
# Real-Time AI & Agentic Systems with Redpanda

Harnessing Agentic AI & Redpanda to Automate, Optimize, and Scale AI-Driven Workflows

# ABOUT ME

**DR. HITESH KUMAR SHARMA**

***Technical Instructor & Consultant***

- Industry/Academic Experience: **16 Years (6 years Exp Kafka, Confluent, Redpanda, Python Development & Testing, 6 years in DevOps, K8S & Docker,4 Years in RPA)**
- **Official Instructor of Confluent and Apache Kafka, Redpanda (50+ Trainings delivered)**
- **Delivered 120+ Trainings (LinkedIn, Snapchat, Walmart, HCL, Accenture etc.)**
- **Delivered training to 30+ Companies**
- **Delivered training in 12+ Countries physically**
- Core Technical Domains: **Java, Test Automation, Confluent, UiPath RPA, DevOps, Cloud Computing, Data Analytics**
- Academic Qualifications**: Ph.D. (CSE), M.Tech (CSE)**
- Certifications:
  - **Confluent Developer, Admin Certified**
  - **UiPath RPA Certified Associate**
  - **Docker Certified Associate**
  - **CKA Certified**
  - **Neo4J Certified Associate**
  - **Maven Certified Professional**
- 5 Books Published
- 45 Patents Published
- 02 Copyright Published

# Redpanda Platform & Operations (DevOps/Platform Focus)

## Redpanda Fundamentals

# AGENDA

- What is Redpanda?
- Broker architecture and IDs
- Kafka basics: topics, partitions, producers/consumers, offsets
- Redpanda vs Apache Kafka performance insights
- Raft consensus algorithm
- Single binary design, thread-per-core model
- Performance metrics on hardware

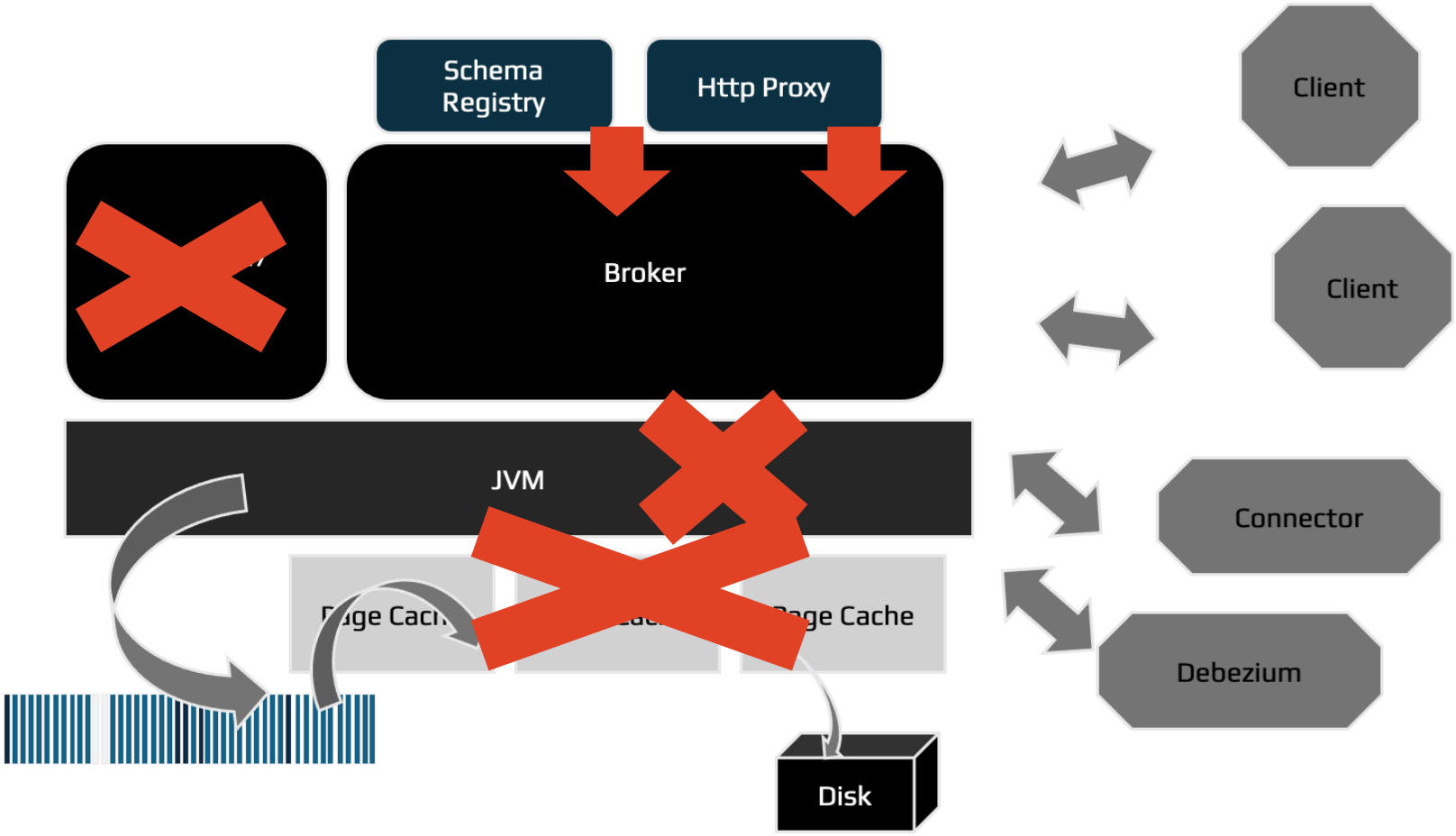
# What is Redpanda?

**Redpanda** is a high-performance streaming data platform, fully compatible with **Kafka APIs** but simpler to run, faster, and does not require ZooKeeper.

- Low latency and high throughput.
- Kafka API compatible (no code changes needed).
- Simplified architecture (no ZooKeeper).
- uns anywhere – on-premises, cloud, or containers.

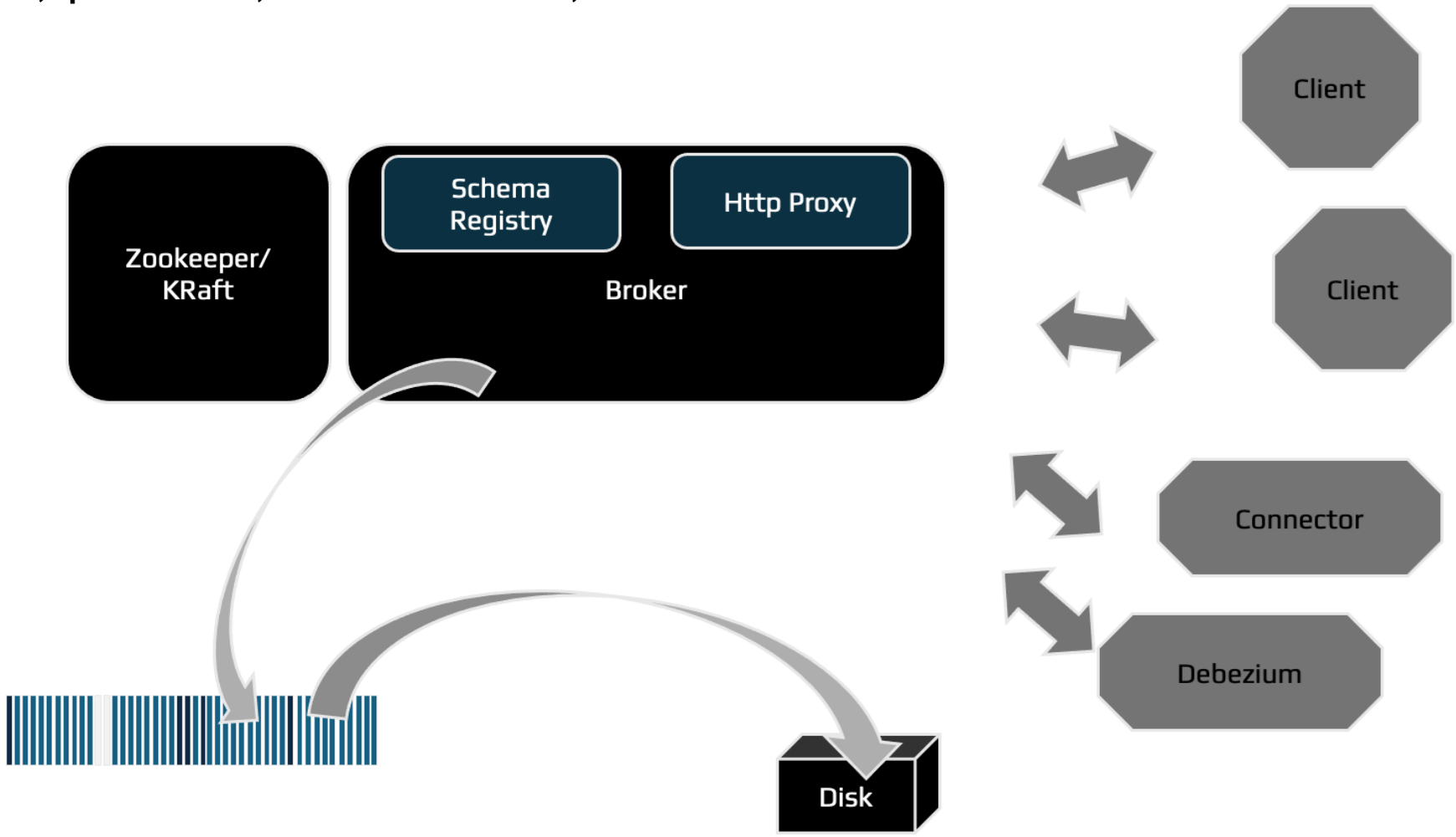
# Redpanda Broker architecture

Simple, powerful, cost-efficient, reliable



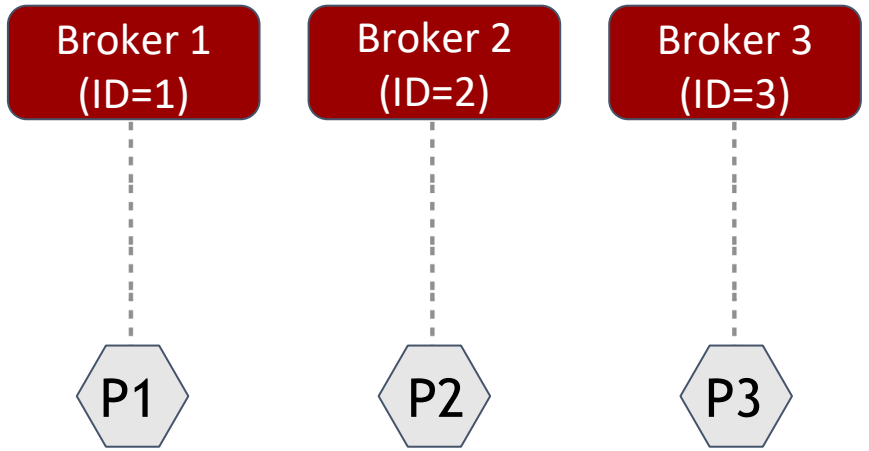
# Redpanda Broker architecture

Simple, powerful, cost-efficient, reliable



# Redpanda Broker IDs

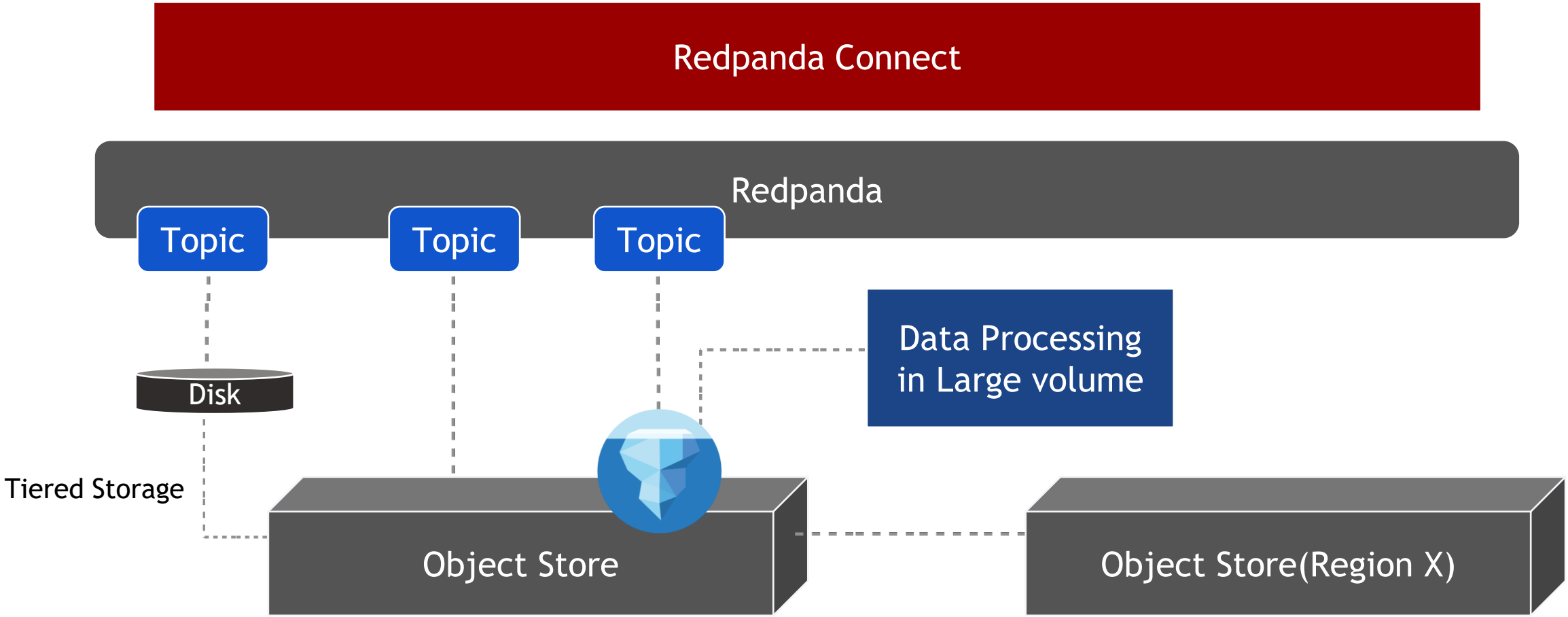
- A **Broker ID** is a **unique identifier** assigned to each broker (server/node) in the Redpanda cluster.
- It helps the cluster **distinguish one broker from another**.
- Even if a broker restarts, the same ID ensures the cluster recognizes it as the same node.




# Redpanda Connect

Redpanda Connect is a framework for moving data in and out of Redpanda without writing custom code.

# Redpanda One



# Kafka basics

**Topics**

Logical channels where messages are published.

**Partitions**

Topics are split into partitions for scalability and parallelism.

**Producers**

Applications that publish (write) data to topics.

**Consumers**

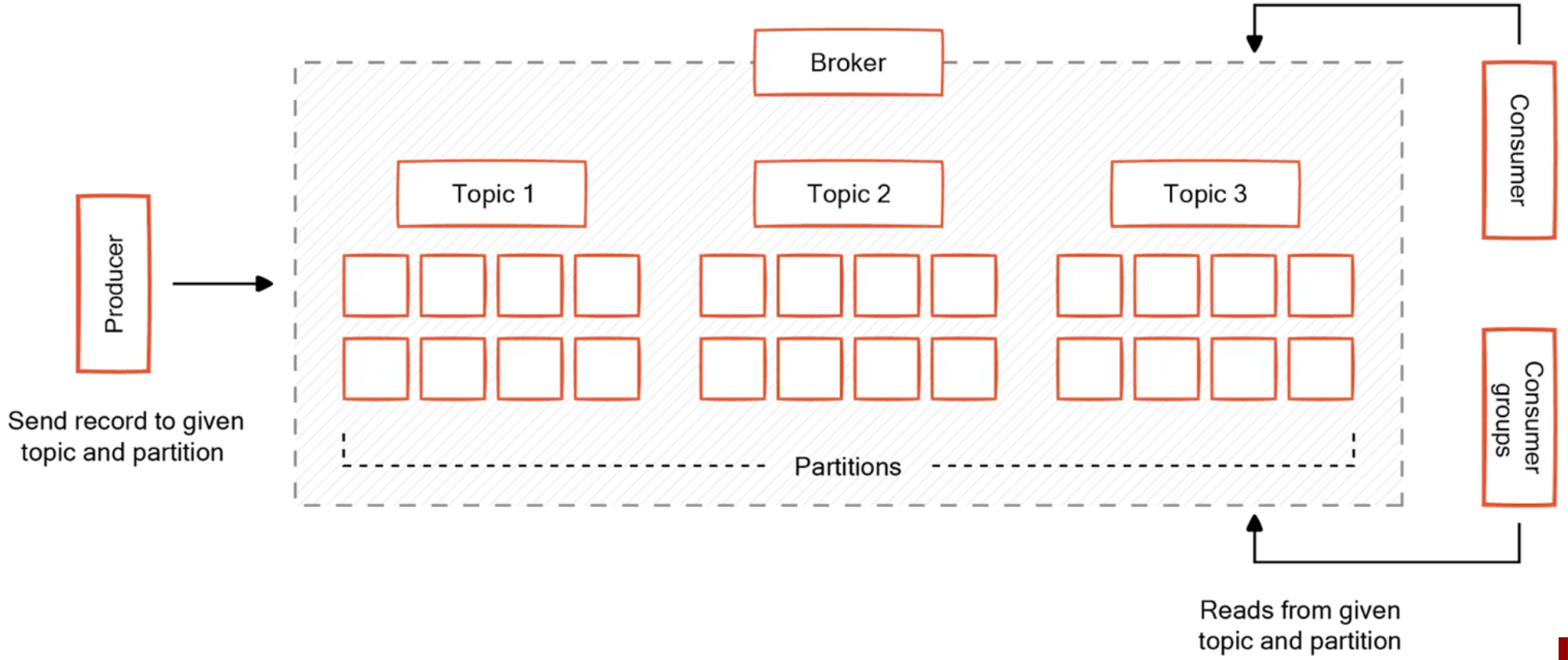
Applications that subscribe (read) data from topics.

**Offsets**

Unique sequence numbers within partitions that track message position.

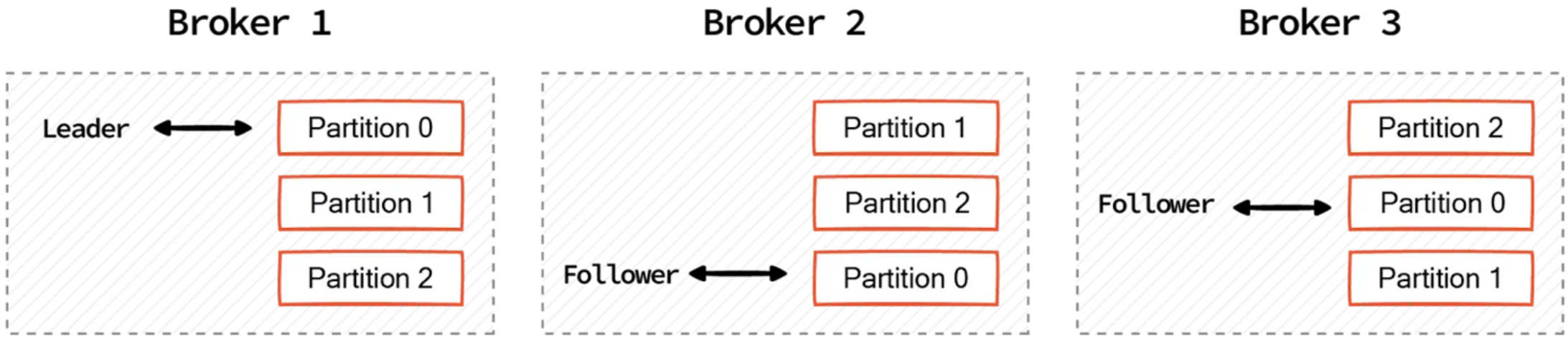
# Kafka Topics

Partitions are ordered, append-only logs that store subsets of a topic's data. Multiple partitions allow Kafka to process messages in parallel.



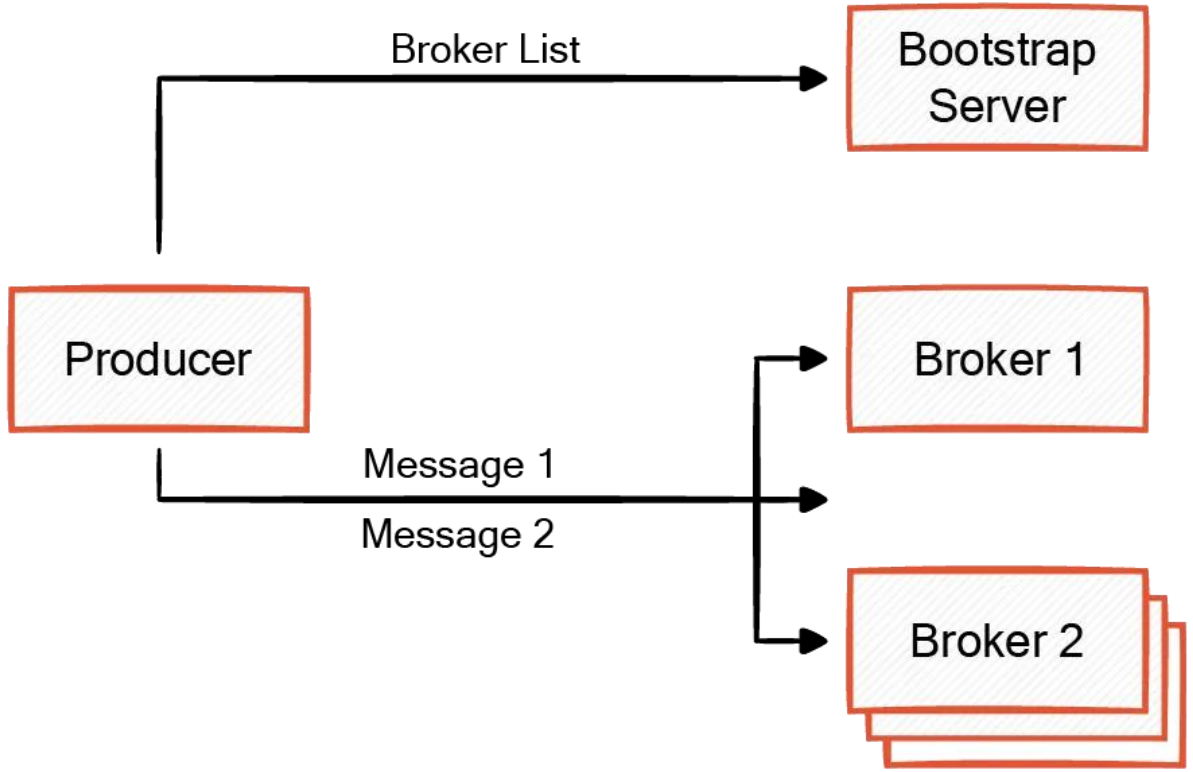
# Kafka Partitions

Partitions are ordered, append-only logs that store subsets of a topic's data. Multiple partitions allow Kafka to process messages in parallel.



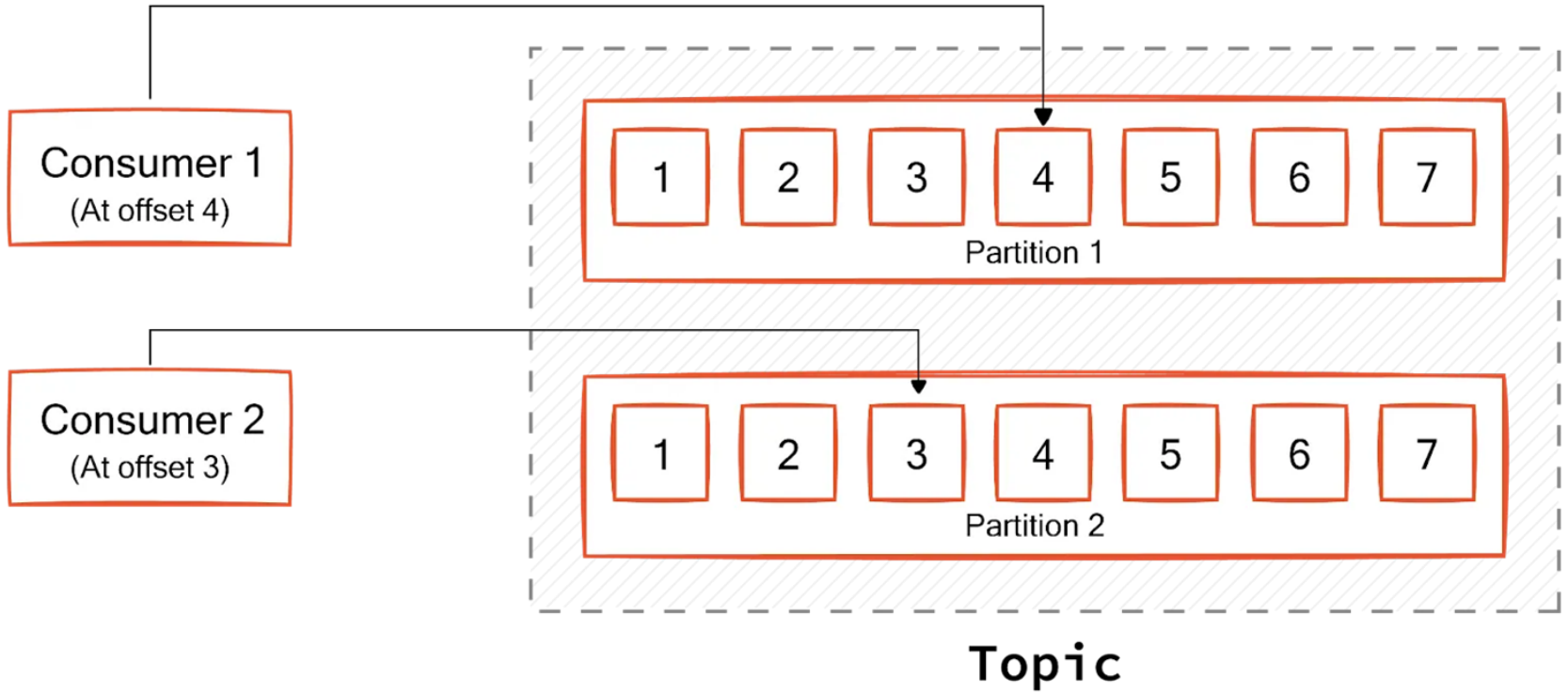
# Kafka producers/consumers

When processing data using Apache Kafka's architecture, components connected to Kafka are producers or consumers.
Producers store messages (or events) in Kafka, and consumers read these messages.



# Kafka Offset

Offset is a continuously increasing identifier that represents the order of a message from the beginning of the partition.



# Redpanda vs Apache Kafka performance insights

## Redpanda

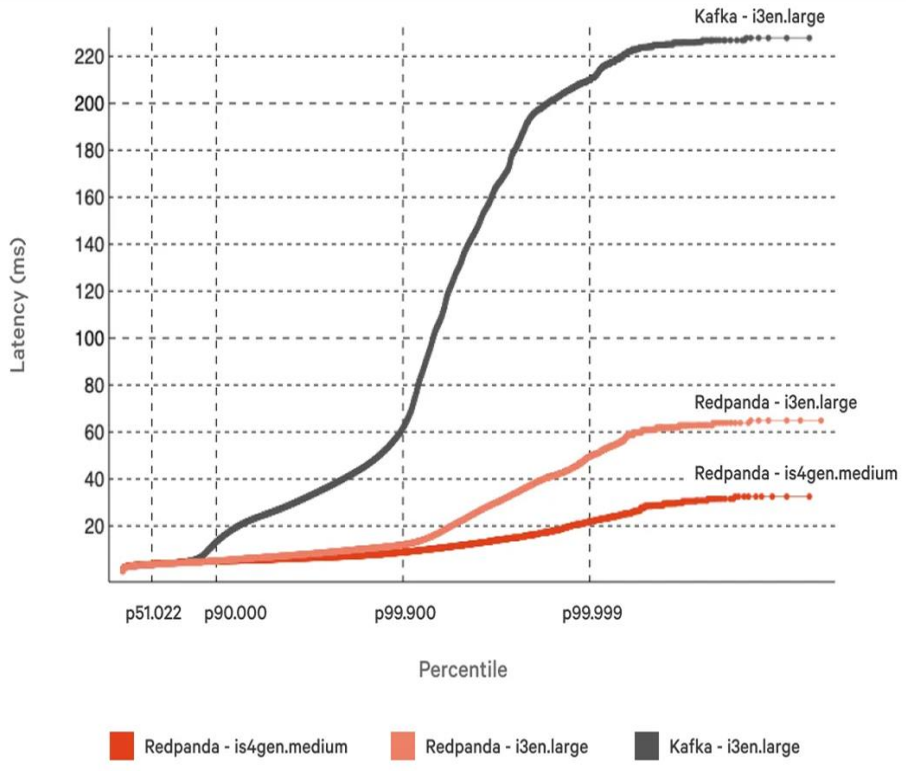
Ultra-low, stable latency (up to 70× faster), breaks 1 GB/s on just 3 nodes, efficient even on small ARM instances.

## Apache Kafka

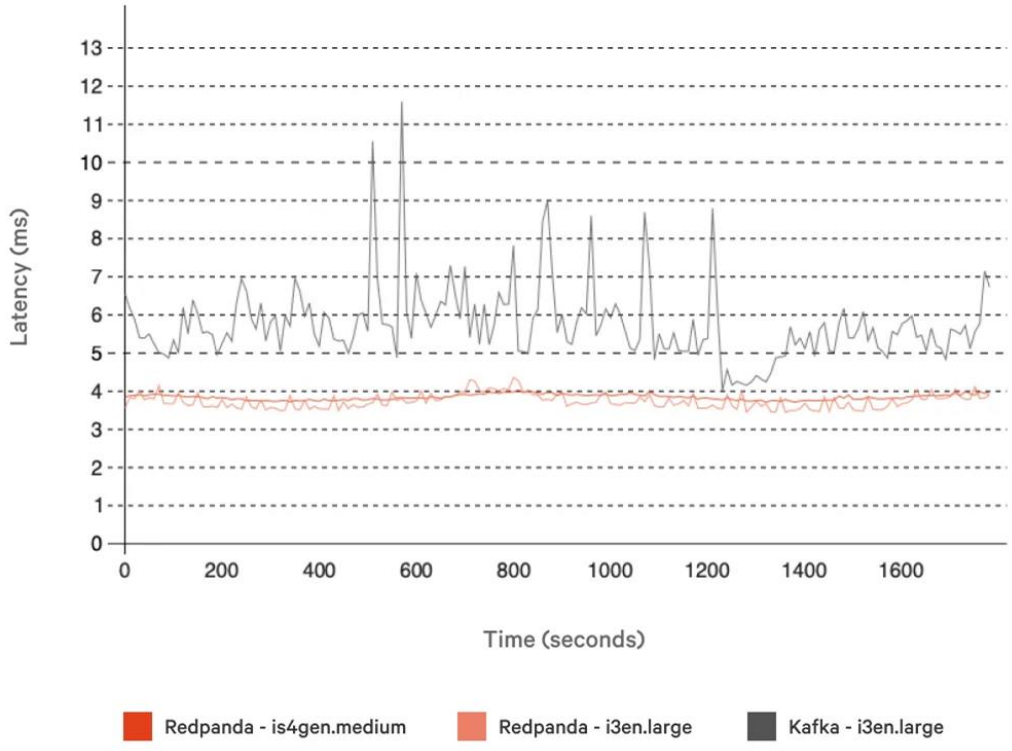
Higher latency, needs 2–3× more nodes and hardware to reach similar throughput, struggles on ARM.

| Workload size/Throughput | Kafka P99.99 | Redpanda P99.99 | Redpanda faster by |
|---|---|---|---|
| *Small - 50MB/s* | *164.57ms (3 nodes)* | *13.91ms (3 nodes)* | *12x* |
| *Medium - 500MB/s* | *388.66ms (4 nodes)* | *16.658ms (3 nodes)* | *23x* |
| *Large - 1GB/s* | *5509.73ms (6 nodes)* | *79.574ms (3 nodes)* | *70x* |

# Comparing performance at 50MB/sec

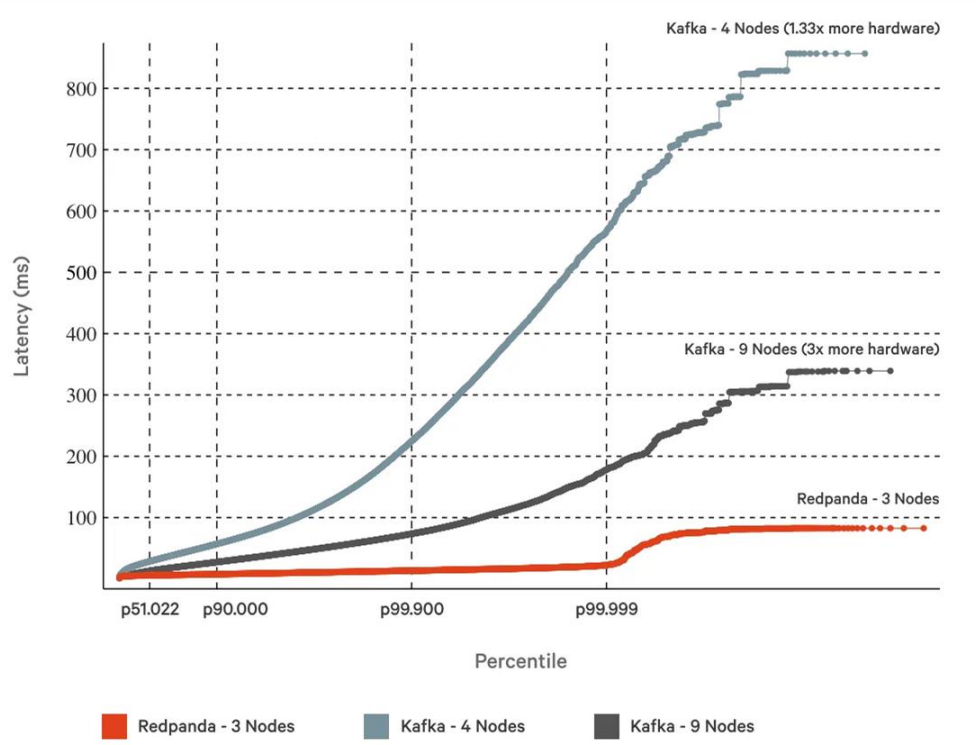



**50MB/sec**: End-to-End Latency
**Percentiles**: lower is better

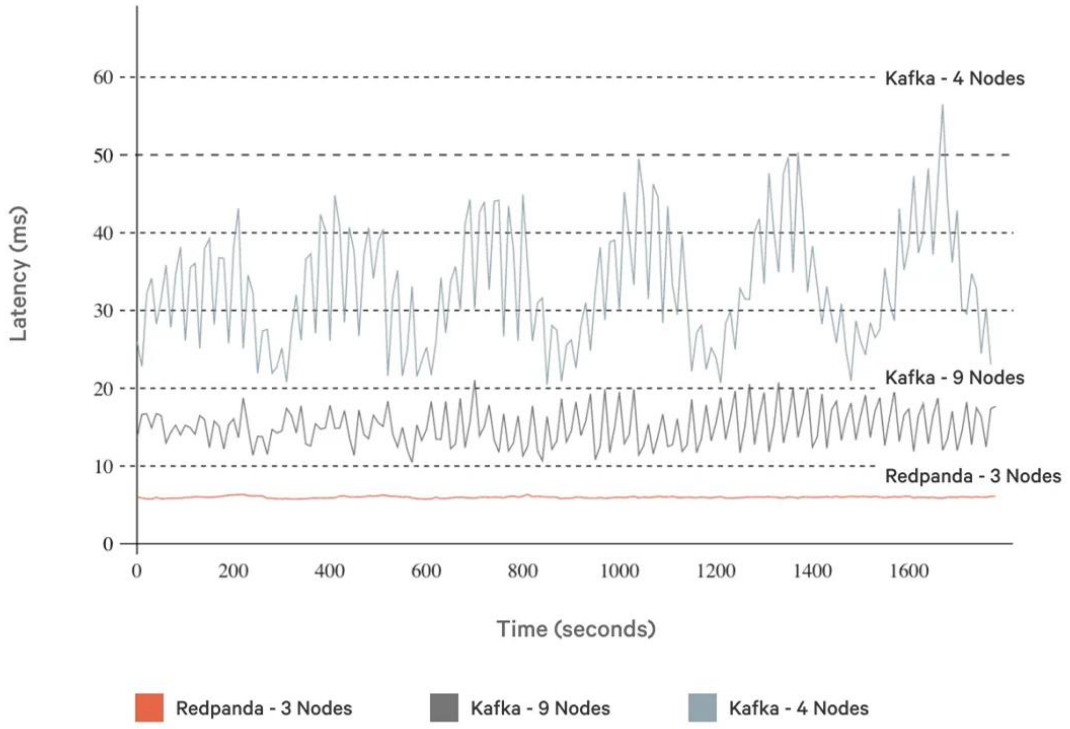



**50MB/sec**: End-to-End Latency
**Average**: lower is better

# Comparing performance at 500MB/sec

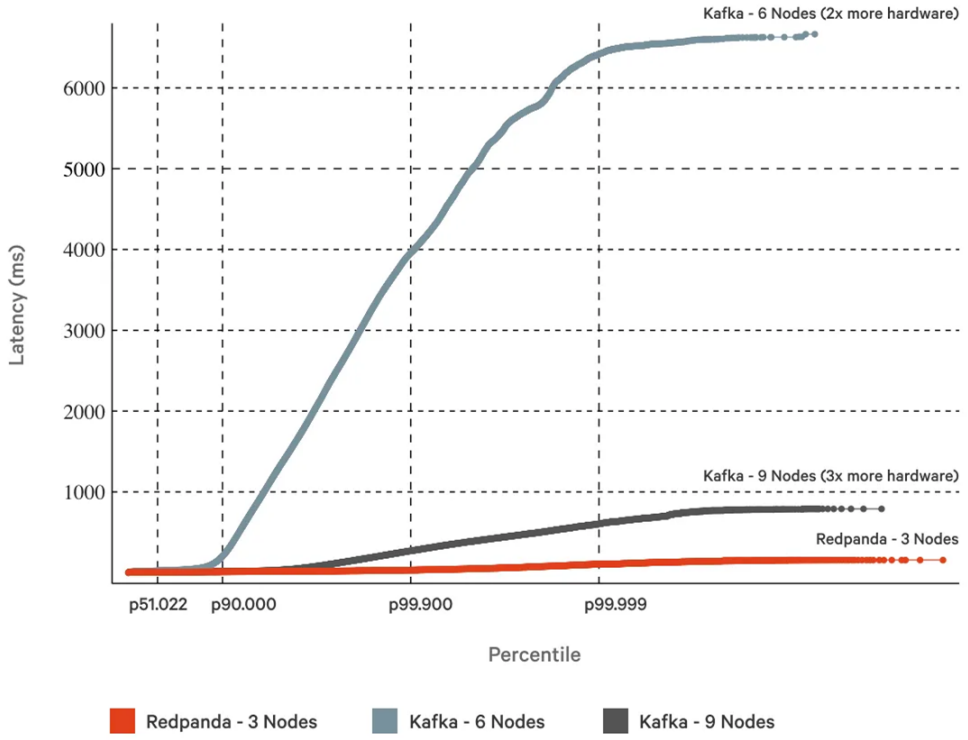



**500MB/sec**: End-to-End Latency
**Percentiles**: lower is better

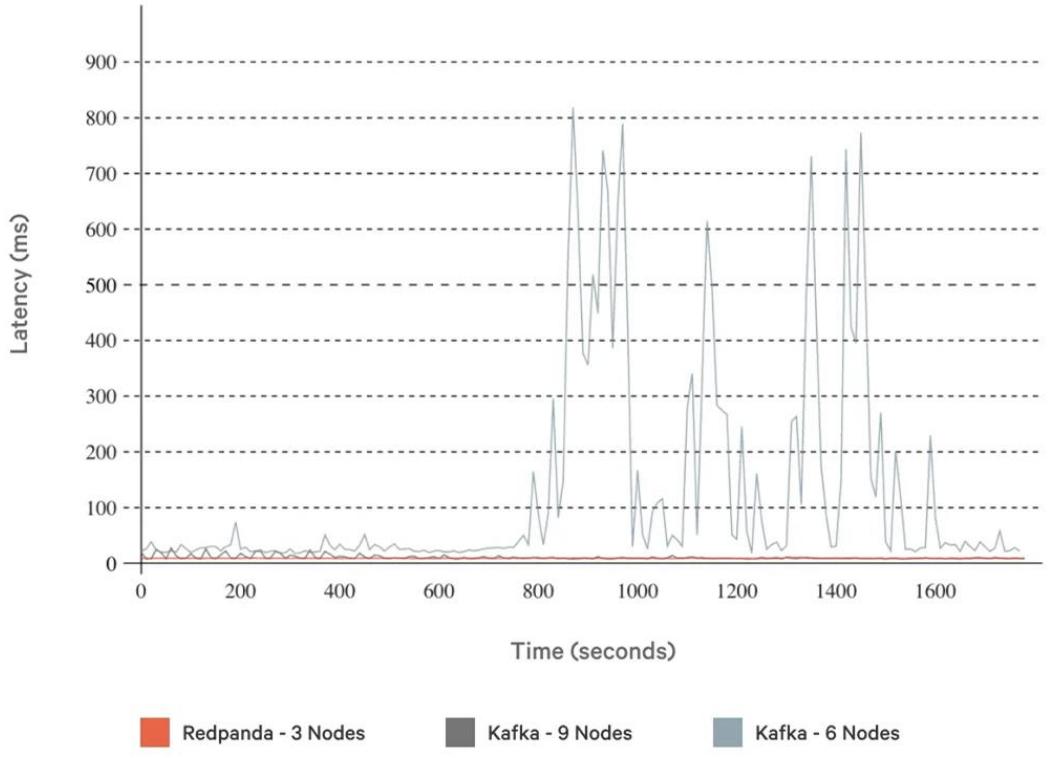



**500MB/sec**: End-to-End Latency
**Average**: lower is better

# Comparing performance at 1GB/sec




**1GB/sec**: End-to-End Latency
**Percentiles**: lower is better




**1GB/sec**: End-to-End Latency
**Average**: lower is better

# Raft consensus algorithm

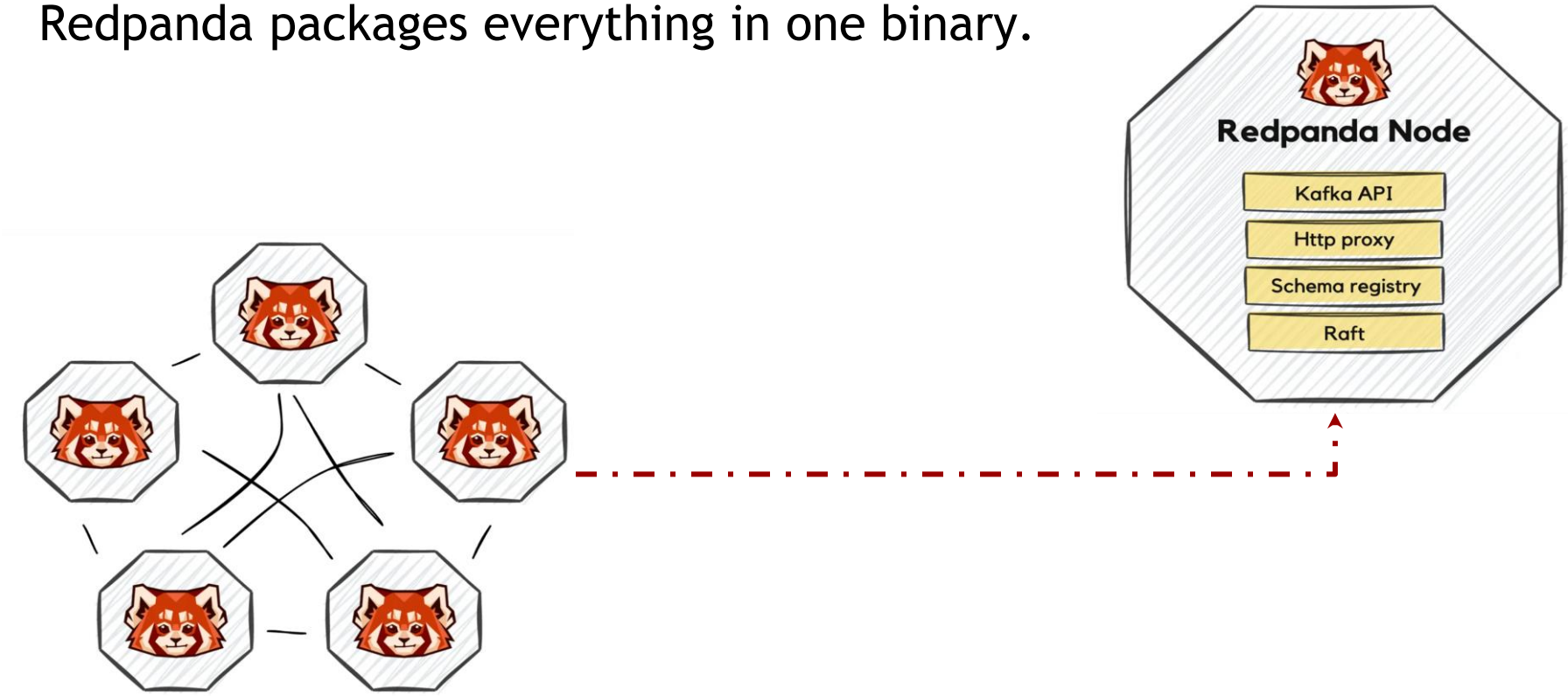
Redpanda uses the well-known Raft distributed consensus algorithm as a foundation for the distributed log.

Some of the custom extensions that exist in Redpanda Raft are:

- priority based voting
- prevotes
- reconfiguration
- learners support
- out of band heartbeat generation
- flush debouncing
- support for Apache Kafka®'s ACKS

# Single binary design

- Redpanda is distributed as a single executable binary.
- Unlike Kafka, which needs JVM, ZooKeeper, and multiple configs, Redpanda packages everything in one binary.



# Thread-per-core model

**Redpanda** uses a thread-per-core architecture

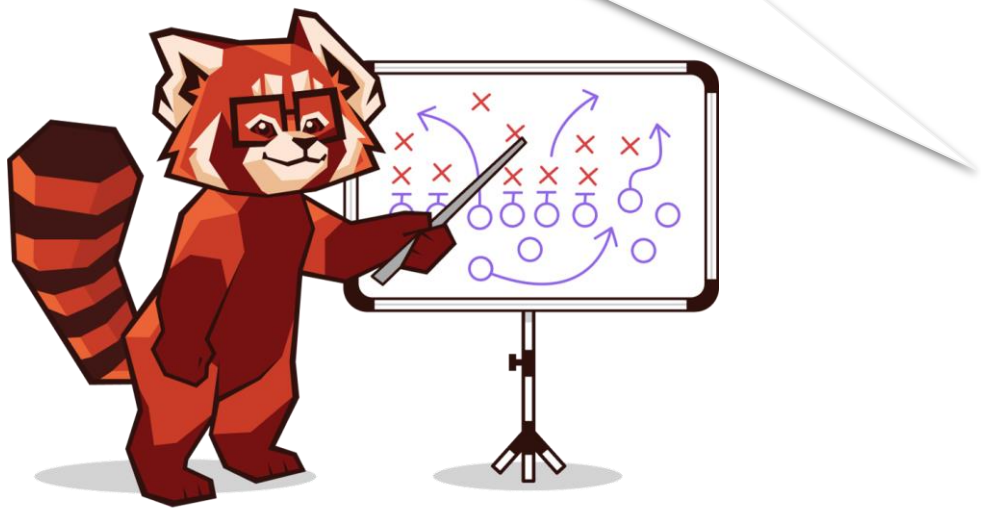
Instead of many threads competing across CPUs , each CPU core runs its dedicated thread, These threads communicate via message passing instead of locks.

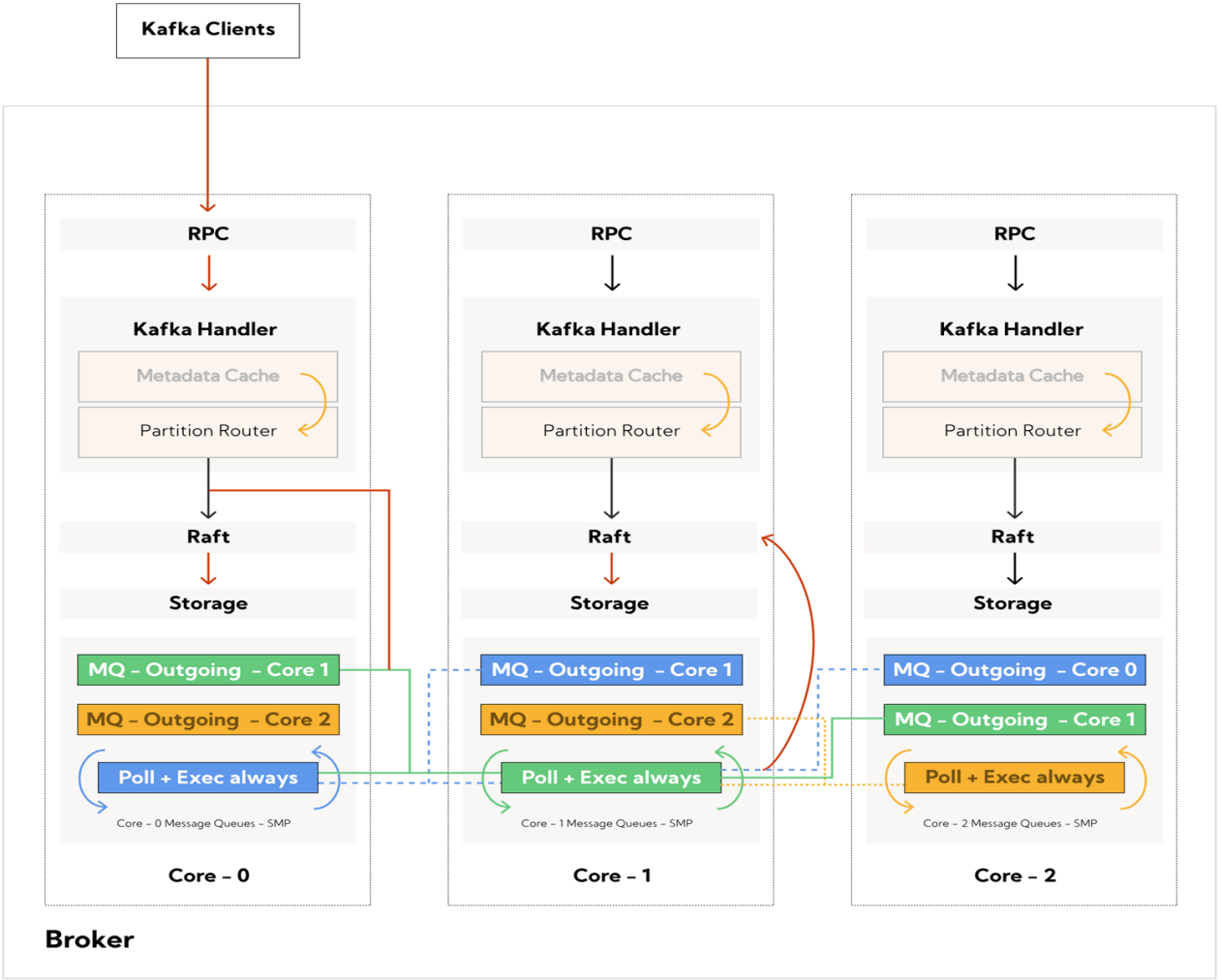
**Benefits:**

- **Ultra-low latency** → avoids lock contention and CPU bottlenecks.
- **Predictable performance** → each core works independently.
- **High throughput** → can process millions of messages/sec efficiently.

# Request flow architecture

Redpanda uses a single pinned thread per core to handle all tasks—network polling, async I/O, events, timers, and compute—ensuring no task blocks longer than 500 microseconds to avoid added latency.





# Performance metrics on hardware

In Redpanda, the C++ engine and thread-per-core model maximize hardware utilization for higher throughput, lower latency, and better efficiency.

## Why Hardware Efficiency Matters:

- Redpanda can deliver 4–10x better performance than Kafka on the same hardware.
- Reduces operational cost by doing more with fewer brokers.
- Well-suited for latency-sensitive applications (trading systems, IoT, real-time analytics).

# Redpanda Platform & Operations (DevOps/Platform Focus)

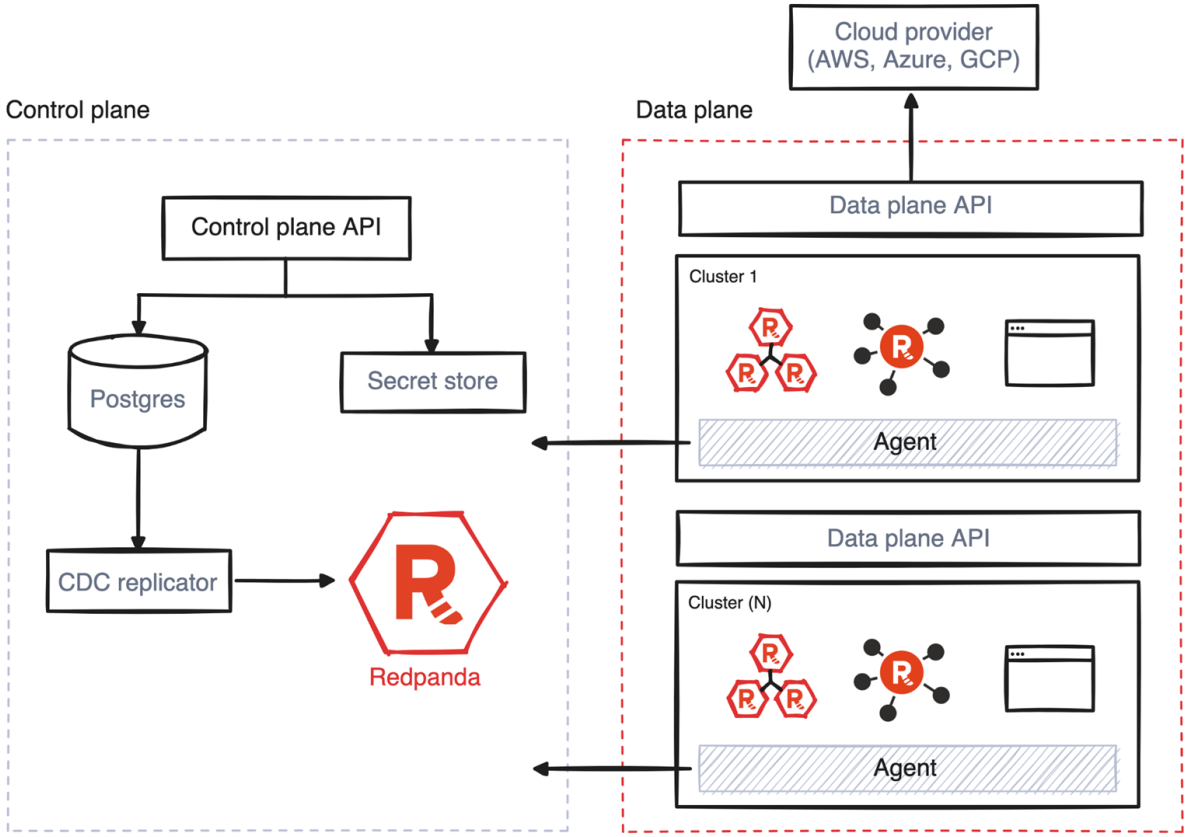
## Resilience Concepts in a BYOC Deployment

# AGENDA

- Replication, partitions, compaction overview
- Designing for durability and high availability in managed context
- Monitoring, SLAs, Dead-Letter Queues, retry handling
- Observability patterns post-MSK lessons

# Redpanda BYOC

In BYOC, the control plane manages clusters, policies, and RBAC, while the data plane runs in your VPC to handle topics, consumer groups, connectors, and schemas.



# Why Resilience Matters

**Real-world outages:** finance transactions dropped, compliance gaps.


Even if outages happen in the real world, using **Redpanda** in a **BYOC** setup ensures you still get enterprise-grade resilience while keeping control over your cloud infrastructure.

# Core Building Blocks

**Replication:**

Data copied across brokers for durability

**Partitions:**

Enable parallelism + fault isolation

**Log Compaction:**

Keeps latest state while saving storage

**High Availability:**

Cluster-level fault tolerance

**Mini quiz:**

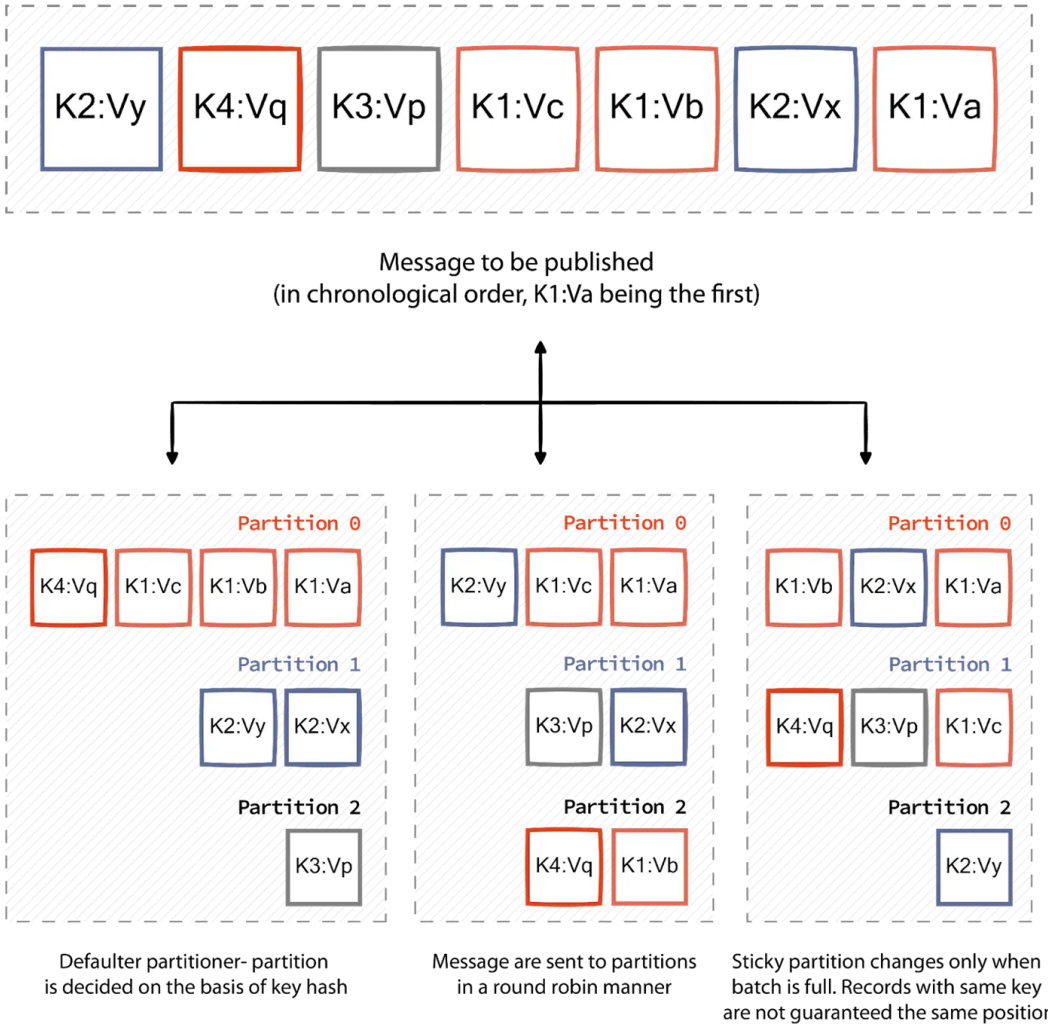
"If RF=2 and 1 broker fails, do we lose data?"

# Consumers and Kafka partitions

Kafka guarantees in-order delivery within a partition, and producer partitioning strategies decide which partition a message is written to.

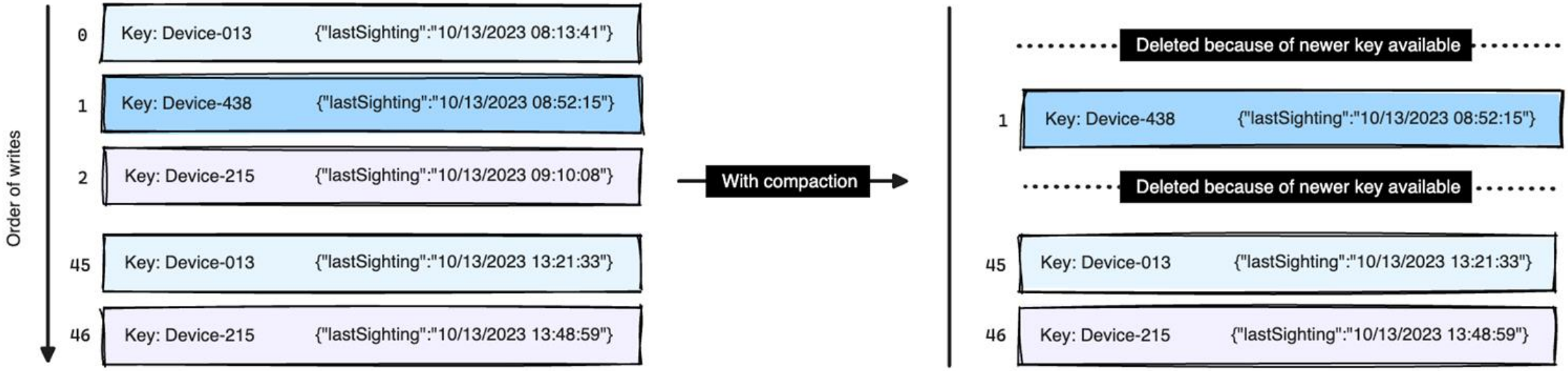
## Kafka example message:

- **Key:** "Temperature sensor"
- **Value:** "Recorded a temperature of 25°C"
- **Timestamp:** "April 26, 2024 at 1:00 p.m."



# Redpanda compaction overview

**Compaction** in **Redpanda** saves storage by keeping only the latest record per key and deleting older duplicates, ensuring topics reflect the most recent state.



# Durability and high availability in managed context

Redpanda is designed to ensure data integrity and high availability (HA), even at high-throughput levels.

**Deployment strategies:**

| Failure | Impact | Mitigation strategy |
|---|---|---|
| **Broker failure** | Loss of function for an individual broker or for any virtual machine (VM) that hosts the broker | Multi-broker deployment |
| **Rack or switch failure** | Loss of brokers/VMs hosted within that rack, or loss of connectivity to them | Multi-broker deployment spread across multiple racks or network failure domains |
| **Data center failure** | Loss of brokers/VMs hosted within that data center, or loss of connectivity to them | Multi-AZ or replicated deployment |
| **Region failure** | Loss of brokers/VMs hosted within that region, or loss of connectivity to them | Geo-stretch (latency dependent) or replicated deployment |

# HA features in Redpanda

**Replica Synchronization**

Ensures data consistency using Raft quorum-based replication.

**Rack Awareness**

Distributes replicas across racks or zones to avoid single points of failure.

**Partition Leadership**

Raft heartbeat and leader elections maintain availability during failures.

**Producer Acknowledgment**

acks settings define durability vs latency trade-offs in message delivery.

**Partition Rebalancing**

Automatically redistributes partitions when brokers are added, removed, or fail.

**Tiered Storage & Disaster Recovery**

Provides long-term retention and fast recovery of data after failures.

# Monitor Redpanda

**Redpanda** exports metrics through two endpoints on the Admin API port (default: 9644) for you to monitor system health and optimize system performance.

**Metrics Endpoints in Redpanda:**

## Metrics (Internal Metrics)

- Legacy endpoint with many low-level internal metrics
- Best for development, testing, and deep analysis

## Public_metrics (Recommended)

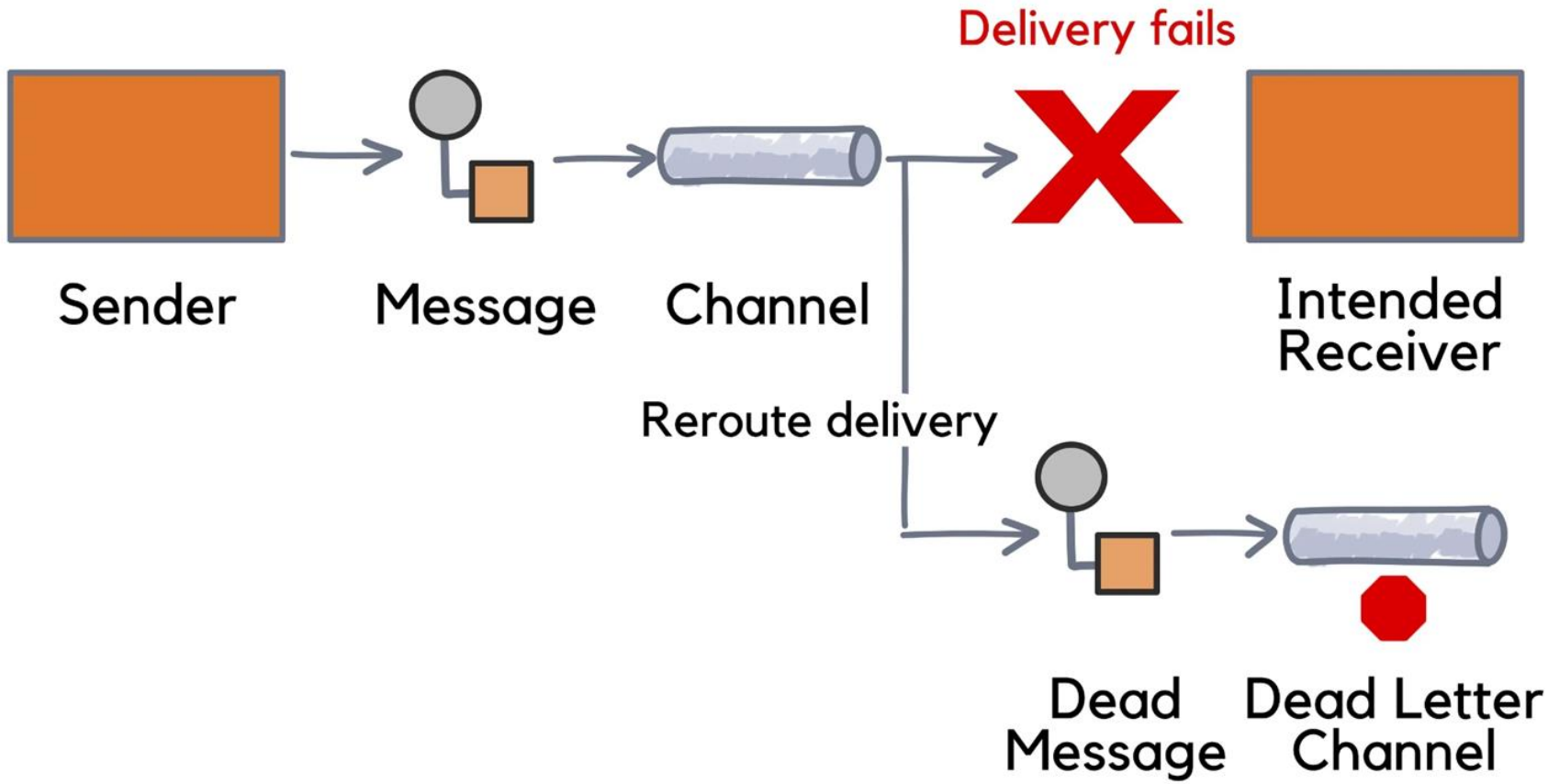
- Smaller, curated set of key metrics
- Faster to query and cheaper to ingest
- Suitable for production monitoring

# Use Redpanda monitoring

Redpanda provides a repository with examples of monitoring Redpanda with Prometheus and Grafana: [redpanda-data/observability]

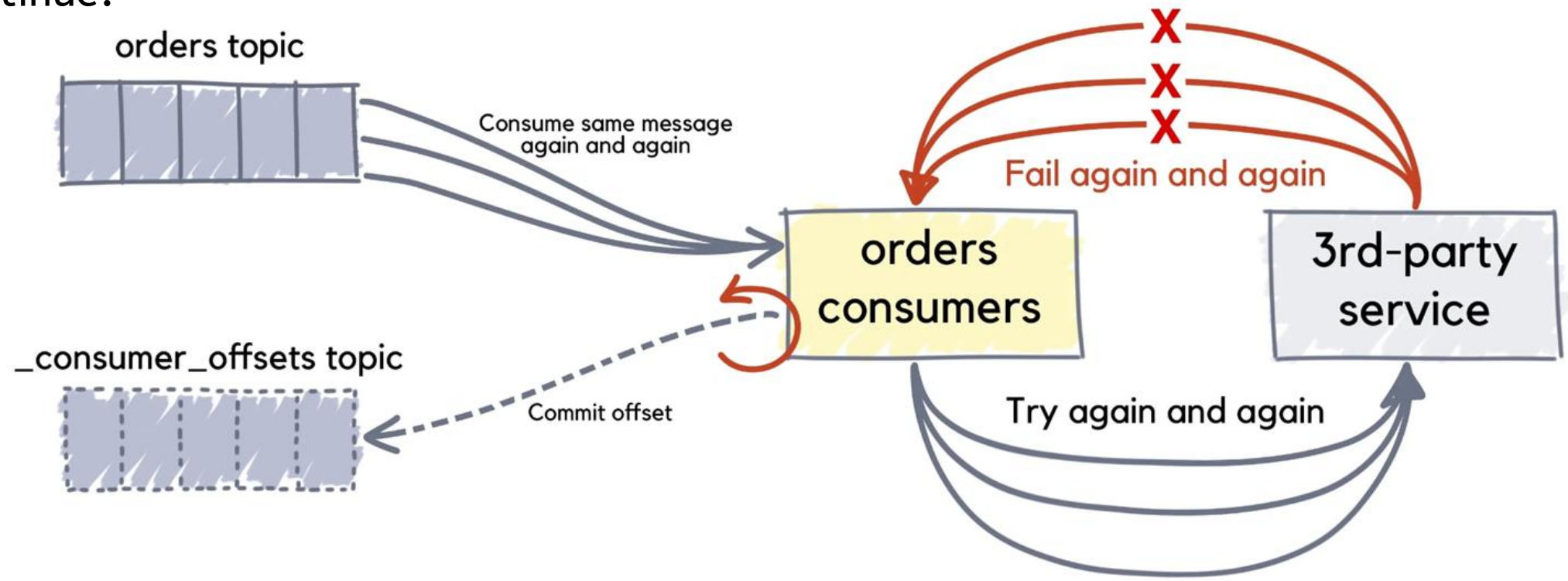
# Dead-Letter Queue

Redpanda doesn’t provide a built-in DLQ but lets you configure any topic as a custom dead-letter queue.



# Retry handling

The best way to handle transient errors is to retry the operation several times, using fixed or incremental backoff intervals. If all retries fail, the message should be redirected to the DLT to prevent blocking and allow processing to continue.



# Observability patterns post-MSK lessons

| | |
|---|---|
| **Metrics first** | Track lag, throughput, and latency on a unified dashboard to spot issues fast. |
| **Structured logging** | Enrich logs with topic/partition/offset and request IDs for quick correlation. |
| **Tracing** | Use distributed tracing to follow messages end-to-end and pinpoint bottlenecks. |
| **ConsuaAlerts & SLOsmers** | Tie alerts to SLOs (e.g., lag, error rate, p99 latency) to act before users feel pain. |
| **Failure patterns** | Detect retry storms, DLQ growth, and replica issues and trigger auto-remediation. |

# Redpanda Platform & Operations (DevOps/Platform Focus)

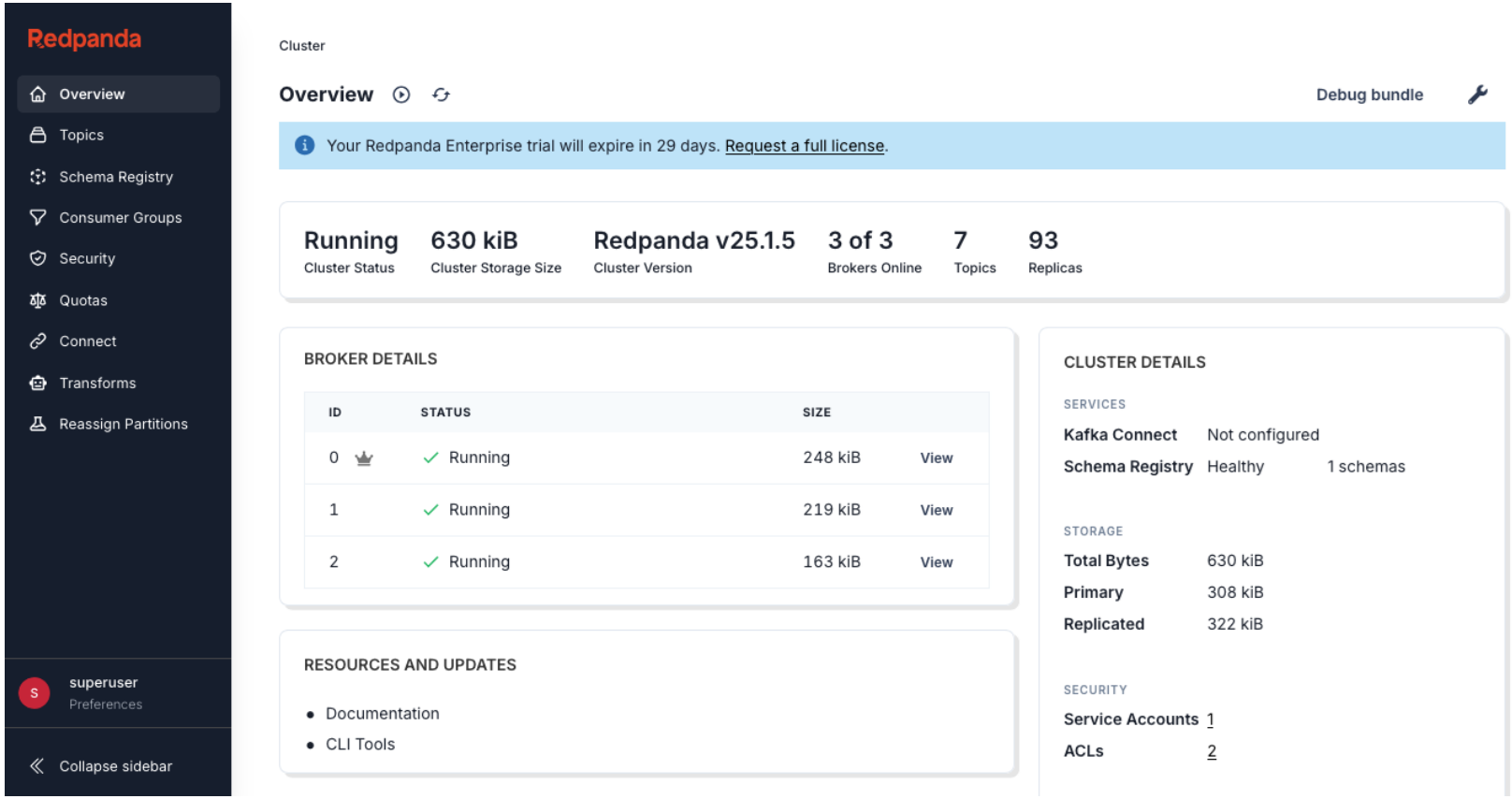
## Redpanda Console Deep Dive (with Terraform)

# AGENDA

- Console features for management, monitoring, debugging
- Admin tasks: GUI topic setup/editing
- Consumer lag, group metrics, time-travel debugging
- Access control, security configuration
- Terraform usage to automate console/admin tasks

# Redpanda Console features

Redpanda Console is a web UI for managing and monitoring Redpanda clusters, making it easier to handle streaming data, troubleshoot issues, and optimize workloads.



# Console features for managemen

- Create, edit, and delete topics
- Manage consumer groups and offsets
- Configure cluster settings
- Set up and manage access control
- Monitor and troubleshoot workloads

# Debugging features

Redpanda Console provides a web-based interface that aids in debugging and troubleshooting Redpanda clusters and Kafka-compatible workloads.

| | | |
|---|---|---|
| Consumer lag monitoring | → | detect slow consumers. |
| Message inspection | → | replay historical messages. |
| Time-travel debugging | → | view, filter, and analyze records. |
| Group metrics | → | track consumer group performance. |
| Error visibility | → | identify failed connectors or schema issues. |

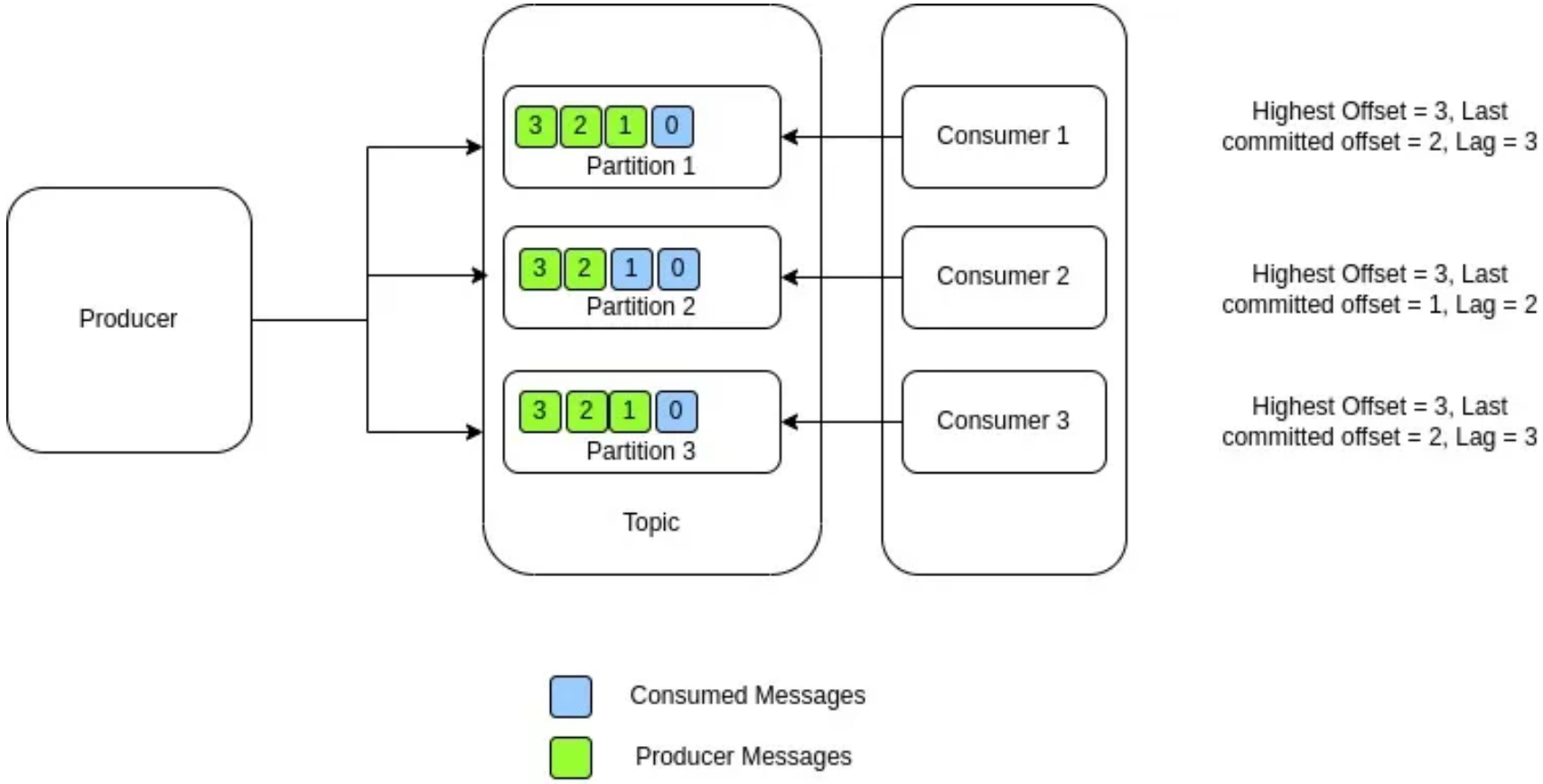
# Admin Tasks: GUI Topic Setup & Editing

Redpanda Console provides an intuitive web-based interface that makes it easy for administrators to manage topics without relying on CLI commands. Through the GUI, you can:

- **Create Topics** – Set up new topics with partitions, replication, and advanced configurations.
- **Edit Topics** – Update topic settings, scale partitions, or adjust replication as needed.
- **Delete Topics** – Remove unused topics to keep the cluster clean and efficient.
- **Provides Visibility** – View real-time topic configurations at a glance.
- **Reduces Errors** – Minimize risks compared to manual CLI commands.
- **Simplifies Management** – Manage topics easily through a no-code GUI.

# Consumer lag in Redpanda

Consumer lag is the difference between the last offset stored by the broker and the last committed offset for that partition.



# Group Metrics in Redpanda

## Why Monitor Consumer Groups?

1. Ensure consumers keep up with producers
2. Detect performance bottlenecks early
3. Maintain healthy stream processing

# Group Metrics in Redpanda

## Core Metrics

1. **redpanda_kafka_consumer_group_committed_offset**
   Last committed offset for a group on a partition

2. **redpanda_kafka_consumer_group_consumers**
   Number of active consumers in the group

3. **redpanda_kafka_consumer_group_topics**
   Number of topics subscribed to by the group

1. **redpanda_kafka_group_lag_max**
   Maximum lag across any partition in the group

2. **redpanda_kafka_group_lag_sum**
   Total lag across all partitions in the group

3. **redpanda_kafka_max_offset**
   Latest broker offset for a partition (used to calculate lag)

# Time-travel debugging

Time-travel debugging in Redpanda is a feature primarily provided through the Redpanda Console.

- **Explore historical messages** – Navigate past messages to see data at specific points in time.
- **Identify & debug problems** – Pinpoint messages or events that caused errors or unexpected behavior.
- **Gain visibility** – Visualize and analyze data streams interactively to spot anomalies.

# Access control

Redpanda uses Access Control Lists (ACLs) and Role-Based Access Control (RBAC) to manage user permissions and access to resources.

## Access Control Lists (ACLs)

Access control lists (ACLs) provide a way to configure fine-grained access to provisioned users. ACLs work with SASL/SCRAM and with mTLS with principal mapping for authentication.

## Role-Based Access Control (RBAC)

Role-based access control (RBAC) lets you manage user permissions at scale with a simple interface, working alongside all supported authentication methods. It ensures secure, consistent, and centralized access management across resources.

# Configure Authentication

Authentication verifies users and apps connecting to Redpanda, with APIs supporting multiple methods and configurable listeners.

| API | Supported Authentication Methods |
|---|---|
| Kafka API | • SASL<br>  ○ SASL/SCRAM<br>  ○ SASL/PLAIN<br>  ○ SASL/OAUTHBEARER (OIDC)<br>  ○ SASL/GSSAPI (Kerberos)<br>• mTLS |
| Admin API | • Basic authentication<br>• OIDC |
| HTTP Proxy (PandaProxy) | • Basic authentication<br>• OIDC |
| Schema Registry | • Basic authentication<br>• OIDC |

# Terraform for Redpanda Console/Admin Automation

## Why Terraform?

Codify Redpanda configuration as **Infrastructure as Code (IaC)**

Ensure **consistency and repeatability** across environments

Enable **version control** and easy rollbacks via Git

Automate **cluster + console tasks** with a single terraform apply

# Terraform for Redpanda Console/Admin Automation

## What Can Be Automated?

Provision and configure **topics** (partitions, replication, retention)

Apply **ACLs and RBAC policies** for secure access

Manage **connectors, schemas, and consumer groups**

Deploy **multi-environment setups** (Dev / Stage / Prod)


# Redpanda Platform & Operations (DevOps/Platform Focus)

## Local Development with Docker

# AGENDA

- Running Redpanda and Console via Docker/Docker Compose
- Producing/consuming test messages locally
- Local connector testing and debugging pipelines

# Docker/Docker Compose & Redpanda

- Simplifies deployment with containers
- No manual installation required
- Easy to scale and test locally
- Consistent dev/test environments

# Steps to Run

## 1. Install Docker & Docker Compose

- Ensure Docker engine is running

## 2. Create a docker-compose.yml

- Define Redpanda broker service
- Define Redpanda Console service

## 3. Start Services

- docker-compose up -d

## 4. Access Redpanda Console

- Open http://localhost:8080 in browser

## 5. Manage Topics & Monitor Metrics

- Use Console UI for topics, consumers, ACLs, connectors

With just one file + docker-compose up, you get a **fully functional Redpanda cluster + Console** running locally.

# Use Cases for Local Setup

- **Developer testing:**

  Quickly test producers/consumers without cloud infra

- **Demo/Training environments:**

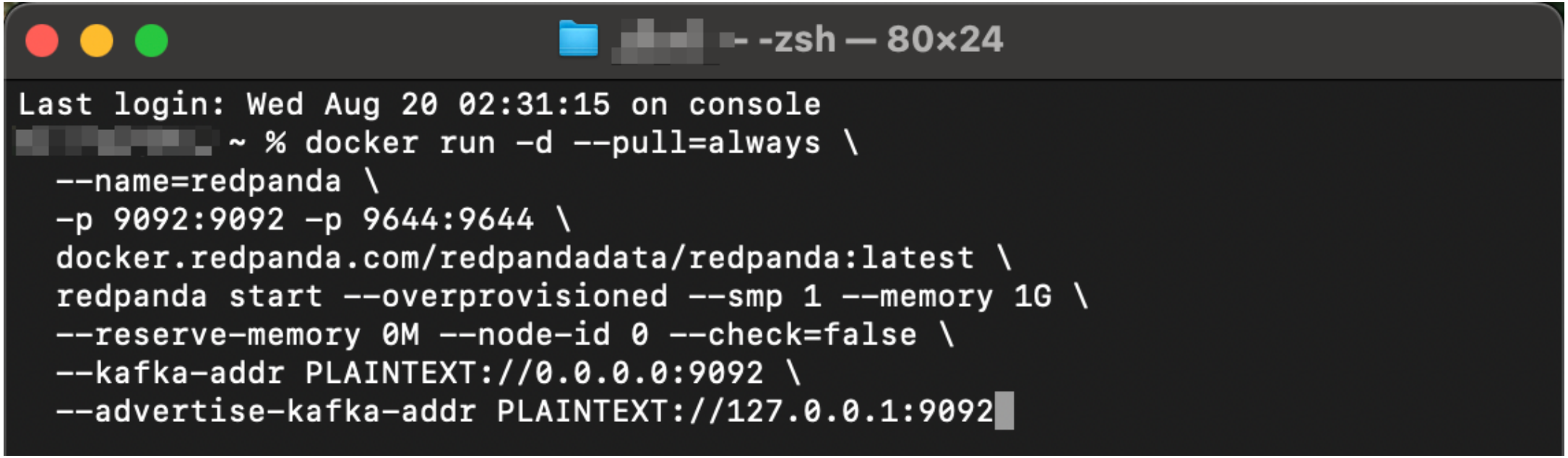
  Spin up Redpanda + Console in seconds

- **CI/CD pipelines:**

  Validate Kafka apps against a real broker in containerized tests

# Producing/consuming test messages locally

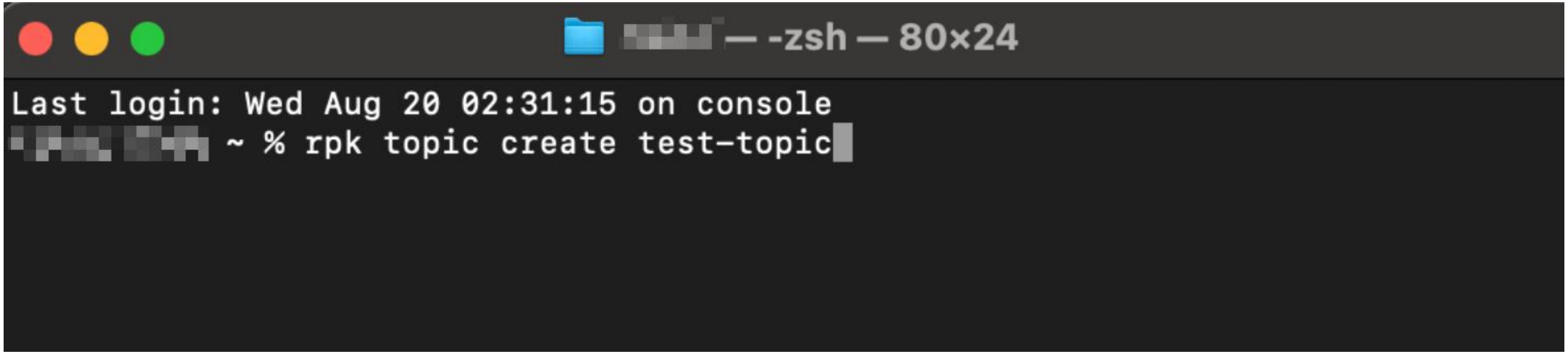
**Step 1:** Start Redpanda




This command runs a **single-node Redpanda cluster** in Docker with 1 CPU, 1GB RAM, exposing **9092 (Kafka)** and **9644 (Admin)** for easy local testing.

# Producing/consuming test messages locally

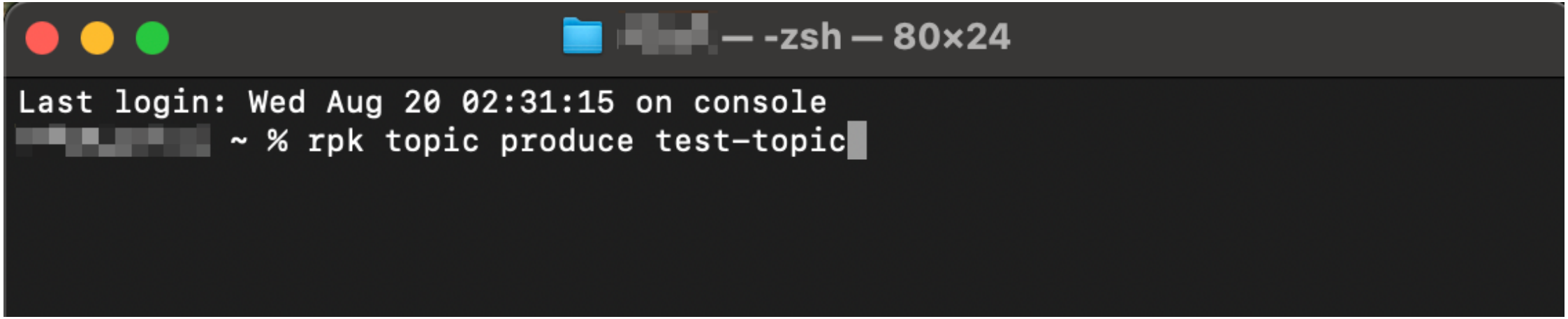
**Step 2:** Create a Topic




- rpk is the Redpanda CLI tool (Redpanda Keeper).
- topic create is the command to create a new topic in Redpanda.
- test-topic is the name of the topic being created, which will be used later for producing (sending) and consuming (reading) test messages.

# Producing/consuming test messages locally

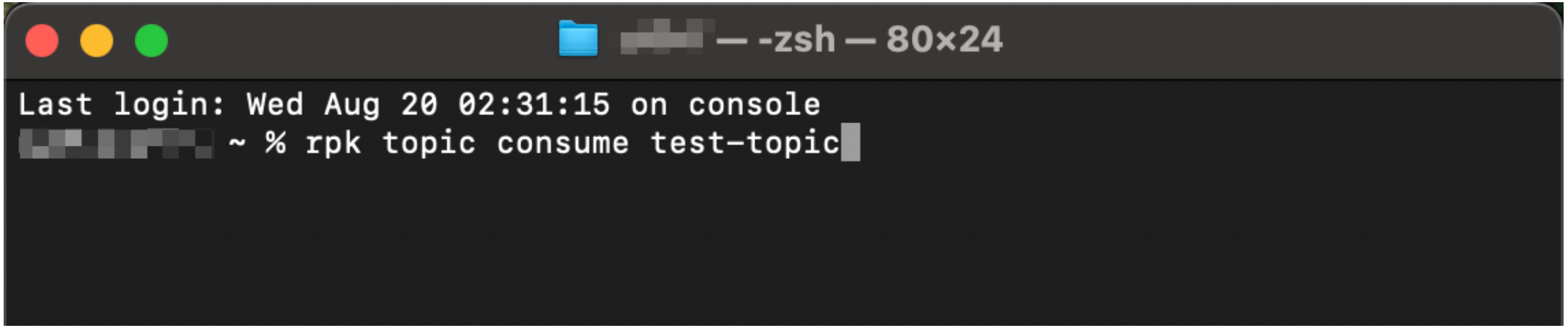
**Step 3:** Produce Messages




- **rpk** → Redpanda's CLI tool (similar to Kafka's kafka-console-producer.sh).
- **topic produce** → Command to produce (send) messages into a topic.
- **test-topic** → The topic name where the messages will be written.

# Producing/consuming test messages locally

**Step 4:** Consume Messages




- rpk → The Redpanda CLI tool.
- topic → Subcommand to manage topics.
- consume → Starts a consumer that listens for messages.
- test-topic → The name of the topic to consume from.

# Why Local Testing?

1. Safe environment to test connectors before production
2. Faster debugging cycle with minimal setup
3. Identify schema, serialization, and transformation issues early
4. Reduce risk of failures in live systems

# Local Setup

**Step: 1**

Run **Redpanda + Connect + Console** via Docker Compose

**Step: 2**

Place connector JARs in /var/lib/kafka-connect/plugins

**Step: 3**

Create **test topics:** test-input, test-output

**Step: 4**

Use **rpk** CLI to produce and consume test events


THANK YOU