

Reacting to an Event-Driven World

Grace Jansen

IBM Developer Advocate

@gracejansen27

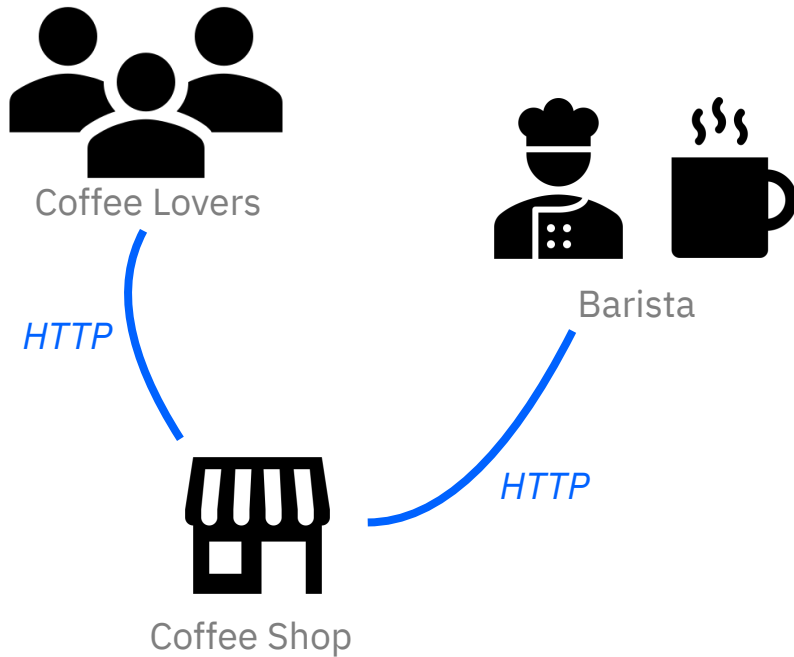
The logo for CONF42, featuring the word "CONF" in a bold, sans-serif font with a stylized planet and ring system integrated into the letter "O", followed by the number "42" in a similar bold font. The entire logo is white and set against a dark gray rectangular background.

Let's get some coffee...

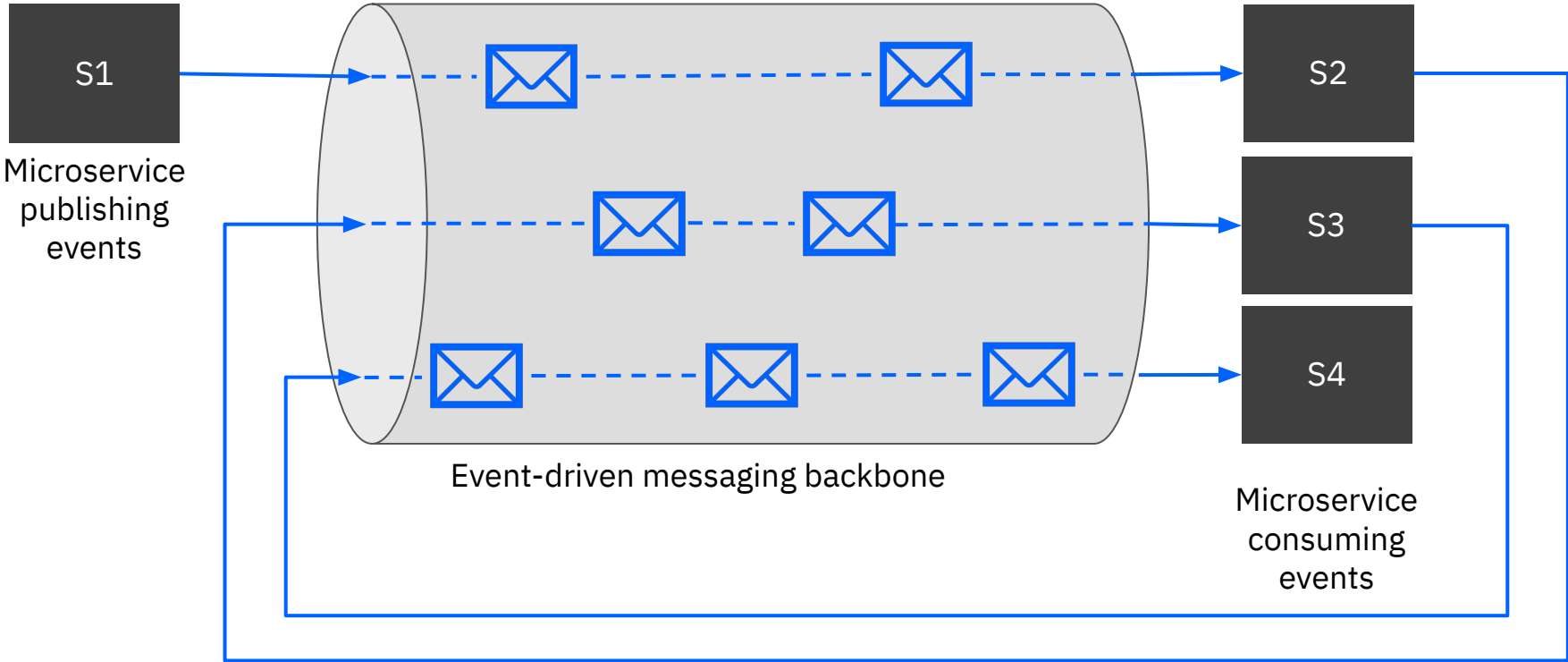


Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>



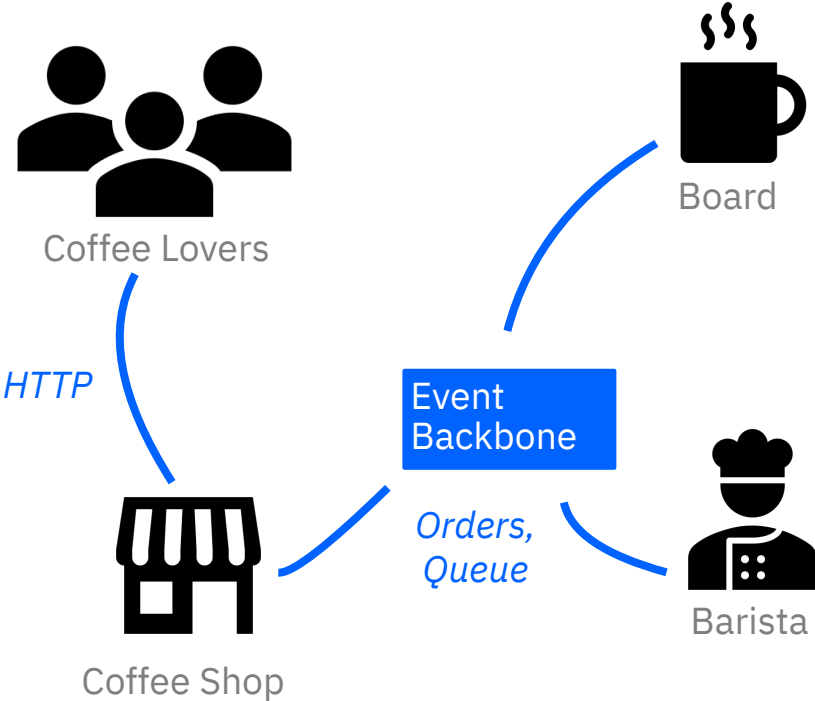
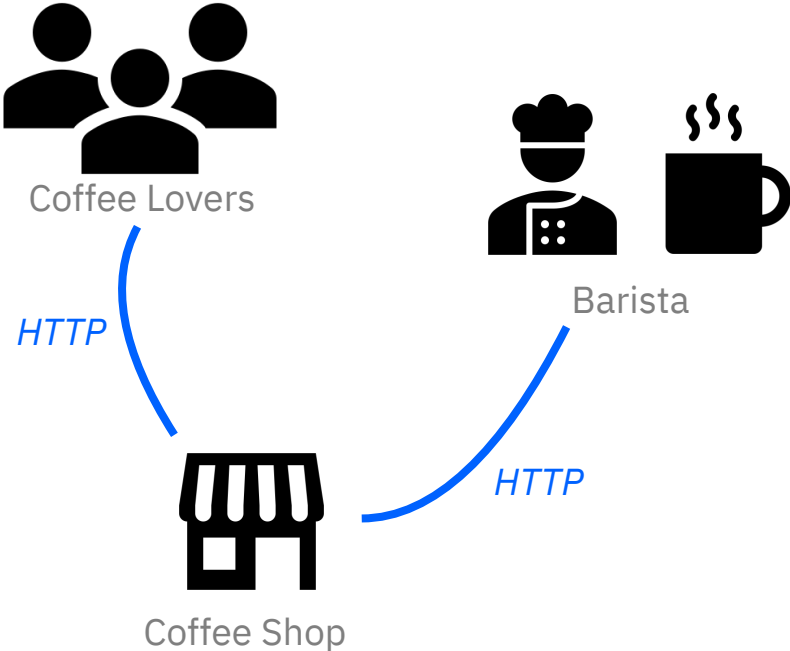
Event Driven Architecture



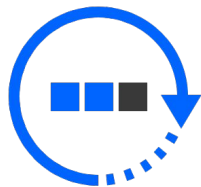
<http://ibm.biz/AdvantagesOfEDA>

Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>



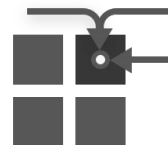
Apache Kafka is an **open source, distributed streaming platform**



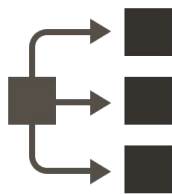
Stream history



Immutable data



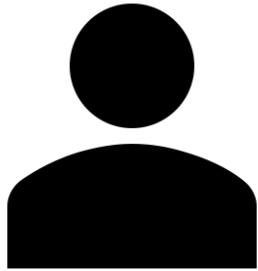
Highly available



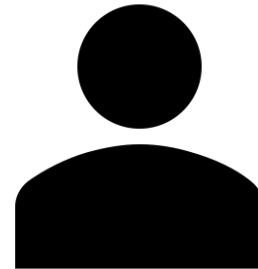
Scalable consumption

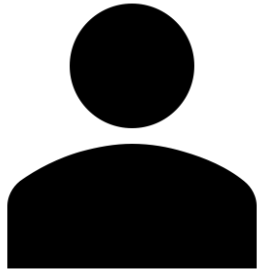


Scalable

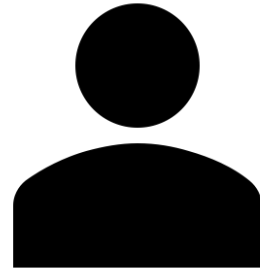


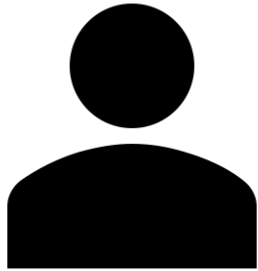
Q: Is your coffee shop non-blocking and highly responsive?





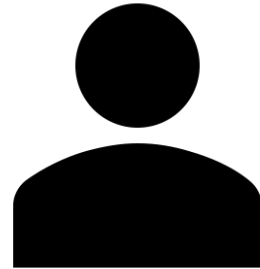
Q: Is your **microservice system** non-blocking and highly responsive?

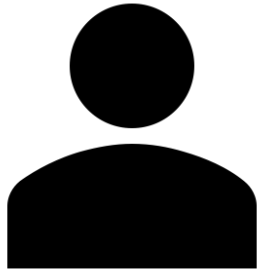




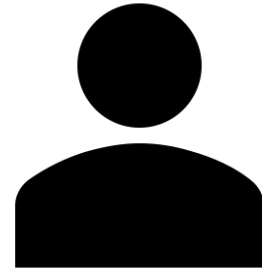
Q: Is your microservice system non-blocking and highly responsive?

A: Yes I'm using Kafka!



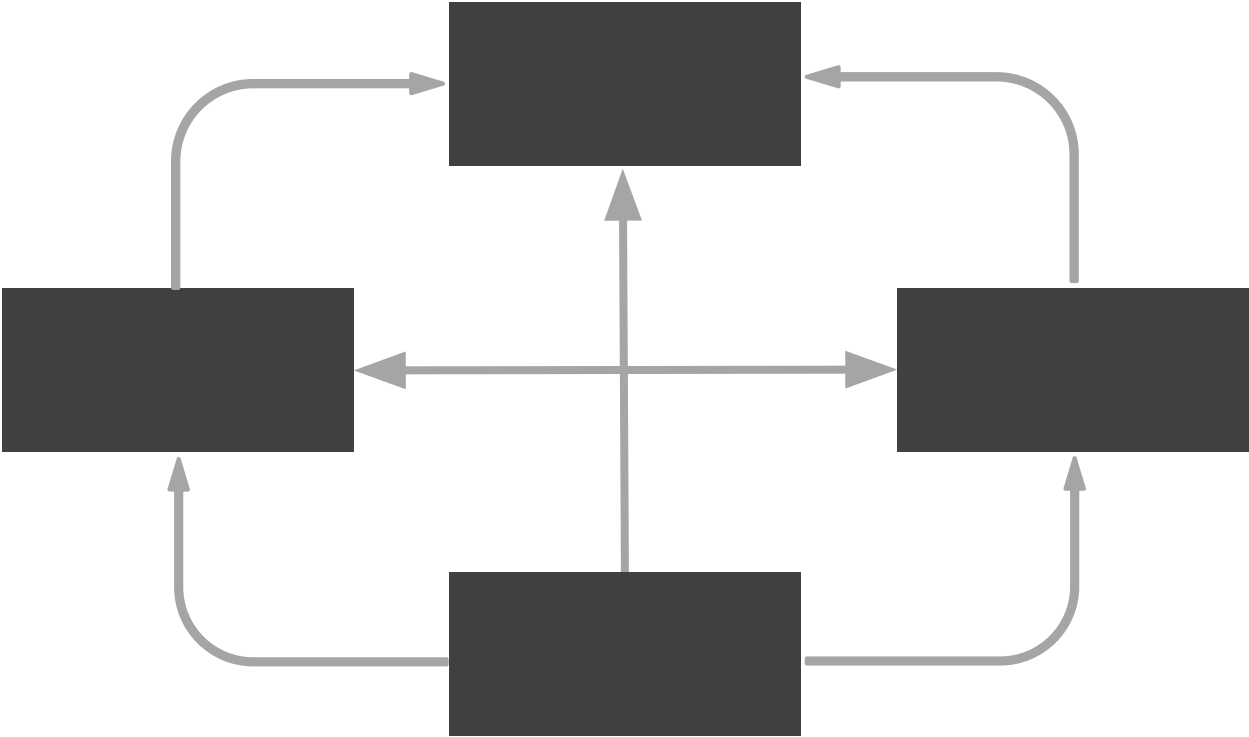


Q: Is your microservice system
non-blocking and highly responsive?

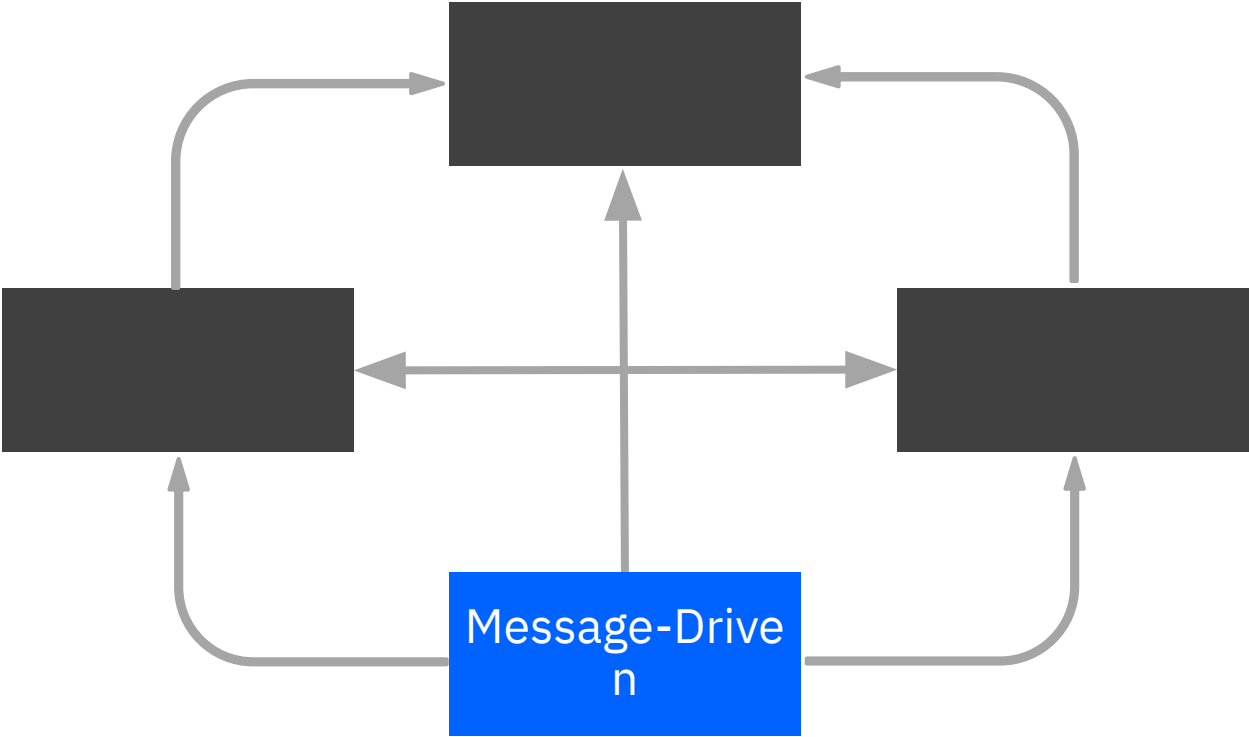


Reactive systems

Reactive Manifesto



Reactive Manifesto



Messages

“An item of data sent to a specific location.”

Events

“A signal emitted by a component upon reaching a given state.”

A message can contain an encoded event in its payload.

Apache Kafka is an **open source, distributed streaming platform**



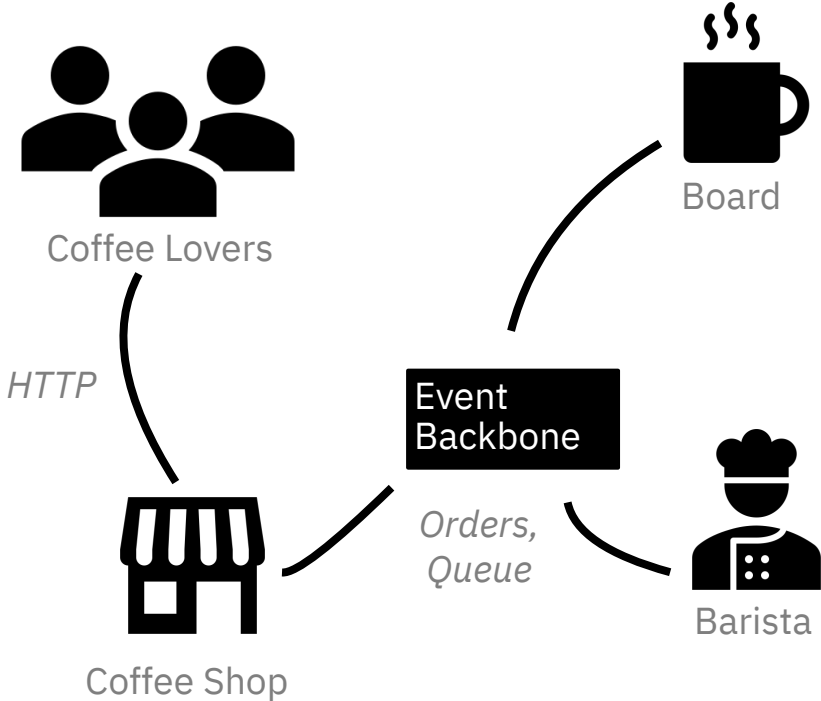
Publish and subscribe to streams of records

Store records in durable way

Process streams of records as they occur

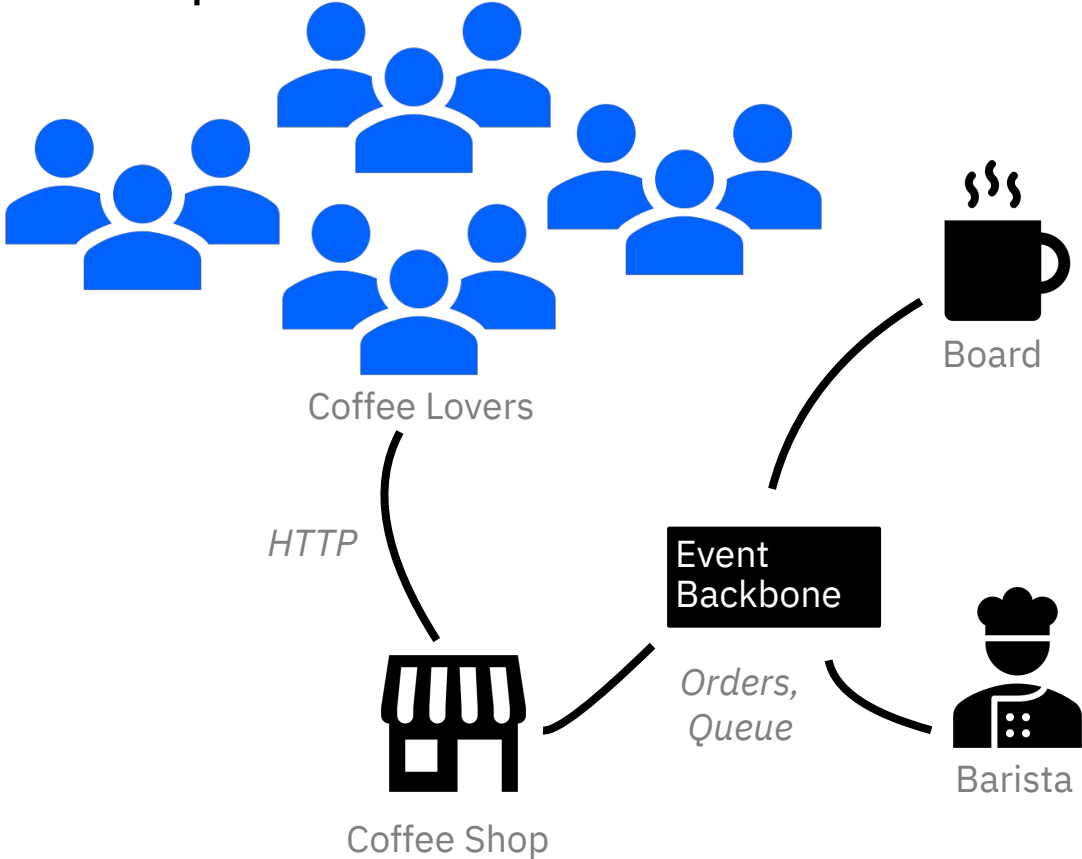
Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>



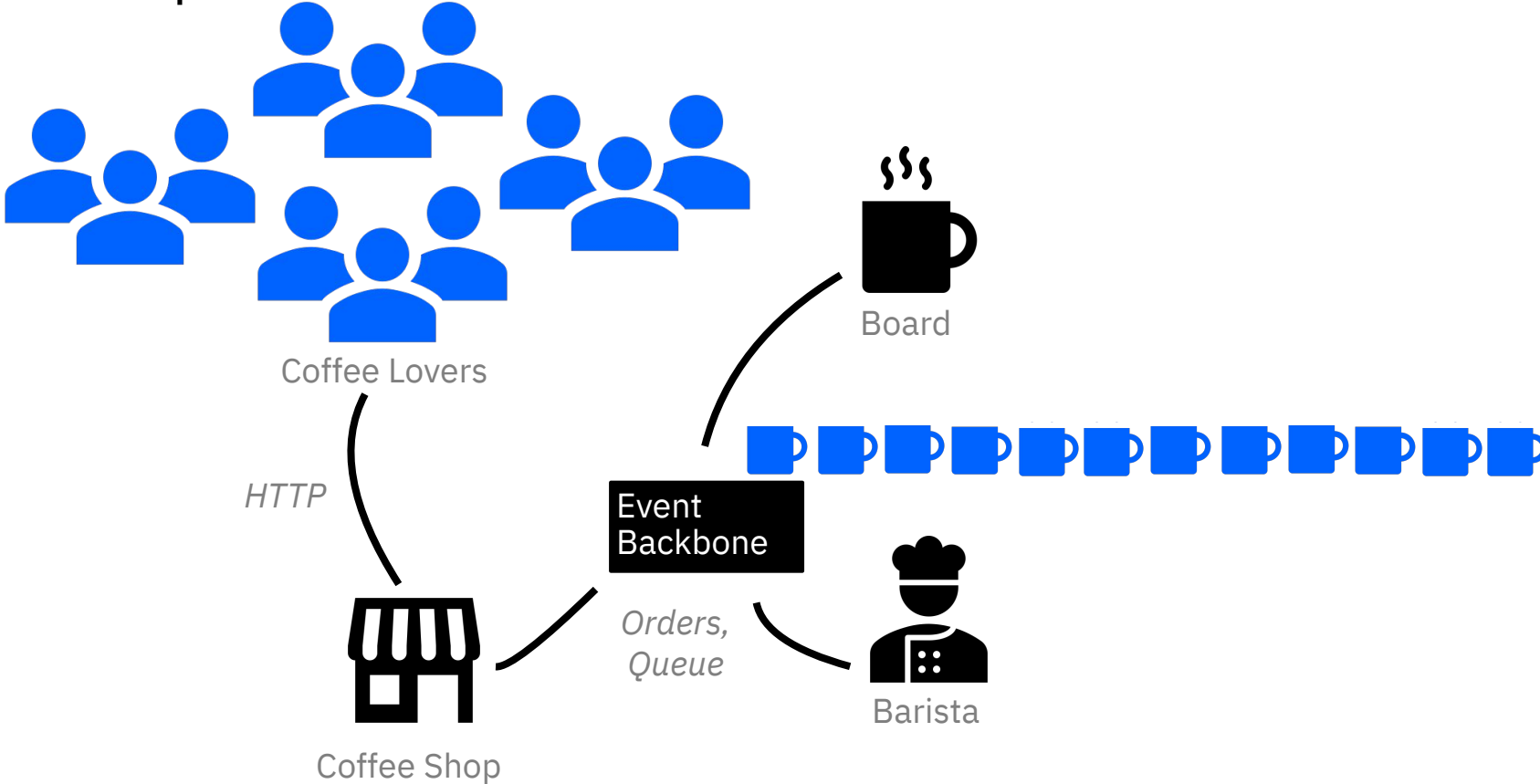
Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>

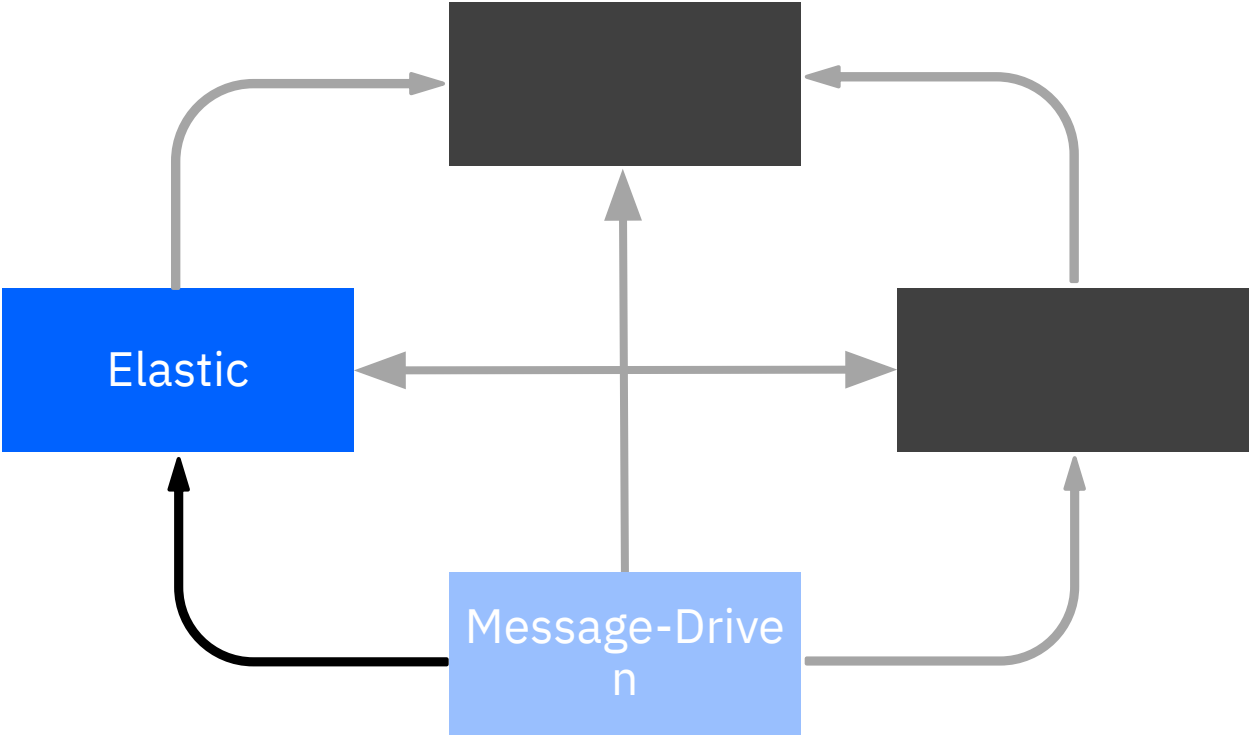


Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>

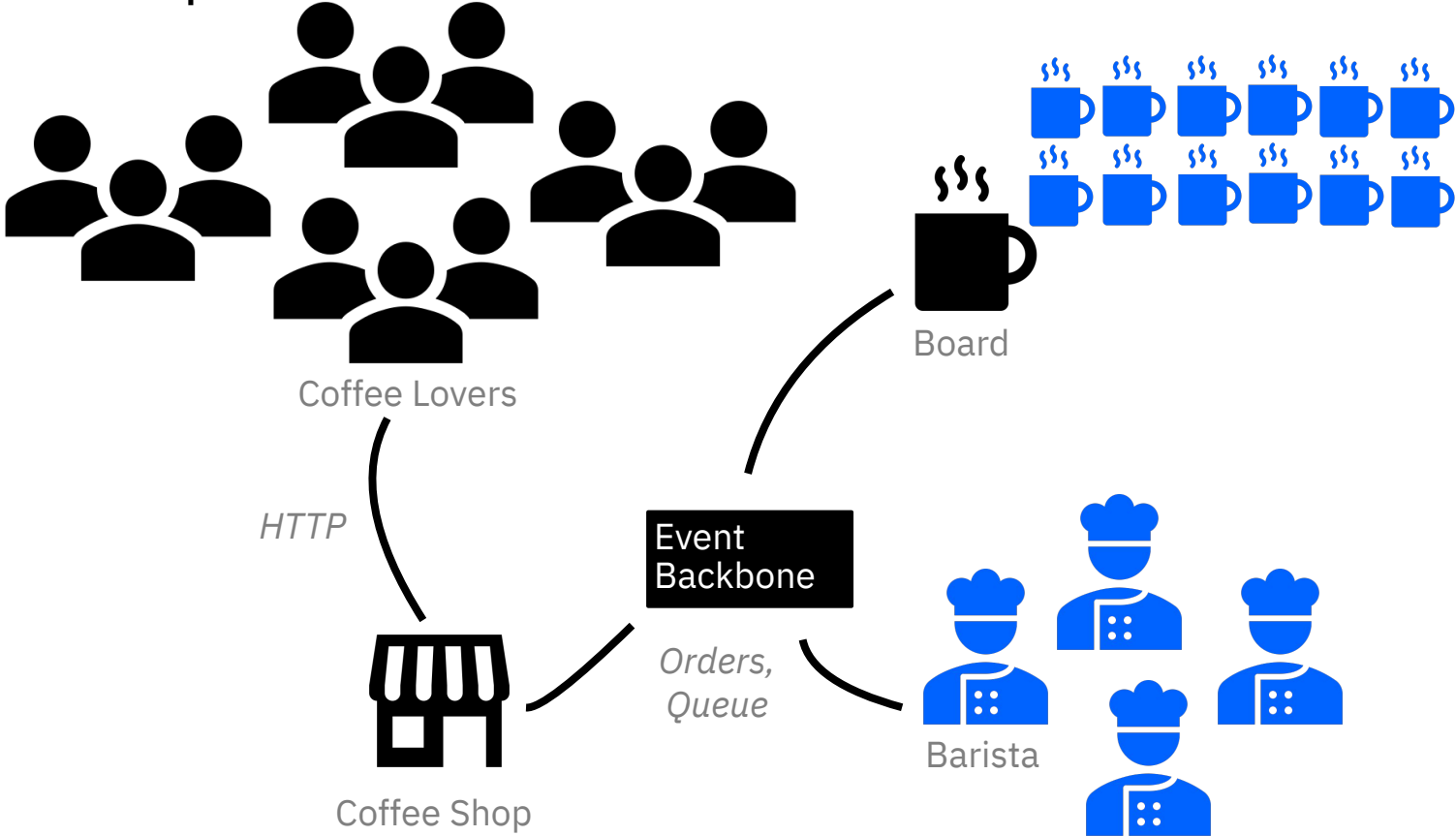


Reactive Manifesto



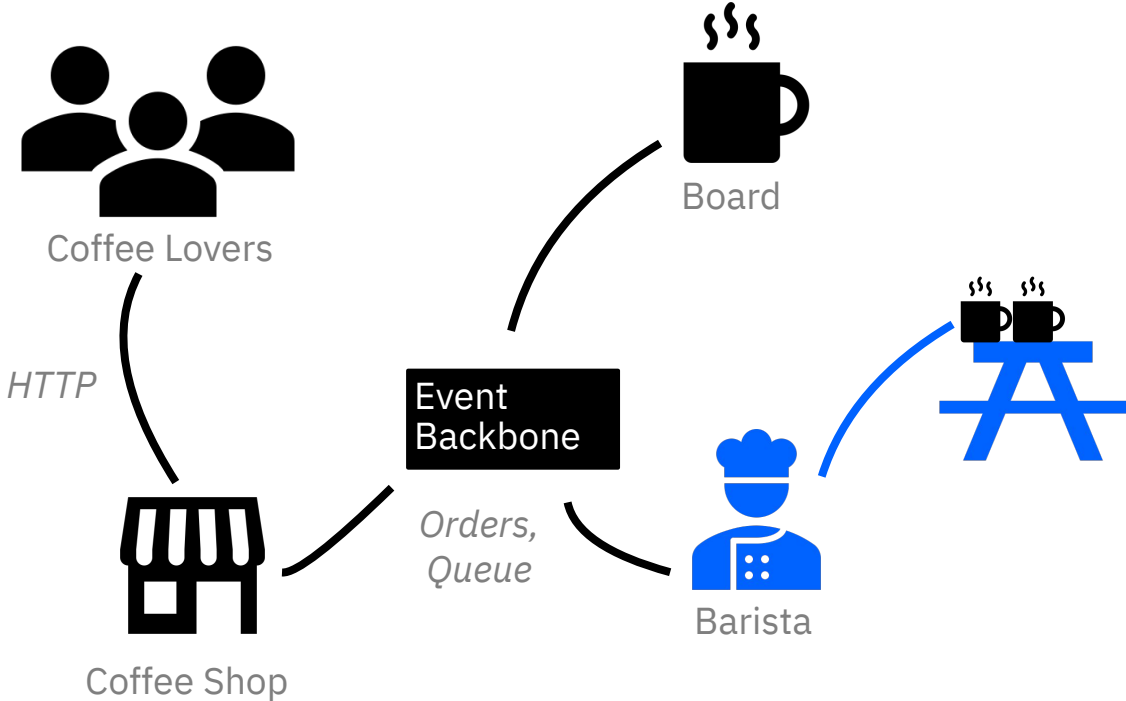
Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>



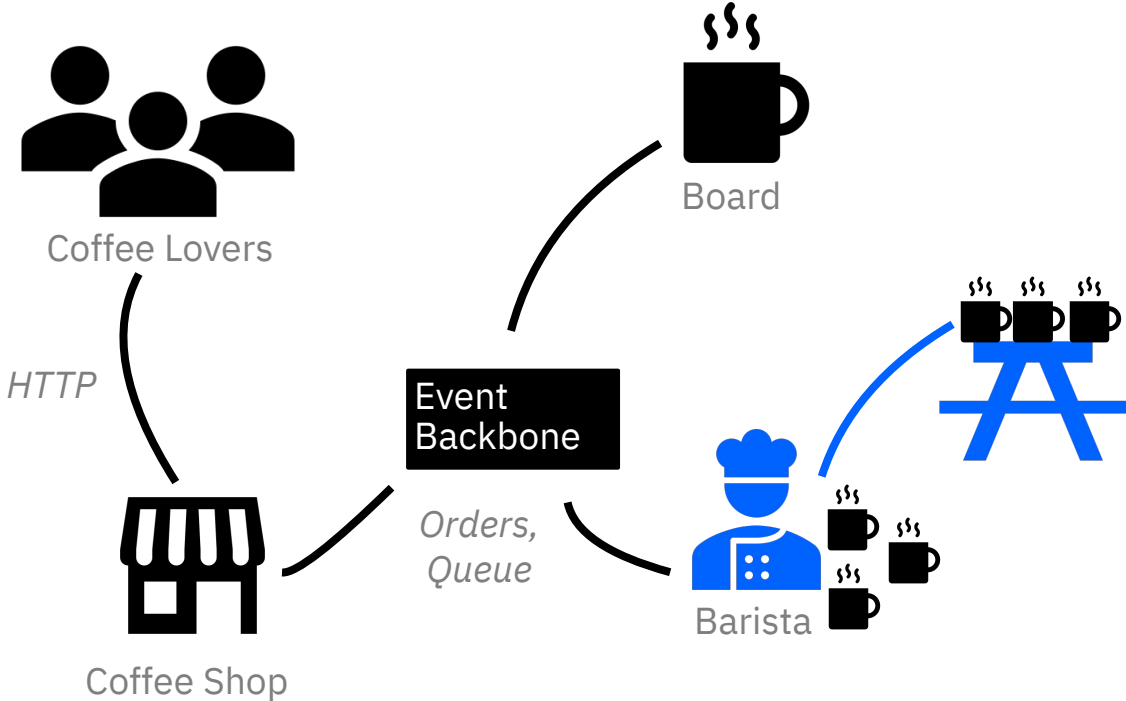
Barista Example:

<https://github.com/cescoffier/quarkus-coffeeshop-demo>

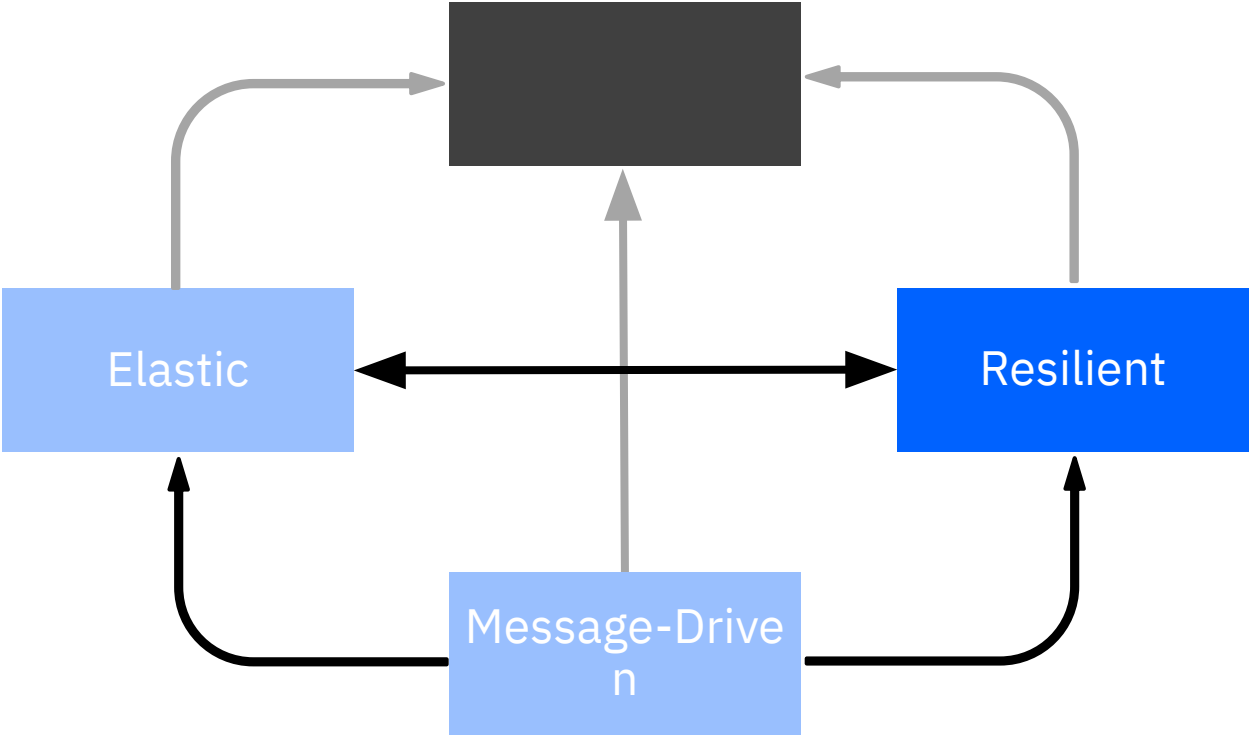


Barista Example:

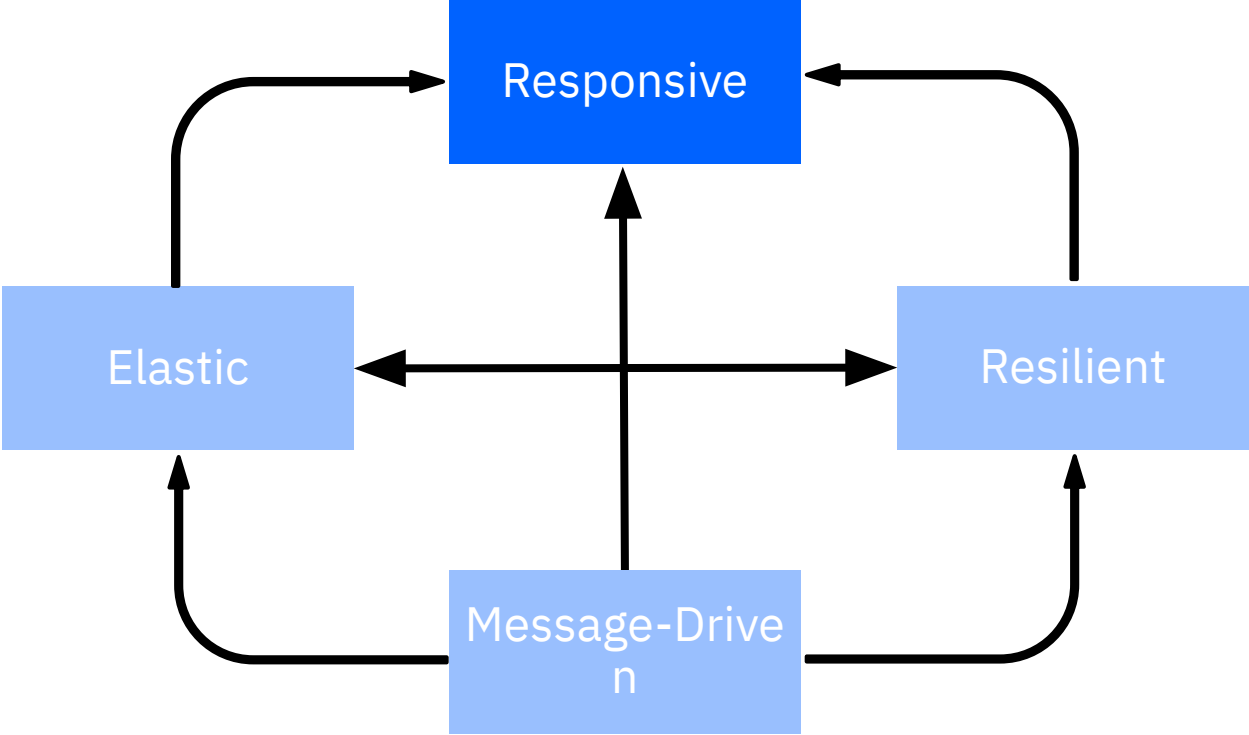
<https://github.com/cescoffier/quarkus-coffeeshop-demo>



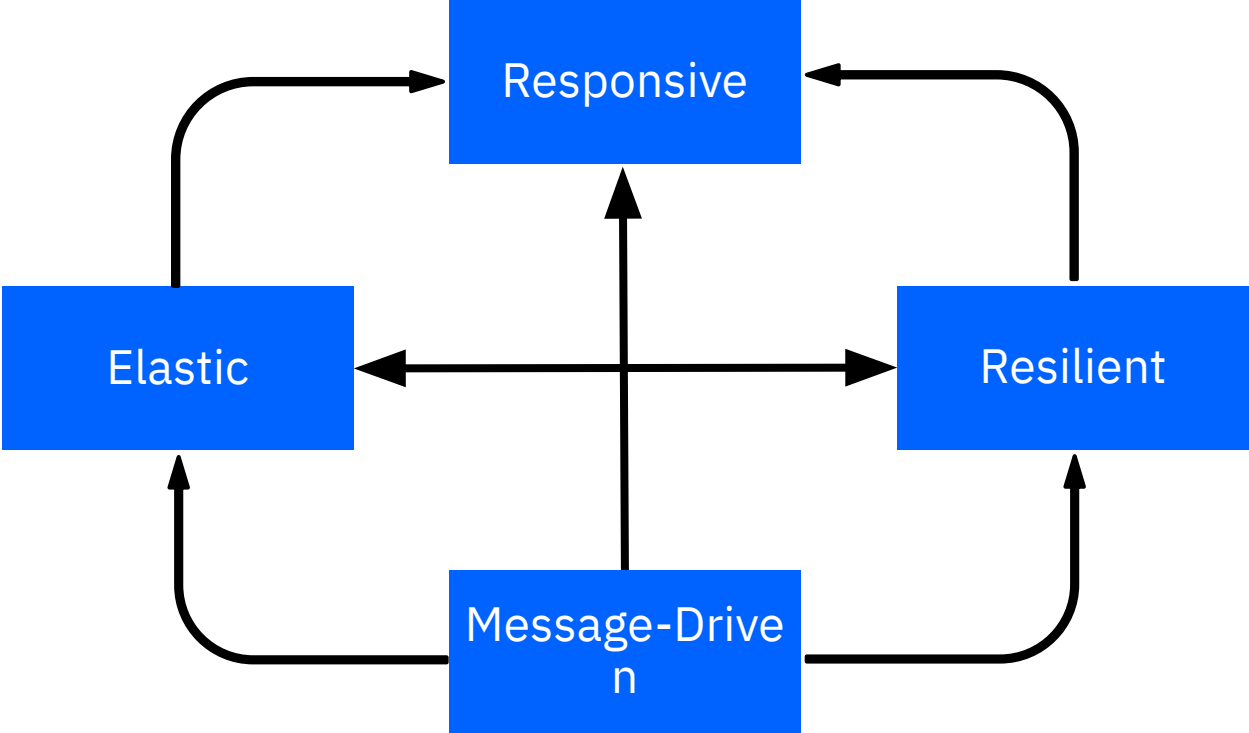
Reactive Manifesto



Reactive Manifesto

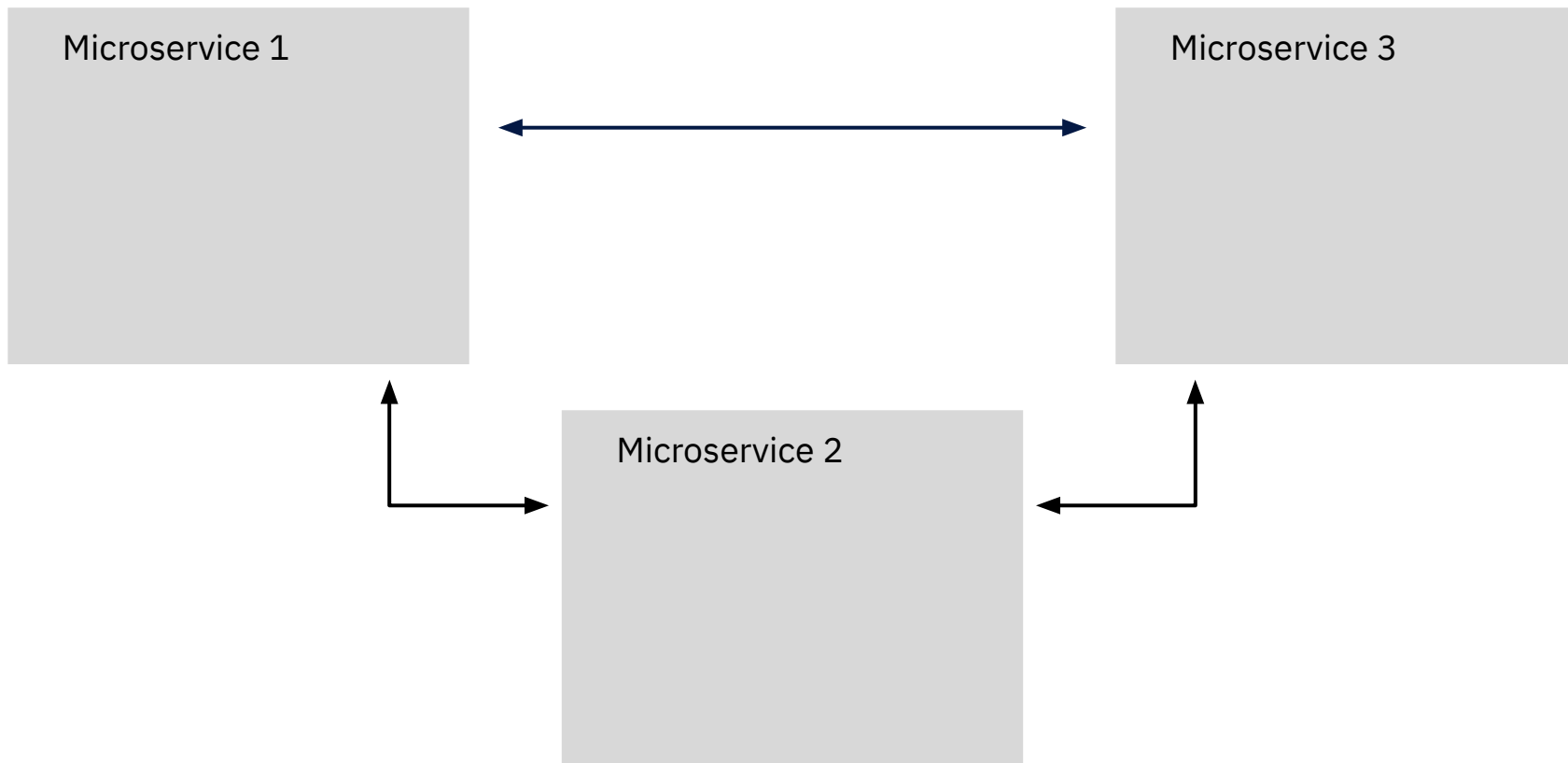


Reactive Manifesto

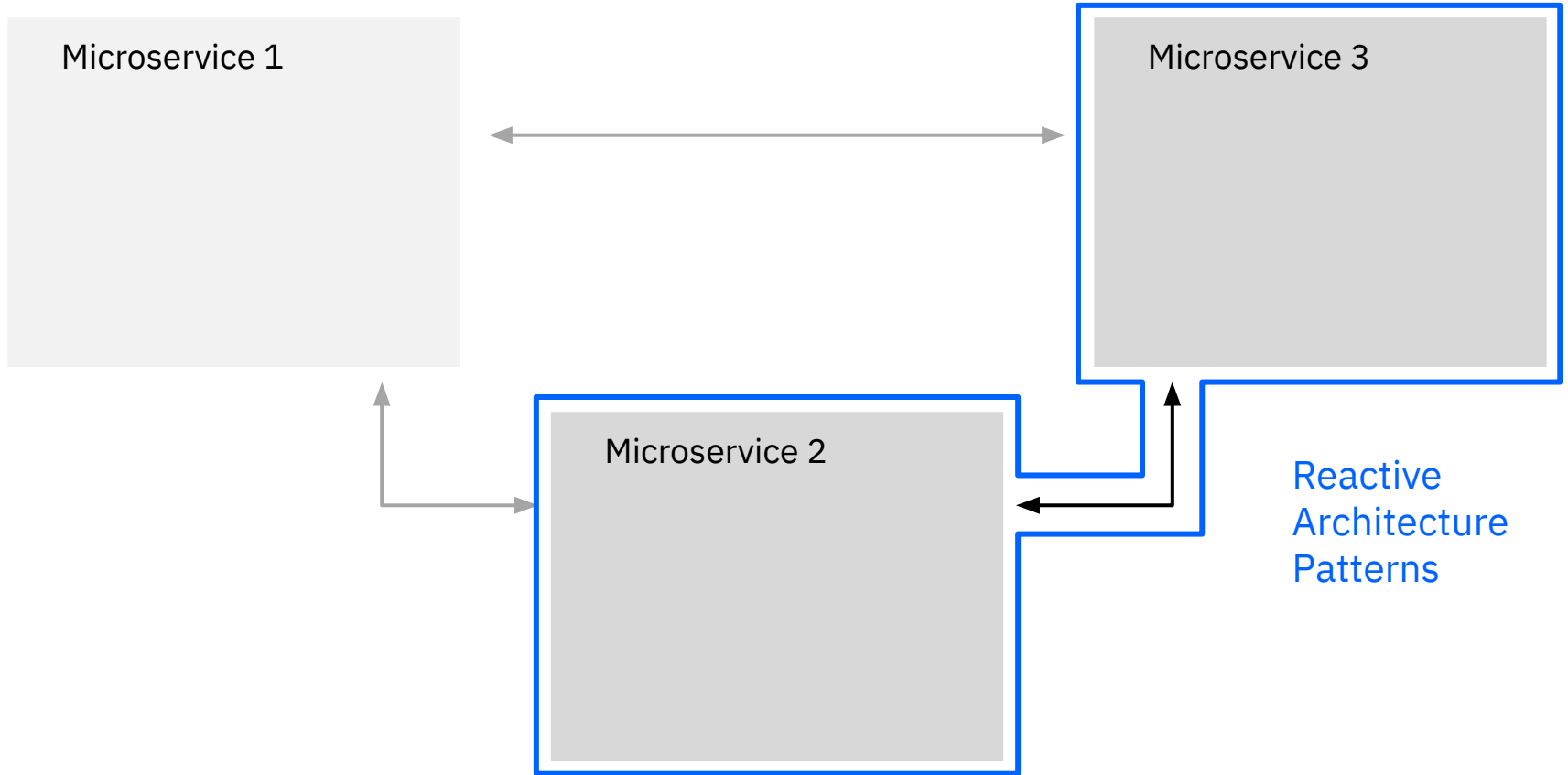


Building reactive systems

How do we make a highly responsive app?

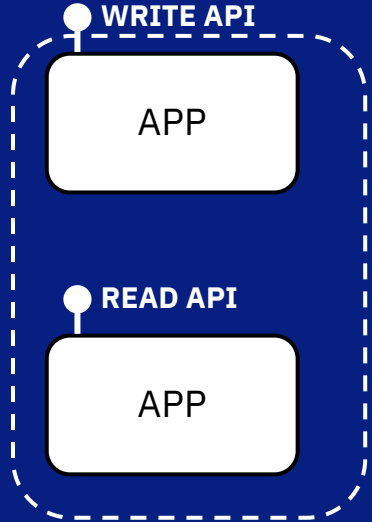


How do we make a highly responsive app?

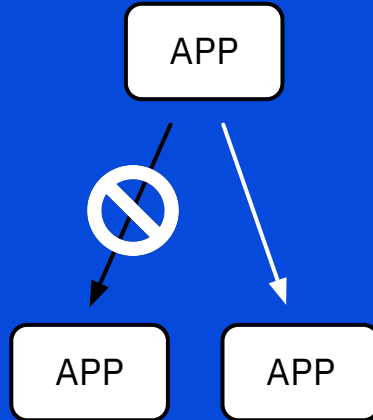


Reactive Architecture design patterns

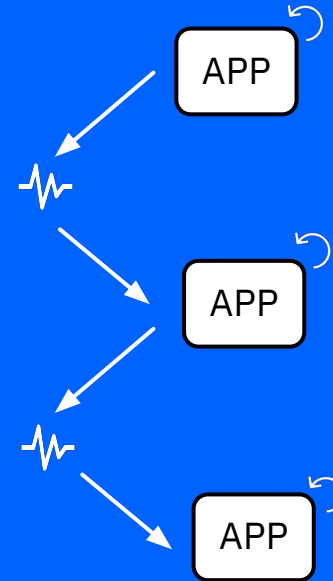
CQRS



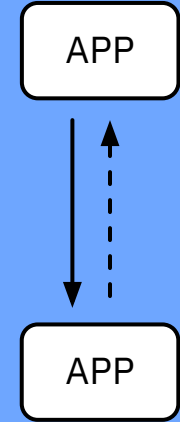
Circuit breaker



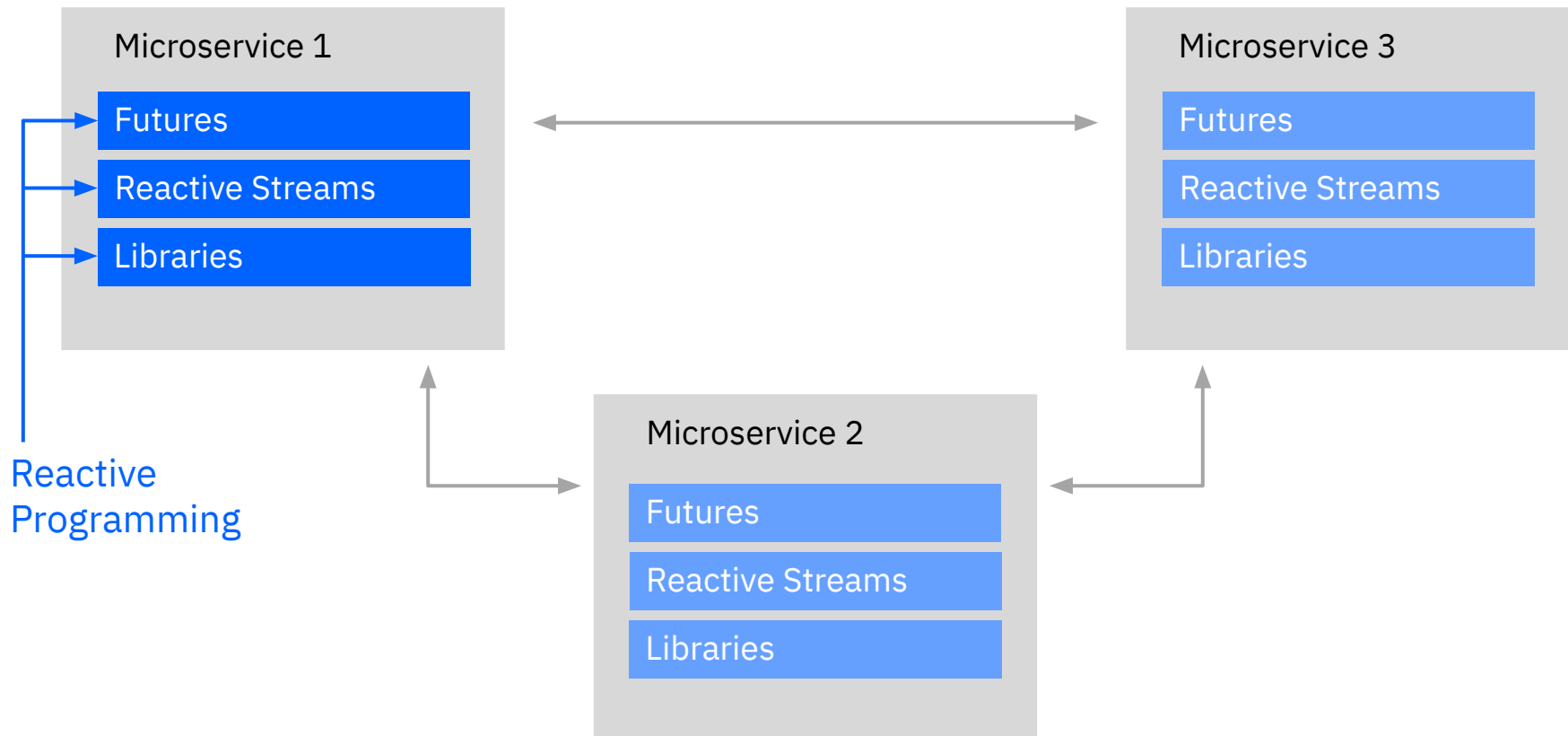
Saga



Back pressure



How do we make a highly responsive app?



Reactive Programming

A subset of [asynchronous programming](#) and a paradigm where the [availability of new information drives the logic forward](#) rather than having control flow driven by a thread-of-execution.

Reactive Programming Patterns

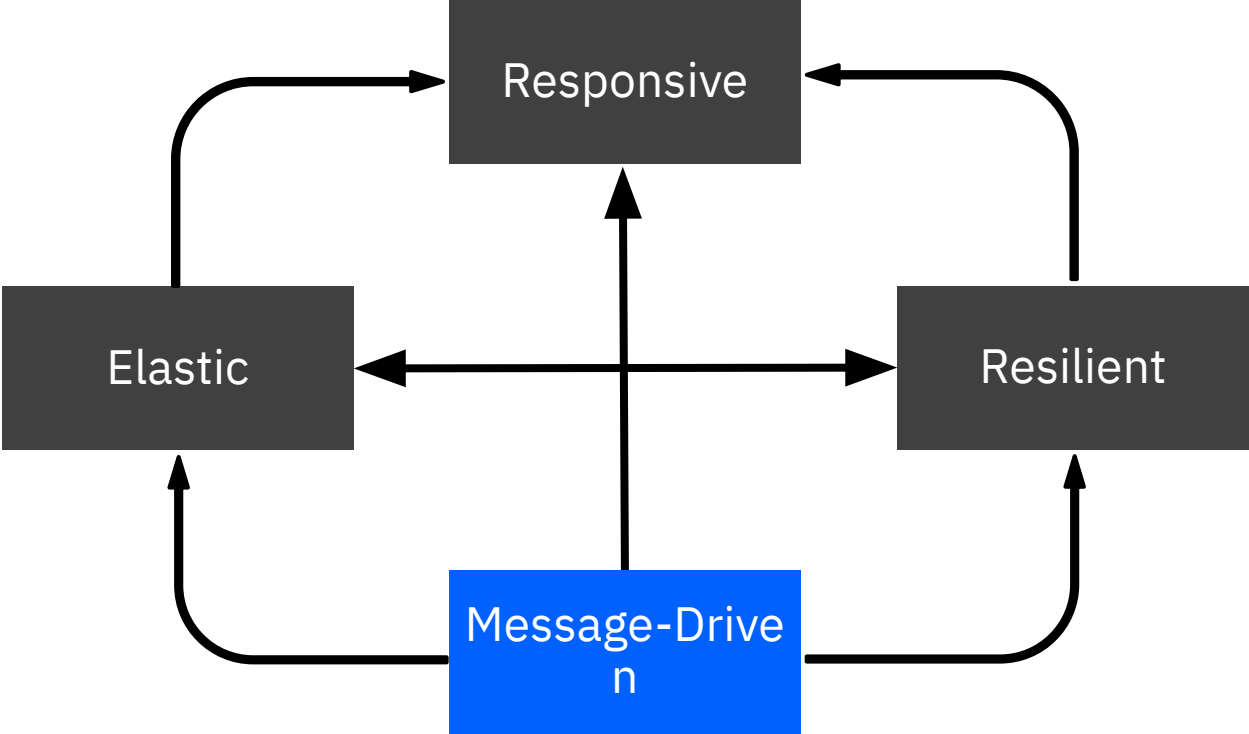
Futures: a **promise** to hold the result of some operation once that operation completes

Reactive programming libraries: for composing asynchronous and event-based programs. (e.g. RxJava, SmallRye Mutiny)

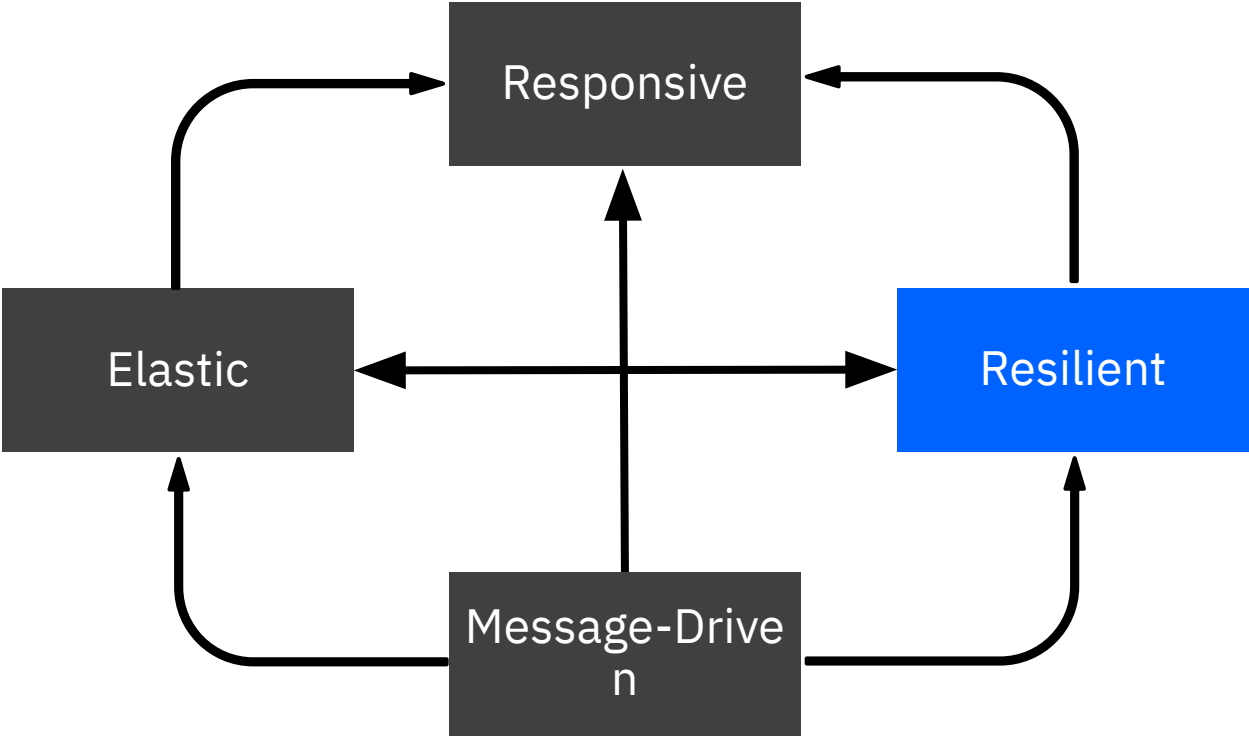
Reactive Streams: a programming concept for **handling asynchronous data streams** in a non-blocking manner while providing backpressure to stream publishers

Utilising Kafka in reactive systems

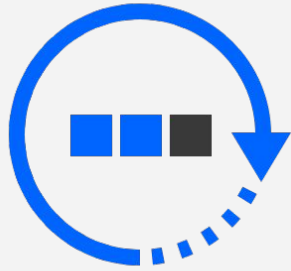
Resiliency in Kafka



Resiliency in Kafka



Message Retention and Data Persistence

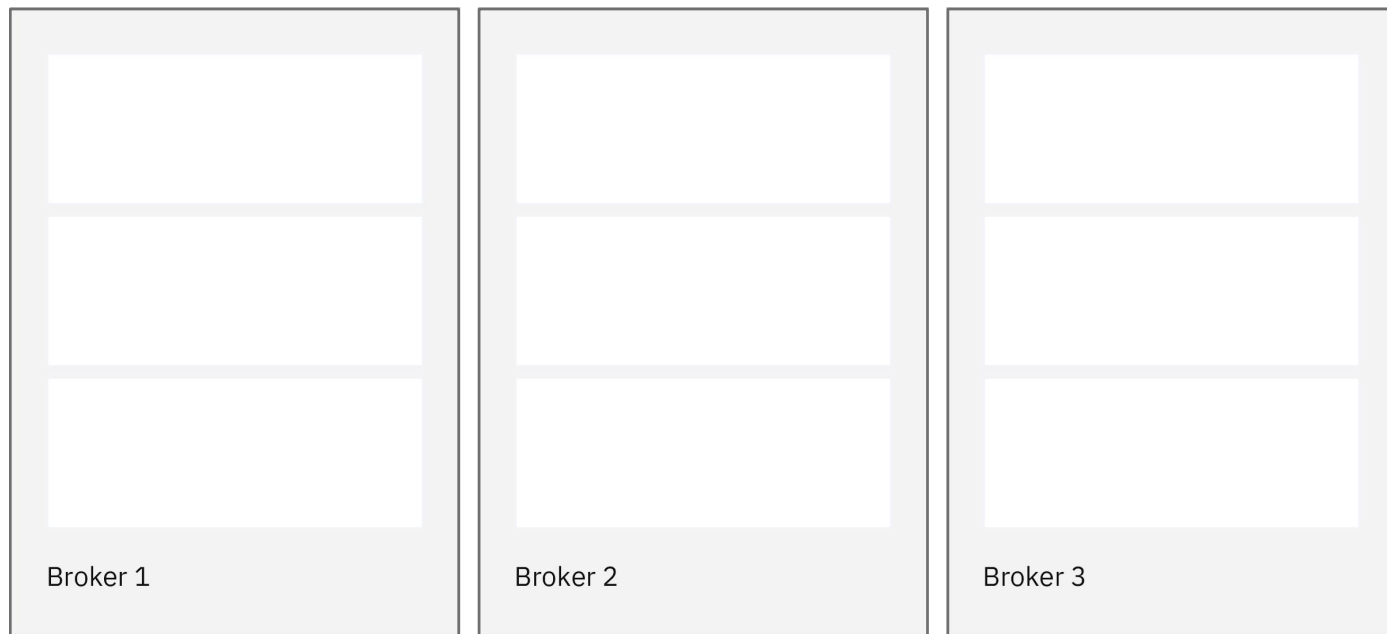


Stream history

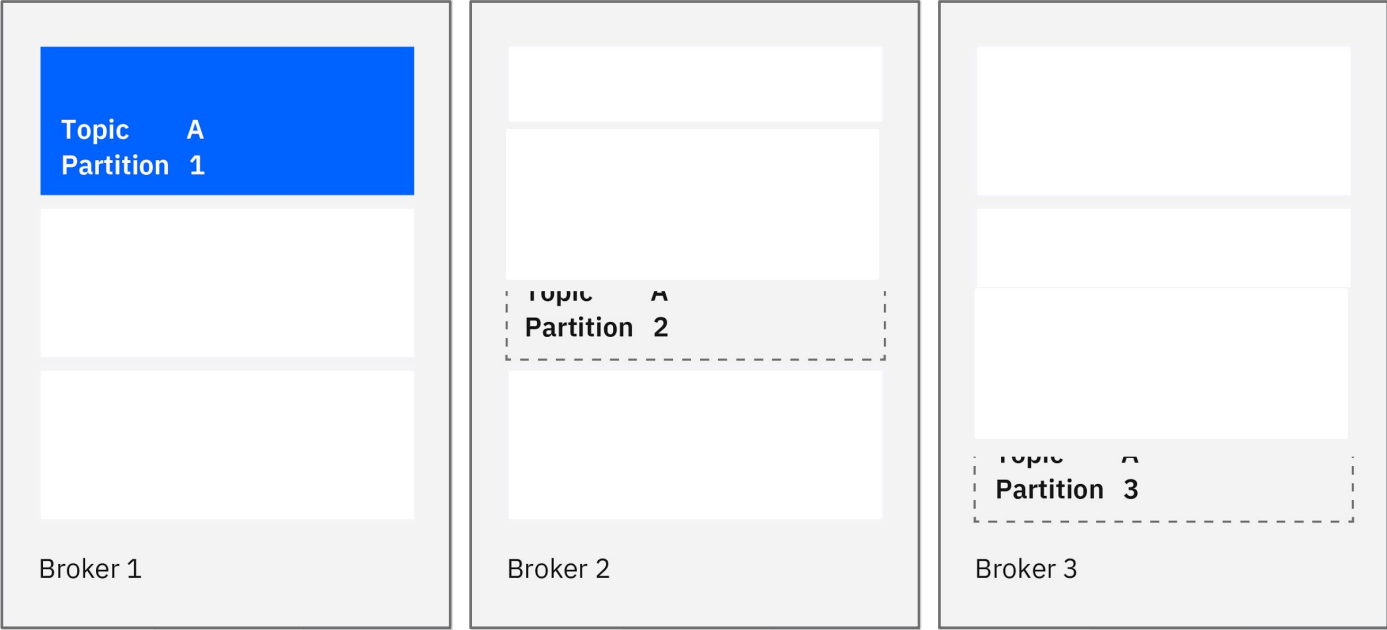


Immutable Data

Resilient Kafka Clusters

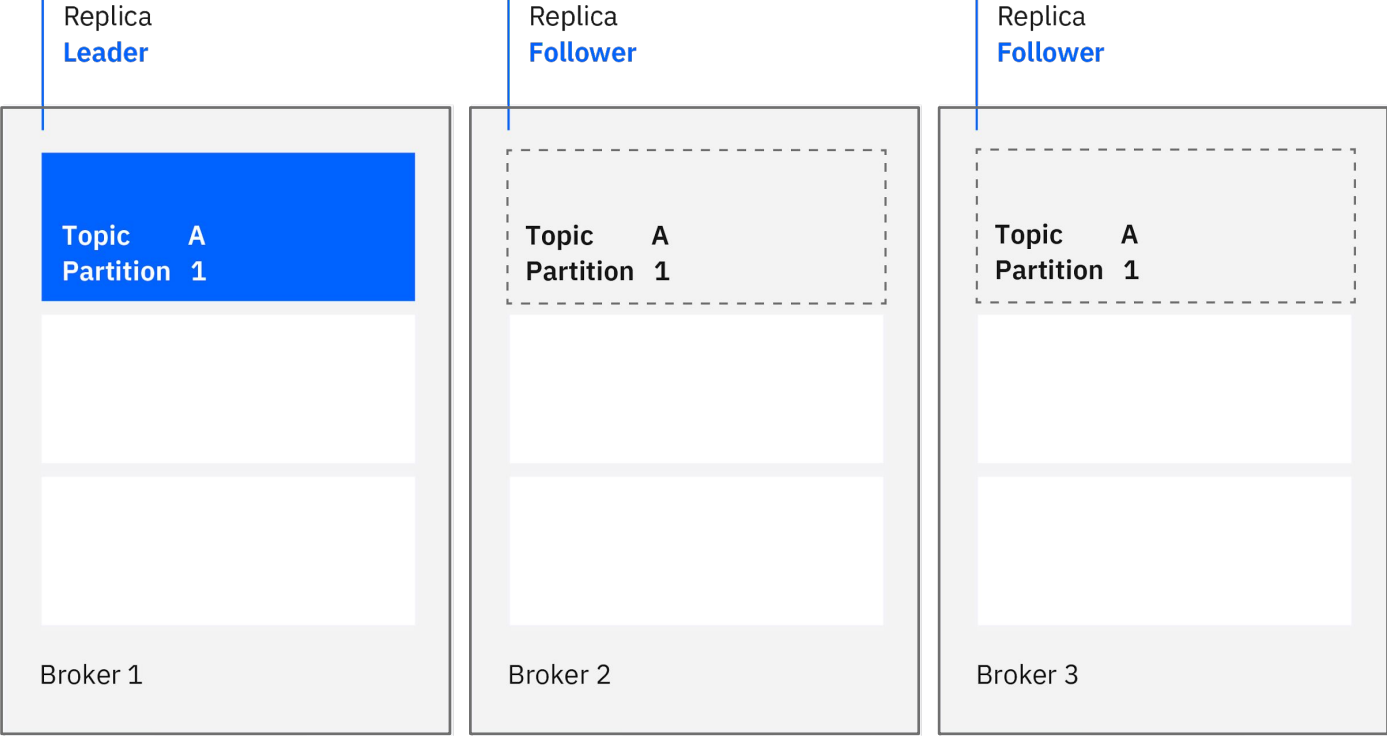


Resilient Kafka Clusters



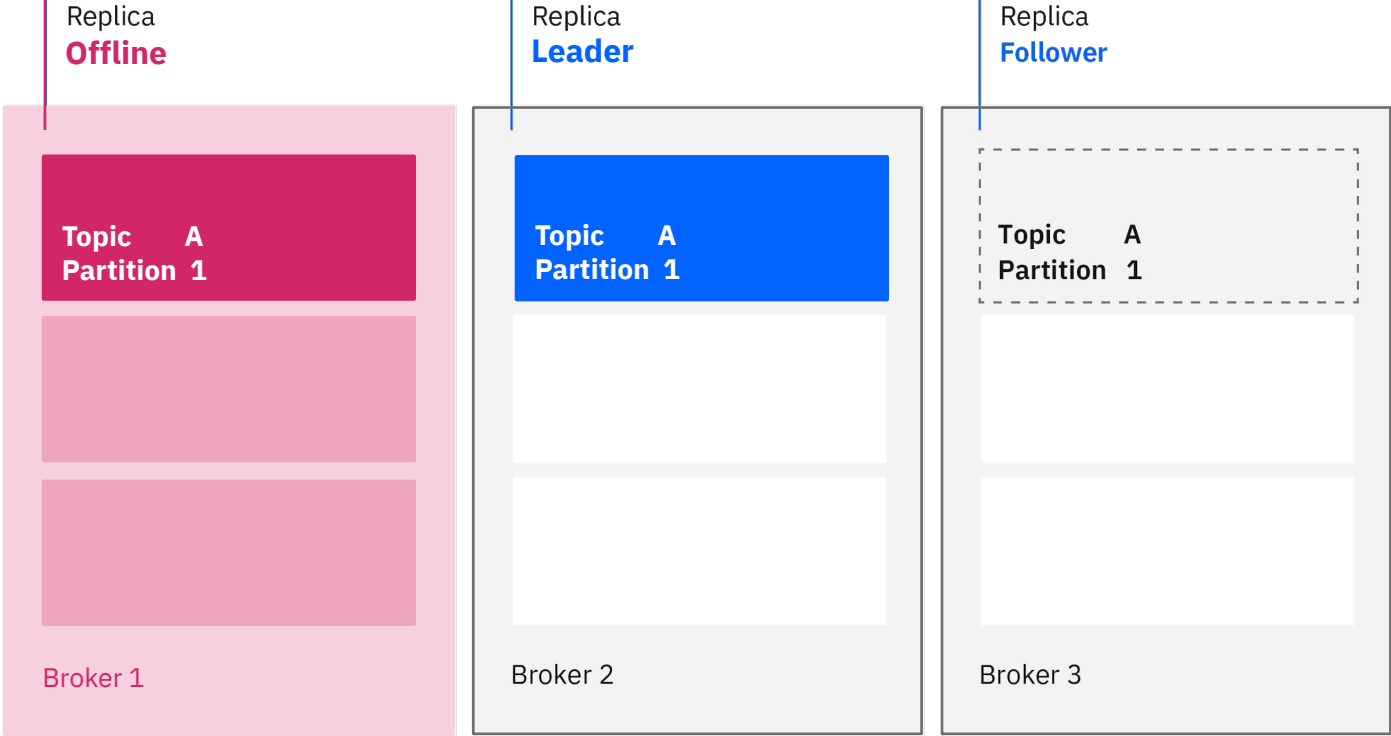
Kafka Cluster

Resilient Kafka Clusters



Kafka Cluster

Resilient Kafka Clusters



Kafka Cluster

Resilient Producers

Delivery guarantees:

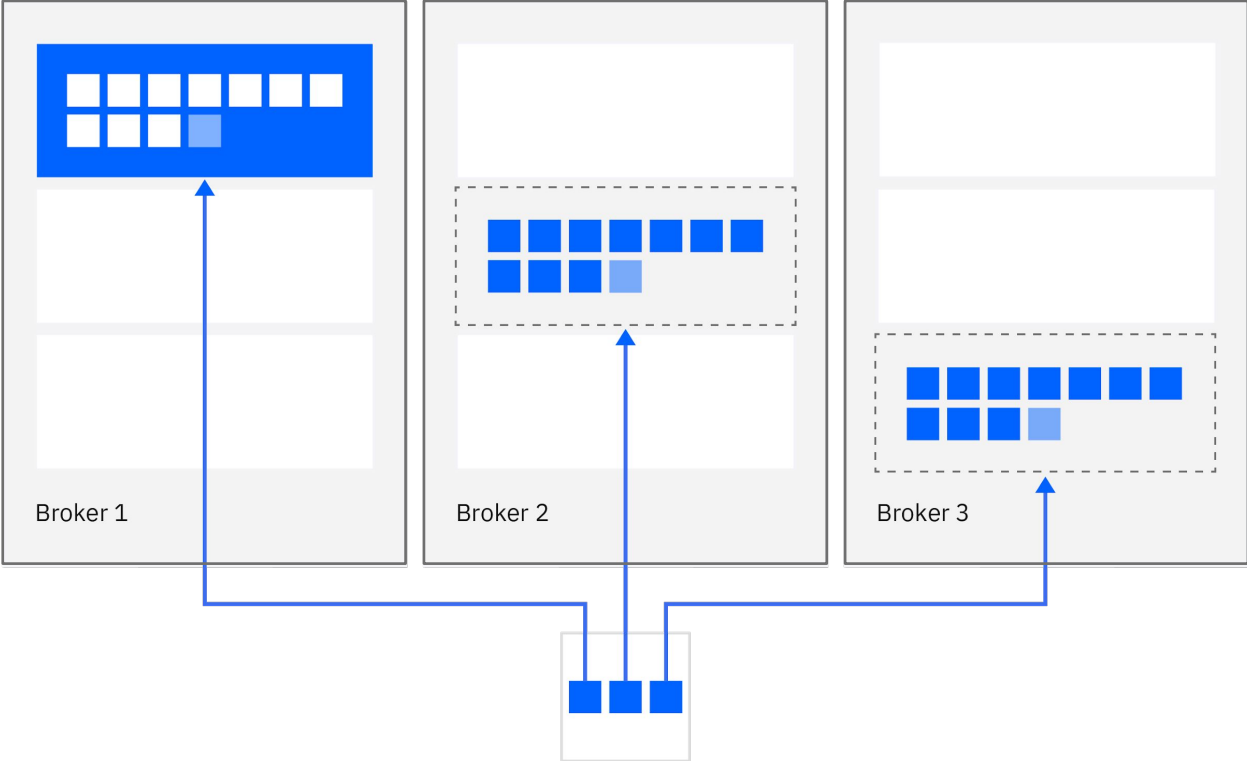
At most once

At least once

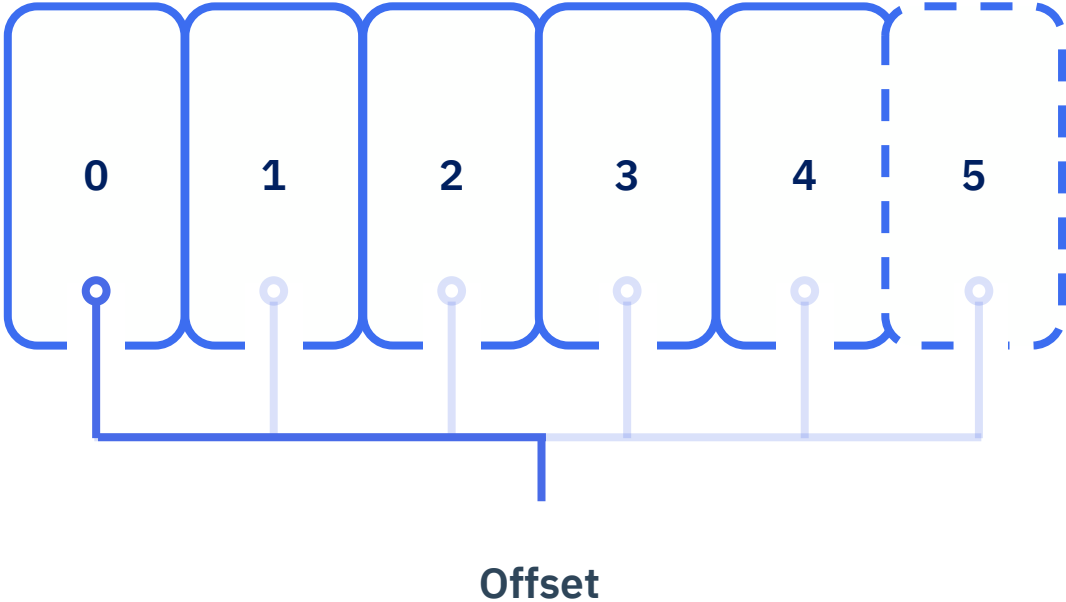
Configuration:

Acks

Retries



Resilient Consumers



Baristas with auto commit



Coffee

Cappuccino

Latte

TOPIC



Barista

Baristas with auto commit

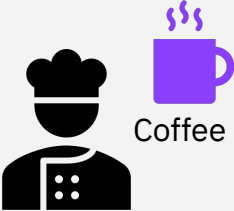


Coffee

Cappuccino

Latte

TOPIC



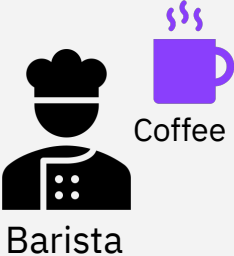
Coffee

Barista

Baristas with auto commit



TOPIC



Baristas with auto commit



TOPIC



Barista

Baristas with auto commit



Coffee

Cappuccino

Latte

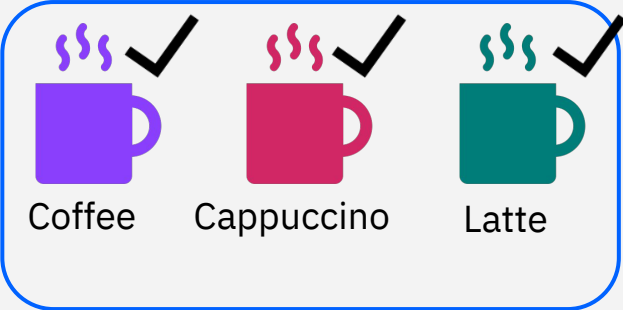
TOPIC



Cappuccino

Barista

Baristas with auto commit



TOPIC

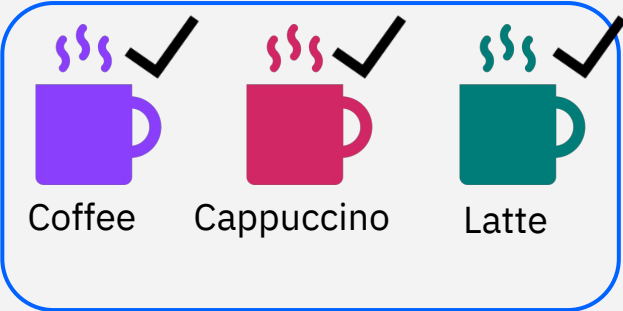


Barista



Cappuccino Latte

Baristas with auto commit



TOPIC



Barista



Cappuccino Latte

Baristas with manual commit

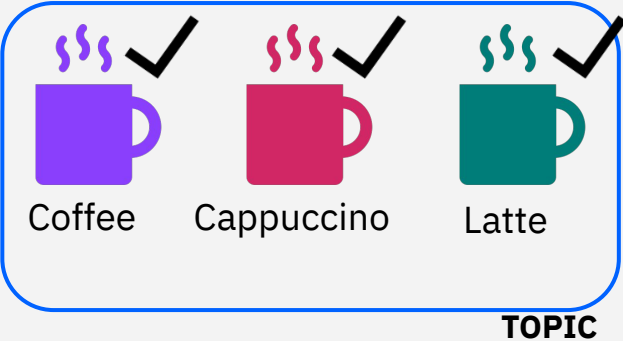


TOPIC



Barista

Baristas with auto commit

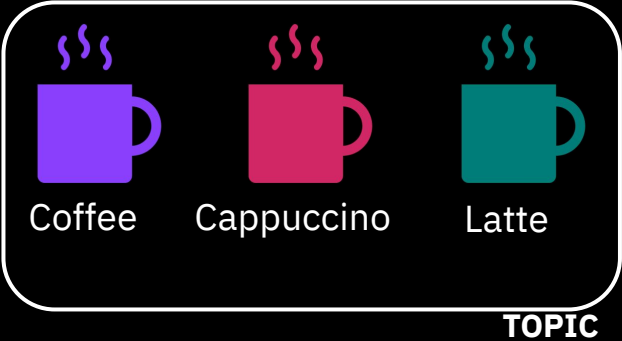


Barista



Cappuccino Latte

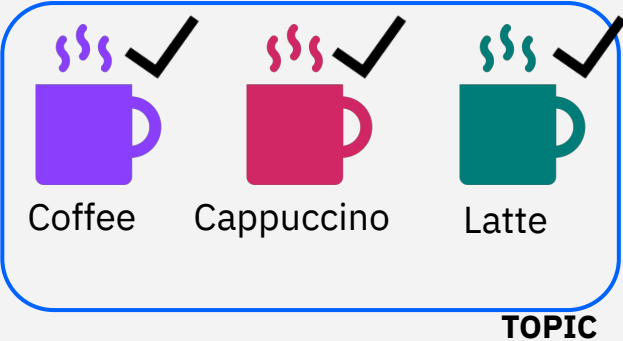
Baristas with manual commit



Barista

Coffee

Baristas with auto commit

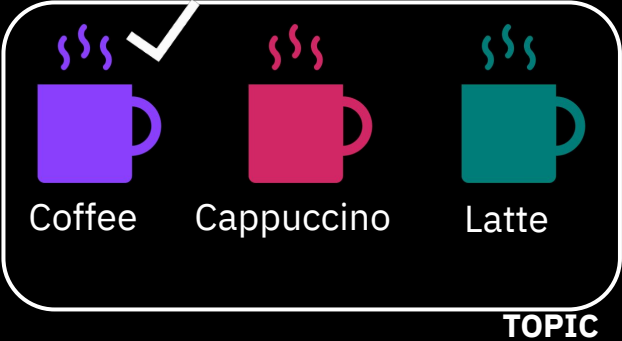


Barista



Cappuccino Latte

Baristas with manual commit

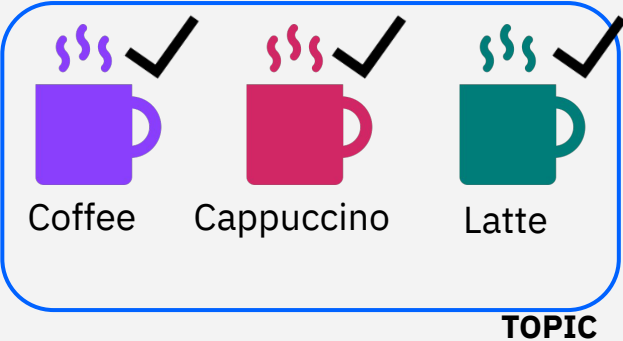


Barista



Coffee

Baristas with auto commit

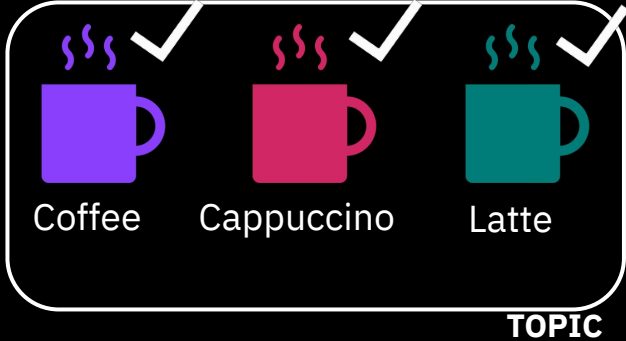


Barista



Cappuccino Latte

Baristas with manual commit

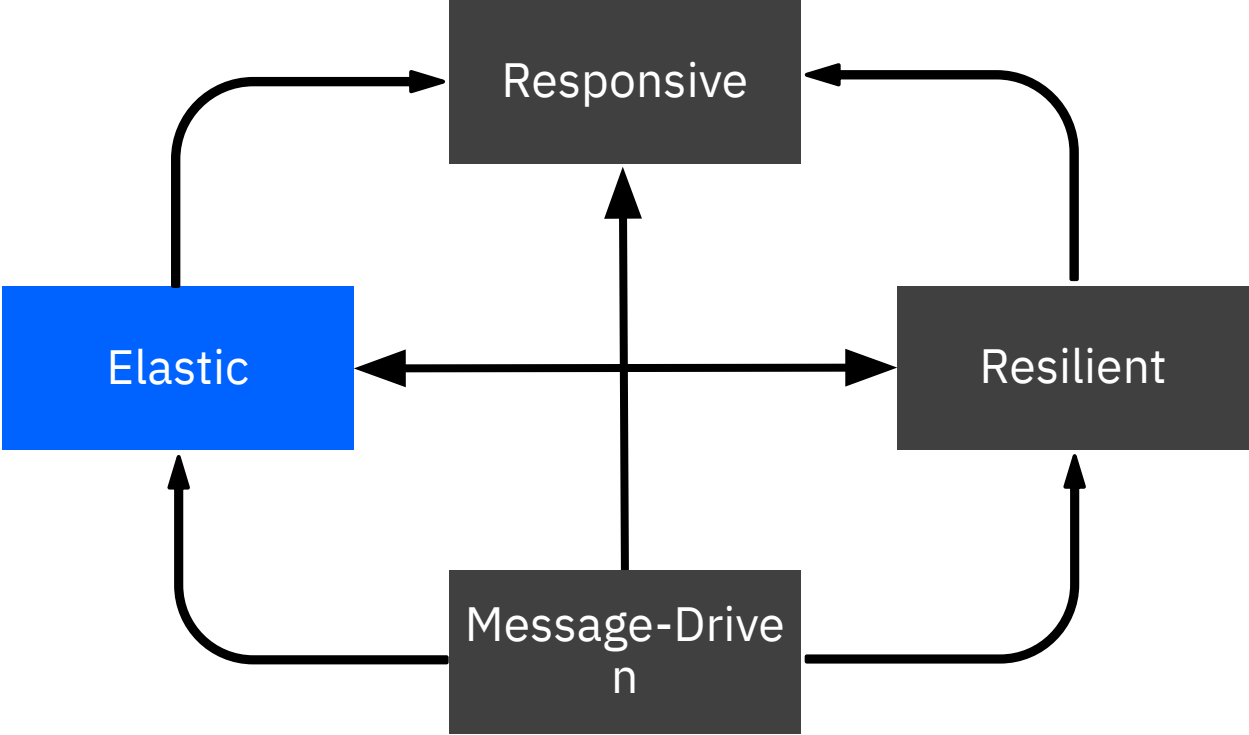


Barista

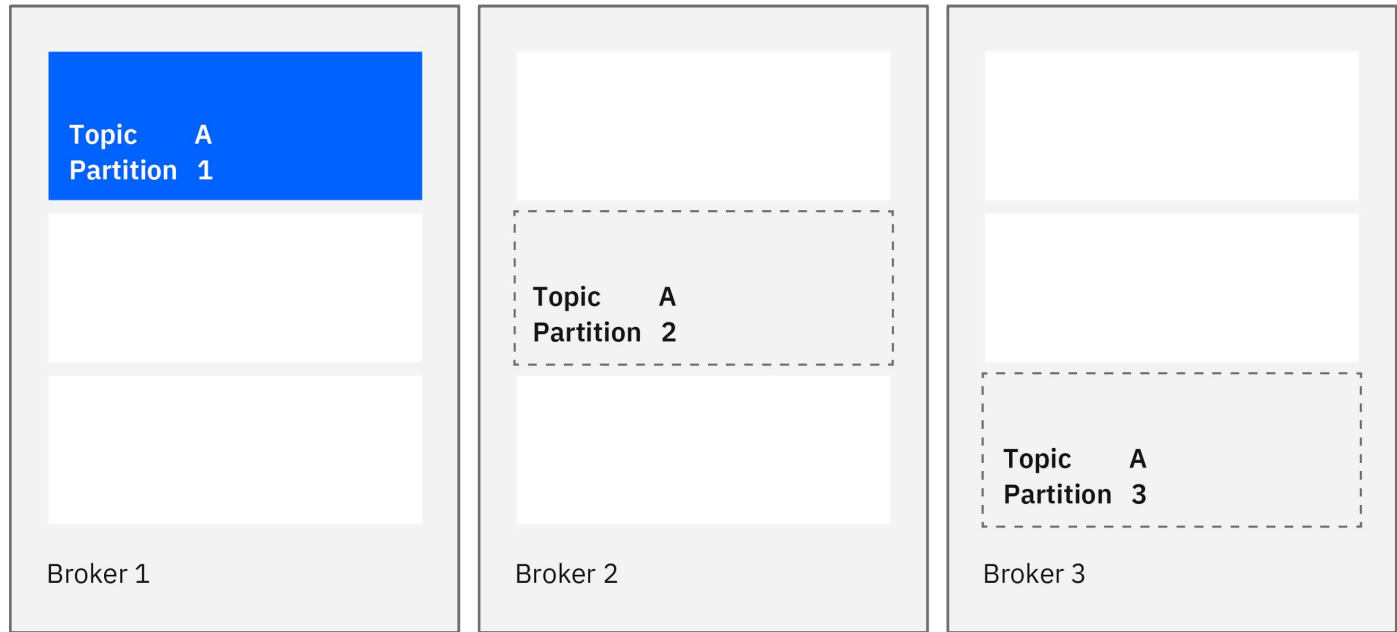


Coffee Cappuccino Latte

Scalability in Kafka

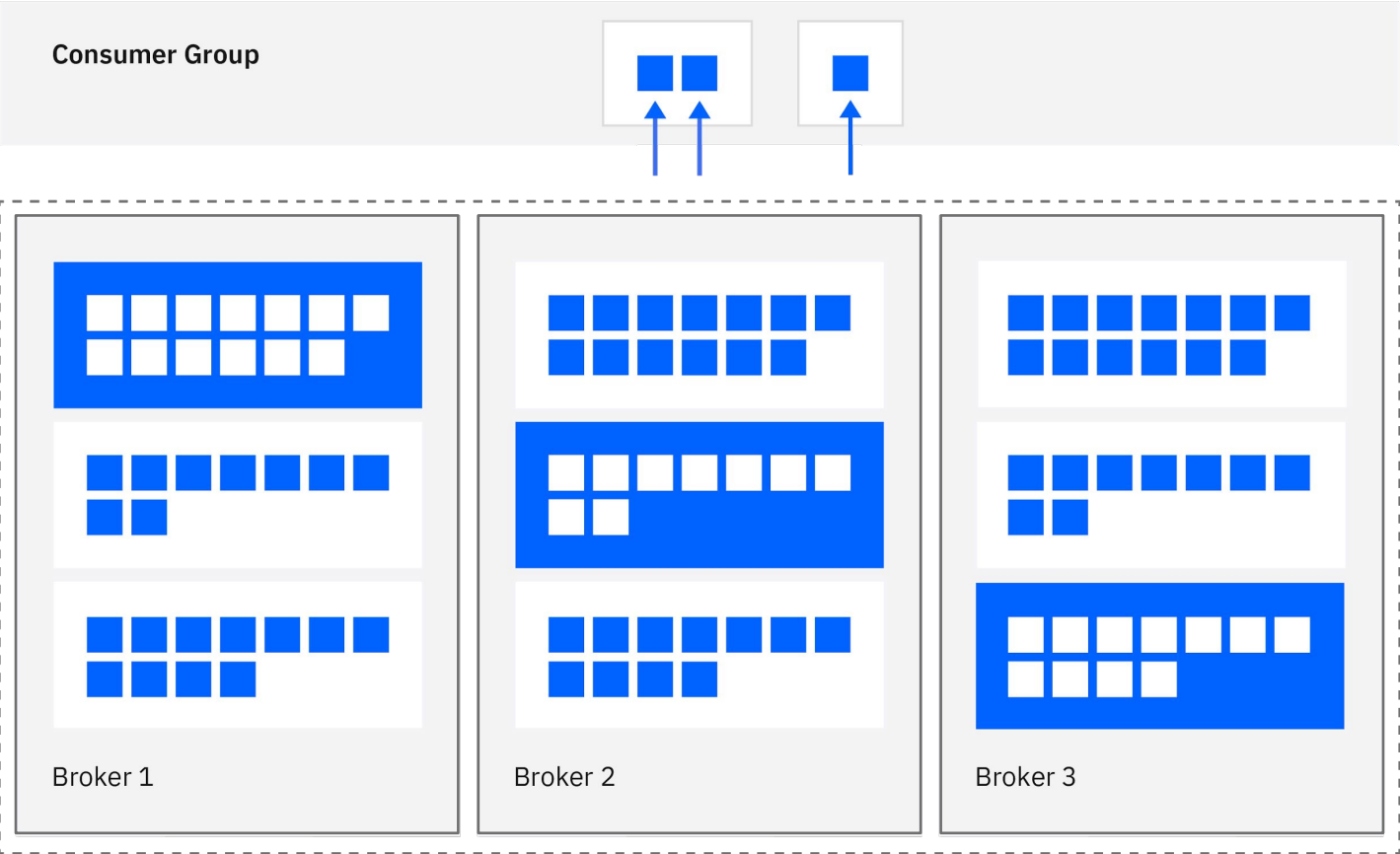


Scalability in Kafka



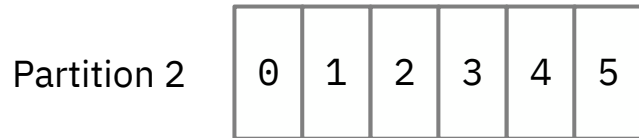
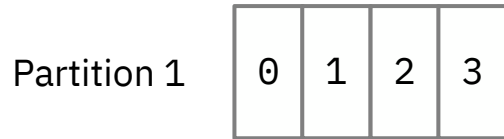
Kafka Cluster

Elasticity in Consumers



Consumer Groups

Topic



Consumer group A

Consumer

Consumer

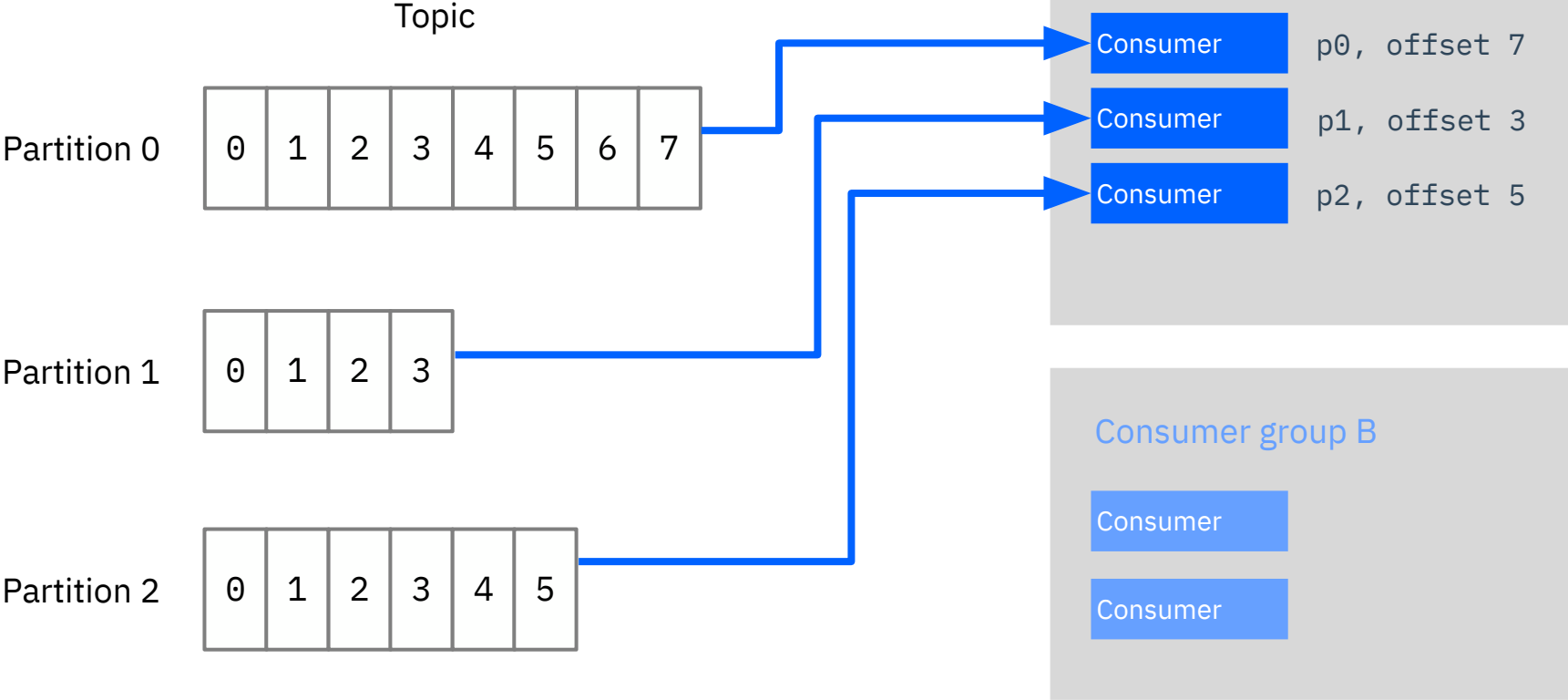
Consumer

Consumer group B

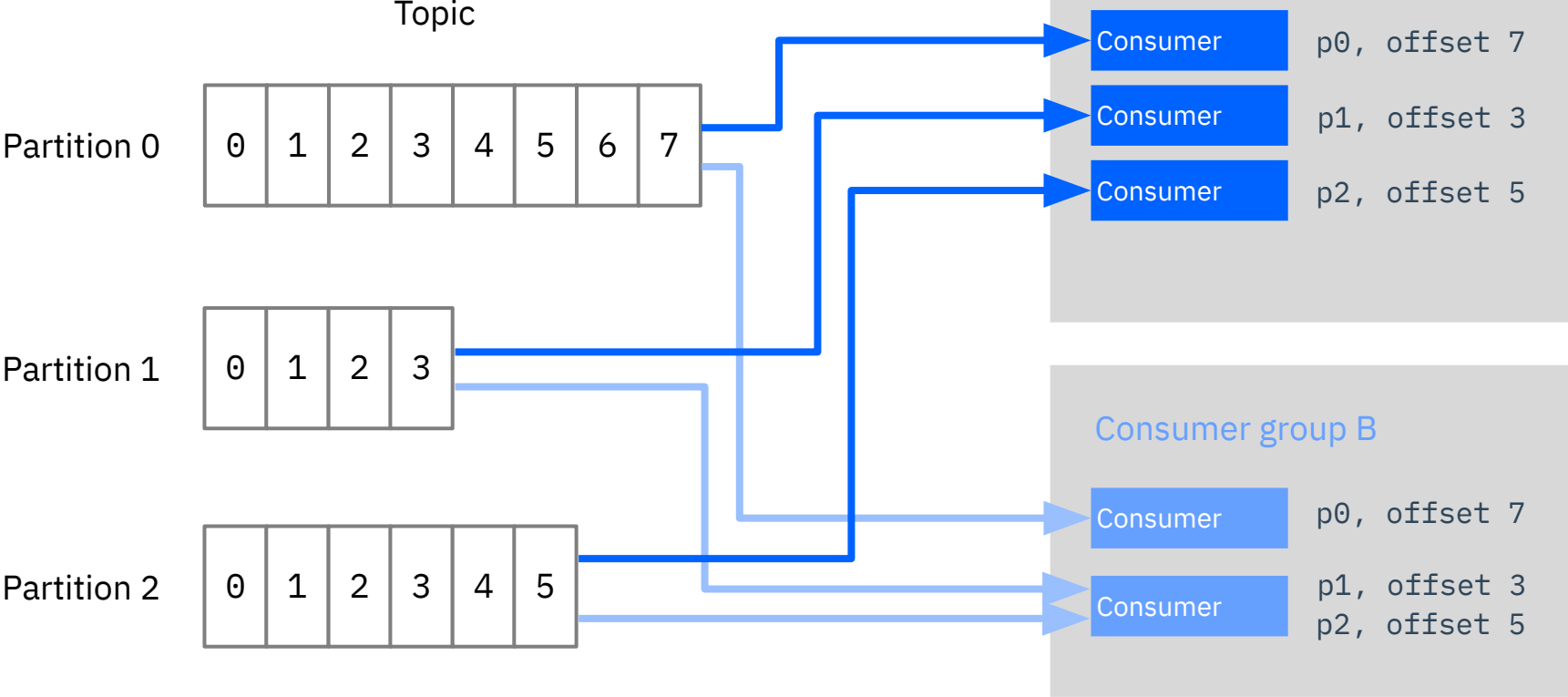
Consumer

Consumer

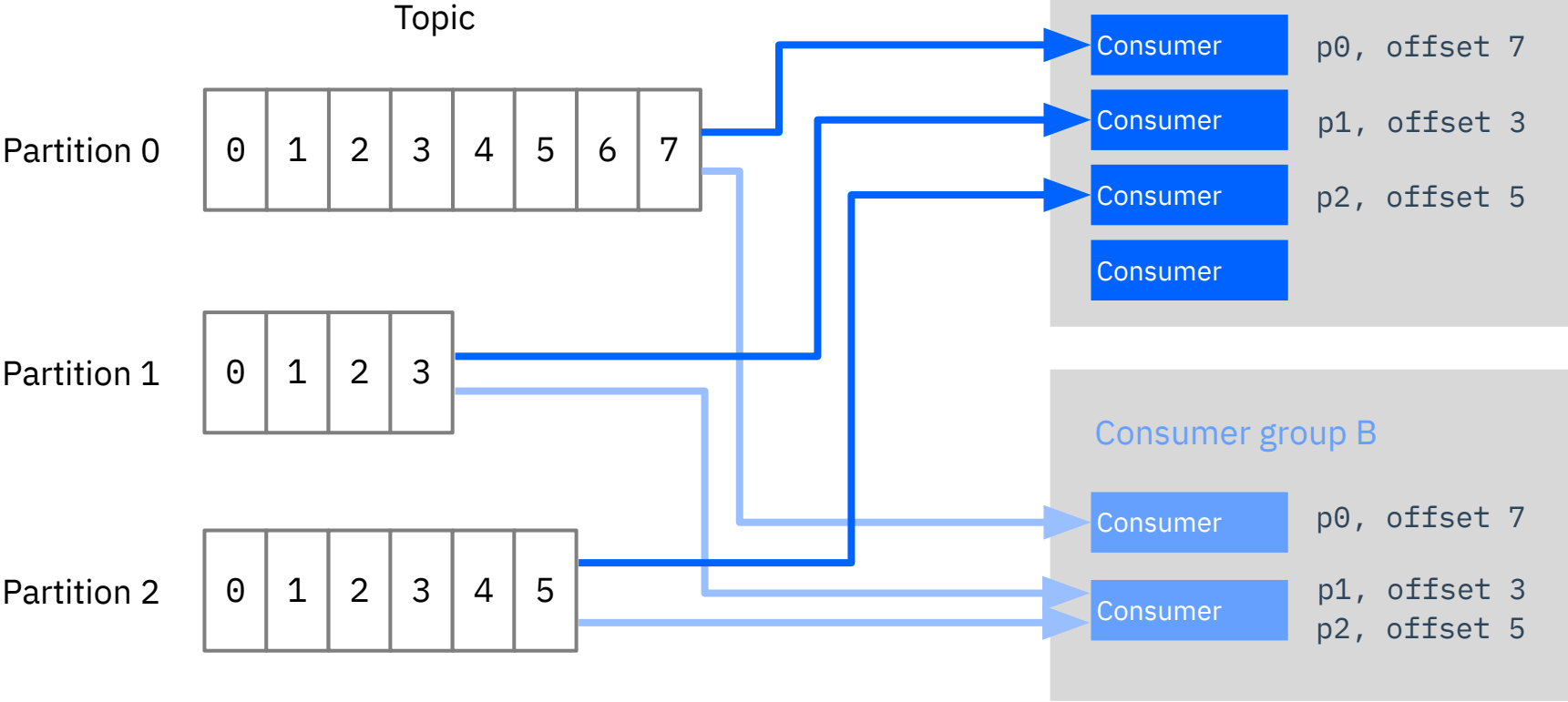
Consumer Groups



Consumer Groups



Consumer Groups



Writing reactive Kafka applications

org.apache.kafka.clients.producer

Class KafkaProducer<K,V>

java.lang.Object

org.apache.kafka.clients.producer.KafkaProducer<K,V>

All Implemented Interfaces:

java.io.Closeable, java.lang.AutoCloseable, Producer<K,V>

```
public class KafkaProducer<K,V>
extends java.lang.Object
implements Producer<K,V>
```

A Kafka client that publishes records to the Kafka cluster.

The producer is *thread safe* and sharing a single producer instance.

Here is a simple example of using the producer to send records to a Kafka cluster:

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("retries", 0);
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

```
Producer<String, String> producer = new KafkaProducer<>(props);
```

org.apache.kafka.clients.consumer

Class KafkaConsumer<K,V>

java.lang.Object

org.apache.kafka.clients.consumer.KafkaConsumer<K,V>

All Implemented Interfaces:

java.io.Closeable, java.lang.AutoCloseable, Consumer<K,V>

```
public class KafkaConsumer<K,V>
extends java.lang.Object
implements Consumer<K,V>
```

A client that consumes records from a Kafka cluster.

This client transparently handles the failure of Kafka brokers, and transparently adapts as topic partitions it fetches migrate with broker to allow groups of consumers to load balance consumption using consumer groups.

The consumer maintains TCP connections to the necessary brokers to fetch data. Failure to close the consumer after use will leak safe. See [Multi-threaded Processing](#) for more details.

Cross-Version Compatibility

This client can communicate with brokers that are version 0.10.0 or newer. Older or newer brokers may not support certain features like `offsetsForTimes`, because this feature was added in version 0.10.1. You will receive an `UnsupportedVersionException` when running broker version.

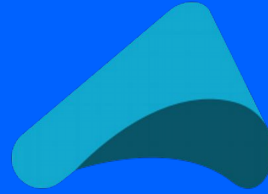
Offsets and Consumer Position

Reactive Frameworks for Kafka

Alpakka Kafka Connector

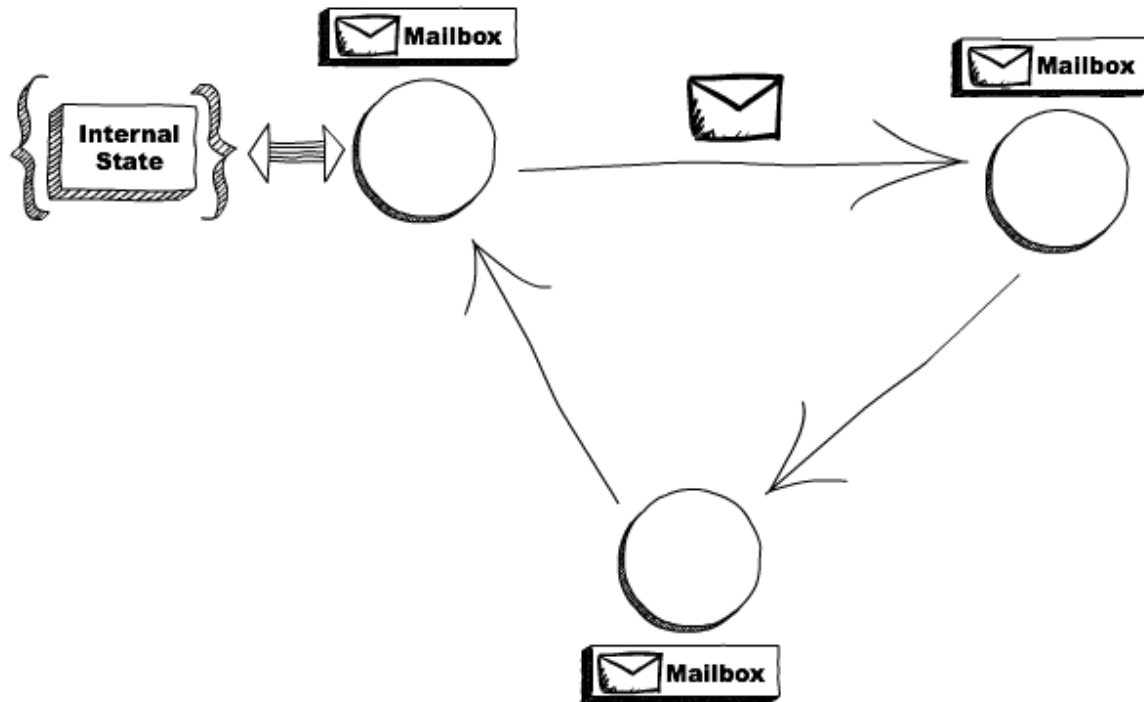
MicroProfile Reactive Messaging

Vert.x Kafka Client



VERT.X

Alpakka Kafka Connector API



Eclipse MicroProfile

An open-source community specification for
Enterprise Java microservices

A community of:
individuals
organizations
vendors

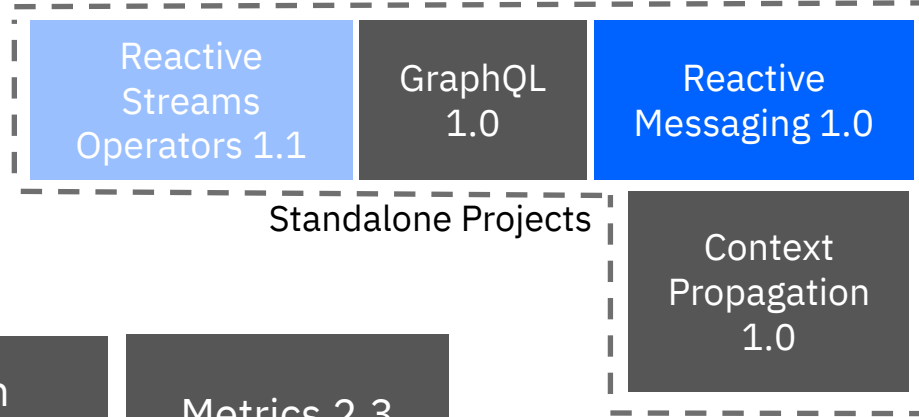


MicroProfile 3.3 Stack



Reactive Streams

Not a MicroProfile spec



Open Tracing 1.3

Health Check 2.2

Metrics 2.3

Open API 1.1

Fault Tolerance 2.1

JWT Propagation 1.1

Config 1.4

JAX-RS 2.1

CDI 2.0

JSON-P 1.1

JSON-B 1.0

Rest Client 1.4

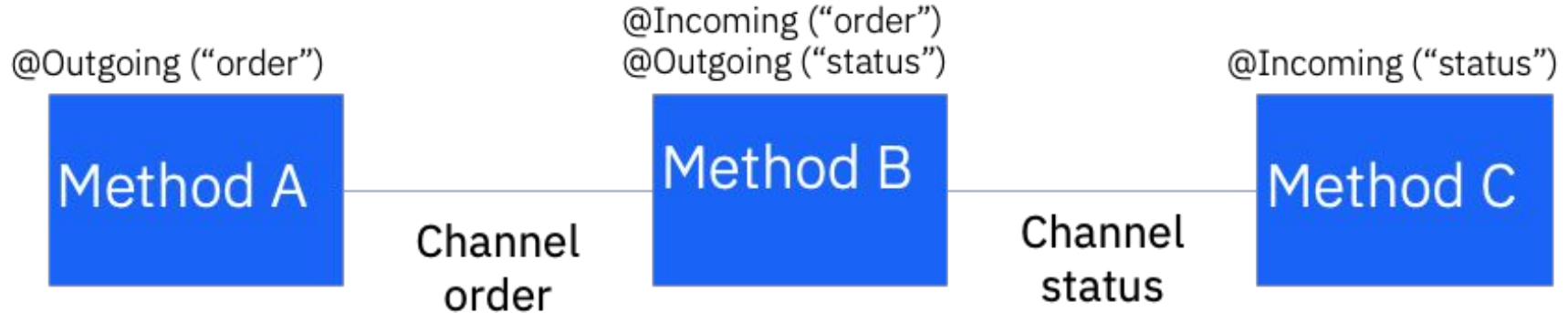
MicroProfile Reactive Messaging

Application's beans contain methods annotated with `@Incoming` and `@Outgoing` annotations

The annotated methods are connected by named *channels*

A *channel* is a name indicating which source or destination of messages is used. *Channels* are opaque Strings.

`@Incoming` and `@Outgoing` annotations are matched up by channel names



What is Eclipse Vert.x?

Polyglot Tool-kit

Based on Reactor pattern

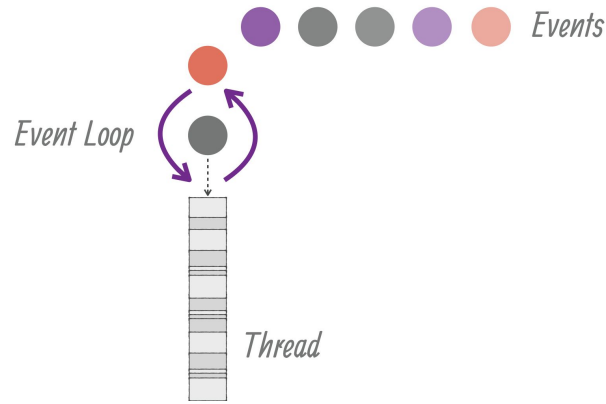
Runs on the JVM

Non-blocking

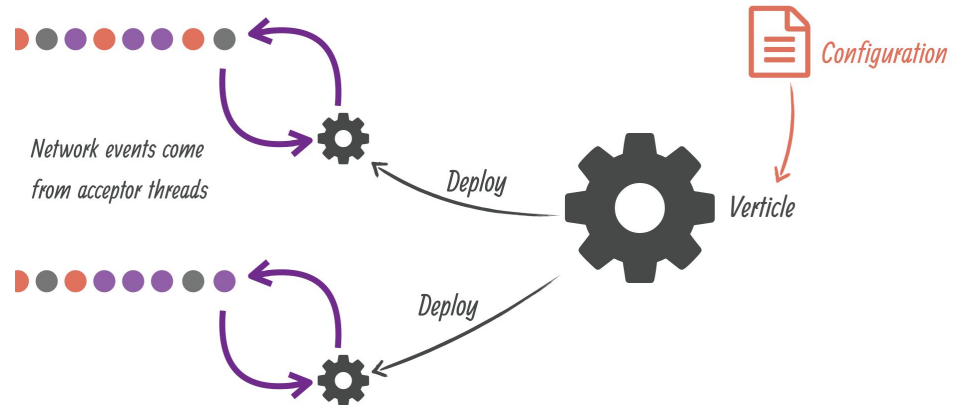
Event-driven

Includes distributed event-bus

Code is single-threaded

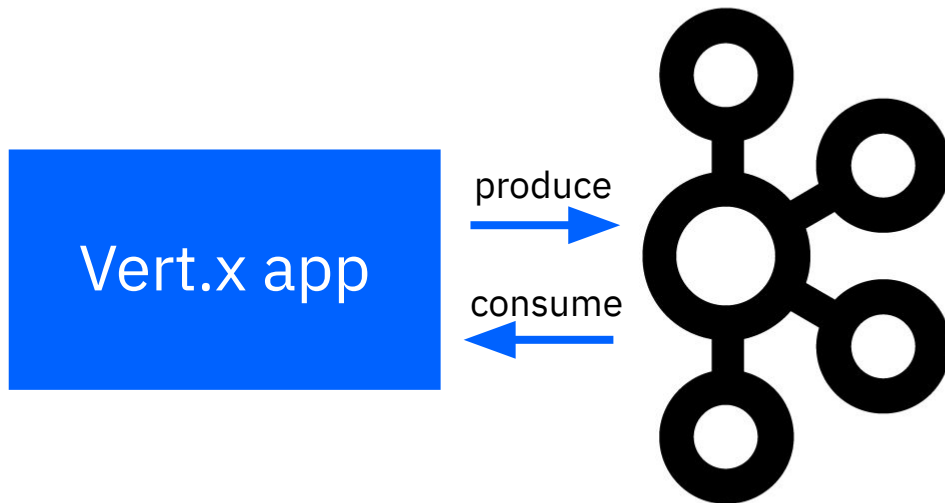


VERT.X



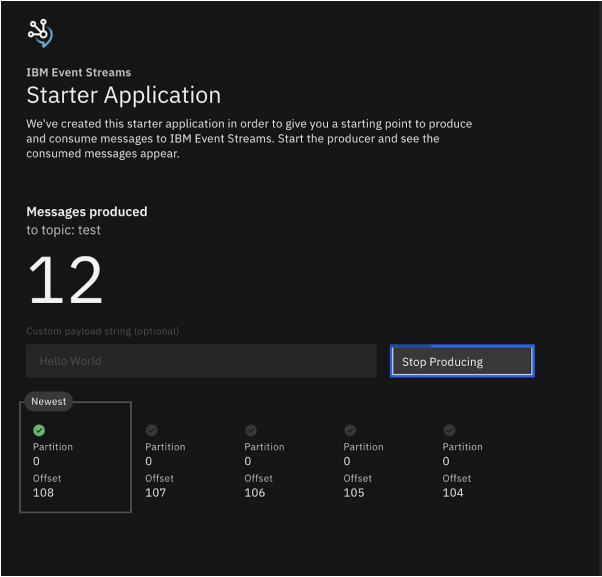
Vert.x Demo App

Demo app



<https://github.com/ibm-messaging/kafka-java-vertx-starter>

Demo app



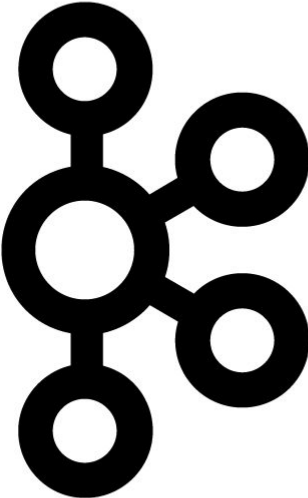
websocket



produce



consume



<https://github.com/ibm-messaging/kafka-java-vertx-starter>

Branch: master

Go to file

Code

matthew-chirgwin committed 2ac005c 4 days ago ... 32 commits 7 branches 1 tag

.github	fix: release workflow trigger	4 days ago
docs	feat: Add UI	18 days ago
src	feat: Add UI	18 days ago
ui	build(deps): Bump carbon-components from 10.14.0 to 10.15.0...	6 days ago
.editorconfig	feat: Simplify idioms	4 months ago
.gitignore	feat: Add UI	18 days ago
CODE_OF_CONDUCT.md	fix: Update contributing guidelines	4 months ago
CONTRIBUTING.md	feat: Add UI	18 days ago
LICENSE	feat: Version 0.0.1 of the app	5 months ago

About

Starter Java app for testing connection to Apache Kafka with Vert.x

Readme

Apache-2.0 License

Releases 1

Release 1.0.0 Latest 18 days ago

Contributors 6



https://github.com/ibm-messaging/kafka-java-vertx-starter



IBM Event Streams

Starter Application

We've created this starter application in order to give you a starting point to produce and consume messages to IBM Event Streams. Start the producer and see the consumed messages appear.

Messages produced

to topic: demo

00

Sample message value

Start producing

No messages produced

Produce messages to Kafka to see them here.

Messages consumed

from topic: demo

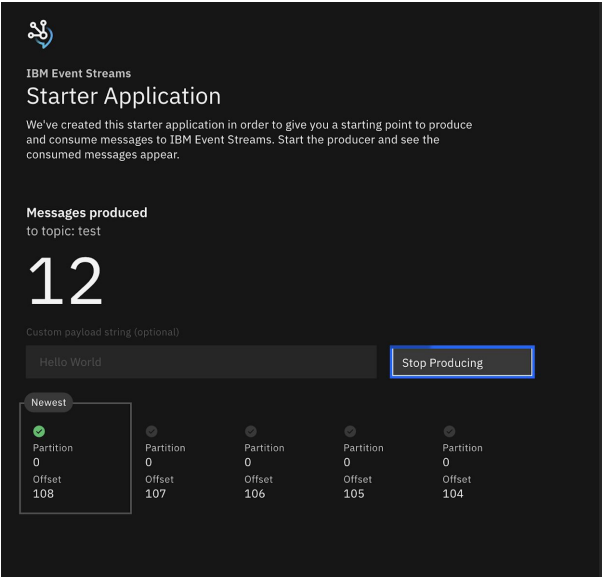
00

Start consuming

No messages consumed

Consume messages from Kafka to see them here.

Producing Records



IBM Event Streams
Starter Application

We've created this starter application in order to give you a starting point to produce and consume messages to IBM Event Streams. Start the producer and see the consumed messages appear.

Messages produced
to topic: test

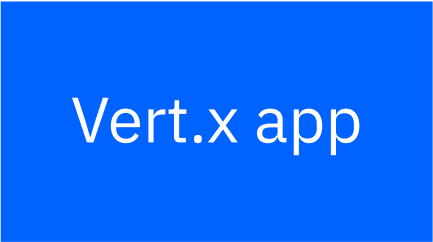
12

Custom payload string (optional)
Hello World

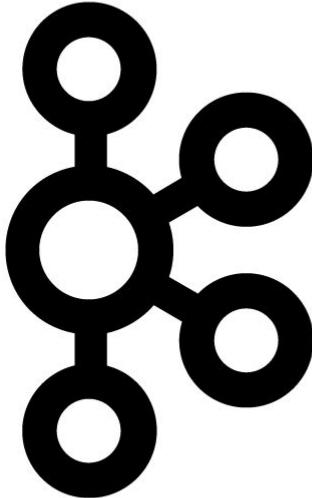
Newest				
<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Partition 0	Partition 0	Partition 0	Partition 0	Partition 0
Offset 108	Offset 107	Offset 106	Offset 105	Offset 104

Start
Stop

websocket



produce



Producing Records

```
67     String payload = customMessage;
68     KafkaProducerRecord<String, String> record = KafkaProducerRecord.create(topic, payload);
        KafkaProducerRecord<String, String> record = KafkaProducerRecord.create(topic, payload);
        payload {}", topic, payload);
        public abstract Future<RecordMetadata> send(KafkaProducerRecord<String, String> record)
71     kafkaProducer
72     .send(record)
73     .onSuccess(metadata -> {
74         JsonObject kafkaMetaData = new JsonObject()
75             .put("topic", metadata.getTopic())
76             .put("partition", metadata.getPartition())
77             .put("offset", metadata.getOffset())
78             .put("timestamp", metadata.getTimestamp())
79             .put("value", payload);
80         vertx.eventBus().send(Main.PERIODIC_PRODUCER_BROADCAST, kafkaMetaData);
81     })
82     .onFailure(err -> {
83         logger.error("Error sending {}", payload, err);
84         vertx.eventBus().send(Main.PERIODIC_PRODUCER_BROADCAST, new JsonObject().put("status", "ERROR"));
85     });
86 }
87 }
88
```

Consuming and Processing Records

Apache Kafka Java Client:

```
while (consuming) {
    ConsumerRecords<String, String> records = kafkaConsumer.poll(Duration.ofMillis("1000"));
    for (ConsumerRecord<String, String> record : records) {
        DemoConsumedMessage message = new DemoConsumedMessage(record.topic(), record.partition(),
            record.offset(), record.value(), record.timestamp());
        currentSession.getBasicRemote().sendObject(message); //send record along websocket
    }
}
```

Consuming and Processing Records

Apache Kafka Java Client:

```
while (consuming) {  
    ConsumerRecords<String, String> records = kafkaConsumer.poll(Duration.ofMillis("1000"));  
    for (ConsumerRecord<String, String> record : records) {  
        DemoConsumedMessage message = new DemoConsumedMessage(record.topic(), record.partition(),  
            record.offset(), record.value(), record.timestamp());  
        currentSession.getBasicRemote().sendObject(message); //send record along websocket  
    }  
}
```

Consuming and Processing Records

Apache Kafka Java Client:

```
while (consuming) {  
    ConsumerRecords<String, String> records = kafkaConsumer.poll(Duration.ofMillis("1000"));  
    for (ConsumerRecord<String, String> record : records) {  
        DemoConsumedMessage message = new DemoConsumedMessage(record.topic(), record.partition(),  
            record.offset(), record.value(), record.timestamp());  
        currentSession.getBasicRemote().sendObject(message); //send record along websocket  
    }  
}
```

Consuming and Processing Records

Vert.x Kafka Client

```
KafkaConsumer<String, JsonObject> kafkaConsumer = KafkaConsumer.create(vertx, kafkaConfig);
kafkaConsumer.handler(record -> {
    JsonObject payload = new JsonObject()
        .put("topic", record.topic())
        .put("partition", record.partition())
        .put("offset", record.offset())
        .put("timestamp", record.timestamp())
        .put("value", record.value());
    vertx.eventBus().send(webSocket.textHandlerID(), payload.encode());
});
```

Consuming and Processing Records

Vert.x Kafka Client

```
KafkaConsumer<String, JsonObject> kafkaConsumer = KafkaConsumer.create(vertx, kafkaConfig);
kafkaConsumer.handler(record -> {
    JsonObject payload = new JsonObject()
        .put("topic", record.topic())
        .put("partition", record.partition())
        .put("offset", record.offset())
        .put("timestamp", record.timestamp())
        .put("value", record.value());
    vertx.eventBus().send(webSocket.textHandlerID(), payload.encode());
});
```

Flow Control

Vert.x Kafka Client

```
websocket.handler(buffer -> {  
    String action = buffer.toJsonObject().getString("action", "none");  
    if ("start".equals(action)) {  
        kafkaConsumer.resume(partition);  
    } else if ("stop".equals(action)) {  
        kafkaConsumer.pause(partition);  
    }  
});
```


Experiences writing a reactive Kafka application

Our journey to reactive - transforming a microservices Kafka application

By [Grace Jansen](#), Kate Stanley

Published June 1, 2020

Apache Kafka is an extremely powerful tool for streaming large quantities of data and enabling asynchronous, non-blocking communication within systems. However, when building applications that use Kafka it can be hard to immediately test whether Kafka is working as it should. To help make this process easier, we created a Kafka starter app designed to enable you to test your Kafka deployment, with a fully functioning UI.

ibm.biz/ExperiencesWritingAReactiveKafkaApp

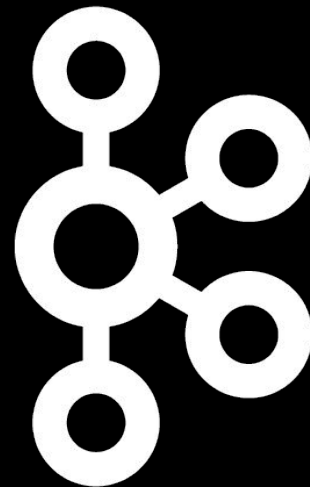
Summary

Non-reactive + Kafka != reactive

Consider Kafka configuration for the best reactive system

The open-source reactive community is on hand to help!

Reactive toolkits and frameworks can provide additional benefits




MicroProfile Reactive labs

QuickLab

Module 1


Creating reactive Java
microservices

 Try any time
→

QuickLab

Module 2


Testing reactive Java
microservices

 Try any time
→

QuickLab

Module 3


Consuming RESTful services
asynchronously with
template interfaces

 Try any time
→

QuickLab

Module 4

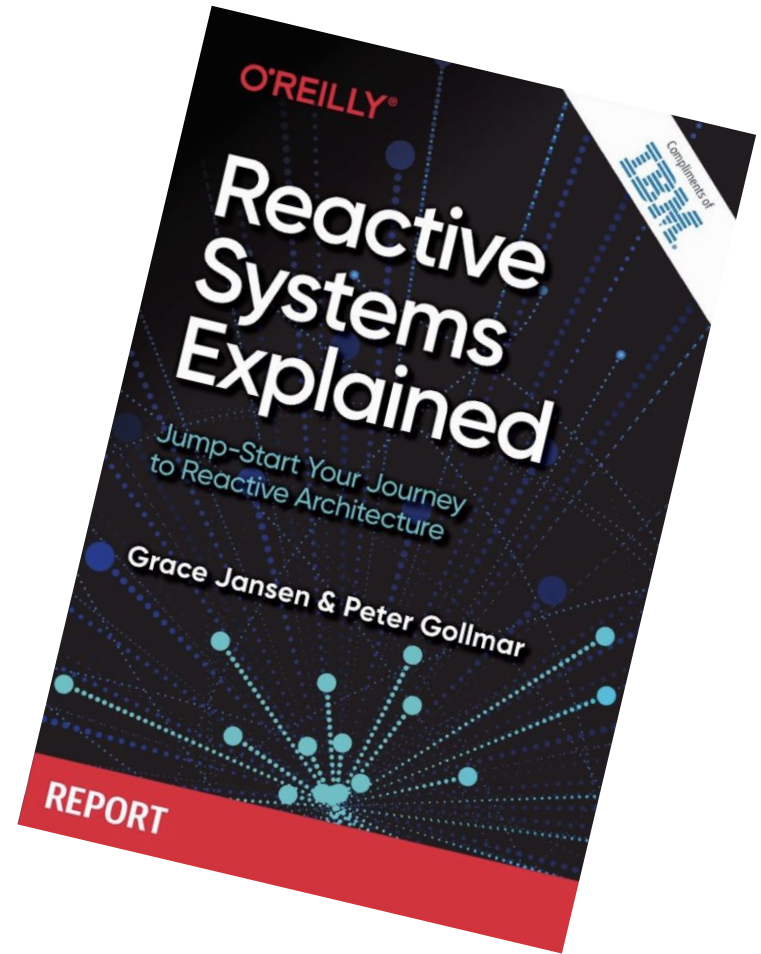
Integrating RESTful services
with a reactive system

 Try any time
→

ibm.biz/reactive-java-labs

Reactive Systems Explained

ibm.biz/ReactiveReport



Thank you

Grace Jansen | @gracejansen27



Reactive resources:

<https://ibm.biz/IntroToReactive>

<https://ibm.biz/GettingStartedWithReactive>

Getting started with Kafka:

<https://kafka.apache.org/quickstart>

<https://strimzi.io>

Reactive Kafka libraries

© 2021 IBM Corporation

<https://vertx.io/docs/vertx-kafka-client/java/>