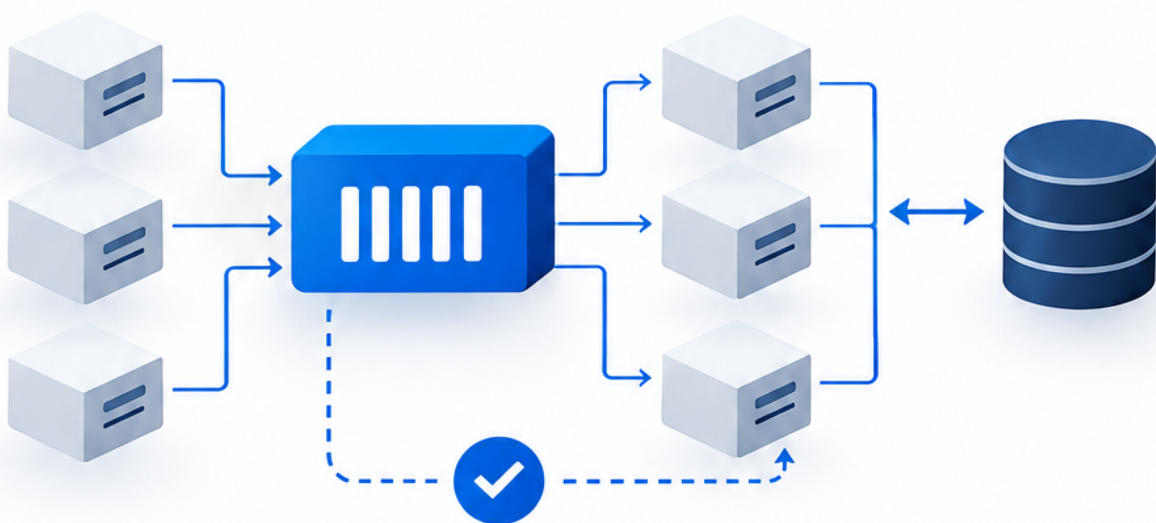


SOFTWARE ENGINEERING
IN PRACTICE

Message Queues **and** Distributed Transactions

Architecture, Implementation,
Reliability and Production Operations



PATRICK COLEMAN

Message Queues and Distributed Transactions

Architecture, Implementation, Reliability and Production Operations

Steve Publications

This book is available at

<https://leanpub.com/messagequeuesanddistributedtransactions>

This version was published on 2026-10-10



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

Architecture, Implementation, Reliability and Production Operations	1
Introduction	2
What this book covers	2
Technology choices	3
Prerequisites	4
How to read this book	4
The reliability mindset	4
Chapter 1: The Problem with Synchronous Communication	6
The synchronous trap: tight coupling and cascading failures	6
Latency, throughput and resource exhaustion under load	7
The network is not a memory bus: partitions, timeouts and partial failures	8
Decoupling time, space and synchronization with messages	9
When synchronous communication is still the right choice	10
Chapter 1 Summary	12
Chapter 2: Core Messaging Concepts	13
Producers, consumers, brokers and message brokers versus message buses	13
Topics, queues, partitions and exchange patterns	14
Point-to-point versus publish-subscribe semantics	14
Message anatomy: headers, keys, bodies and metadata	15
Acknowledgments, offsets and delivery semantics	16
Chapter 2 Summary	18
Chapter 3: Delivery Guarantees and Their Meaning	19
At-most-once: semantics, use cases and failure modes	19
At-least-once: duplicate delivery, retries and crash windows	19

CONTENTS

Effectively-once processing: idempotency, deduplication and application guarantees	19
The broker-consumer-database coordination problem	19
Why true end-to-end exactly-once is a myth in practice	19
Chapter 3 Summary	20
Chapter 4: The CAP Theorem, PACELC and Messaging Trade-offs . .	21
CAP revisited: consistency, availability, partition tolerance in messaging	21
PACELC: understanding trade-offs during normal and failure conditions	21
How messaging systems position themselves on the spectrum . . .	21
Throughput, latency, durability and operational complexity	21
Choosing consistency levels for different business needs	21
Chapter 4 Summary	23
Chapter 5: Apache Kafka	24
The log, partitions, replication and the controller	24
Producers: batching, compression, acknowledgments and ordering	24
Consumers, consumer groups, offsets and rebalancing	24
Kafka transactions: write-ahead logs, transactional IDs and exactly-once semantics	24
Limits: retention, replay constraints, latency tail behavior and operational overhead	24
Chapter 5 Summary	25
Chapter 6: RabbitMQ	26
Exchanges, queues, bindings and routing keys	26
Message durability, confirmations and negative acknowledgments	26
Clustering, mirrored queues and quorum queues	26
The transaction API, publisher confirms and reliability trade-offs	26
When to choose RabbitMQ and when not to	26
Chapter 6 Summary	27
Chapter 7: NATS and JetStream	28
The NATS core protocol and supercluster architecture	28
Subjects, wildcards and request-reply patterns	28

CONTENTS

JetStream: stream storage, consumers and durable subscriptions	28
Deliveries guarantees, ack policies and message retention	28
Chapter 7 Summary	29
Chapter 8: Redis Streams, Amazon SQS/SNS, Apache Pulsar and Database-Backed Queues	30
Redis Streams: lightweight persistence with Redis familiarity	30
Amazon SQS and SNS: managed, simple and opinionated	30
Apache Pulsar: unified messaging and streaming with tiered storage	30
Relational-database-backed queues: polling, locking and simplicity trade-offs	30
Comparative summary: protocol, delivery, ordering, durability and ops complexity	30
Chapter 8 Summary	32
Chapter 9: Technology Selection Framework	33
Defining selection criteria: workload patterns, data volume, latency SLAs and team skills	33
Scenario analysis: high-throughput event streaming, reliable task queues, pub-sub eventing	33
Stack recommendations: Kafka-first, RabbitMQ-first, managed-cloud-first, lightweight	33
Multi-broker architectures and interoperability challenges	33
Migration considerations and technology debt	33
Chapter 9 Summary	35
Chapter 10: The Reference Application	36
Business domain: orders, inventory, payments, shipping and notifications	36
Initial synchronous architecture and its failure modes	36
Target microservice decomposition	36
Data models, APIs and consistency requirements	36
Technology stack decision: choosing the reference stack	36
Chapter 10 Summary	37
Chapter 11: Producers and Consumer Design	38

CONTENTS

Producer configuration: retries, acknowledgments, batching and compression	38
Consumer design: processing loops, batching and graceful shutdown	38
Serialization choices: JSON, Protobuf and schema registries	38
Error handling: transient versus permanent failures	38
Connection management and resource limits	38
Chapter 11 Summary	40
Chapter 12: Message Ordering and Partitioning	41
Ordering guarantees within partitions	41
Partition-key design for the order-processing domain	41
Hot partitions and load imbalance	41
Ordering across partitions and aggregates	41
Trade-offs between ordering, parallelism and latency	41
Chapter 12 Summary	42
Chapter 13: Retries, Dead-Letter Queues and Poison Messages	43
Retry strategies: exponential backoff, jitter and circuit breakers	43
Dead-letter queues: routing, inspection and replay	43
Poison messages: detection, isolation and remediation	43
Retry limits and escalation	43
Observability for failure patterns	43
Chapter 13 Summary	44
Chapter 14: Idempotency and Deduplication	45
Idempotent operations in the order-processing domain	45
Idempotency keys and deduplication tables	45
Set-based versus record-based deduplication	45
Handling idempotency with distributed locks	45
Idempotency as a design principle, not a fix	45
Chapter 14 Summary	46
Chapter 15: Flow Control, Backpressure and Batching	47
Backpressure: signals, pauses and graceful degradation	47
Consumer rate limiting and prefetch tuning	47
Batching: producer batching and consumer batching	47
Memory management and resource exhaustion	47

CONTENTS

Coordinated flow control across services	47
Chapter 15 Summary	48
Chapter 16: ACID and Local Transactions	49
ACID properties: atomicity, consistency, isolation, durability	49
Isolation levels: read uncommitted through serializable	49
Concurrency control: locking versus MVCC	49
Transaction isolation anomalies	49
When local transactions are sufficient	49
Chapter 16 Summary	50
Chapter 17: The Saga Pattern	51
The distributed transaction problem across services	51
Choreographed sagas: event-driven compensating transactions	51
Orchestrated sagas: central coordinator and state machines	51
Saga design for the order-processing reference application	51
Choosing orchestration versus choreography	51
Chapter 17 Summary	52
Chapter 18: Two-Phase Commit and XA	53
Two-phase commit: prepare and commit phases	53
Transaction coordinators and participant failures	53
Three-phase commit: reducing blocking	53
XA specification and database support	53
Why two-phase commit is rarely the right choice in microservices	53
Chapter 18 Summary	54
Chapter 19: The Transactional Outbox Pattern	55
The crash window between commit and publish	55
Outbox table design and transactional writes	55
Publishing from the outbox: polling versus change data capture	55
Idempotent event handling and consumer guarantees	55
Chapter 19 Summary	56
Chapter 20: Change Data Capture and Event Sourcing	57
Change data capture: WAL readers, Debezium and log-based CDC	57
CDC pipeline architecture and latency	57

CONTENTS

Event sourcing: events as the source of truth	57
Combining CDC with the outbox pattern	57
Trade-offs: complexity, auditability and debugging	57
Chapter 20 Summary	58
Chapter 21: Distributed Transaction Failure Modes	59
Crash windows: commit without publish, publish without commit . . .	59
Duplicate delivery and compensating transaction idempotency . . .	59
Concurrent processing and race conditions	59
Network partitions and split-brain scenarios	59
Recovery procedures and state reconciliation	59
Chapter 21 Summary	60
Chapter 22: Deployment and Infrastructure	61
Docker Compose for local development	61
Kubernetes deployment: StatefulSets, storage and networking . . .	61
Resource planning: CPU, memory, disk IOPS and networking	61
Infrastructure as code and environment parity	61
Multi-region deployment and data locality	61
Chapter 22 Summary	62
Chapter 23: Security and Access Control	63
Authentication: SASL, TLS and cloud IAM integration	63
Authorization: topic-level and queue-level access control	63
Encryption: in transit and at rest	63
Secrets management and credential rotation	63
Least privilege and audit logging	63
Chapter 23 Summary	64
Chapter 24: Observability: Logging, Metrics and Tracing	65
Structured logging with correlation IDs	65
Metrics: throughput, latency, queue depth and error rates	65
Distributed tracing across services and message hops	65
Dashboards and alerting: SLOs and meaningful thresholds	65
Debugging message flow and dead-letter analysis	65
Chapter 24 Summary	66

Chapter 25: Reliability, Scaling and Disaster Recovery	67
High availability: replication, failover and rolling upgrades	67
Horizontal scaling: brokers, partitions and consumers	67
Capacity planning and load testing methodology	67
Backup, restore and disaster recovery procedures	67
Incident response and postmortem practices	67
Chapter 25 Summary	68
Chapter 26: Anti-Patterns, Migration and Evolution	69
Anti-patterns: chatty services, message bloating and hidden synchronous calls	69
Schema evolution and backward compatibility	69
Migrating between messaging technologies	69
Breaking change management in event-driven systems	69
When to abandon messaging and simplify	69
Chapter 26 Summary	70
Conclusion: The Reliability Mindset	71
Summary of core principles	71
Choosing simplicity where possible, sophistication where necessary	71
Continuous learning and staying current	71
Final recommendations for practice	71
References	72

Architecture, Implementation, Reliability and Production Operations

This book teaches software engineers, backend developers, distributed-systems architects and platform engineers how to design, implement and operate reliable message-driven systems and distributed transactions. It combines rigorous theoretical foundations with practical engineering guidance, complete working implementations and realistic production scenarios. You will learn the underlying mechanics of major messaging platforms, the patterns for building eventually consistent distributed transactions and the operational practices for running these systems in production. A consistent reference application demonstrates how architectural decisions evolve from a simple implementation to a resilient production system.

Introduction

At 03:17 on a Thursday morning, the payment service crashed. Within forty five seconds the order service fell behind. Within two minutes the inventory reservations started timing out. Within ten minutes the monitoring alerts converged on a single truth: the system had more messages than it could process and more failures than it could tolerate.

The on-call engineer restarted the payment service. Messages flooded in. The service crashed again. The engineer added workers. More messages. Another crash. Each remediation attempted to solve the symptom. The actual problem was architectural: the system assumed the network would be reliable, dependencies would respond predictably and every message would be processed exactly once. None of those assumptions held under load.

This is not a hypothetical scenario. Every distributed system eventually faces these conditions: networks partition, services crash, messages arrive out of order, consumers process messages twice, transactions fail halfway through and operators are awakened at inconvenient hours to untangle failures that should have been handled automatically.

Message queues and distributed transactions are the engineering response to that reality. They do not prevent failures. They make systems resilient to failures that are already happening.

What this book covers

This book is structured in five parts.

Part One establishes the foundations: why synchronous communication breaks at scale, what messaging abstractions mean in practice, what delivery guarantees actually deliver and how distributed-systems theory maps to architectural decisions.

Part Two examines the technology landscape. You will read detailed architectural analyses of Apache Kafka, RabbitMQ, NATS, Apache Pulsar, Redis

Streams, Amazon SQS and SNS and database-backed queues. Each technology receives treatment proportional to its relevance: internal architecture, persistence models, replication mechanisms, delivery semantics, ordering guarantees, scalability limits, operational requirements and the workloads it handles well or poorly. This is not a feature comparison table. It is engineering analysis to support real technology selection decisions.

Part Three walks through message processing patterns: producers, consumers, partitioning, ordering, retries, dead letter queues, idempotency, back-pressure and flow control. These patterns appear in every message-driven system. Getting them right separates systems that degrade gracefully from systems that collapse.

Part Four covers distributed transactions. Local ACID transactions do not span microservices. This section covers the saga pattern, two-phase commit and its limitations, the transactional outbox pattern, change data capture and event sourcing. Each approach receives an honest treatment of its failure modes, consistency guarantees and operational trade-offs.

Part Five addresses production operations: deployment on Kubernetes, security, observability, monitoring, scaling, disaster recovery and incident response. A messaging system that works in development but cannot be operated in production is a liability, not an asset.

A consistent reference application runs throughout the book. It is an order-processing and payment system with order, inventory, payment, shipping and notification services. You will see it evolve from a naive synchronous design through multiple messaging patterns until it becomes a resilient, production-grade system with proper transaction handling, retry policies, observability and operational procedures.

Technology choices

This book uses Go as its primary implementation language. Go has excellent Kafka clients, strong standard library support for concurrency and networking and is widely used in production distributed systems. The primary messaging technology is Apache Kafka, which is the most widely deployed event streaming platform and has the richest ecosystem for transactional messaging. PostgreSQL serves as the relational database for service data and outbox tables. Debezium demonstrates change data capture.

Where alternative technologies demonstrate meaningfully different architectural approaches, the book includes comparative implementations. RabbitMQ appears for scenarios where complex message routing and acknowledgment semantics are central. Amazon SQS and SNS appear for managed cloud deployments. NATS appears for lightweight high-performance use cases. Redis Streams and database-backed queues appear for simple workloads where operational overhead must be minimized.

Every code example uses real libraries at specific versions, includes all necessary supporting components and can be compiled and run. There is no pseudocode presented as production implementation. Every significant class, function, configuration option, transaction boundary, acknowledgment, retry policy and error-handling path is explained.

Prerequisites

This book assumes you are comfortable with programming at a professional level, preferably in Go, Java, or a similar systems language. You should understand basic microservice concepts: independent deployable services, APIs and the database-per-service pattern. Familiarity with relational databases, SQL and basic distributed-systems terminology is helpful but not required. The book introduces distributed-systems concepts when needed.

How to read this book

Read sequentially. Each part builds on concepts from earlier parts. The technology deep dives in Part Two reference the foundational concepts in Part One. The patterns in Parts Three and Four reference technology-specific features. The production operations in Part Five assume familiarity with the patterns and technologies previously covered.

If you are primarily interested in distributed transactions, read Parts One and Four and treat Part Two as reference. If you are primarily interested in messaging infrastructure, read Parts One and Two and use Parts Three and Four as pattern lookup. If you are operating a production system, start with Part Five and jump to specific chapters in earlier parts as needed.

The reliability mindset

The central thesis of this book is simple: reliable distributed systems require disciplined architectural choices and operational rigor. There are no silver bullets. Every technology, pattern and guarantee trades one cost for another. Production resilience comes from understanding and managing these trade-offs explicitly rather than relying on infrastructure abstractions.

This book will make those trade-offs visible. It will show you what guarantees are real and what guarantees are marketing. It will show you where systems break and how to build them so they survive. It will give you the technical depth to evaluate new tools and the practical experience to implement the ones you choose.

The rest of this book delivers on that promise.

Chapter 1: The Problem with Synchronous Communication

Every distributed system starts with a choice: communicate synchronously or asynchronously. Synchronous communication feels natural because it matches how we program single-process applications. Function A calls function B, function B returns a result and function A continues. The mental model is simple. The control flow is obvious. The failure modes are familiar.

That simplicity is an illusion in distributed systems. When function A and function B run in different processes across a network, every assumption about reliability, latency and availability that held locally breaks down. Synchronous communication amplifies those failures. Asynchronous communication with message queues mitigates them.

This chapter examines why synchronous communication becomes problematic at scale and what asynchronous messaging provides as an alternative. Understanding these problems is the foundation for every architectural decision that follows.

The synchronous trap: tight coupling and cascading failures

Consider a typical e-commerce checkout flow implemented with synchronous HTTP calls. The order service receives a create-order request. It must validate inventory, process payment, reserve stock and trigger notifications before returning a success response. Each step is a synchronous call to another service.

This design creates tight coupling in three dimensions: time, space and failure.

Time coupling means the caller must wait for every dependency to complete. If inventory validation takes 200 milliseconds, payment processing takes 800 milliseconds and notification delivery takes 500 milliseconds, the

total response time is at least 1500 milliseconds. In the worst case, any one slow dependency determines the entire response time. This is the straggler problem: the slowest service in the chain becomes the bottleneck for all callers [1].

Space coupling means the caller must know about every dependency. The order service must know the network address of the inventory service, the payment service and the notification service. It must handle each service's API contract. Adding a new dependency requires changing the order service. This coupling creates ripple effects: every change in a downstream service potentially requires changes in every upstream caller.

Failure coupling is the most dangerous form. If the notification service is down, the order service cannot complete the checkout flow. The order fails even though inventory is available and payment succeeds. This is cascading failure: one service's unavailability propagates upstream and takes down the entire system [2].

Production systems face cascading failures regularly. Netflix famously demonstrated this in their early cloud migration by using Chaos Monkey to randomly terminate production instances and observe how failures propagated [3]. Without defensive design, a single failed service takes down the entire system.

The synchronous trap is deceptively easy to fall into. Every microservice architecture guide warns against tight coupling. Every architecture decision starts with the intention to decouple. But synchronous APIs look loosely coupled. They use clean interfaces. They follow REST conventions. They pass typed requests and responses. The coupling is architectural, not syntactic. It lives in the call graph, not the code.

Latency, throughput and resource exhaustion under load

Synchronous communication has fundamentally different resource characteristics than in-process calls. Understanding those characteristics is critical because they determine how systems behave under load.

Every synchronous HTTP call consumes a thread or event-loop slot on the caller, a thread or slot on the callee, a connection in each endpoint's connection pool and CPU time for serialization and network handling. Under heavy load, these resources are finite and become bottlenecks.

Consider a service that handles 1000 requests per second and makes three synchronous calls per request. The service must sustain 3000 outgoing calls per second. If each call takes 100 milliseconds, the service needs at least 300 concurrent connections. If the called service takes 200 milliseconds during peak load, the caller needs 600 connections. Add multiple caller services making requests to the same service and the required capacity grows rapidly.

This is the thread exhaustion problem. In thread-per-request models, each blocked synchronous call occupies a thread that cannot serve other requests. When the thread pool is exhausted, the system starts rejecting requests or queueing them in memory until the process runs out of memory [4].

Connection pool exhaustion is equally common. Most production services limit outgoing connections to prevent overwhelming their dependencies. When the pool is exhausted, calls block waiting for an available connection. If all calls block, the pool remains full and the system deadlocks.

Latency distribution makes these problems worse. Average latency is irrelevant for capacity planning. The tail latency matters. If a service responds in 50 milliseconds for 99 percent of requests but 5 seconds for 1 percent of requests, those slow requests consume 100 times more resources and determine the capacity requirements [5].

Synchronous systems also amplify latency across call chains. A chain of five synchronous calls, each averaging 50 milliseconds, produces 250 milliseconds of latency before any business logic executes. Add network jitter, garbage collection pauses and load balancing overhead and the actual latency is often five to ten times higher than the sum of averages.

These resource constraints create a feedback loop. Higher latency consumes more resources. Fewer available resources cause higher latency. The system collapses when the feedback loop reaches a tipping point where latency growth outpaces resource release [6].

The network is not a memory bus: partitions, timeouts and partial failures

The fundamental insight of distributed systems is that the network is unreliable. This is not marketing language. It is a physical reality that every distributed system must accommodate.

Network latency varies. A call that normally takes 5 milliseconds may take 500 milliseconds during congestion. TCP retransmissions add unpredictable delays. DNS resolution failures add more.

Network partitions occur. Data centers lose connectivity. Rack switches fail. Cloud providers experience zone-level outages. When a network partition isolates a subset of services, the partitioned services cannot communicate with the rest of the system [7].

Timeouts are a blunt instrument for handling unreliable networks. Setting a timeout too low causes premature failures. Setting it too high causes callers to wait for requests that will never complete. Every timeout decision is a trade-off between responsiveness and false positives [8].

Partial failures are more insidious than complete failures. A service may respond to health checks but fail on specific operations. A service may succeed on the write path but fail on the read path. A service may process some requests correctly while silently dropping others. Synchronous callers cannot distinguish between these scenarios and healthy operation without additional instrumentation [9].

The fallacies of distributed computing, articulated by L. Peter Deutsch and others at Sun Microsystems, remain accurate: the network is reliable, latency is zero, bandwidth is infinite, the network is secure, topology does not change, there is one administrator, transport cost is zero and the network is homogeneous [10]. Synchronous architectures assume most of these fallacies. They fail when reality contradicts the assumptions.

Decoupling time, space and synchronization with messages

Asynchronous messaging with message queues addresses the coupling problems of synchronous communication by introducing an intermediate store. Producers send messages to the queue. Consumers retrieve messages when they are ready. The producer and consumer do not need to be online simultaneously. They do not need to agree on timing. They do not need to know each other's identities.

This decoupling provides three benefits.

Temporal decoupling means the producer and consumer operate on different time scales. A producer can send messages faster than consumers process them and the queue absorbs the burst. A consumer can start later and catch up by processing the backlog. A consumer can crash and restart without requiring the producer to retry [11].

Spatial decoupling means the producer does not need to know the consumer's location. The producer sends to a named queue or topic. The consumer subscribes to that queue or topic. Adding a new consumer does not require changing the producer. Removing a consumer does not affect the producer. Services can be discovered, relocated, or scaled independently [12].

Failure decoupling means one service's failure does not automatically cause another service's failure. If the consumer is down, messages accumulate in the queue. If the producer crashes mid-batch, the broker tracks which messages were confirmed. The caller receives an immediate response without waiting for all dependencies to complete. Cascading failures are contained within boundaries [13].

The trade-off is latency and consistency. Synchronous calls provide low latency and immediate consistency: the caller knows immediately whether the operation succeeded. Asynchronous messaging introduces latency: the caller must wait for eventual processing. Consistency becomes eventual: the system converges on a correct state over time rather than maintaining it at every instant [14].

This trade-off is intentional. It is the core decision of asynchronous architecture: accept eventual consistency and higher latency in exchange for resilience, scalability and decoupling. Systems that need immediate consistency for every operation should not use asynchronous messaging. Systems that need to survive dependency failures and scale independently should use it.

When synchronous communication is still the right choice

Asynchronous messaging is not universally superior. Synchronous communication remains appropriate in several scenarios.

User-facing operations requiring immediate feedback often benefit from synchronous calls. A login request should return quickly with a clear success

or failure. A search query should return results without requiring the user to poll for completion. These operations have predictable latency requirements and users expect immediate responses [15].

Read-your-write consistency requirements may necessitate synchronous communication. If a user creates an order and immediately expects to see it in their order list, the order service should update the database and return before responding. Eventual consistency introduces complexity for users who expect immediate visibility [16].

Simple systems with few dependencies and low scale requirements may not need the overhead of message infrastructure. Adding a message queue introduces operational complexity, monitoring requirements and failure modes that do not exist with direct calls. If a system works reliably with synchronous communication and has clear scalability paths, introducing messaging may be premature optimization [17].

Short-lived workflows where all steps are critical may be simpler synchronously. A payment authorization that must validate, authorize and capture in a single atomic operation is more straightforward as a synchronous transaction than as an asynchronous saga. The complexity of distributed compensation may outweigh the resilience benefits for simple workflows [18].

The key is to evaluate each communication path on its own requirements. Not every service call should become an asynchronous message. Not every message should become a synchronous call. The right architecture uses both patterns where each is appropriate.

Understanding when synchronous communication fails and when it succeeds provides the context for the rest of this book. The next chapter defines the core concepts of messaging systems so we can discuss specific technologies and patterns with shared terminology.

Chapter 1 Summary

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

Chapter 2: Core Messaging Concepts

Before analyzing specific technologies and implementing patterns, we need shared terminology. Message queuing and event streaming systems use similar concepts but name them differently. This chapter defines the core abstractions so the rest of the book can reference them precisely.

Producers, consumers, brokers and message brokers versus message buses

Every messaging system has four roles.

The producer creates messages and sends them to the messaging system. Producers are typically services that generate events: an order service that publishes order-created events, a payment service that publishes payment-completed events, or a background worker that queues jobs for later processing.

The consumer receives messages and processes them. Consumers subscribe to queues or topics and pull messages when they are ready to process them. Multiple consumers can subscribe to the same queue or topic. Each consumer processes messages independently.

The broker stores and routes messages between producers and consumers. Brokers provide persistence so messages survive consumer crashes and restarts. They provide routing logic so messages reach the correct consumers. They provide coordination so multiple consumers can share the load without processing the same message twice.

Some systems use a bus architecture instead of a broker. A message bus is an abstraction layer that manages messaging infrastructure on behalf of services. Services interact with the bus API rather than individual brokers. The bus handles topic creation, queue management, routing rules and failover transparently.

The distinction matters architecturally. Message brokers are infrastructure components that require operational management. Message buses are

abstractions that hide infrastructure complexity but introduce a single point of control and potential vendor lock-in [1].

Topics, queues, partitions and exchange patterns

Messaging systems organize messages using different abstractions depending on the delivery semantics required.

A queue is a first-in-first-out buffer where messages are delivered to exactly one consumer. When multiple consumers subscribe to the same queue, each message is delivered to one consumer. This is the point-to-point pattern. Queues are appropriate for work distribution: background jobs, order processing, payment handling. The key property is that each message is consumed exactly once by exactly one consumer [2].

A topic is a broadcast channel where messages are delivered to all subscribers. Every consumer subscribed to a topic receives every message. This is the publish-subscribe pattern. Topics are appropriate for event distribution: notifying multiple services that an order was created, broadcasting configuration changes, distributing audit logs. The key property is that every subscriber receives every message [3].

Partitions divide a topic or queue into independent segments. Each partition is an ordered sequence of messages. Partitions enable parallelism: multiple consumers can process different partitions simultaneously. Partitions enable scaling: adding more partitions allows more consumers to process messages in parallel. The trade-off is that ordering is guaranteed only within a partition, not across partitions [4].

Exchange patterns define how messages are routed. Direct routing delivers messages to queues based on exact matching criteria. Topic routing delivers messages to queues based on pattern matching against message properties. Fanout routing delivers messages to all queues bound to an exchange. These patterns appear in systems like RabbitMQ where routing complexity is a first-class feature [5].

Kafka treats every topic as both a queue and a topic. Consumer groups provide point-to-point semantics within a topic: each message is delivered to one consumer in the group. Multiple consumer groups can subscribe to the same topic for publish-subscribe semantics: each group receives all messages [6].

Point-to-point versus publish-subscribe semantics

The choice between point-to-point and publish-subscribe determines how services interact and how failures propagate.

Point-to-point semantics are appropriate when work needs to be divided among consumers. An order-fulfillment queue distributes orders among workers. Each order is processed by exactly one worker. Scaling adds more workers to the same queue. Each worker processes a subset of orders. This is the classic work-queue pattern [7].

Publish-subscribe semantics are appropriate when events need to be broadcast. An order-created event should be visible to the notification service, the analytics service, the recommendation service and the fraud-detection service. Each service receives the same event and processes it independently. Adding a new service does not require changing the producer [8].

Many systems need both. An order processing system uses point-to-point semantics for order fulfillment (one worker processes each order) and publish-subscribe semantics for event notification (all interested services receive order events). The architectural decision is which events go into which pattern [9].

Message anatomy: headers, keys, bodies and metadata

Messages have structure beyond the payload. Understanding message anatomy is important because these components determine routing, ordering and processing behavior.

The body contains the message payload: the actual data being transmitted. Bodies should be self-contained and include enough context for the consumer to process the message without external lookups. Message bodies should be serialized in a format both producer and consumer understand: JSON, Protocol Buffers, or Avro are common choices [10].

Headers are metadata attached to the message that controls routing and processing. Headers can specify routing keys, content types, correlation IDs, TTL and priority. Headers are typically small and structured. They are examined by the broker for routing decisions [11].

Keys determine partition placement. Messages with the same key are always routed to the same partition. This guarantees ordering for messages related to the same entity: all messages for order ID 123 go to the same partition so they are processed in order. Keys should be chosen carefully because they determine data distribution and potential hot partitions [12].

Metadata is system-generated information: timestamps, message IDs, sequence numbers, offsets and broker identifiers. Metadata enables consumers to track progress, detect duplicates and recover from failures. Offsets uniquely identify messages within partitions and are the basis for consumer position tracking in Kafka [13].

Message IDs enable deduplication and tracing. A globally unique message ID lets consumers detect duplicates and lets operators trace a message through the system. Message IDs should be generated by the producer and included in every message [14].

Acknowledgments, offsets and delivery semantics

Acknowledgments are the mechanism by which consumers confirm message processing. Without acknowledgments, the broker cannot know whether a message was processed successfully and must either redeliver it or drop it.

Positive acknowledgments confirm successful processing. The consumer processes the message and sends an acknowledgment to the broker. The broker removes the message from the queue or marks it as consumed. Positive acknowledgments prevent message loss: if a consumer crashes before acknowledging, the broker redelivers the message [15].

Negative acknowledgments indicate processing failure. The consumer cannot process the message and sends a negative acknowledgment. The broker can either discard the message or requeue it for another attempt. Negative acknowledgments enable the dead-letter queue pattern: messages that fail repeatedly are moved to a separate queue for inspection [16].

Automatic acknowledgments happen before the consumer processes the message. The broker considers the message consumed as soon as it is delivered. Automatic acknowledgments provide higher throughput but risk message loss if the consumer crashes during processing [17].

Manual acknowledgments happen after the consumer processes the message. The consumer explicitly sends an acknowledgment when processing suc-

ceeds. Manual acknowledgments prevent message loss but reduce throughput because the broker holds messages until acknowledged [18].

Offsets are the position of a consumer within a partition. In Kafka, offsets are sequential integers assigned to each message in a partition. Consumers commit offsets to track how far they have progressed. Offset commits can be automatic at intervals or manual after processing. Offset management determines whether consumers process messages multiple times or skip messages during recovery [19].

Delivery semantics describe the broker's guarantee about message delivery. At-most-once delivery means a message is delivered zero or one times: no duplicates but potential loss. At-least-once delivery means a message is delivered one or more times: no loss but potential duplicates. Exactly-once delivery means a message is delivered exactly one time: no loss and no duplicates. Each guarantee has different implementation complexity and performance characteristics [20].

These core concepts appear in every messaging system. Specific technologies implement them differently and provide different trade-offs. The next chapter examines delivery guarantees in detail because they determine correctness and are the most commonly misunderstood aspect of message processing.

Chapter 2 Summary

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

Chapter 3: Delivery Guarantees and Their Meaning

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

At-most-once: semantics, use cases and failure modes

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

At-least-once: duplicate delivery, retries and crash windows

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

Effectively-once processing: idempotency, deduplication and application guarantees

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

The broker-consumer-database coordination problem

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

Why true end-to-end exactly-once is a myth in practice

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

Chapter 3 Summary

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

Chapter 4: The CAP Theorem, PACELC and Messaging Trade-offs

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

CAP revisited: consistency, availability, partition tolerance in messaging

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

PACELC: understanding trade-offs during normal and failure conditions

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

How messaging systems position themselves on the spectrum

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

Throughput, latency, durability and operational complexity

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

Choosing consistency levels for different business needs

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

Chapter 4 Summary

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

Chapter 5: Apache Kafka

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

The log, partitions, replication and the controller

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

Producers: batching, compression, acknowledgments and ordering

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

Consumers, consumer groups, offsets and rebalancing

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

Kafka transactions: write-ahead logs, transactional IDs and exactly-once semantics

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

Limits: retention, replay constraints, latency tail behavior and operational overhead

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

Chapter 5 Summary

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

Chapter 6: RabbitMQ

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

Exchanges, queues, bindings and routing keys

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

Message durability, confirmations and negative acknowledgments

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

Clustering, mirrored queues and quorum queues

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

The transaction API, publisher confirms and reliability trade-offs

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

When to choose RabbitMQ and when not to

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

Chapter 6 Summary

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

Chapter 7: NATS and JetStream

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

The NATS core protocol and supercluster architecture

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

Subjects, wildcards and request-reply patterns

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

JetStream: stream storage, consumers and durable subscriptions

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

Deliveries guarantees, ack policies and message retention

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

Chapter 7 Summary

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

Chapter 8: Redis Streams, Amazon SQS/SNS, Apache Pulsar and Database-Backed Queues

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

Redis Streams: lightweight persistence with Redis familiarity

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

Amazon SQS and SNS: managed, simple and opinionated

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

Apache Pulsar: unified messaging and streaming with tiered storage

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

Relational-database-backed queues: polling, locking and simplicity trade-offs

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

Comparative summary: protocol, delivery, ordering, durability and ops complexity

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

Chapter 8 Summary

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

Chapter 9: Technology Selection Framework

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

Defining selection criteria: workload patterns, data volume, latency SLAs and team skills

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

Scenario analysis: high-throughput event streaming, reliable task queues, pub-sub eventing

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

Stack recommendations: Kafka-first, RabbitMQ-first, managed-cloud-first, lightweight

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

Multi-broker architectures and interoperability challenges

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

Migration considerations and technology debt

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

Chapter 9 Summary

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

Chapter 10: The Reference Application

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

Business domain: orders, inventory, payments, shipping and notifications

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

Initial synchronous architecture and its failure modes

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

Target microservice decomposition

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

Data models, APIs and consistency requirements

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

Technology stack decision: choosing the reference stack

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

Chapter 10 Summary

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

Chapter 11: Producers and Consumer Design

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

Producer configuration: retries, acknowledgments, batching and compression

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

Consumer design: processing loops, batching and graceful shutdown

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

Serialization choices: JSON, Protobuf and schema registries

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

Error handling: transient versus permanent failures

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

Connection management and resource limits

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

Chapter 11 Summary

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

Chapter 12: Message Ordering and Partitioning

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

Ordering guarantees within partitions

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

Partition-key design for the order-processing domain

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

Hot partitions and load imbalance

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

Ordering across partitions and aggregates

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

Trade-offs between ordering, parallelism and latency

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

Chapter 12 Summary

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

Chapter 13: Retries, Dead-Letter Queues and Poison Messages

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

Retry strategies: exponential backoff, jitter and circuit breakers

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

Dead-letter queues: routing, inspection and replay

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

Poison messages: detection, isolation and remediation

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

Retry limits and escalation

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

Observability for failure patterns

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

Chapter 13 Summary

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

Chapter 14: Idempotency and Deduplication

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

Idempotent operations in the order-processing domain

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

Idempotency keys and deduplication tables

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

Set-based versus record-based deduplication

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

Handling idempotency with distributed locks

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

Idempotency as a design principle, not a fix

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

Chapter 14 Summary

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

Chapter 15: Flow Control, Backpressure and Batching

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

Backpressure: signals, pauses and graceful degradation

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

Consumer rate limiting and prefetch tuning

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

Batching: producer batching and consumer batching

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

Memory management and resource exhaustion

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

Coordinated flow control across services

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

Chapter 15 Summary

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

Chapter 16: ACID and Local Transactions

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

ACID properties: atomicity, consistency, isolation, durability

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

Isolation levels: read uncommitted through serializable

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

Concurrency control: locking versus MVCC

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

Transaction isolation anomalies

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

When local transactions are sufficient

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

Chapter 16 Summary

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

Chapter 17: The Saga Pattern

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

The distributed transaction problem across services

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

Choreographed sagas: event-driven compensating transactions

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

Orchestrated sagas: central coordinator and state machines

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

Saga design for the order-processing reference application

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

Choosing orchestration versus choreography

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

Chapter 17 Summary

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

Chapter 18: Two-Phase Commit and XA

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

Two-phase commit: prepare and commit phases

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

Transaction coordinators and participant failures

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

Three-phase commit: reducing blocking

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

XA specification and database support

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

Why two-phase commit is rarely the right choice in microservices

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

Chapter 18 Summary

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

Chapter 19: The Transactional Outbox Pattern

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

The crash window between commit and publish

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

Outbox table design and transactional writes

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

Publishing from the outbox: polling versus change data capture

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

Idempotent event handling and consumer guarantees

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

Chapter 19 Summary

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

Chapter 20: Change Data Capture and Event Sourcing

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

Change data capture: WAL readers, Debezium and log-based CDC

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

CDC pipeline architecture and latency

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

Event sourcing: events as the source of truth

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

Combining CDC with the outbox pattern

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

Trade-offs: complexity, auditability and debugging

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

Chapter 20 Summary

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

Chapter 21: Distributed Transaction Failure Modes

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

Crash windows: commit without publish, publish without commit

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

Duplicate delivery and compensating transaction idempotency

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

Concurrent processing and race conditions

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

Network partitions and split-brain scenarios

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

Recovery procedures and state reconciliation

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

Chapter 21 Summary

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

Chapter 22: Deployment and Infrastructure

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

Docker Compose for local development

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

Kubernetes deployment: StatefulSets, storage and networking

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

Resource planning: CPU, memory, disk IOPS and networking

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

Infrastructure as code and environment parity

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

Multi-region deployment and data locality

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

Chapter 22 Summary

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

Chapter 23: Security and Access Control

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

Authentication: SASL, TLS and cloud IAM integration

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

Authorization: topic-level and queue-level access control

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

Encryption: in transit and at rest

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

Secrets management and credential rotation

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

Least privilege and audit logging

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

Chapter 23 Summary

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

Chapter 24: Observability: Logging, Metrics and Tracing

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

Structured logging with correlation IDs

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

Metrics: throughput, latency, queue depth and error rates

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

Distributed tracing across services and message hops

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

Dashboards and alerting: SLOs and meaningful thresholds

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

Debugging message flow and dead-letter analysis

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

Chapter 24 Summary

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

Chapter 25: Reliability, Scaling and Disaster Recovery

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

High availability: replication, failover and rolling upgrades

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

Horizontal scaling: brokers, partitions and consumers

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

Capacity planning and load testing methodology

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

Backup, restore and disaster recovery procedures

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

Incident response and postmortem practices

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

Chapter 25 Summary

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

Chapter 26: Anti-Patterns, Migration and Evolution

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

Anti-patterns: chatty services, message bloating and hidden synchronous calls

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

Schema evolution and backward compatibility

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

Migrating between messaging technologies

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

Breaking change management in event-driven systems

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

When to abandon messaging and simplify

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

Chapter 26 Summary

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

Conclusion: The Reliability Mindset

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

Summary of core principles

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

Choosing simplicity where possible, sophistication where necessary

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

Continuous learning and staying current

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

Final recommendations for practice

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

References

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