at <https://leanpub.com/event-drivensystemsattheedge>.