Integration Layers

Central Region Lunch and Learn

Red Hat

Stephen Nimmo

Staff Specialist Solution Architect

20+ years of software development and delivery experience
BBA from Stephen F. Austin State University
MBA from University of Houston, specializing in energy trading and risk

Technical Skills

- High Level Software Architecture
  - API Development and Management
  - Event Driven Architecture
  - Monolith to Microservice
  - Enterprise Integration Patterns
- Kubernetes/OpenShift Platform
- Extensive DevOps Experience
  - CICD, Automation, Culture, Team Management, GitOps
- Java and .NET - Server Side Development
  - Quarkus, Vert.x, Spring, .NET Core, C#, Entity Framework, JMS, AMQP, Kafka, Data Caches
  - Some python, node.js, angular
- Agile coach - Former Scrum Master and Product Owner

Business Domains

- Commodity Trading - Front and Middle Office
  - Futures, Options, Swaps, Risk
  - Exchange integration, Confirmations, Risk Modeling
  - Compliance - Dodd Frank, Trade Surveillance
- Natural Gas Pipelines
  - Allocations and Balancing, Scheduling, Contracts
- Financial Services
  - Equities trading, low latency trading, FIX infrastructure
- Retail Power
  - Trading - hedging and forecasting, Contracts, Products, Pricing, Meter Data Management
- Petrochemical Inspections

Email: nimmo@redhat.com
Mastodon/Twitter: @stephennimmo
Blog: https://stephennimmo.com

Red Hat

Agenda

- Patterns for Integration
- Streaming ETL
- Best Practices

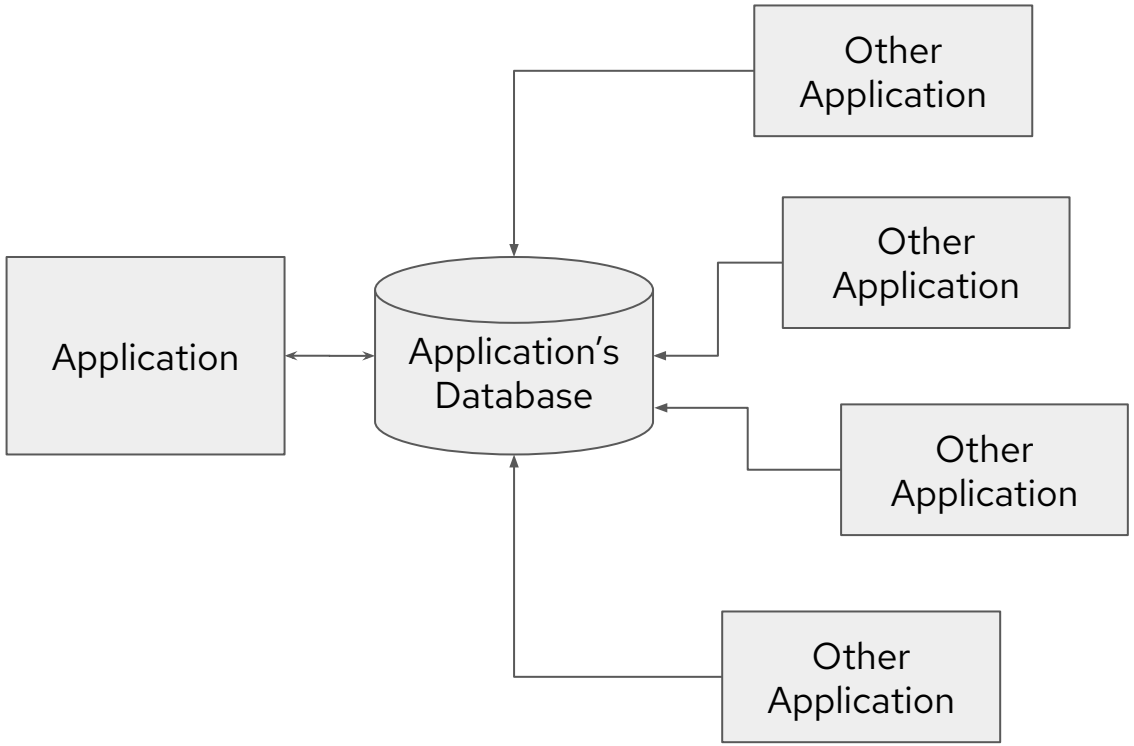
Patterns for Integration

This isn't integration

Stop doing this

What Happens When:

- Everyone is polling at the same times
- The other applications start executing huge joins against the database
- The execution plans and indexing structures become overloaded to optimize for all the other applications
- The primary application team needs to make structural changes to the database schema



Red Hat

Characteristics of Good Integration

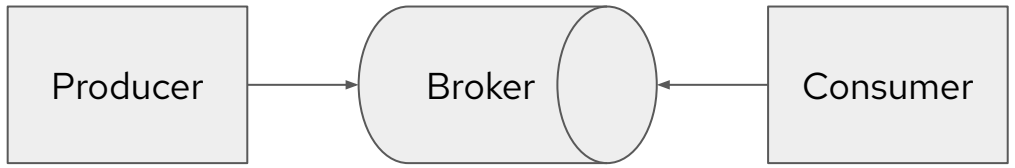
Agility

- Abstraction - the systems should not be aware of the internal implementations of the other systems
- Clear Boundaries - each system should have clearly defined boundaries both in terms of functionality and structure
- Standardized - boundaries should be defined using stable, industry standard protocols and structures
- Easily Testable - the integration logic should be able to be easily validated using automated testing capabilities

Producer and Consumer Pattern

Canonical Transformation Pattern

Messages are formatted using a domain driven, system agnostic canonical data model

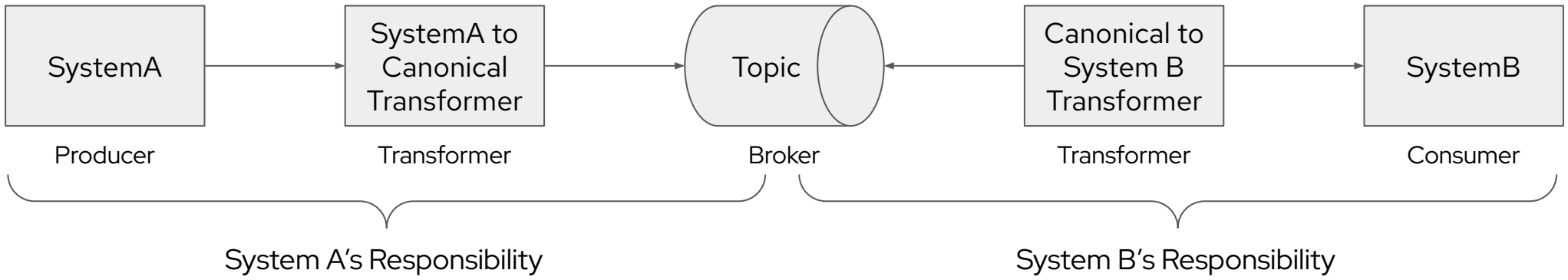


Canonical data models not only include details about the structure, but also concerns around standardization of data values as well such as enumerated data types and entity keys

- The **Producer** publishes the data to the broker in a canonical format
- The **Consumer** consumes the canonically formatted data and performs any additional work necessary
- The **Producer** is unaware of any **Consumer** systems
- The **Consumer** is unaware of the **Producer** system(s)
- However it's rarely this simple...

Basic Producer and Consumer Pattern

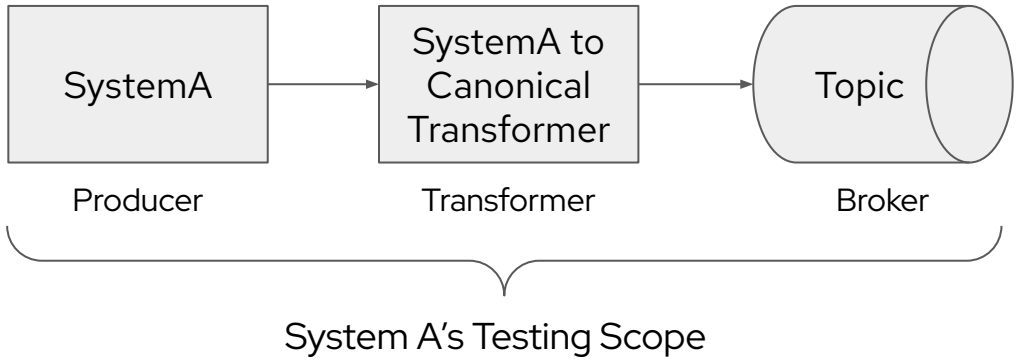
Canonical Transformation Pattern



In this example, System B not only has no knowledge of System A's internal implementation details, but in fact does not even know System A exists.

Basic Producer and Consumer Pattern

Single System Responsibility



When changes are made to System A, as long as the set of automated regression tests prove that the same expected canonical data model events are created, then all downstream systems can be confident as well

- System A does not require knowledge of the canonical data formatting requirements
  - Great for use in legacy or COTS systems
- Transformer logic is isolated and highly testable via mocking
- System testing becomes much more valuable as initiation of an event can be tested from end to end
- System A can now make huge changes to it's internal implementation without affecting any other systems

COTS = Commercial Off the Shelf

What's the Payoff?

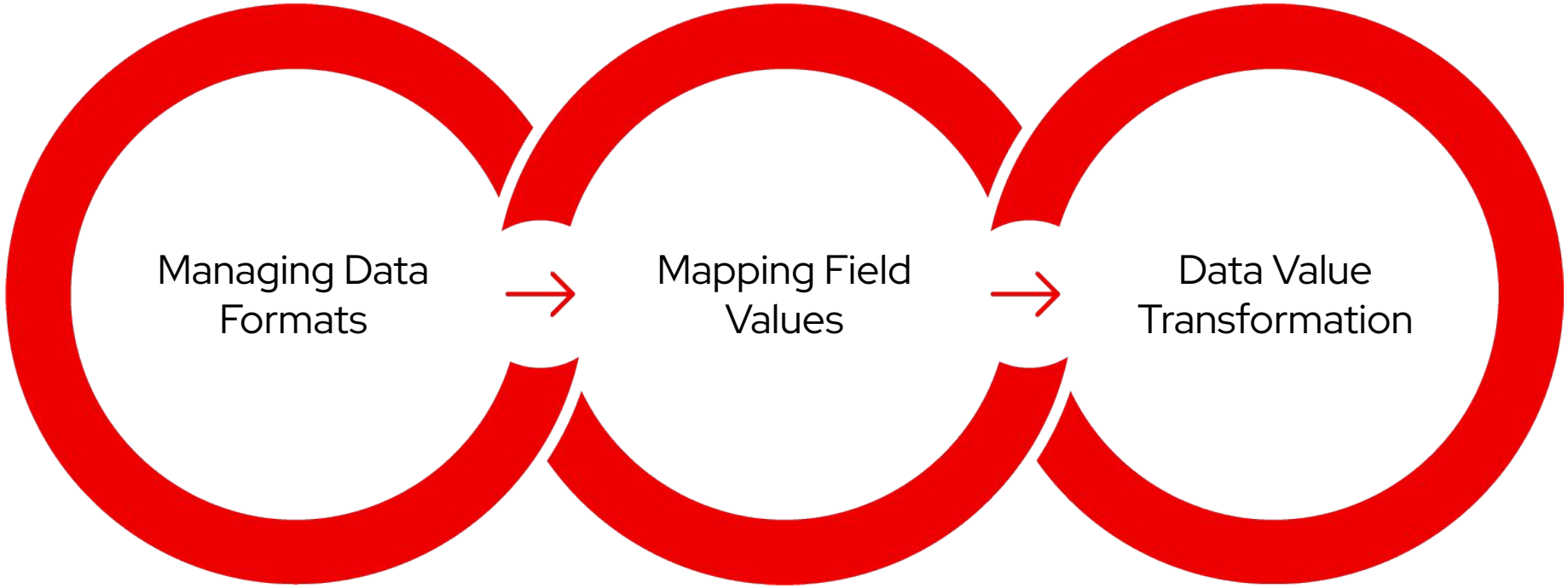
Huge

- Reduced or eliminated cross-team dependencies
- Highly agile integration components
- New producers and consumers of event streams without any infrastructure changes or additional message routing logic
- Economies of scale created through shared components such as support systems and shared transformation libraries

Streaming ETL

The Big Three of Transformation

The Foundation of Integration



Red Hat

Managing Data Formats

Schema Management

- Schemas should be defined using industry standard protocols
  - Ex. JSON, AVRO, Protobuf
- Schemas should not only be stored in version control, but they should also be stored in a generally available service registry
- Schemas should be versioned and follow compatibility rules
  - Backward, Forward, Full, Transitive
- The organization should have created and communicated the policies and procedures related to introduction of breaking changes into schemas including deprecation windows

Mapping Field Values

Structure to Structure Field Mappings

GUI-based field mappers and other low code techniques sound like a perfect fit but beware of the edge use cases requiring additional complexity.

Most times, it's easier just to code it.

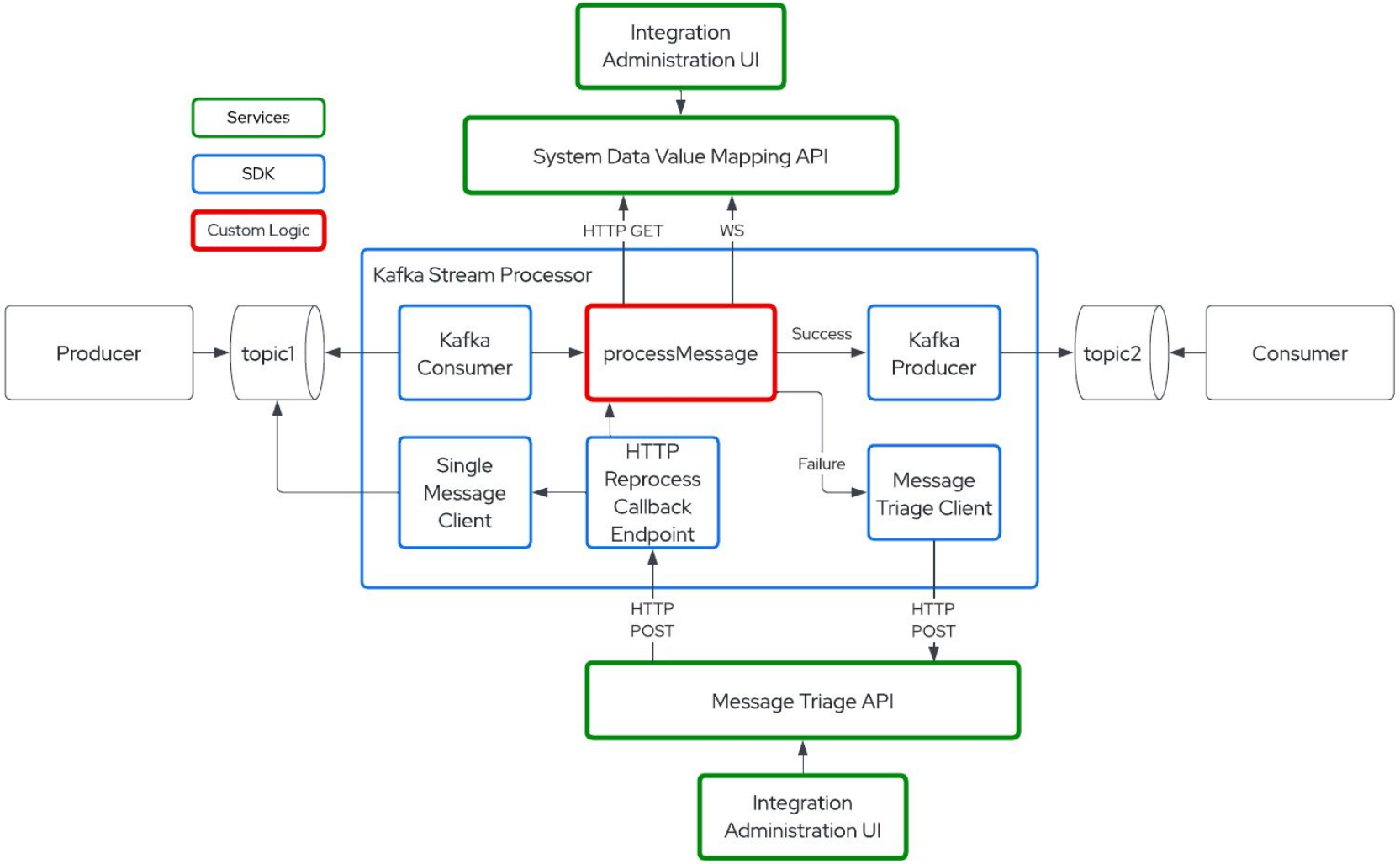
- The next step in transformation involves mapping data from fields on the source data structure to the target data structure
- While a majority of these mappings tend to be simple field-to-field data copies, there can be additional complexities such as a single source field that could result in multiple target fields or multiple source fields could result in a single target field
- As the cyclomatic complexity increases, so do the requirements to create and maintain test classes to sufficiently cover the known use cases

Data Value Transformations

The Hard Part

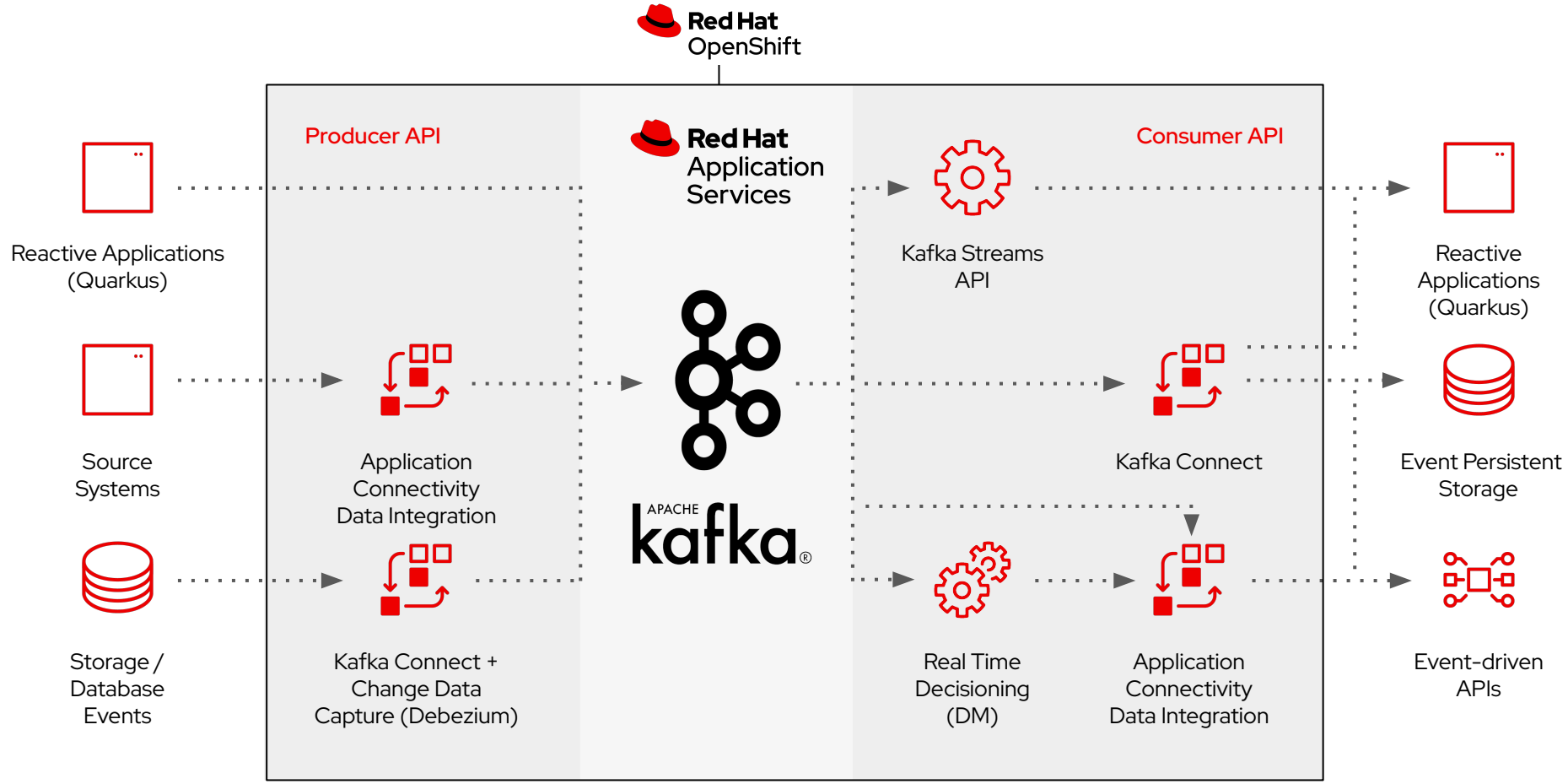
- Different systems have different values for data. Some of these are simple and trivial, while some are complex.
  - **Simple:** String capitalization, date formats
  - **Complex:** Differing enumerated values, domain entity keys
- These data value transformations are present on both sides of the event stream as producing systems convert to canonical and consuming systems convert from canonical
- There will need to be a system for not only maintaining these mappings between systems but also for handling message transformation errors associated with missing mappings

Example Streaming ETL Pattern



Red Hat

Event Management Bus



Source

Red Hat

Red Hat Technology Stack

Supporting Streaming ETL

Red Hat AMQ Streams

The Red Hat® AMQ streams component is a massively scalable, distributed, and high-performance data streaming platform based on the Apache Kafka project.

Red Hat OpenShift Container Platform

Red Hat OpenShift, the industry's leading hybrid cloud application platform powered by Kubernetes

Red Hat Fuse

Red Hat® Fuse is an agile, lightweight, and modern integration platform that enables rapid integration across the extended enterprise

Red Hat Runtimes

Red Hat Runtimes is a set of products for developing and maintaining cloud-native applications. It offers lightweight runtimes and frameworks, like Quarkus.

- Apache Kafka
- Kafka Connect
- Red Hat Service Registry
- Red Hat Single Sign-On
- Apache Camel
- Red Hat Quarkus

Best Practices

Best Practices for Integration

People, Processes, Tools

- Start by building the community
- Focus on domain driven canonical interface definitions for eventing
- Fully automated CICD pipelines for integration components
- Practices and communications regarding API and schema management organizational behaviors
    - Schema compatibility, deprecations and ways of working
- Create economies of scale through innersourcing shared resources such as canonical data structure libraries and building shared services

Thanks

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat