Red Hat Summit

Connect

Messaging czy Streaming?

Andrzej Kowalczyk

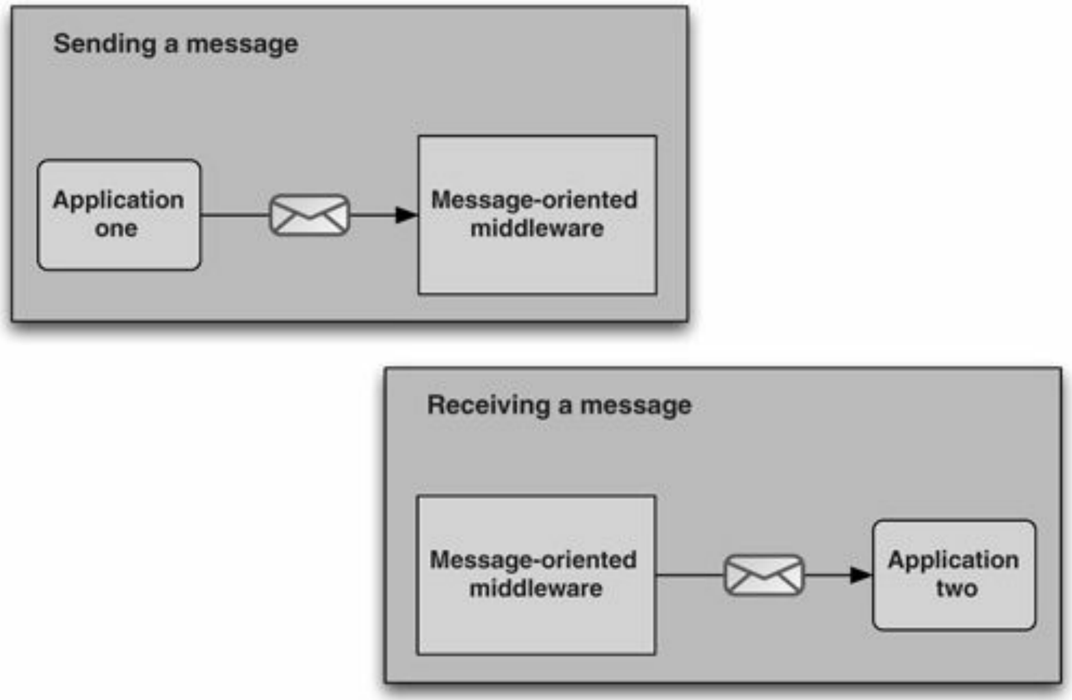
Red Hat | intel.

Event Driven Architecture

Introduction: Events

What is Event-Driven Architecture?

Event-Driven Architecture (EDA) is a way of designing applications and services to respond to real-time information based on the sending and receiving of event notifications



Source: https://www.gartner.com/en/information-technology/glossary/eda-event-driven-architecture :
Image: ActiveMQ in Action https://www.manning.com/books/activemq-in-action

Why Event-Driven Architecture or EDA?

Mirrors the real world

The real world is event-driven. Systems generate and respond to events in everyday life, e.g., the human central nervous system.

Reduced coupling

Traditional RPC-style service architecture results in tightly-bound services. Changes to the application flow typically require service code changes. EDA allows new functionality to be added by adding services that consume existing event streams.

Encapsulation

Microservices concepts have grown in popularity due to the ability for service teams to develop services in isolation. EDA means that service designers need not be aware of how events are consumed.

Fine-grained scaling

Services can be independently scaled up and down to meet the event volume.

Near real-time latency

Customers increasingly expect a near real-time experience. Polling on APIs is a delicate trade-off between responsiveness and load. EDA allow apps to react in near real-time without compromise.

Source:https://developers.redhat.com/topics/event-driven/

What is an event?

Event an action or occurrence recognized by software, often originating asynchronously from the external environment, that may be handled by the software

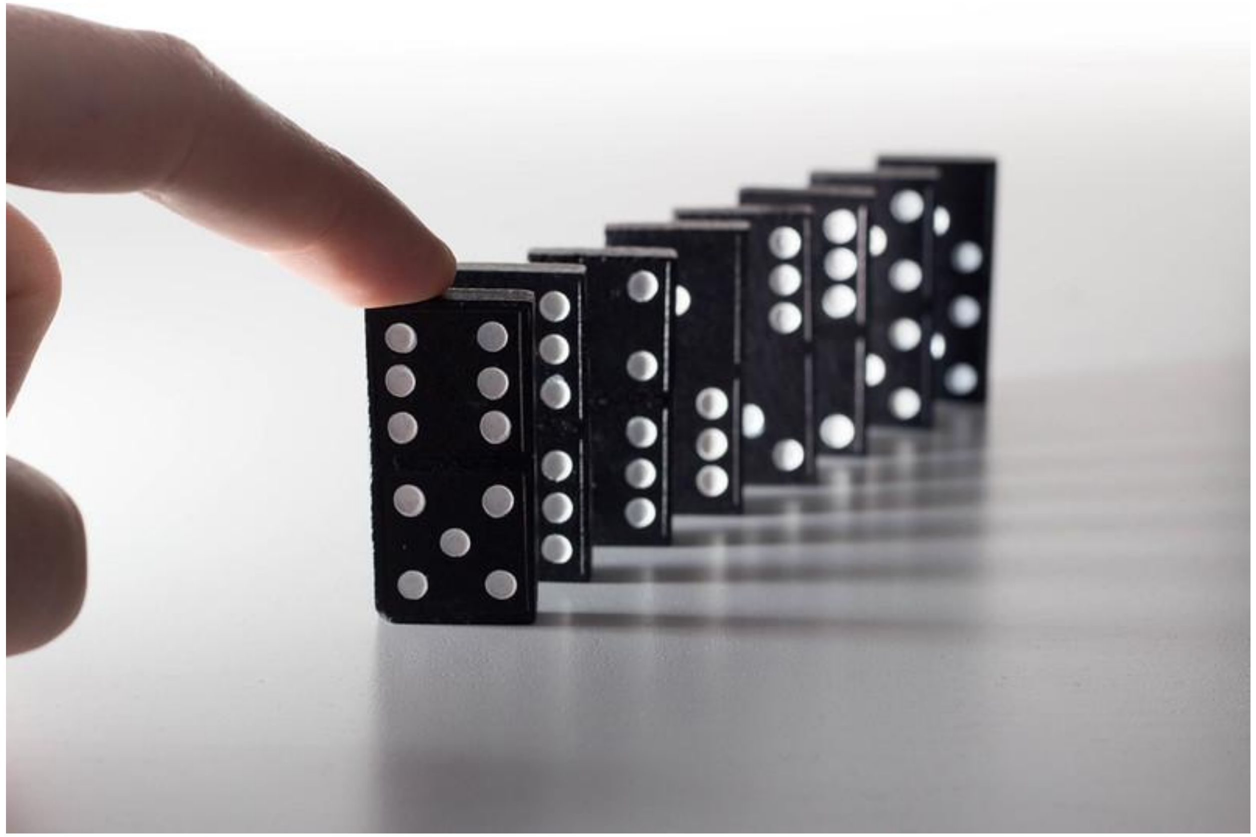


Image: https://foto.wuestenigel.com/hand-ready-to-push-domino-pieces/

What is an event?

Event

Immutable state and value of a particular entity, which occurred during operation among services.

Command

Async form of Remote Procedure Call, contains instructions telling recipient what to do, may cause a change of state.

Query

Similar to commands, queries expect a response returning the results, but do not cause any change in state.

Source: https://developers.redhat.com/topics/event-driven/

Types of event consumption patterns

Volatile

The event needs to be disseminated to all consumers online at time of publication. Not persisted.

Durable

Events stored durably until read by all registered consumers. Traditional store-and-forward brokers.

Replayable

Events stored durably for specific period of time or storage capacity. Consumers can move back and forth of the stream.

Source: https://developers.redhat.com/topics/event-driven/

How to choose the right implementation?

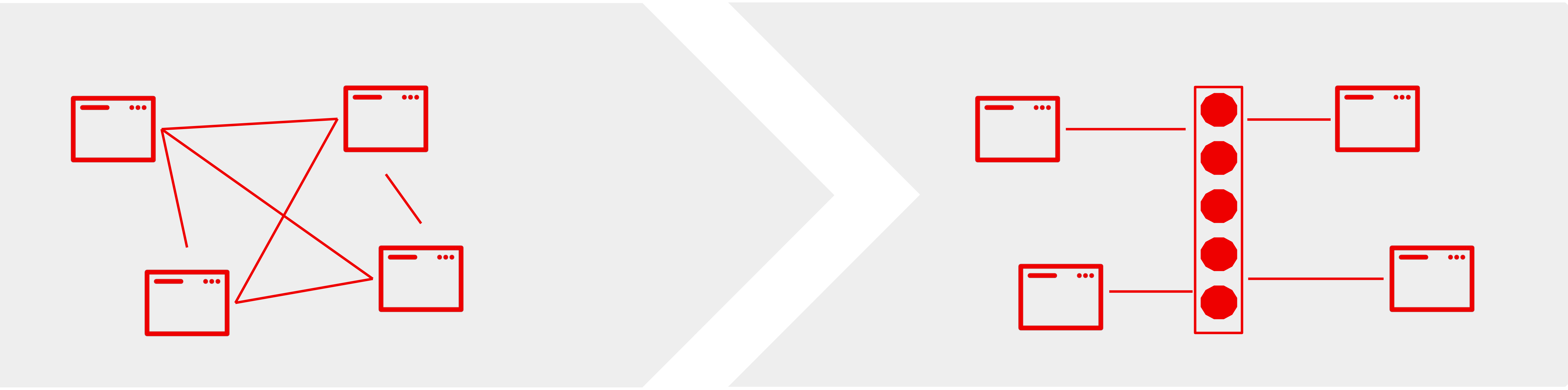
EXAMPLES	BEST FIT PATTERNS	POSSIBLE PATTERNS	UNSUITABLE PATTERNS
Traditional messaging broker w/ Queuing & Pub/Sub Rich feature set inc. JMS 2.0 Message Brokers like Apache ActiveMQ Artemis	DURABLE EVENTS	VOLATILE EVENTS COMMAND QUERY	REPLAYABLE EVENTS
Message router 1:1 (anycast) and 1:many (multicast) Secure messaging backbone for hybrid cloud Message Routers like Apache Qpid Dispatch Router	COMMAND QUERY VOLATILE EVENTS		DURABLE EVENTS REPLAYABLE EVENTS
Enterprise distribution of Apache Kafka Simplified deployment on Kubernetes/OpenShift Like Kafka on Kubernetes based on Strimzi	REPLAYABLE EVENTS DURABLE EVENTS	COMMAND QUERY	VOLATILE EVENTS TRANSACTED/FILTERED EVENTS

Source: https://developers.redhat.com/topics/event-driven

Event-driven Microservices

Microservices Async Communication

Foundation for event-driven microservices



- High availability: No dependency on other services
- Autonomy in services with independent evolution

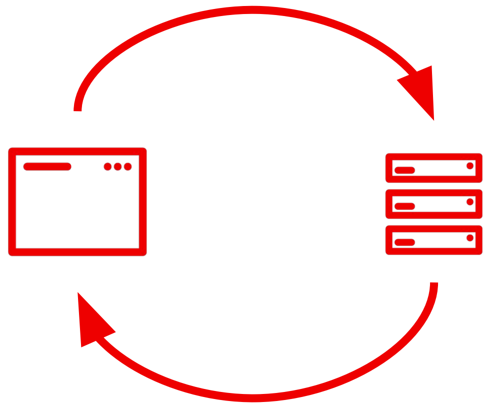
Source:

Event-driven architecture use cases

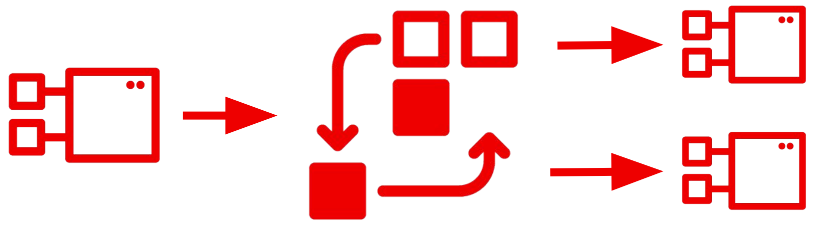
- Reactive notification
- Behavior capture
- Cache store
- Complex event processing
- Command query responsibility segregation (CQRS)
- Streaming between data centers
- Auditing

Source:

Request-reply & Event-driven



Synchronous & ephemeral
Low composability
Simplified model
Low tolerance to failure
Best practices evolved as REST

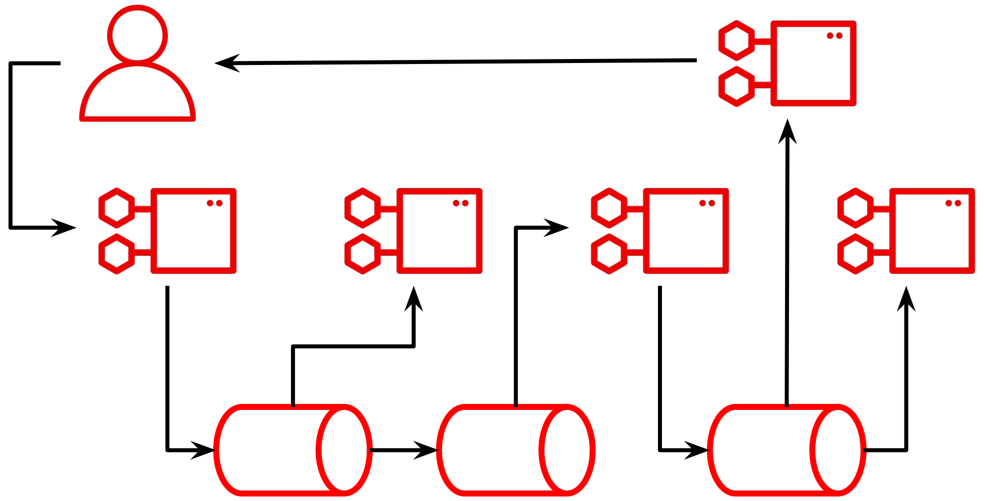


Asynchronous and persistent
Decoupled
Highly composable
Complex model
High tolerance to failure
Best practices are still evolving

Source:

Connect loosely-coupled microservices

Remain agile with event-centric microservice architecture



Connect microservices and stay agile

- Publish events to Kafka brokers and decouple the data from the event-consuming services
- Meet event volumes by independently scaling up and down your microservices
- Avoid hard-coding integrations and connections between microservices applications

Comparing Traditional Messaging and Event Streaming

Messaging

- Store-and-forward
- Individual message exchanges (transactions, acknowledgment, error handling/DLQs), P2P/competing consumer support
- Publish-subscribe support

Trade-offs

- No replay support
- Requires fast and/or highly available storage infrastructure
- No ordering at scale

Event Streaming

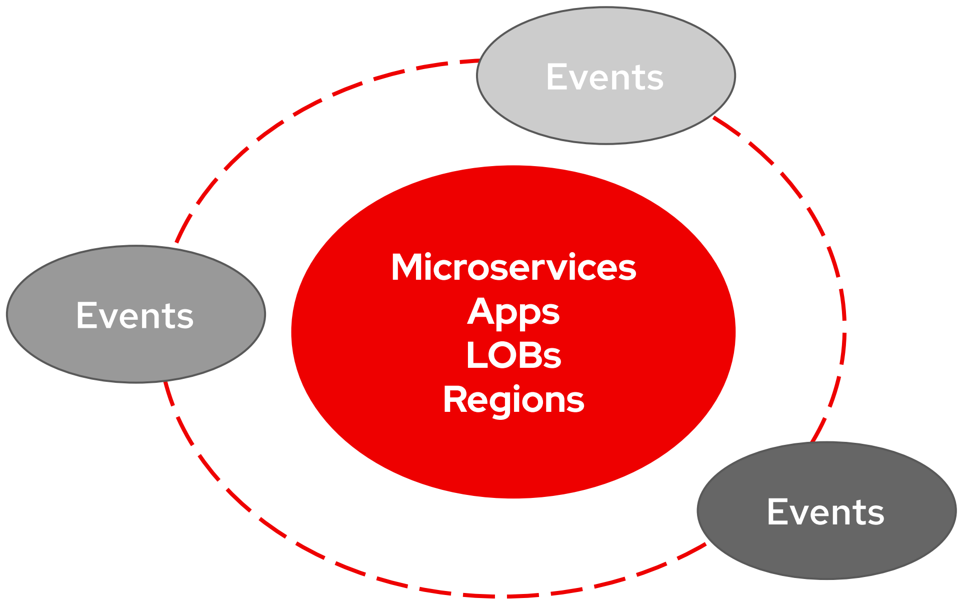
- Long-term persistence, semantic partitioning, large publisher/subscriber imbalances, replay and late-coming subscribers
- Shared nothing data storage model
- Repeatable ordering at scale

Trade-offs

- No out of order acknowledge, weak support for request-response
- Larger data footprint and extremely fast storage access
- High initial entry cost

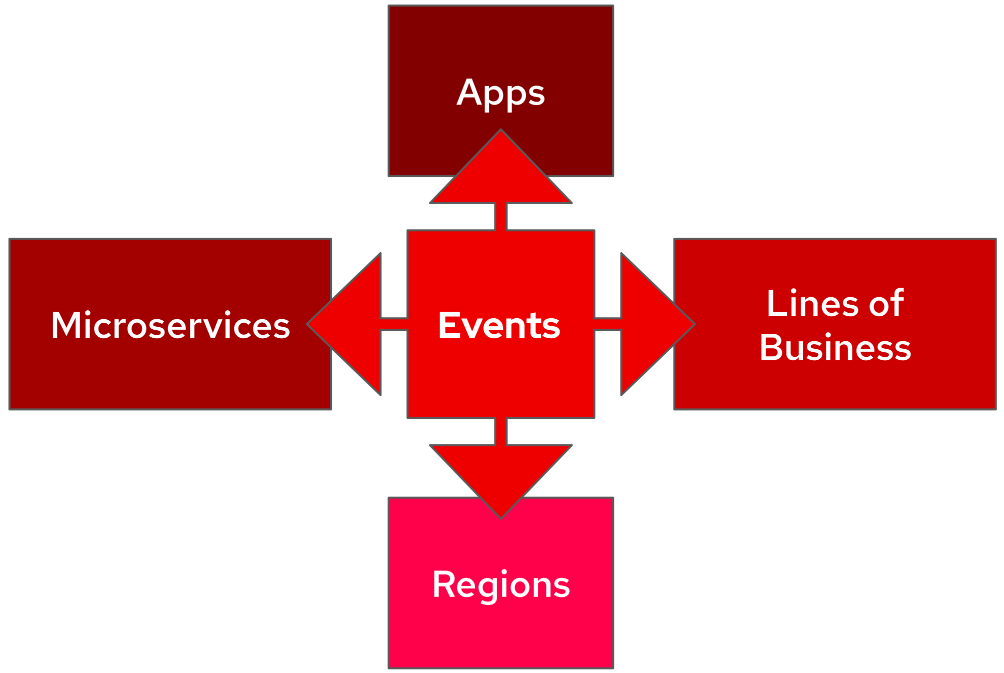
Source

Thinking about event-driven architecture



System and data-centric

Events are designed to respond to ad-hoc connectivity needs



Event-centric

Events are first class citizens that describe the interactions in the enterprise

Source

Event Stream Processing

Use Cases and Applications

Web Site Activity Tracker

Rebuild user activity tracking pipeline as a set of real-time publish-subscribe feeds.

Metrics

Aggregation of statistics from distributed applications to produce centralized feeds of operational data.

Log Aggregation

Centralized collection of log files in a highly-available store, enabling real-time streaming access to log activity.

Stream Processing

Enables continuous, real-time applications built to react to, process, or transform streams.

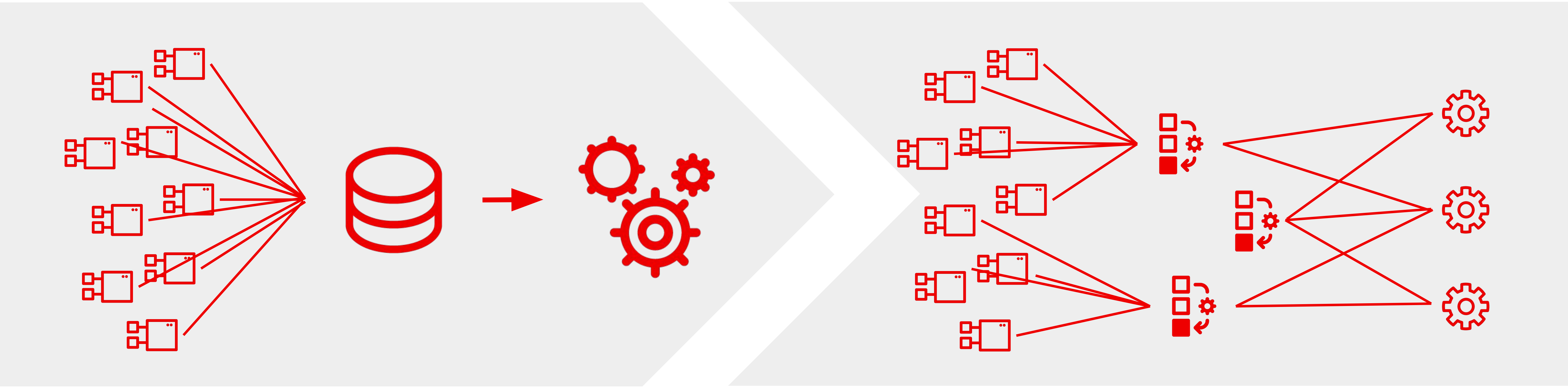
Data Integration

Captures streams of events or data changes and feeds these to other data systems.

Source: https://kafka.apache.org/uses

Streaming Data Processing

Data processing from replayable streams



- Stateless and Stateful processing
- Reading data from multiple streams

Source:

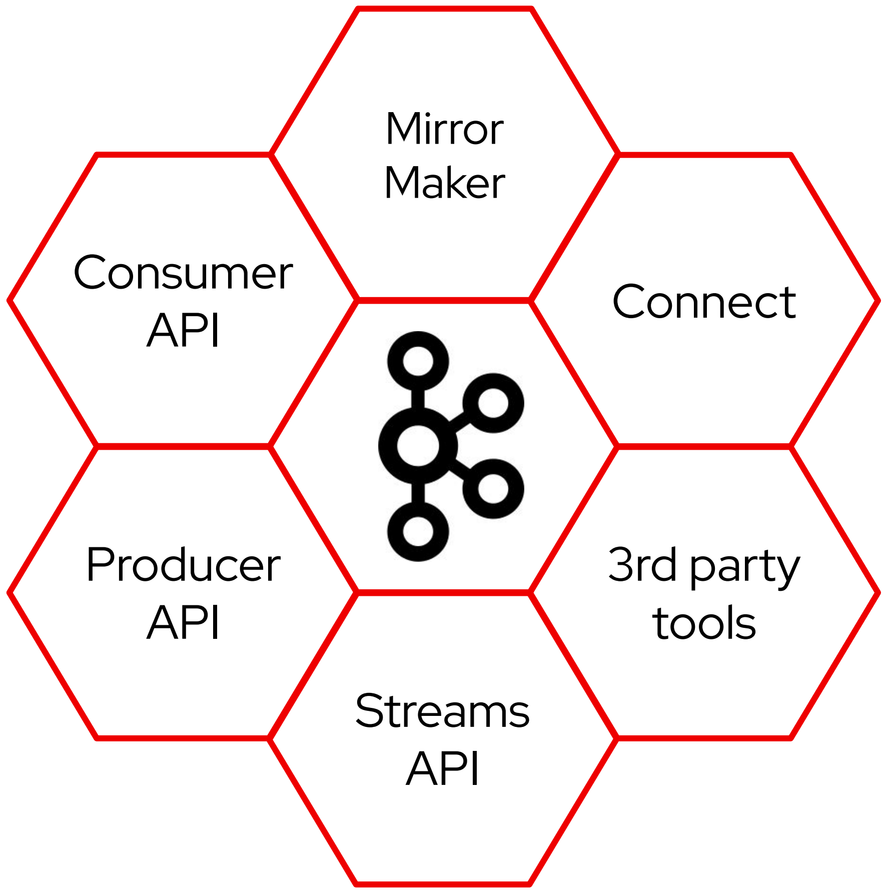
What is Apache Kafka?

Apache Kafka is a distributed system designed for streams. It is built to be an horizontally-scalable, fault-tolerant, commit log, and allows distributed data streams and stream processing applications.

Source

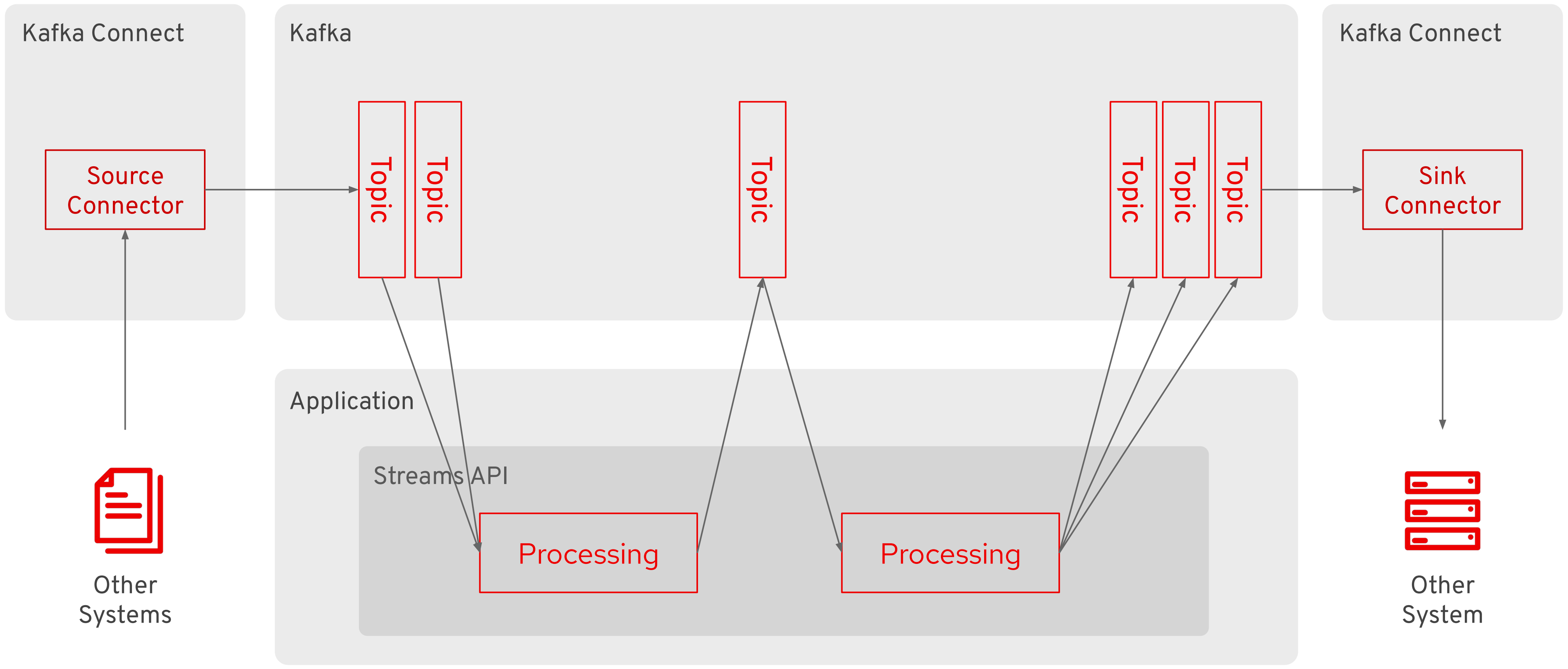
Apache Kafka ecosystem

- Kafka Core
  - Broker
  - Producer API, Consumer API, Admin API
  - Management tools
- Kafka Connect
- Kafka Streams API
- Mirror Maker / Mirror Maker 2
- REST Proxy for bridging HTTP and Kafka
- Schema Registry



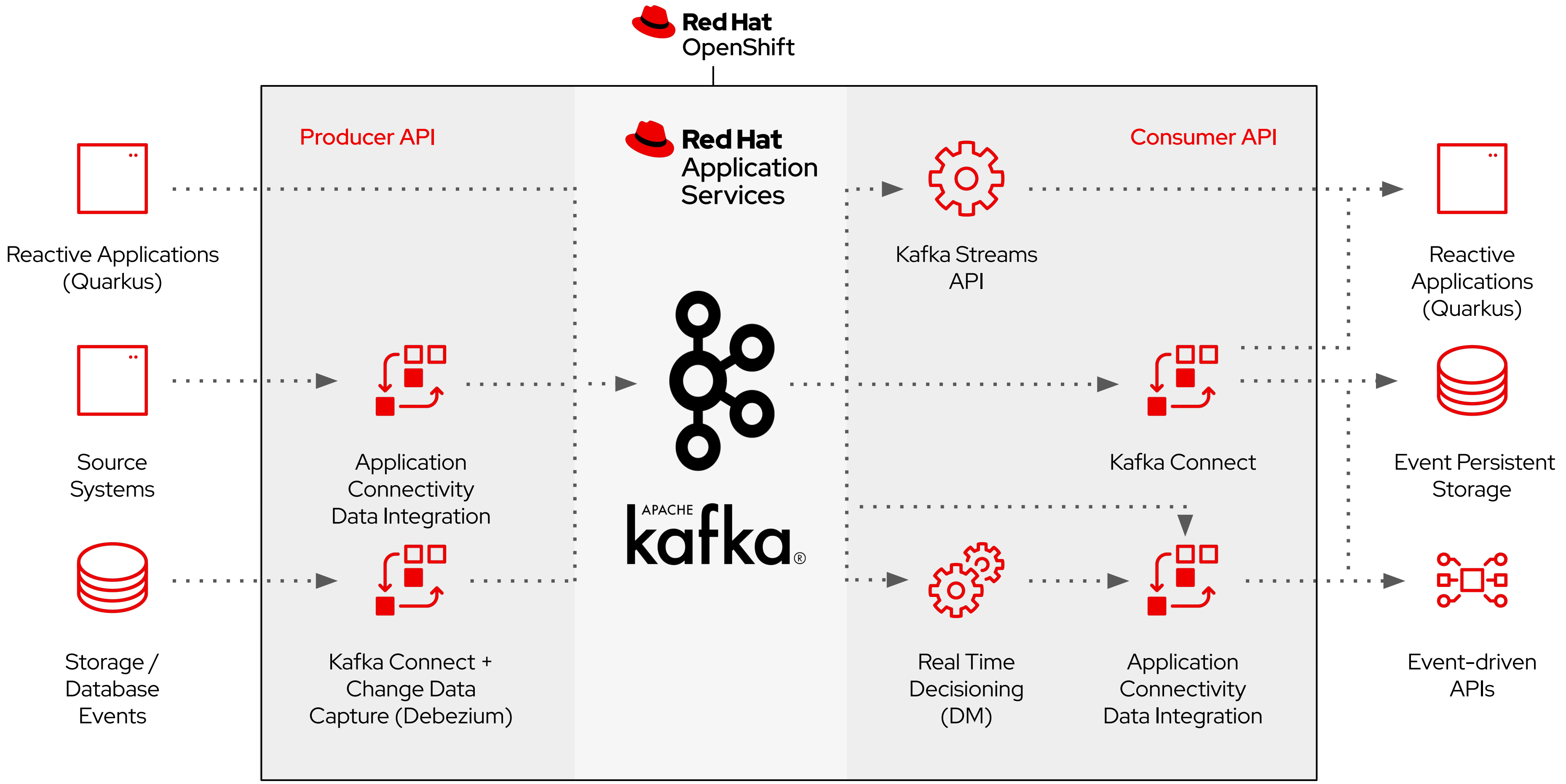
Source

Kafka Connect + Kafka Streams API



Source

Event Management Bus


Red Hat OpenShift
Producer API
Red Hat Application Services
Consumer API
Reactive Applications (Quarkus)
Source Systems
Storage / Database Events
Application Connectivity Data Integration
Kafka Connect + Change Data Capture (Debezium)
APACHE kafka
Kafka Streams API
Kafka Connect
Real Time Decisioning (DM)
Application Connectivity Data Integration
Reactive Applications (Quarkus)
Event Persistent Storage
Event-driven APIs

Source

Service Registry

Service Registry

Use Cases

Schema Registry

Schema registry for Kafka serializers/deserializers.

API Designs

API specification registry for API consumers.

Shared Data Types

Shared data types (schemas) across API and Event driven architectures.

Source:
https://github.com/Apicurio/apicurio-registry

Service Registry

Targeted Requirements for Enterprise Registry

Flexibility

Available for different solutions like Events (Kafka), API management (3scale) and EIPs (Fuse)

Inclusive Artifacts

Support different artifacts: Avro, Protobuf, JSONSchema, OpenAPI, AsyncAPI, among others

API Usage

Target usage for OpenAPI, Async API, REST CRUD, UI, Search

Enterprise Ready

Provide Lifecycle Management, Monitoring & Metrics, Operators, Pluggable Storage, validations.

Source:
https://github.com/Apicurio/apicurio-registry

Service Registry Key Features

Publish, discover and reuse artifacts using a registry service

Artifact management - reusable artifacts are documented, and made available for browsing and downloading.

Version control with compatibility validation - APIs and schemas are checked for validity and compatibility before updates are allowed.

Schema validation - Schemas are checked for valid content, syntax and semantics when published to the registry.

Support for multiple schema formats - Apache Avro, JSON schema, Protobuf, OpenAPI, AsyncAPI, GraphQL, WSDL & XSD.

Compatibility with CNCF and schema registry APIs - compatibility layers are exposed easier application integration and recognition.

Also Available as a service, managed by Red Hat SRE - 24x7 premium support and fully managed highly-available (multi-AZ) infrastructure and daily operations.

Service Registry for your Kafka Topics

Map your Kafka topics to the appropriate schemas

Governance and centralization

Increased service reuse by providing a source of truth for all schemas.

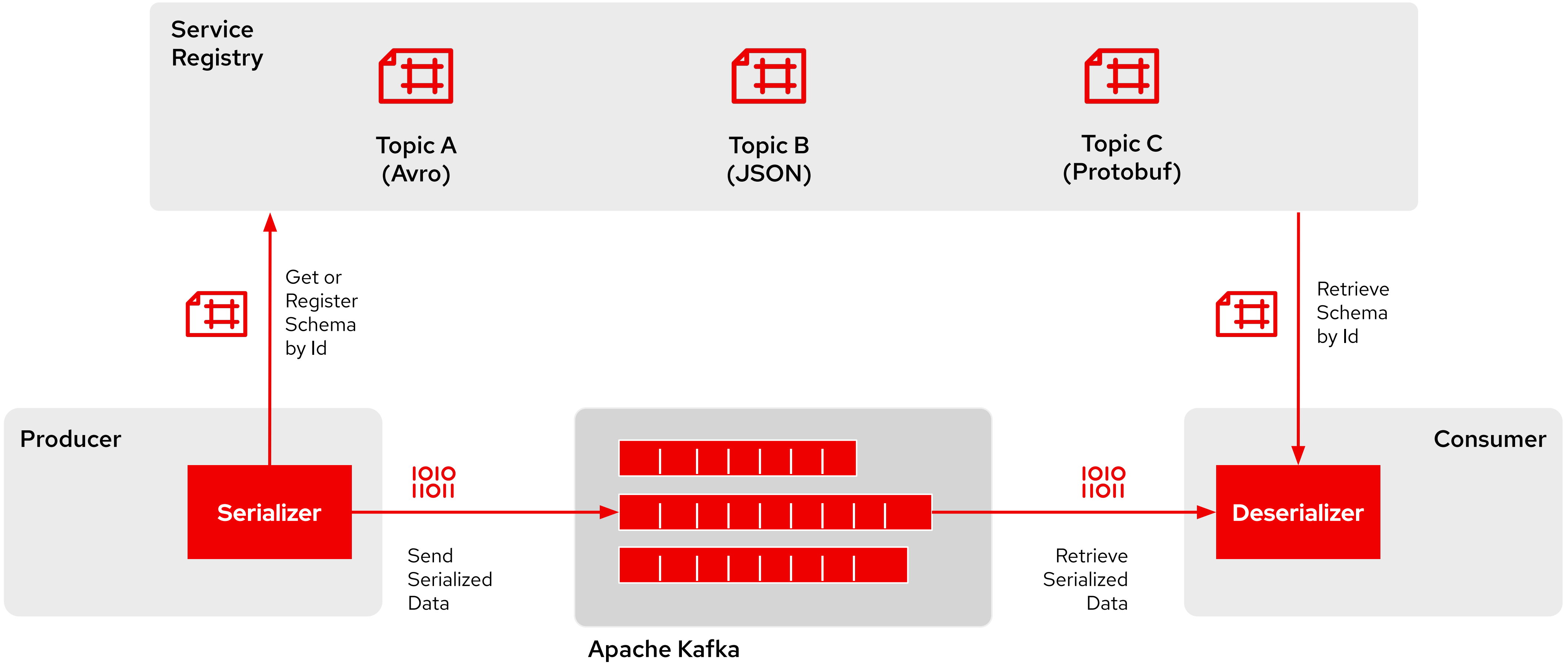
Decoupling

Improve application efficiency and reduce costs by decoupling the schema from client applications.

Discoverability

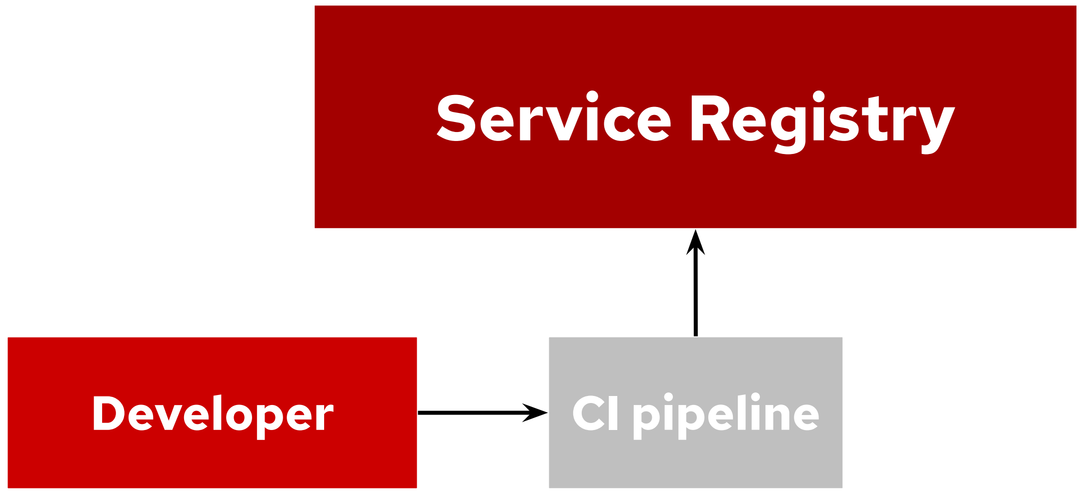
Increase visibility over the schema catalog for serialization/deserialization and compatibility.

Apache Kafka schema management with service registry



Service Registry for your APIs

Enable teams to set API standards and support partners on API discoverability and usability



- **Governance** – increased service reuse by becoming the source of truth for Open API specifications.
- **Discoverability** – increase visibility over the API library ensuring reusability and compatibility.

Rules for content validation and version compatibility

Govern how your registry content evolves over time

The goal of rules is to prevent invalid content from being added to the registry.

- Rules can be configured in service registry for each artifact.
- Each rule has a name and configuration information.
- The registry maintains the list of rules for each artifact and the list of global rules
- Rules are enforced when new artifacts are added to the registry.

Rules can check on:

- Invalid syntax for a given artifact type (for example, AVRO or PROTOBUF)
- Valid syntax, but semantics violate a specification
- Incompatibility, when new content includes breaking changes relative to the current artifact version

Service Registry

Registry Rules

Validation Rules

- Artifact must have valid content or server will reject it
- Can check for valid syntax
- Can also check for valid semantics (for some artifact types)

Compatibility Rules

- Determines whether an update is allowed based on configured compatibility requirement setting.
- Multiple compatibility options including Backwards and Forwards compatible
- Only relevant for updates (checks the new version against the previous version)
- Controls the evolution of a single Artifact over time

Source:
https://github.com/Apicurio/apicurio-registry

Red Hat AMQ

Event and Message based Integration

Event notification

Enterprise messaging

Modern, scale-out architecture

Support microservice applications

Bridge to legacy systems

Enable large scale solutions

Source

Overview

Streams

Streams simplifies the deployment, configuration, management and use of Apache Kafka on OpenShift using the Operator concept

Interconnect

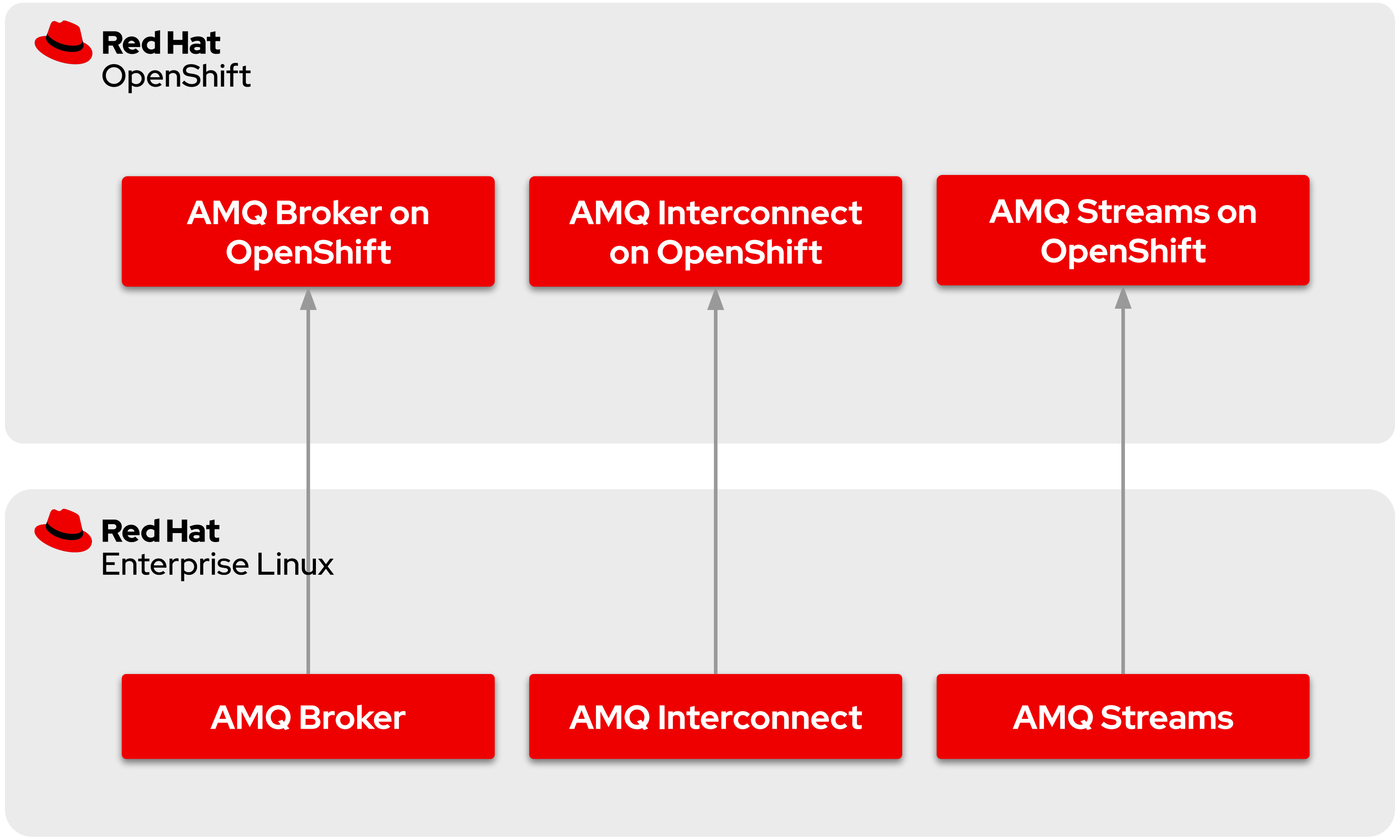
Message router to build large-scale messaging networks using the AMQP protocol to create a redundant application-level messaging network

Broker

High-performance messaging implementation based on ActiveMQ Artemis

Source

Overview



Source

AMQ Broker Overview

- Full-featured, message-oriented middleware broker
  - Pure Java, high-performance message broker
  - Flexible persistence: high-performance journal or JDBC
  - High availability: shared SAN or shared-nothing replication
  - Flexible clustering
- Specialized queueing behaviors, message persistence, and manageability
- Multiple protocols and client languages are supported
  - Including AMQP 1.0, MQTT, STOMP, OpenWire, HornetQ
  - Java JMS, C++, .NET, Python, Javascript, NodeJS Clients

Apache ACTIVEMQ Artemis

Source

AMQ Interconnect Overview

- AMQP-native message router
- Network offers shortest-path routing with redundancy
- Can be used standalone or in conjunction with broker
- Influenced by Red Hat MRG Messaging use cases
- 1-to-1 or 1-to-many
- Supports high performance direct messaging
- Only available on RHEL or OpenShift

Apache Qpid™
Dispatch Router

Source

AMQ Streams Overview

Enterprise data streaming platform distribution based on Apache Kafka.

Available standalone on Red Hat Enterprise Linux VMs/bare metal or on OpenShift (based on Strimzi project).

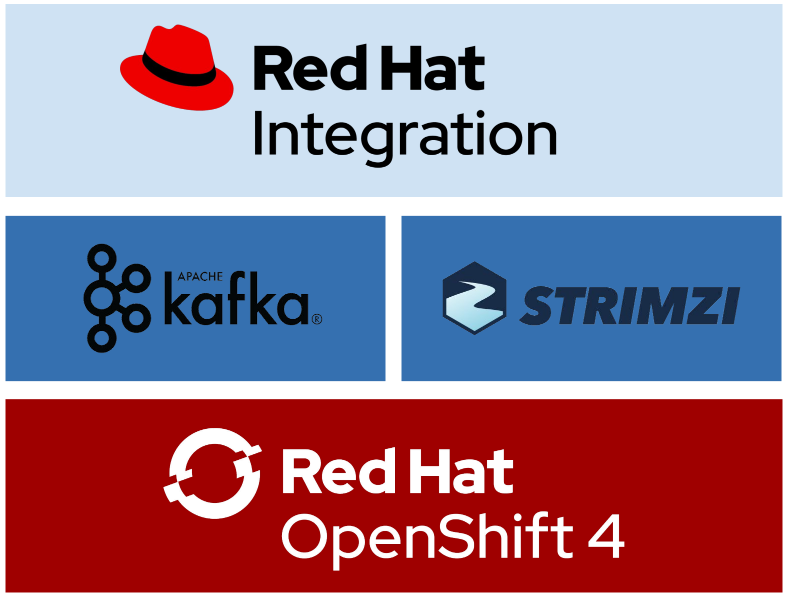
Source

Running Apache Kafka on OpenShift

- Red Hat AMQ streams provides:
  - Container images for Broker, Connect, Zookeeper and MirrorMaker
  - Kubernetes Operators for managing/configuring Apache Kafka clusters, topics and users
  - Kafka Consumer, Producer and Admin clients, Kafka Streams
  - Tools like Cruise Control (TP)
- Upstream Community: Strimzi
  - 100% Open source project licensed under Apache License 2.0
  - Part of the Cloud Native Computing Foundation (CNCF)

Source

Operational Excellence with Red Hat



Kafka optimized for Kubernetes

- Strimzi.io provides Kube-native fit for Kafka
- Member of the CNCF community
- Addresses management and operational complexity of enterprise Kafka architecture via OpenShift Operator

Eventing built into the platform

- Comprehensive catalog of event sources and sinks
- Dynamic scaling and event dispatch based on Knative

Event-Driven applications are more than Kafka

- Reactive framework support based on Quarkus
- Encapsulate business decisions/actions from application logic as Kubernetes services

Source

Red Hat AMQ

A Complete Foundation for the Event-Driven Enterprise

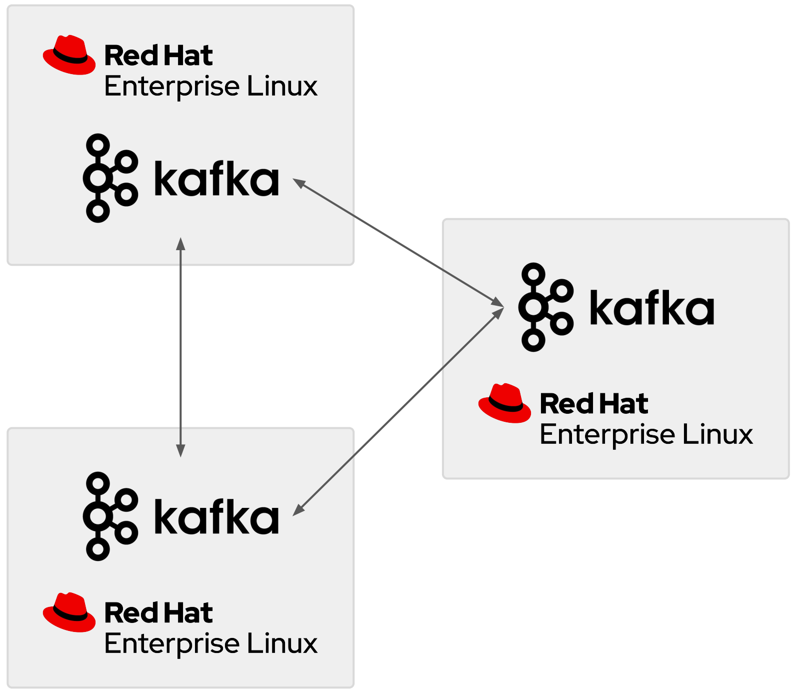
	DESCRIPTION	BEST FIT PATTERNS	POSSIBLE PATTERNS	UNSUITABLE PATTERNS
AMQ BROKER	Traditional messaging broker w/ Queuing & Pub/Sub Rich feature set inc. JMS 2.0 Best-in-class performance Based on Apache ActiveMQ Artemis	DURABLE EVENTS	VOLATILE EVENTS COMMAND QUERY	REPLAYABLE EVENTS
AMQ INTERCONNECT	Message router 1:1 (anycast) and 1:many (multicast) Secure messaging backbone for hybrid cloud Based on Apache Qpid Dispatch Router	COMMAND QUERY VOLATILE EVENTS		DURABLE EVENTS REPLAYABLE EVENTS
AMQ STREAMS	Enterprise distribution of Apache Kafka Simplified deployment on OpenShift Based on Kafka and Strimzi	REPLAYABLE EVENTS DURABLE EVENTS	COMMAND QUERY	VOLATILE EVENTS TRANSACTED/FILTERED EVENTS

Mapping AMQ Components to Patterns

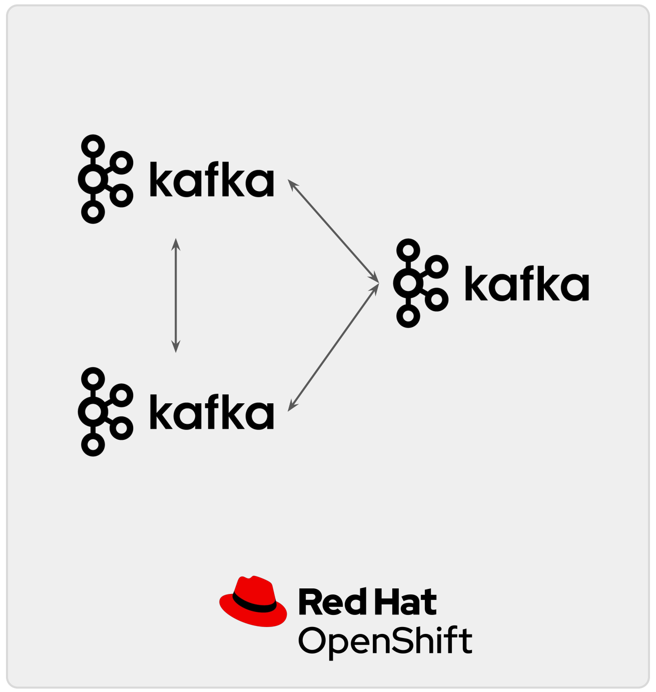
Red Hat | intel.

Kafka Connectivity

AMQ Streams Deployment Options

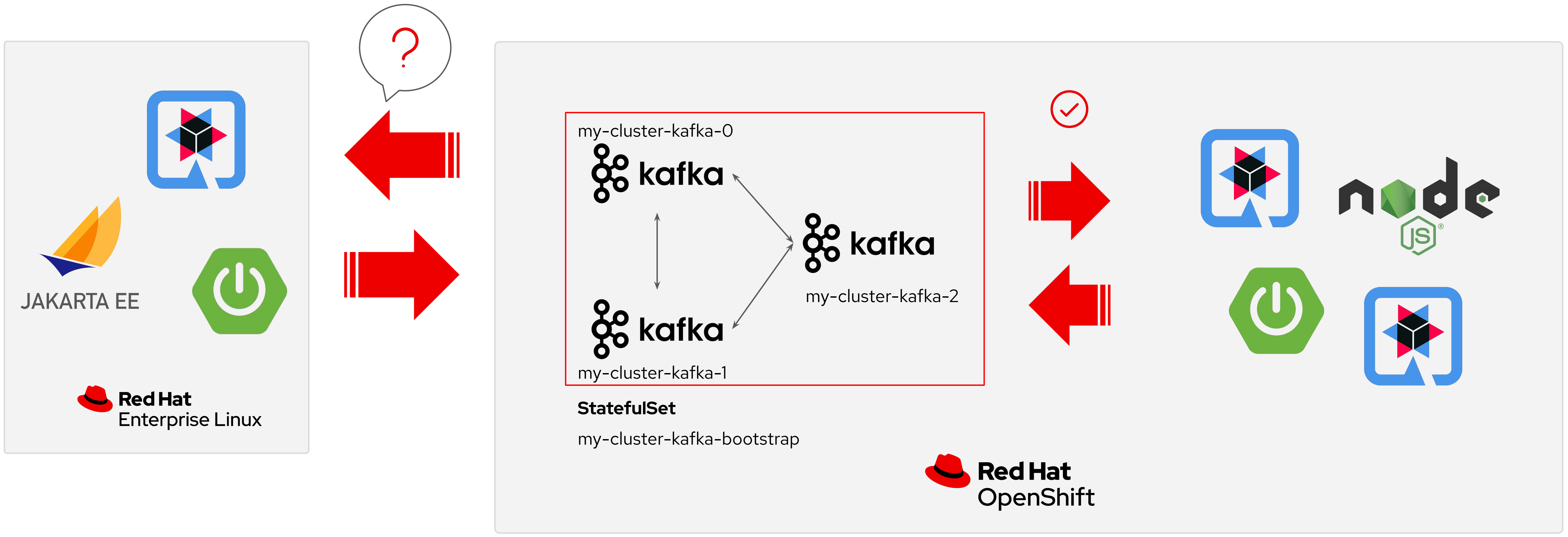


AMQ streams on RHEL

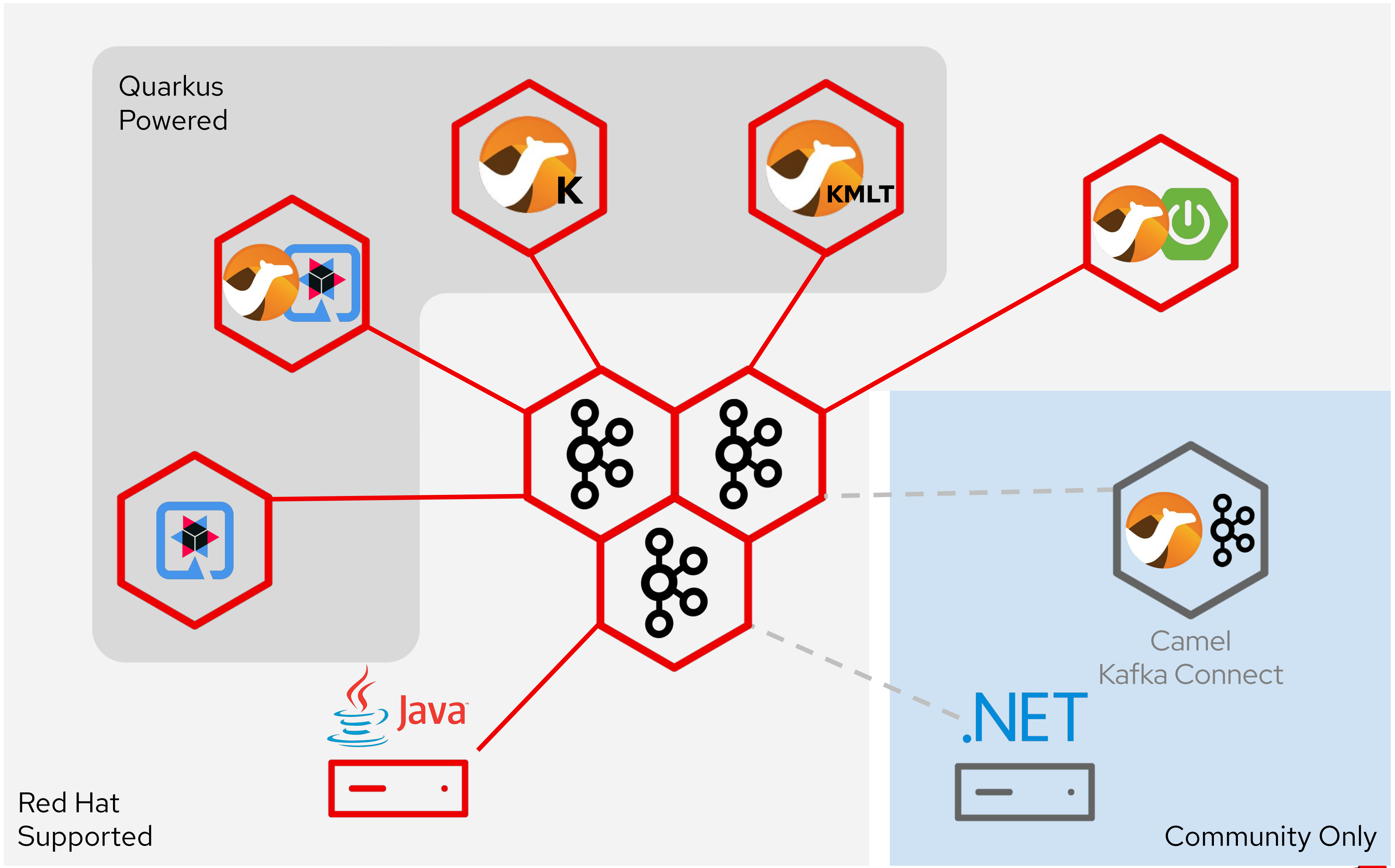


AMQ Streams on OpenShift

AMQ streams on OpenShift



https://developers.redhat.com/blog/2019/06/06/accessing-apache-kafka-in-strimzi-part-1-introduction/


Quarkus
Powered
K
KMLT
Camel
Kafka Connect
Java
.NET
Red Hat
Supported
Community Only

Camel Kafka Component

Producing messages to Kafka

```
from("direct:start")
     // Message to send
    .setBody(constant("Message from Camel"))
     // Key of the message
    .setHeader(KafkaConstants.KEY, constant("Camel"))
    .to("kafka:test?brokers=localhost:9092");
```

Consuming messages from Kafka

```
from("kafka:test,test1,test2?brokers=localhost:9092")
    .log("Message received from Kafka : ${body}")
    .log("    on the topic ${headers[kafka.TOPIC]}")
    .log("    on the partition ${headers[kafka.PARTITION]}")
    .log("    with the offset ${headers[kafka.OFFSET]}")
    .log("    with the key ${headers[kafka.KEY]}")
```

- Traditional Apache Camel component
- Wrapper for the Producer and Consumer API
- Used to retrieve or send events to Apache Kafka
- Part of a Camel route
- Configured using component path and query parameters
- Managed by the user

https://access.redhat.com/documentation/en-us/red_hat_fuse/7.8/html/apache_camel_component_reference/kafka-component

Why use Camel for your Kafka connectivity?

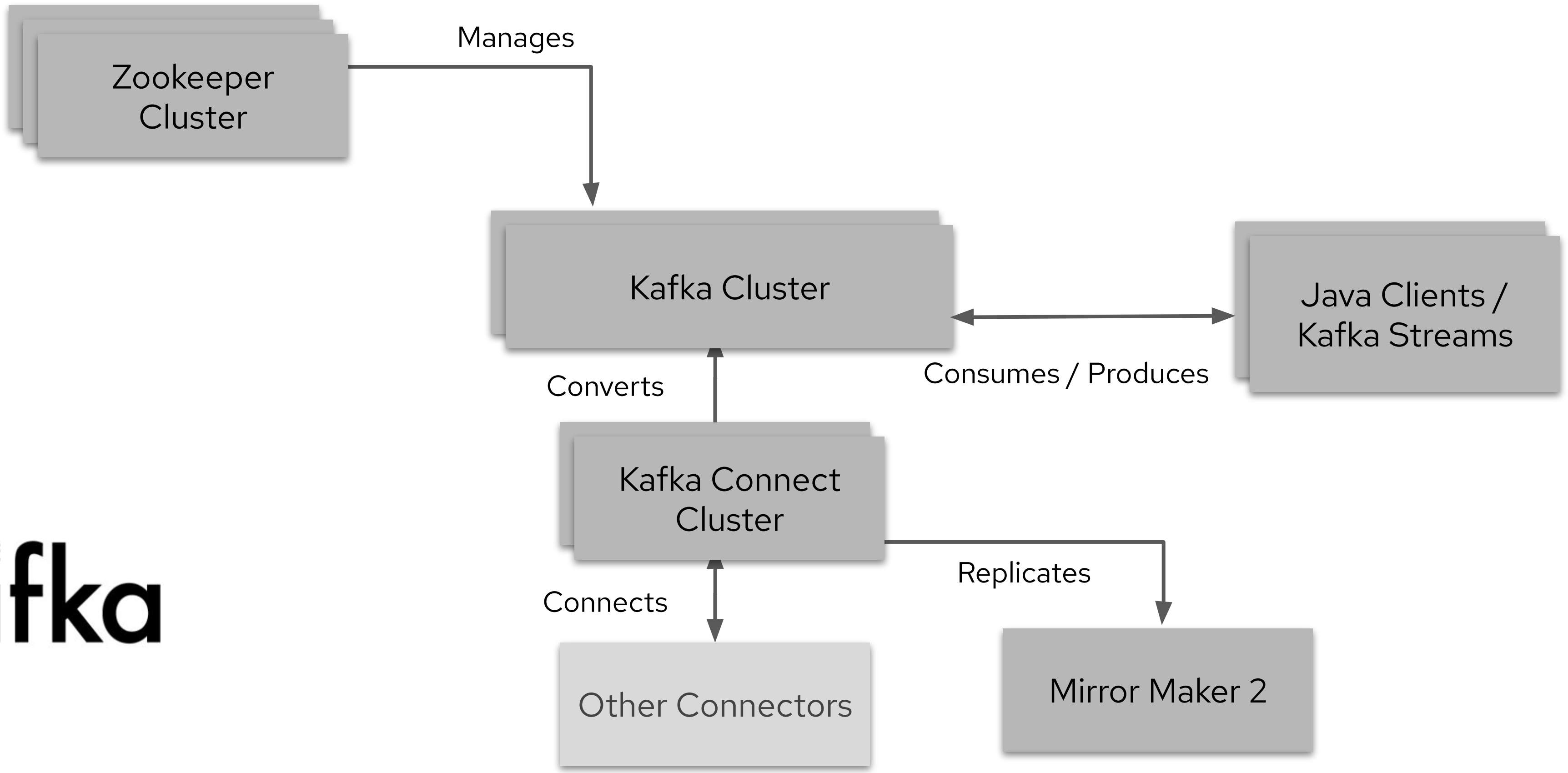
Ingesting data into kafka platform and streaming data out of it

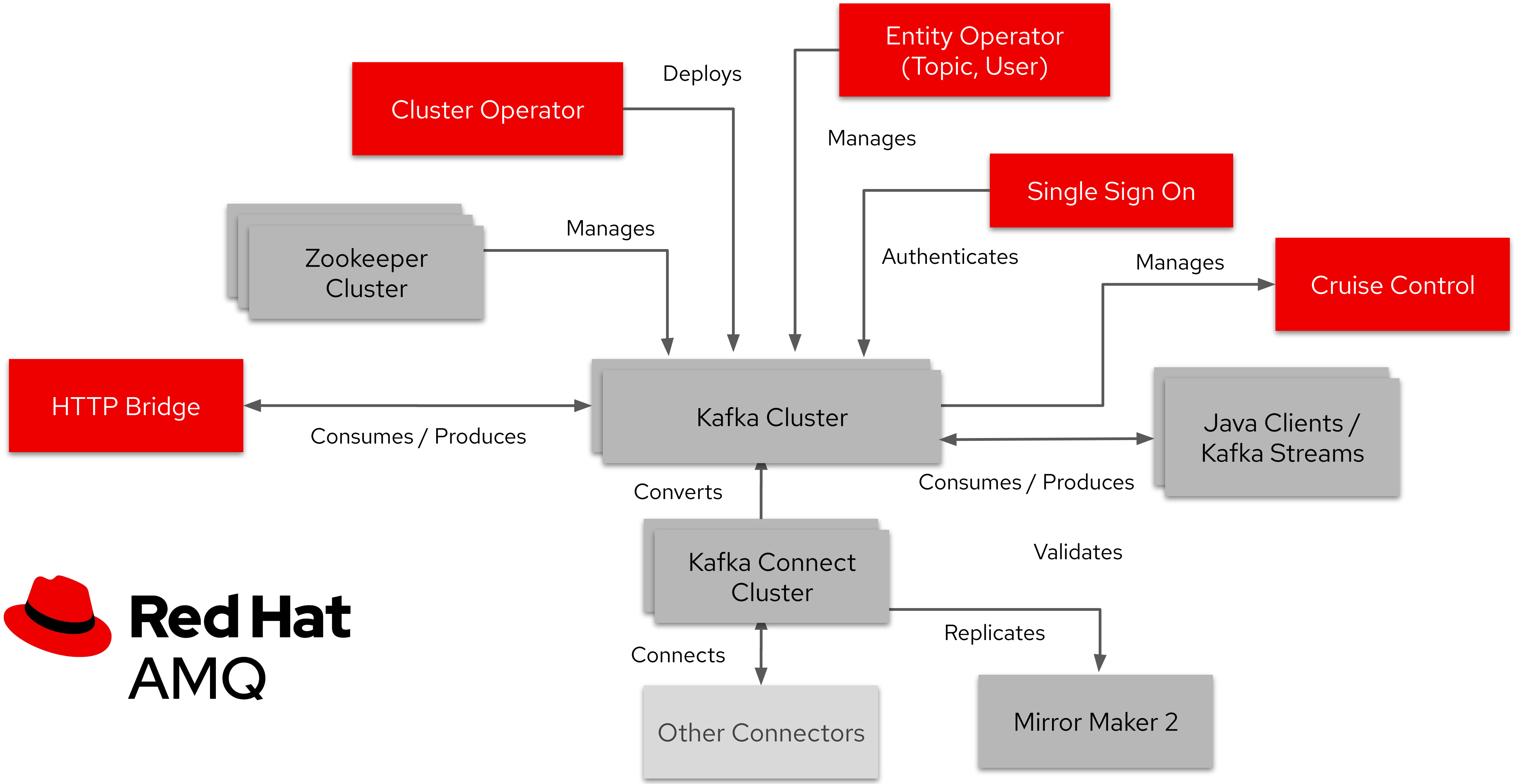
- Consolidate events stored in kafka into a Mongodb instance for reporting purposes (Mongodb Sink)
- Consolidate events stored in kafka into an Elasticsearch instance for analytics purposes (Elasticsearch Sink)
- Ingest transactional log events to further process and aggregate them (files source of syslog source)

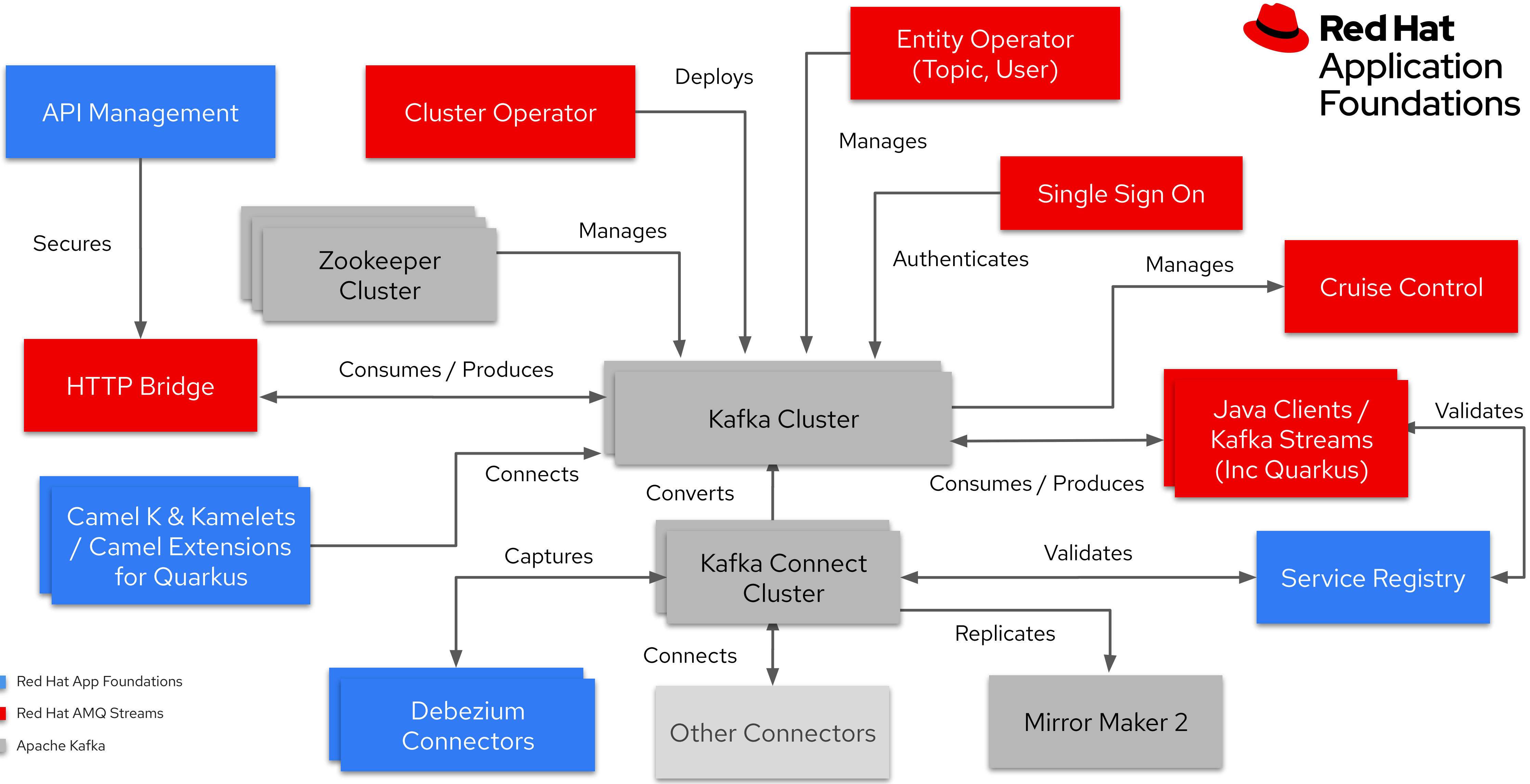
Source:

Red Hat Integration for Apache Kafka

Apache Kafka (Community)




Entity Operator (Topic, User)
Cluster Operator
Deploys
Manages
Single Sign On
Zookeeper Cluster
Manages
Authenticates
Manages
Cruise Control
HTTP Bridge
Kafka Cluster
Java Clients / Kafka Streams
Consumes / Produces
Converts
Consumes / Produces
Kafka Connect Cluster
Validates
Red Hat AMQ
Replicates
Connects
Other Connectors
Mirror Maker 2



🟦 Red Hat App Foundations

🟥 Red Hat AMQ Streams

⬜ Apache Kafka

The complete vision is delivered by Red Hat Integration

Apache Kafka

Community

Red Hat AMQ streams on RHEL

Apache Kafka + Bridge + Cruise Control

Red Hat AMQ streams on OCP

AMQ Streams + OpenShift Operators

Competitor Platform

Apache Kafka + Kubernetes Operator + Kafka Connectors +

KSQL + Replicator + **Control Center**

Red Hat Integration

AMQ Streams + OpenShift Operators + Service Registry + Debezium Connectors + Camel Kafka Connectors + AMQ + Fuse Connectors + API Management + Single Sign On + Quarkus + Serverless (Knative)

Reference: https://www.confluent.io/confluent-community-license-faq/

Messaging czy Streaming?

To zależy :)

Red Hat Summit

Connect

Thank you

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat