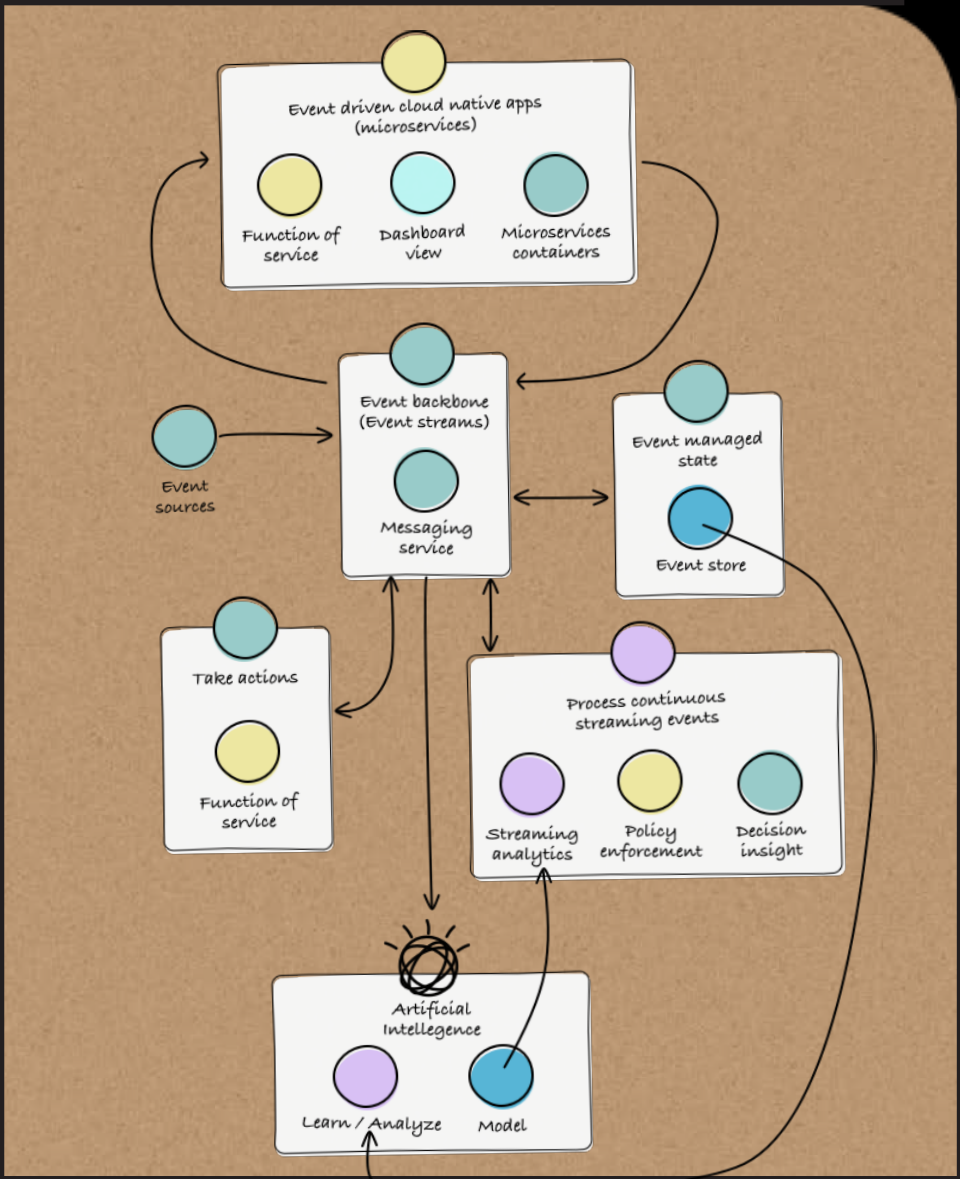



IBM Event Driven

# Field Guide

IBM

Download the current version of the IBM Event Driven Field Guide

https://ibm.biz/ibm-event-driven-field-guide

# Build event-driven solutions

A modern digital business works in real time, informing interested parties of relevant events when they happen, making sense of and deriving insight from an ever-growing number of sources. An intelligent business learns and predicts based on real time events.

**BY NATURE, BUSINESS IS EVENT-DRIVEN**

**Build responsive applications.** Modern business applications bring immediate replies and feedback to the end user or system.

**Access data in real time.** Business teams need access to data to aggregate, score, and act on them in real time.

**Deliver high quality applications.** Modern systems need to be responsive, elastic, resilient, and should deliver consistent quality of service.

**Adopt an asynchronous message-driven backbone.** Event-driven solutions enable loosely coupled components to act together on data by using computing analytics, applying complex rules, or using machine-trained models for scoring.

## What's inside?

This field guide provides a high-level overview of IBM's event-driven approach.

**LEARN IT**
A summary of the concepts.

**GET STARTED**
Tips to start the journey.

# The value of event-driven solutions

Event-driven architecture addresses the loose coupling requirements of microservices and avoids complex communication integration.

## PROCESS EVENTS AS YOU GET THEM

**Mature your adoption of microservices.** Adopt reactive programming coupled with an event backbone that supports the publish/subscribe protocol and enables long-term distributed data persistence for high availability and replayability.

**Modernize your data pipeline.** Adopt real-time data loading, transformation, and publishing to build continuous data streams, which support fraud detection, personalization, recommendations, and business intelligence in near real time.

**Use real time analytics and AI.** Add components to compute statistics, aggregate, or preform stateful computations like time windowing business logic. Perform prescriptive or predictive scoring or generate new facts about the data using AI models.

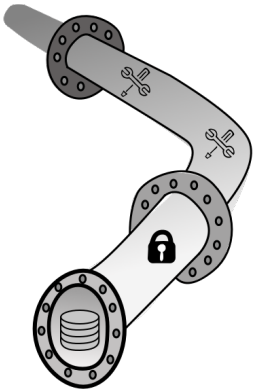
**Reactive Systems.** Adopt message-driven applications to build elastic, resilient, and responsive applications that enhance your user's experience and system reliability.

Learn more

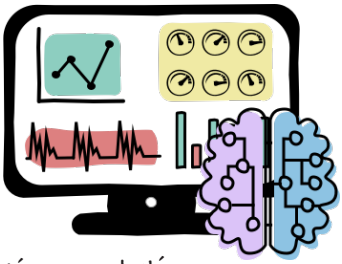
Check out IBM® Event Streams.
https://www.ibm.com/cloud/event-streams



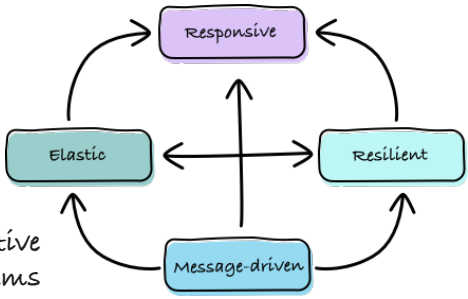
Mature microservices



Data pipeline



Real time analytics
and AI




Reactive
systems

# IBM's unique point of view

IBM products support hybrid cloud and integrated open-source components that act as the foundation of an event-driven system running on lightweight containers deployed to a Kubernetes platform.

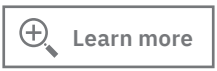
**IBM: BEST-OF-CLASS MESSAGING AND MIDDLEWARE PRODUCTS**

**Support for traditional (MQ) and Apache Kafka pub/sub messaging styles.** The use cases are different but complementary and both styles of messaging are used in modern, cloud-native architecture.

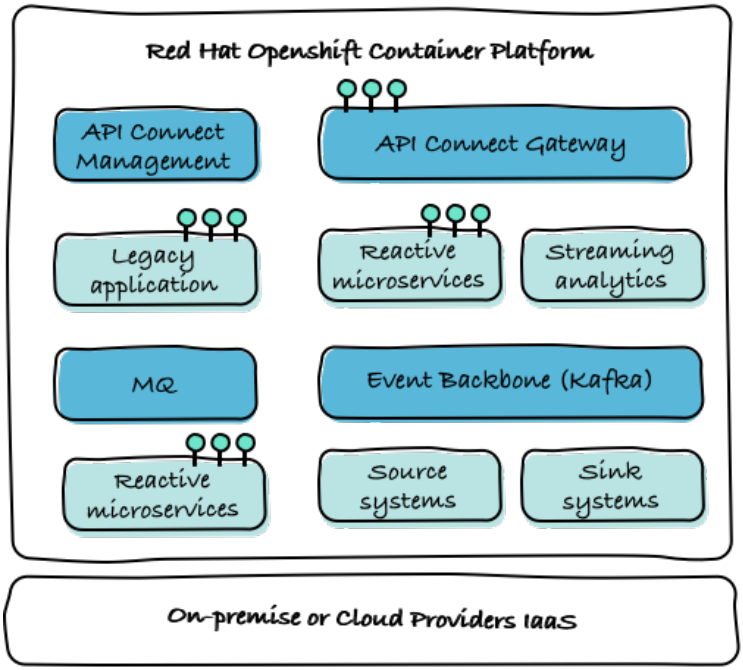
**Adopt message-driven microservices.** Decouple services and enable the continuous visibility of data. Possibilities include real-time recommendations, customer interaction analysis, next best action, fraud detection, auditing, real-time stateful operations on a data stream, and AI applied on each message as events occur.

**Use APIs and event streaming in your event-driven microservices.** AsynchAPI, cloud events, and schema registries help define the contract for message-based interactions.

**Use DevOps tools to build and manage your event-driven solution.** DevOps tools deploy event-driven microservices to Red Hat® OpenShift® with minimum knowledge of Kubernetes.




Check out the IBM Cloud Pak® for Integration.
https://www.ibm.com/cloud/cloud-pak-for-integration




**IBM Cloud Pak for Integration portfolio with event streaming via Apache Kafka, APIs, and Kubernetes (Red Hat OpenShift) work together to support a modern, cloud-native deployment.**

# Architect your solution

The event-driven architecture promotes the production, detection, consumption of, and reaction to events. It can be extended with new components to produce or consume existing events in the system.

**COMPONENTS OF AN EVENT-DRIVEN ARCHITECTURE**

**Event sources**. Generate events and event streams from sources such as 'Internet of Things' devices, web applications, mobile applications, legacy transactional processing, and microservices.

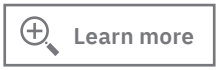
**Event streams & event backbone.** Provides an event backbone supporting publish/subscribe communication, immutable event logs, with stateful event-stream processing.

**IBM Cloud® functions.** Provides a simplified programming model to take action on an event occurrence through a 'serverless', function as a service model.

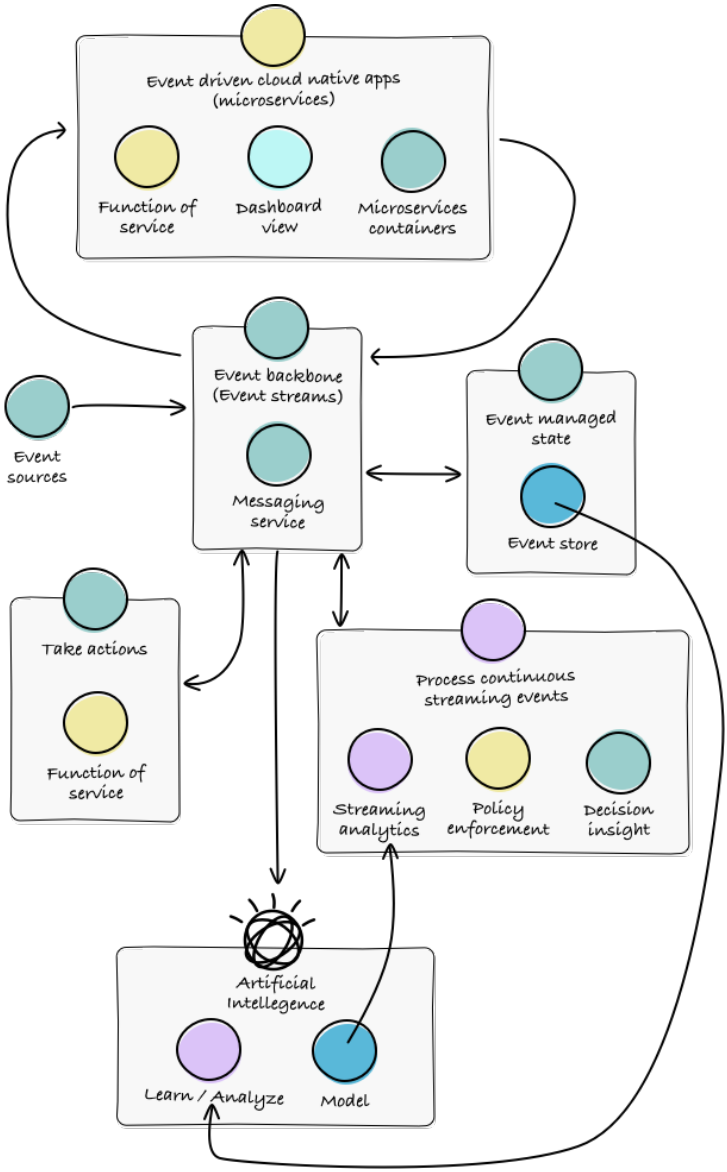
**Streaming analytics.** Continuously ingest and process analytics across multiple event streams. Take action on events or non-events using business rules.

**Event stores.** Optimized persistence (data stores) for append logs to be replicable and replayable.

**Event-driven microservices applications.** Run as serverless functions or containerized workloads connected via pub/sub event communication.




Check out the Event Driven architecture.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/overview


Event driven cloud native apps (microservices)
Function of service
Dashboard view
Microservices containers
Event backbone (Event streams)
Messaging service
Event sources
Event managed state
Event store
Take actions
Function of service
Process continuous streaming events
Streaming analytics
Policy enforcement
Decision insight
Artificial Intellegence
Learn / Analyze
Model

# Real-time analytics

Real-time analytics enables your solution to investigate and understand the events flowing through unbounded, real-time event streams. Streaming applications process the event flow and apply data and analytical functions to information in the stream. Streaming applications are written as multistep flows that are easy to develop and deploy on demand.

**GO THROUGH THE FLOW**

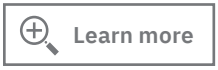
**Ingest.** Gather data from many sources.

**Prepare.** Transform, filter, correlate, aggregate on some metrics, and enrich your data using other data sources. Keep enriched data as new data elements ready to consume by other application.

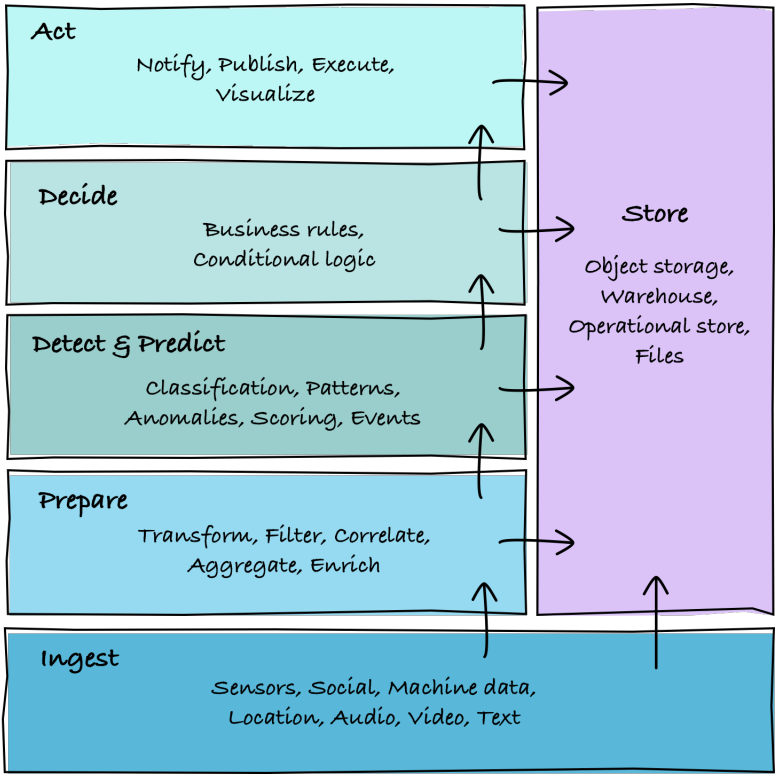
**Detect and predict.** Classify and score data, detect patterns and anomalies, and predict event patterns.

**Decide.** Apply business rules and business logic.

**Act.** Execute an action, a process, a decision, send data to an external system, or publish a notification.

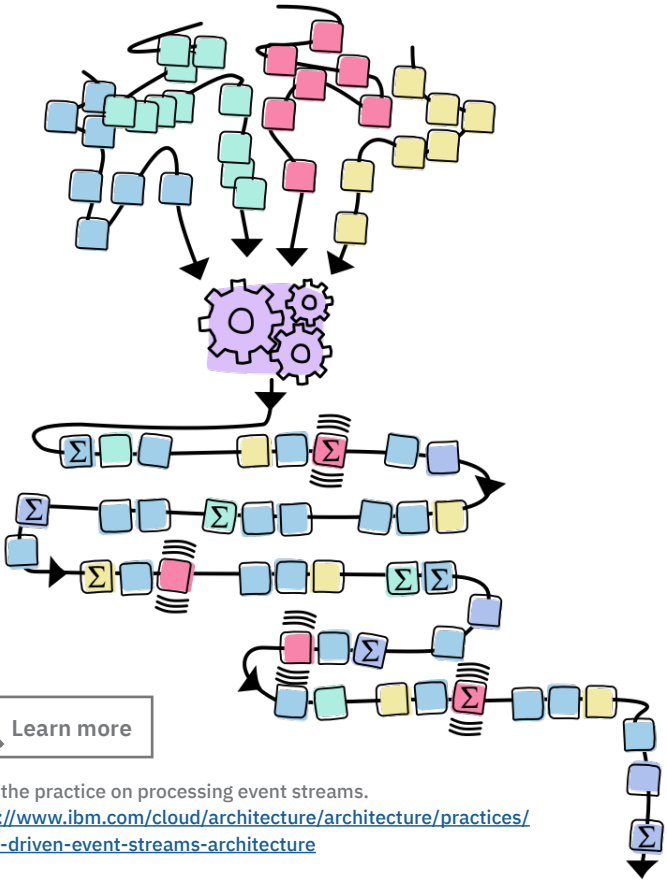



Check out the Data, Analytics, and AI architecture.
https://www.ibm.com/cloud/architecture/architectures/dataAIArchitecture/overview


Act
Notify, Publish, Execute, Visualize
Decide
Business rules, Conditional logic
Detect & Predict
Classification, Patterns, Anomalies, Scoring, Events
Prepare
Transform, Filter, Correlate, Aggregate, Enrich
Ingest
Sensors, Social, Machine data, Location, Audio, Video, Text
Store
Object storage, Warehouse, Operational store, Files

# Basic streaming analytics capabilities

Streaming solutions can be implemented by different technologies, but they all process unbounded event streams that are based on a number of capabilities.



Learn more

Read the practice on processing event streams.
https://www.ibm.com/cloud/architecture/architecture/practices/event-driven-event-streams-architecture

**Continuous event ingestion and analysis.** Continuously inject data from heterogenous sources to the event backbone, via a rich, scalable connector framework, then easily integrate via the Stream API. Different delivery semantics can be supported, including fire and forget, at most one, or exactly once.

**Process multiple event streams.** Deliver multiple parallel processing apps using a scalable event streaming architecture. Deploy each app using container orchestration software. Each app consumes data from streams, processes, and produces new data in a separate stream. Chaining stream processing enriches and aggregates data records to deliver business meaning for other apps to consume.

**Low latency processing, without storing data.** A producer can generate events with low latency processing from the broker cluster. Data persists as an append log for a long time. Stateful operations can persist states in topics in a memory database. Inbound messages from a 'firehose' are delivered, not persisted, and focus on low latency processing. The first level of stream processing computes aggregates that persist and support exactly once delivery.

**Process high-volume, high-velocity streams of data.** Tune your configuration to support high throughput and large records. Apply patterns to keep or drop large messages or use a long term bucket to persist larger files and keep metadata as factual, immutable data in the distributed log within the brokers.

**Continuous query and analysis of the feed.** Streaming applications that compute stateful logic and aggregates are continuously queried using a distributed mechanism. If the query reaches a node with the expected key, records are returned immediately. If not, the target URL is returned so records are found in a second call. The knowledge of key distribution supports continuous interactive queries.

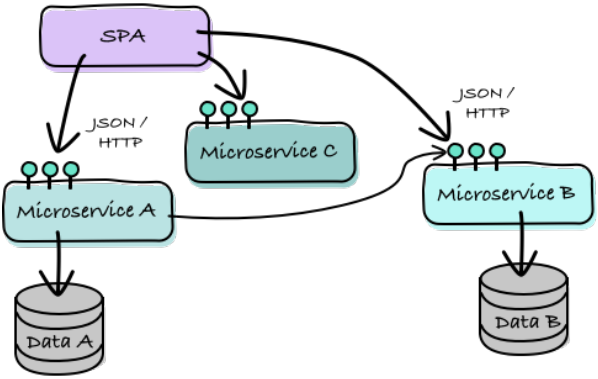
# Microservices: API-driven

Modern microservice-based applications use a combination of synchronous API-driven and asynchronous event-driven communication styles.

### SYNCHRONOUS API-DRIVEN MICROSERVICES

When a microservice (A) needs to access data from microservice (B), it calls an endpoint via an HTTP GET, which leads to strong coupling at the data definition level. This introduces mainenance challenges.

You must assess which microserve implements the join, service A, B, or a new service (C). Service C implements the join by calling two API endpoints and reconciling data using primary keys. It is simpler to use point-to-point synchronous communication; however, as the number of services grows, coupling creates challenges.




Learn more

Read the practice, Event-driven cloud-native applications (microservices). https://www.ibm.com/cloud/architecture/architecture/practices/event-driven-cloud-native-apps-architecture

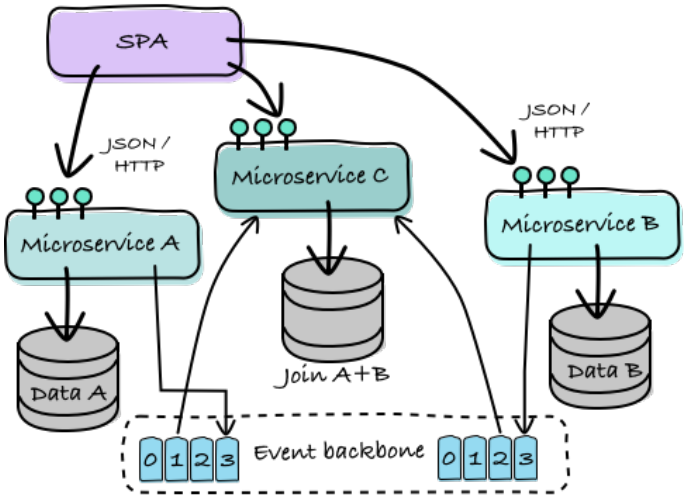
# Microservices:  Event backbone

Communication is asynchronous using a pub/sub protocol when using event-driven microservices. With this approach, the microservices are naturally responsive, elastic, and resilient enhancing the loose coupling nature of microservices.

**EVENT-DRIVEN MICROSERVICES**

When adopting Kafka as the event backbone, data sharing occurs via an event log, which can be kept for a long period, is replayable, and resilient. A consumer can restart and process messages from the last read,  go back in time, or replay from the last good processed input record.

When using domain-driven design and bounded context for microservices, microservice A and B produce facts about their business entities to their respective topic in the event backbone. To support a join, microservice C consumes those facts to build its own data projection.



# Reactive systems

Contemporary solutions based on microservices support dynamic loads and elegantly tolerate failure. The reactive manifesto defines four characteristics that modern, cloud-native applications must support.

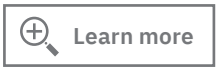
**MESSAGE-DRIVEN. ELASTIC. RESILIENT. RESPONSIVE.**

**Message-driven.** A reactive system architecture is based on an asynchronous, message-driven backbone. The architecture enables loose coupling of applications, which minimizes coherency delays and inter-service communication network latency.

**Elastic.** Reactive systems remain responsive under varying workloads by scaling resource utilization depending on the system load.

**Resilient.** Reactive systems are responsive in the face of failure. This implies distributed systems and the ability to reload state from the message bus.

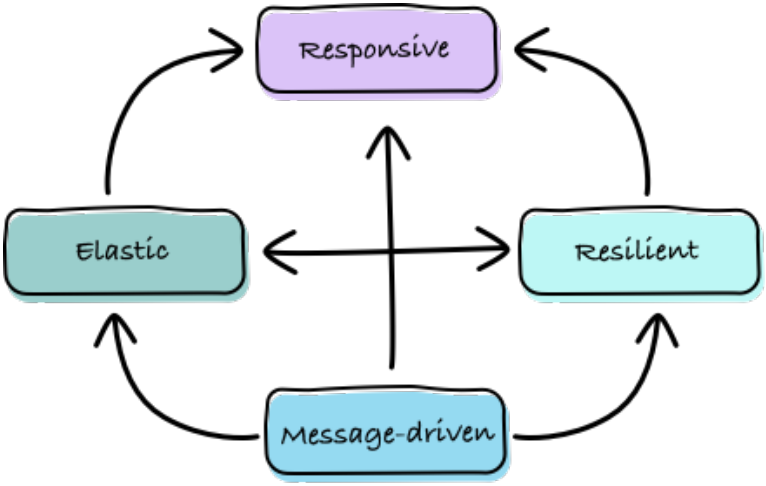
**Responsive.** Reactive systems deliver a consistent quality of service to end users or systems. Modern business applications respond even under heavy workload.




Read about Reactive systems.
https://ibm-cloud-architecture.github.io/refarch-eda/advantages/reactive/

Asynchronous communication helps to support scaling, integration, coupling, and failover. Adopting reactive design and implementation might look complex at first, but it has become a necessity for scalable business solutions.

For example, in e-commerce, many monolithic applications have been redesigned to adopt reactive characteristics that support scaling business needs and respond to sporadic demands.



# Eventual data consistency

Adoption of cloud-native applications requires distributed systems, which can lead to data inconsistency. It is not yet possible to have consistency, availability, and network partition tolerance. When a service supports a POST, PUT, or DELETE operation on its main business entity, the propagation of changes to other microservices interested in that data will eventually be consistent.

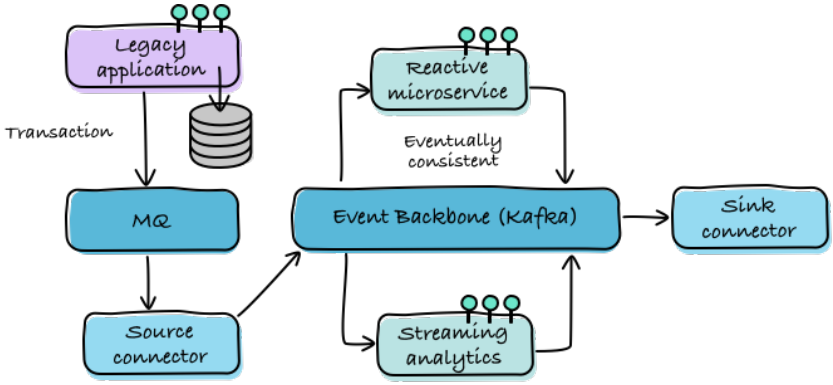
### ALL OBSERVERS SHOULD GET THE SAME RESULT

**Microservices are distributed.** With hundreds of microservice instances running in parallel in distributed data centers, data issues and network partitioning occur. There is a trade off between consistency and availability. For most business applications at the scale of the internet, availability is the priority.

**Eventual data consistency.** Event-driven architecture and proven design patterns like Saga, CQRS, and event sourcing, help to support availability and facilitate eventual consistency. Reading data can be performed anywhere. The event backbone is used for data replication and is the source that builds the eventually consistent data view.

**Ensure consistency on the write model.** The architecture uses transactions to ensure consistency between persisting data and publishing messages to queues.

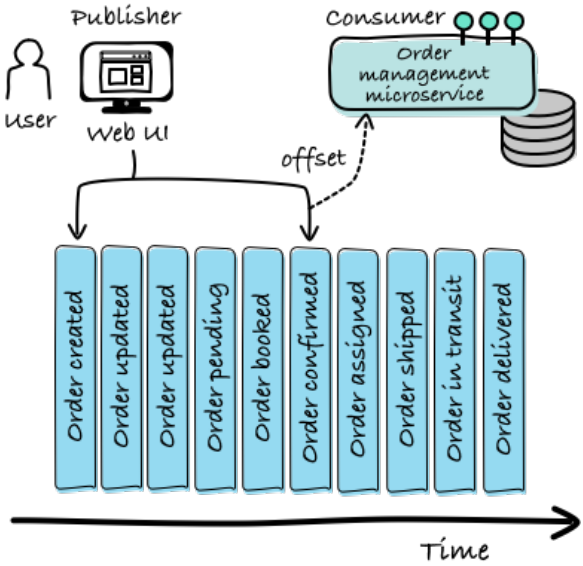

Check out the Event Driven architecture.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/overview




**Consistency requirements are different for each application. The developer has to select the tools they need to support their requirements.**

# Event sourcing design pattern

Most business applications are state-based, where any update changes the business entities. The database keeps the last committed update. Some business applications need to explain how they reached their current state. Event sourcing persists the state of a business entity, such as an order, as a sequence of state-changing events or immutable facts over time in the order of occurrence.

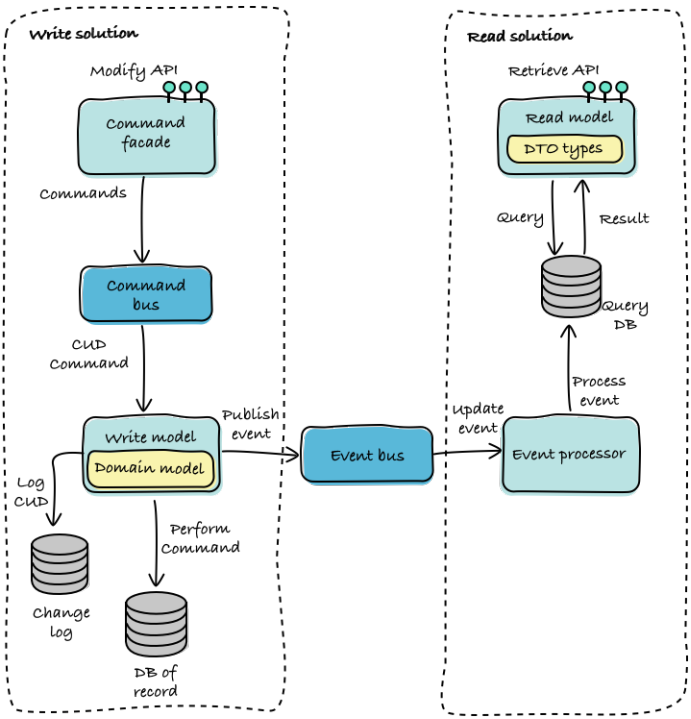



Learn more

Explore the event driven patterns.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/patterns

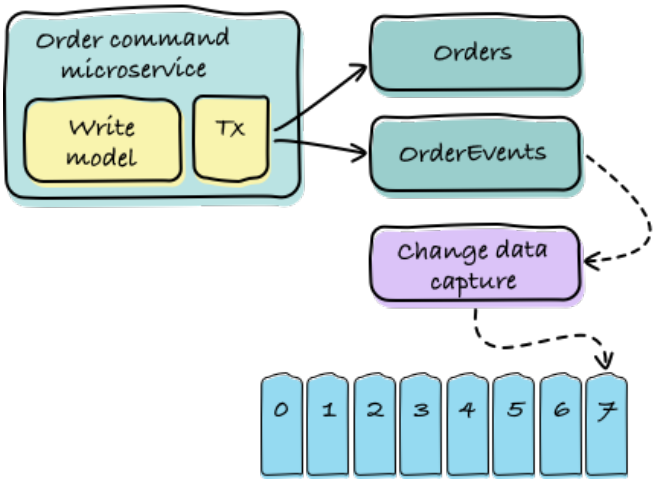
# CQRS design pattern

When a data model is overburdened by managing complex aggregate objects, concurrent updates, and numerous cross-cutting views, it must be refactored. The Command-Query Responsibility Segregation (CRQS) pattern strictly segregates read operations from write operations that update data, making data operations more manageable and simpler to implement. The operations can be developed, optimized, and evolved independently, enabling better scale, performance, and security.



# Outbox design pattern

Typically, an event-driven service command must atomically update database tables and emit events. When transactional queueing products are not available, insert an event into an ‘event’ table. Then a change data capture agent gets the records from the table and publishes them to the event backbone, like Kafka. This is a powerful solution for new app.

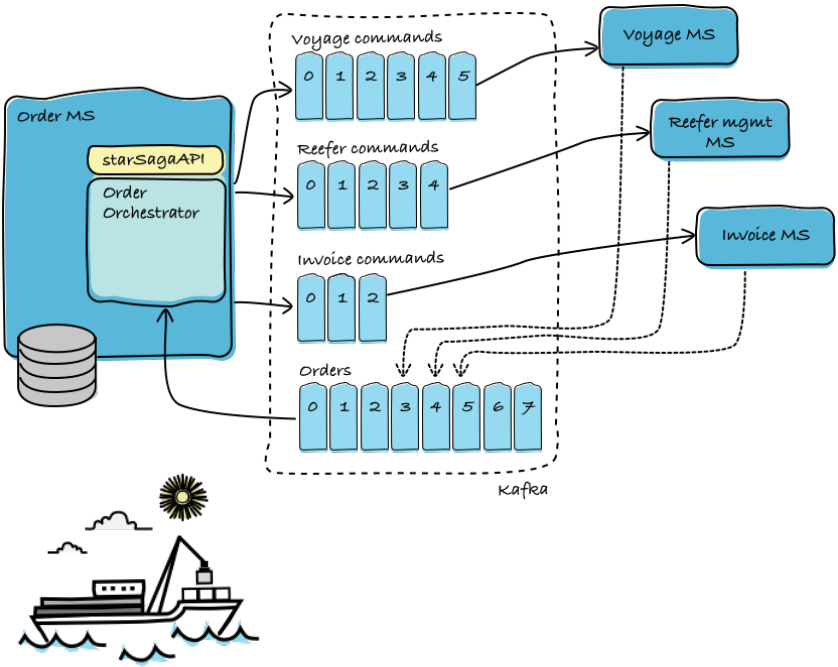



Learn more

Explore the event driven patterns.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/patterns

# Saga design pattern

When using one data source per microservice, it is challenging to support long-running transactions across microservices. The Saga pattern helps solve this by breaking up a long-running transaction into a collection of interwoven sub-transactions. With the Saga Choreography pattern, each service produces and listens to other services' events and decides whether to take action. With the Saga Orchestration pattern, one service is responsible for driving what each participant does and when.

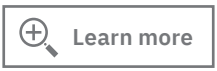


# Data pipeline to data platform

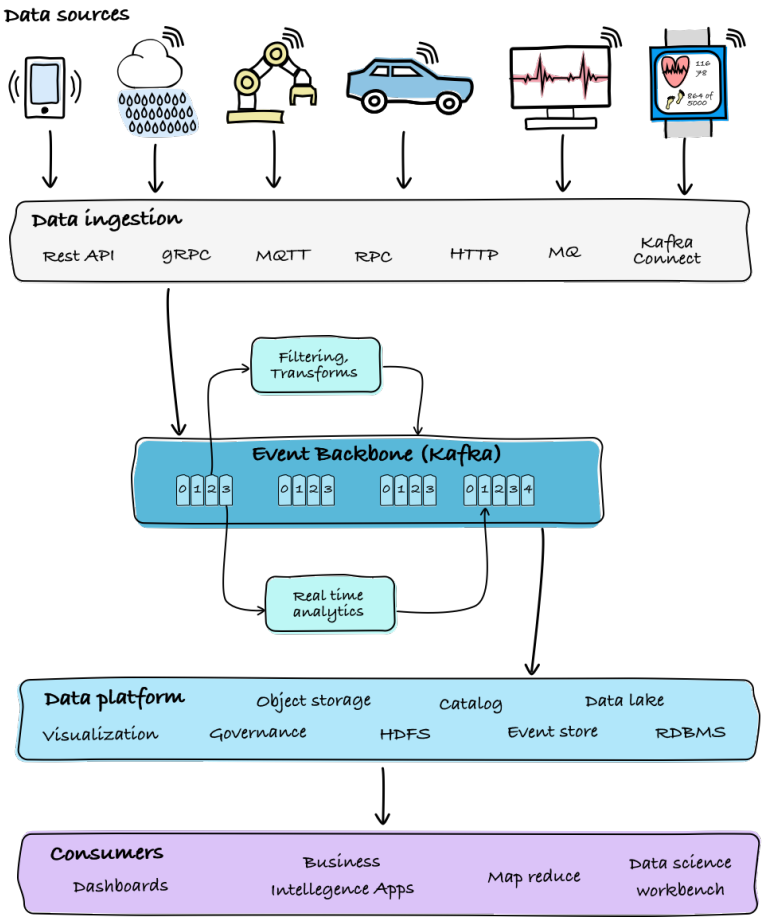
A data pipeline architecture commonly uses an event backbone to buffer data and sink applications. A data pipeline is a series of steps or processes that reliably moves data from a source to a destination. As data moves through the pipeline, it is transformed and enriched by models. The models are consumed by components acting on the data. An enriched model is used by the AI workbench to engineer features and develop a prescriptive scoring model. The pipeline moves data into a data lake for big data processing.

A data lake is a system or repository that stores raw, transformed data. A Lambda architecture handles massive quantities of data by taking advantage of both batch and stream processing methods.

By using Kafka as an event backbone for a data pipeline and a source of truth, you can enhance the architecture. Batch queries and 'map reduce' can address huge quantities of raw data. Streaming queries support real time aggregation and analytics. Microservices generate data and streaming apps aggregate data. Data enrichment and the data lake provide the source for the AI model and scoring.




Learn about modern data lake.
https://ibm-cloud-architecture.github.io/refarch-eda/introduction/reference-architecture/#modern-data-lake


Data sources
116
98
864 of
5000
Data ingestion
Rest API
gRPC
MQTT
RPC
HTTP
MQ
Kafka
Connect
Filtering,
Transforms
Event Backbone (Kafka)
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3 4
Real time
analytics
Data platform
Object storage
Catalog
Data lake
Visualization
Governance
HDFS
Event store
RDBMS
Consumers
Dashboards
Business
Intellegence Apps
Map reduce
Data science
workbench

# Mainframe integration

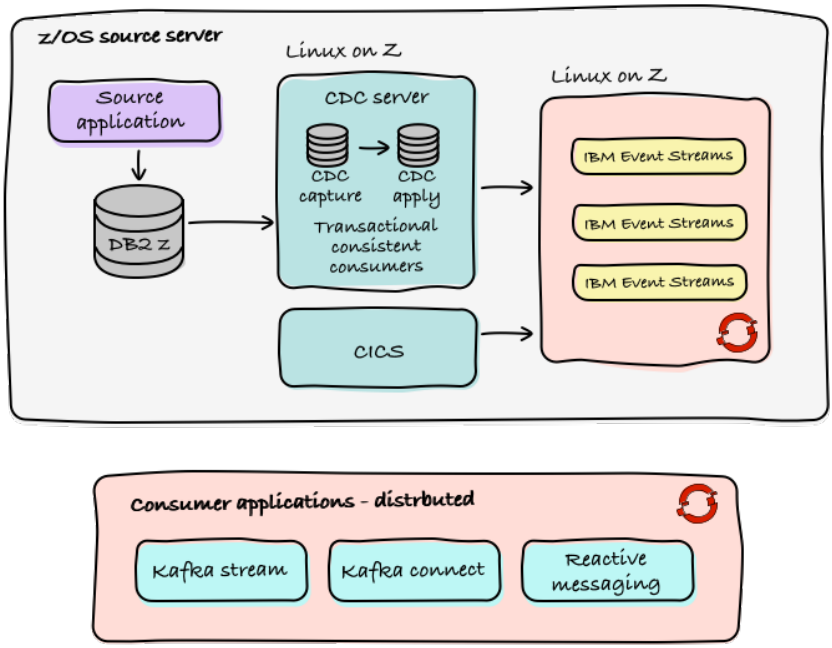
Most mainframe integration uses existing transaction processing to keep the write model on the mainframe. In a distributed hybrid-cloud world, you can move the read model to the cloud. Apache Kafka (IBM Event Streams) and a Change Data Capture (CDC) product can be co-located on the mainframe to reduce operation and runtime cost, and complexity.

**ARCHITECTURE PATTERN: IBM EVENT STREAMS & CDC**

**Improve quality of service**. Auto-scale and balance between Linux nodes while maintaining resilience. For the Kafka service, use IBM Event Streams, the only Kafka variant supported by Linux® on IBM Z®.

**Reduce complexity and management cost.** Use the architecture pattern to reduce latency and increase memory speed. Network traffic is not encrypted, which reduces MIPS by avoiding authentication (TLS) overhead on IBM z/OS®. Avoid network spend, management, and maintenance between servers.

**Ensure transactional integrity.** The CDC server uses the Kafka Transaction Capture Consumer library to maintain transaction integrity while publishing to a Kafka topic. Consumer Information Control System (CICS®) business events are a mechanism for declaratively emitting events from CICS routines.




**This architecture pattern increases the efficiency of the mainframe and creates a data pipeline with transactional integrity. Deploy CDC servers and event stream brokers on Red Hat OpenShift on Linux on Z.**

# Existing system integration

If your existing systems are not designed to submit events, adopt an event-driven architecture. It introduces new opportunities for existing systems to include real-time analytics, a data pipeline, an AI-based scoring agent, and loosely coupled microservices.

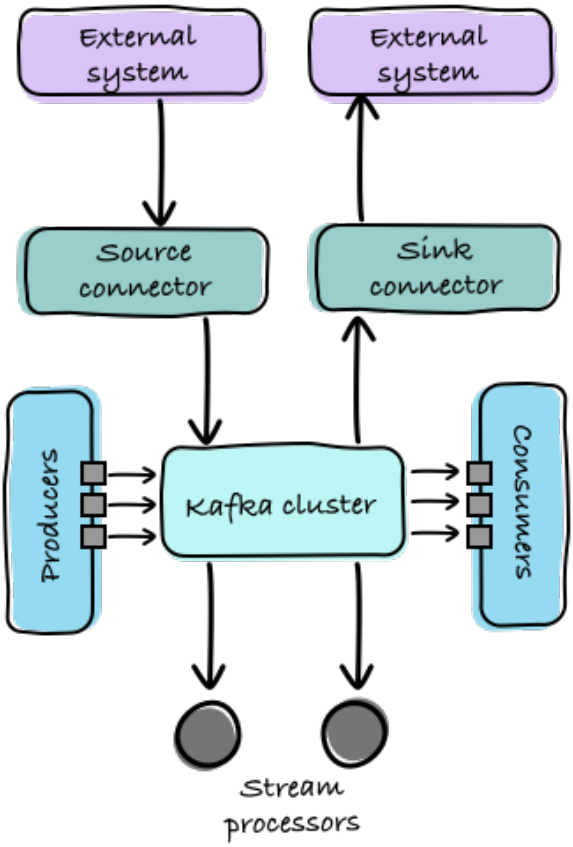
**MORE DATA STREAMED = AGILITY AND OPPORTUNITIES**

**Kafka connector framework.** Use the Kafka connector framework for external system integration. The Kafka connector framework uses long retention topics to keep state of the source or sink connectors, and supports clustering to scale horizontally.

**Kafka cluster.** Use the Kafka connector framework to get new events into a Kafka cluster.

**Open source connectors.** Connectors are available from the open-source community and as part of Apache Camel.

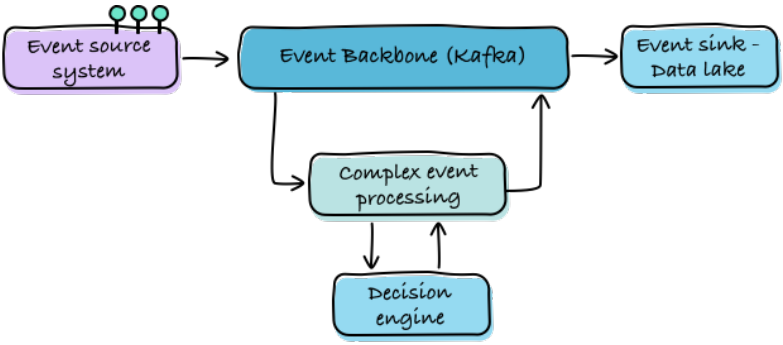

Learn about the IBM Event Driven architecture.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture/overview


External system
External system
Source connector
Sink connector
Producers
Kafka cluster
Consumers
Stream processors

# Integrate, then automate

Event driven solutions need to integrate with existing systems by using either SOA services, MQ queues as source, REST end points, a decision service, or a business process.

**APPLY BUSINESS RULES TO EVENTS AS THEY HAPPEN**




Learn more

Check out the Event-driven architecture.
https://www.ibm.com/cloud/architecture/architectures/eventDrivenArchitecture

For business process applications, BPMN supports the message construct used to trigger a process, including message consumption, generation of intermediate messages, and sending a message to a queue. Most of the business process management products use JMS, which needs queue to topic integration. This is supported by Kafka Connectors.

Integration flows deployed in an Enterprise Service Bus can also be integrated with Kafka and other event backbones in a no code manner. IBM® App Connect supports Kafka integration with simple end point and security configuration. For a Kafka expert, more advanced settings make it easy to send or consume messages.

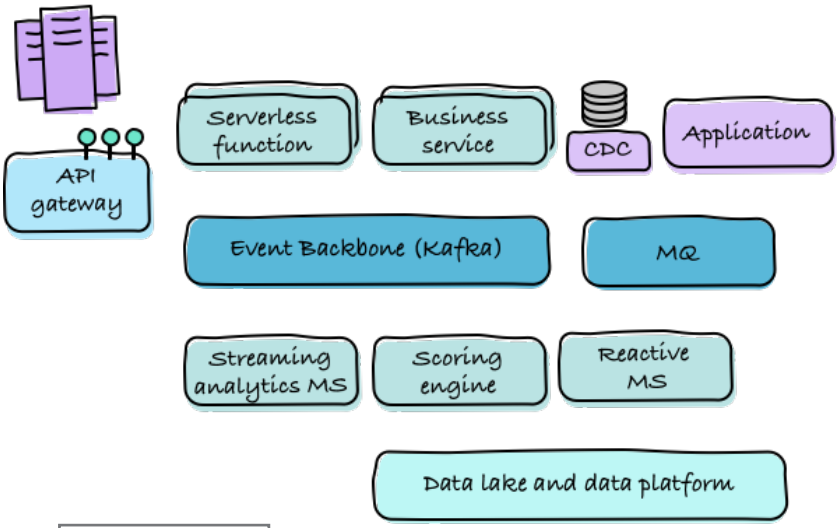
Integrate event and business rules by embedding the rule engine inside the data streaming processing. Integration with Apache Flink or Kafka Streams is possible with IBM® Operational Decision Management so the rule engine can apply business rules on top of events. Business analysts can edit the rules.

The AsynchAPI function in IBM API management helps to define metadata about Kafka Topics so consumer applications have visibility to what is deployed in Kafka and can integrate new events into their application.

# Event-driven solutions in action

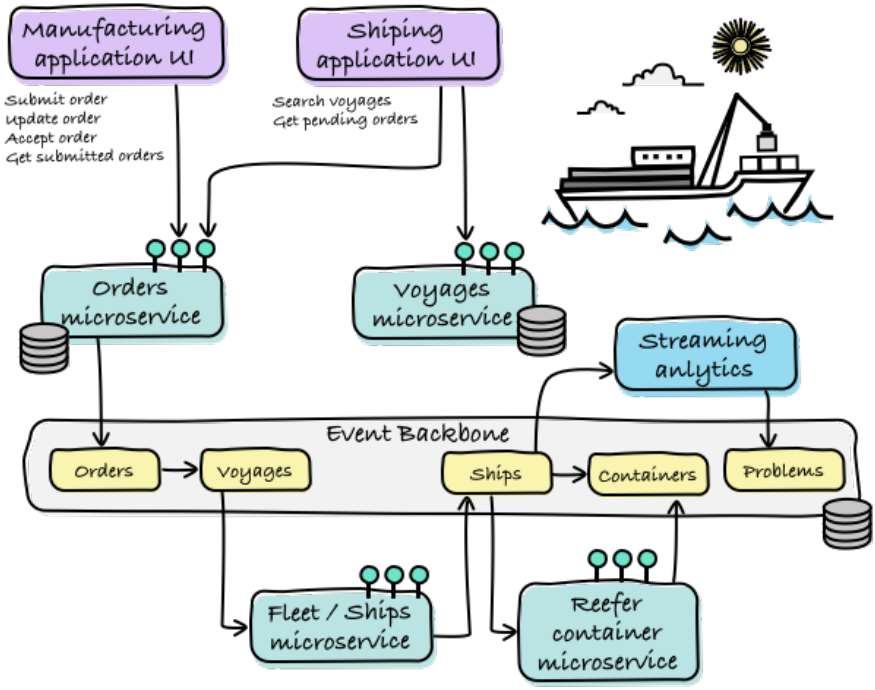
Putting event-driven concepts together in a single architecture is complex. In this example, a business application uses event-driven microservices, choreography to exchange data, a mobile channel, real time analytics via streaming apps, and data transformation and enrichment to a data lake - all integrated with mainframe transactional applications.

**ITS A BIG MAINFRAME EVENT-DRIVEN SOLUTION**




Learn more

Check out the solution.
https://ibm-cloud-architecture.github.io/refarch-kc/


Manufacturing application UI
Shiping application UI
Submit order
Update order
Accept order
Get submitted orders
Search voyages
Get pending orders
Orders microservice
Voyages microservice
Streaming anlytics
Event Backbone
Orders
Voyages
Ships
Containers
Problems
Fleet / Ships microservice
Reefer container microservice

# IBM can help

The IBM Garage™ helps businesses disrupt, innovate, operate, and motivate like a startup. The Garage can help you create inspiring experiences that produce significant business outcomes faster. IBM brings deep industry expertise to deliver successful client implementations.

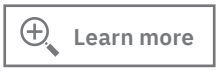
**LET IBM HELP YOU**

**Accelerate your journey with IBM Garage.** Focus on innovation and keep existing resources in place by letting our IBM Garage experts manage your ongoing integration needs.

**Adopt industry solutions.** Make use of IBM Global Business Services industry vertical solutions.

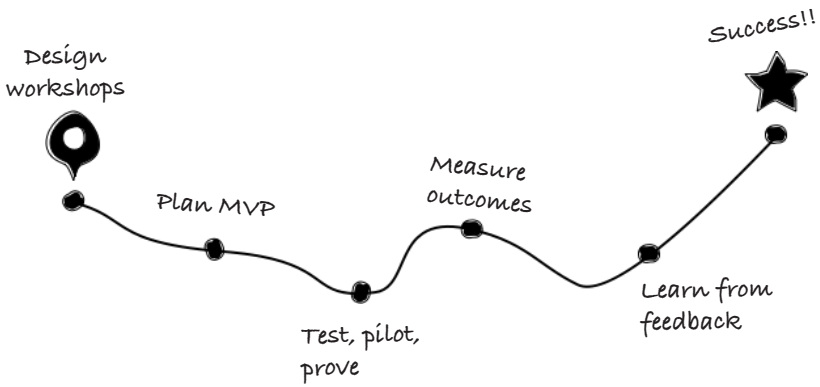
**Deploy and manage integration projects.** Let IBM help you to deploy and manage your solutions.

**Move forward with help from Expert Labs.** Work alongside our experts to assess your portfolio, implement quick, agile change and set the course for your integration future.




Check out the IBM Garage.
https://www.ibm.com/garage

## Engage IBM experts!




**IBM is a trusted partner, providing technology and prescriptive guidance to deliver immediate business value.**

# Notes:

## Learn about the IBM Cloud Pak for Integration

https://www.ibm.com/cloud/cloud-pak-for-integration

## Assess your integration maturity

https://www.ibm.com/account/reg/us-en/signup?formid=urx-48428

## Learn about the IBM Cloud Pak for Automation

https://www.ibm.com/cloud/cloud-pak-for-automation

## Learn more about Red Hat OpenShift

https://www.openshift.com/

## Check out IBM Cloud Paks!!

https://www.ibm.com/cloud/paks

## Visit an IBM Garage

https://www.ibm.com/garage

To support best practices and patterns that demonstrate the value of an event-driven solution, there are two open source reference implementations.

https://ibm-cloud-architecture.github.io/refarch-kc/

# Notices



IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

**IBM Director of Licensing**
**IBM Corporation**
**North Castle Drive, MD-NC119**
**Armonk, NY 10504-1785**
**US**

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

**Trademarks**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.