



Event-Driven Microservices With Apache Kafka and Micronaut

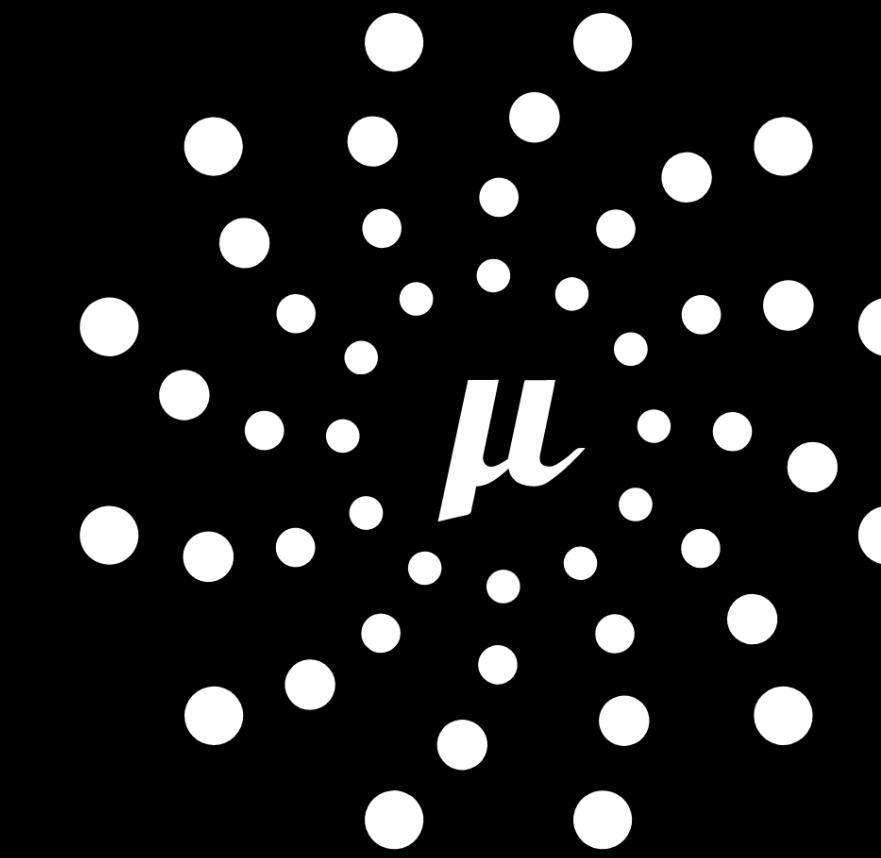
Zachary Klein, Principal Software Engineer, 2GM Team



About me

- ❖ Zachary Klein - Developer & Architect
- ❖ Principal Software Engineer at Object Computing, Inc
- ❖ 12+ years of software development experience
- ❖ OSS contributor
- ❖ Training instructor





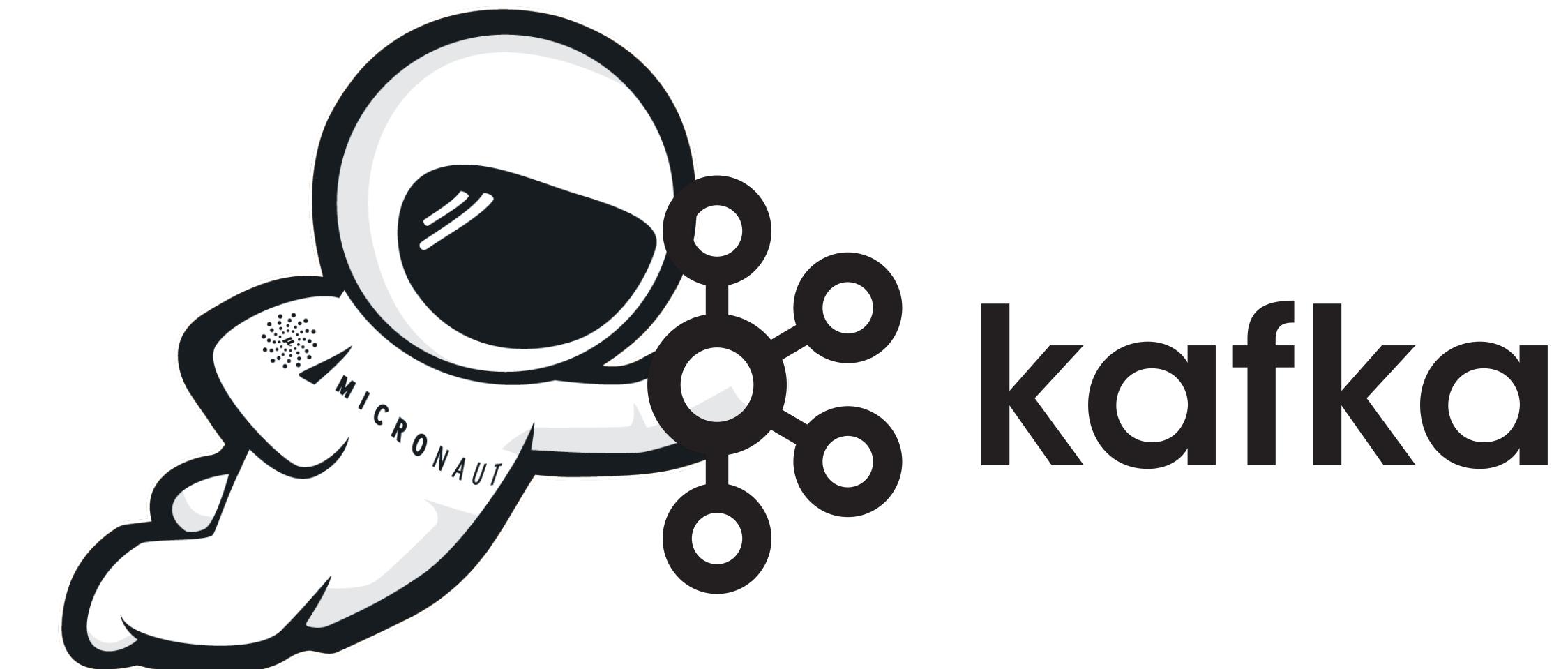
M I C R O N A U T[®]

A modern, JVM-based, full-stack framework for building modular,
easily testable microservice and serverless applications.

Agenda



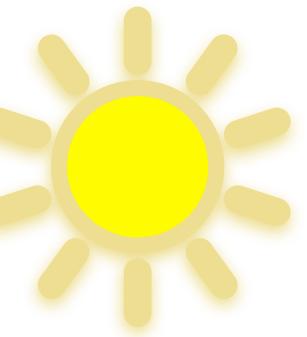
- Introduce Event-Driven Architecture
- Introduce Apache Kafka
- Introduce Micronaut
- Building Event-Driven Microservices using Micronaut Kafka



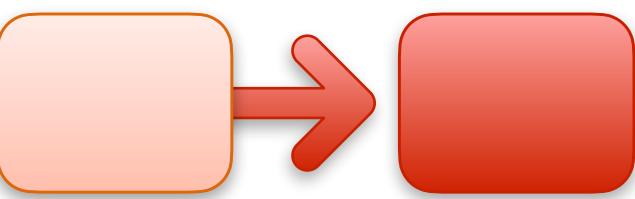
What is Event-Driven Architecture?



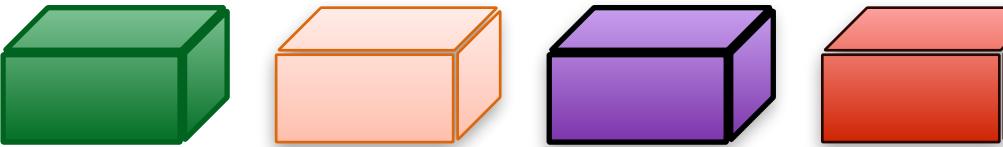
1. Event Notification



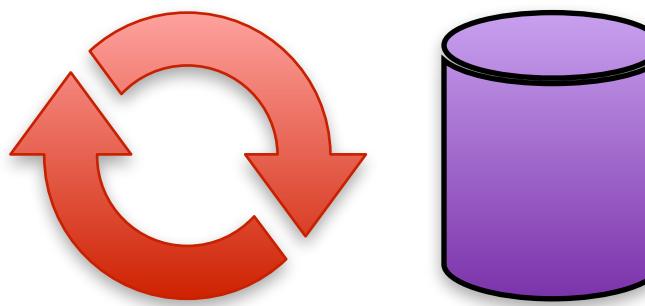
2. State Transfer



3. Event Sourcing



4. CQRS*



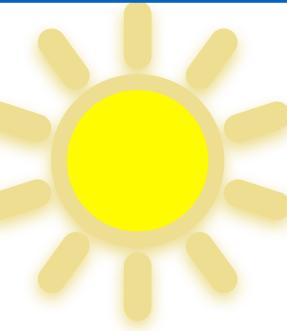
* Command & Query Responsibility Segregation

What is Event-Driven Architecture?

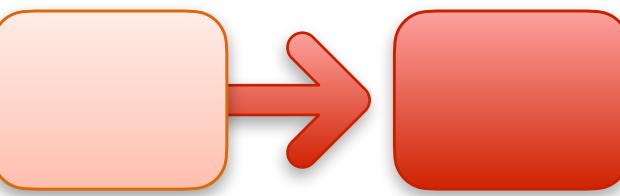


OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

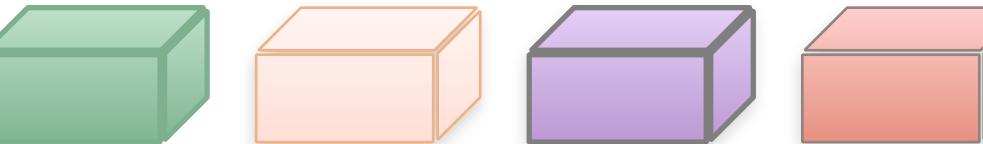
1. Event Notification



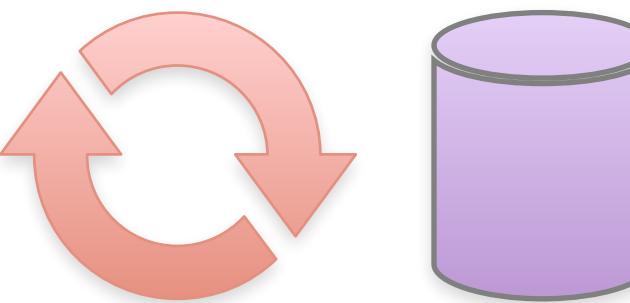
2. State Transfer



3. Event Sourcing



4. CQRS*



* Command & Query Responsibility Segregation

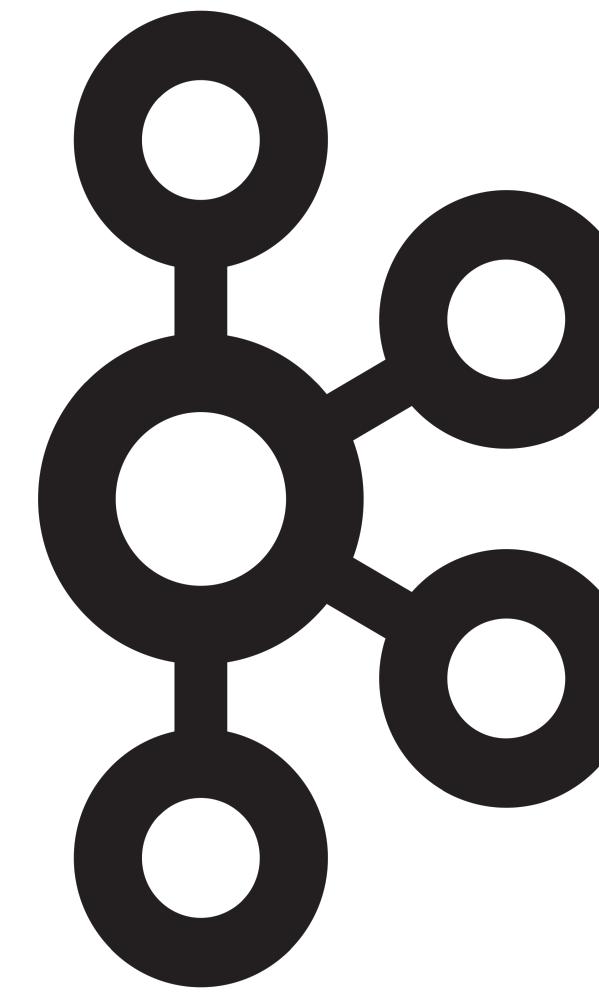
Event-Driven architecture: Key points



- Events are the Source of Truth
- Event-modeling: define the “shape” of your data
 - Schemas (e.g., with Apache Avro)
- Single-Writer Principle: Only one service writes to a given topic, N services read from it
- Event-Driven does not equal No Database



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



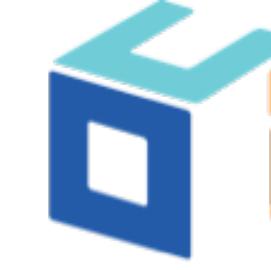
kafka

What is Apache Kafka?

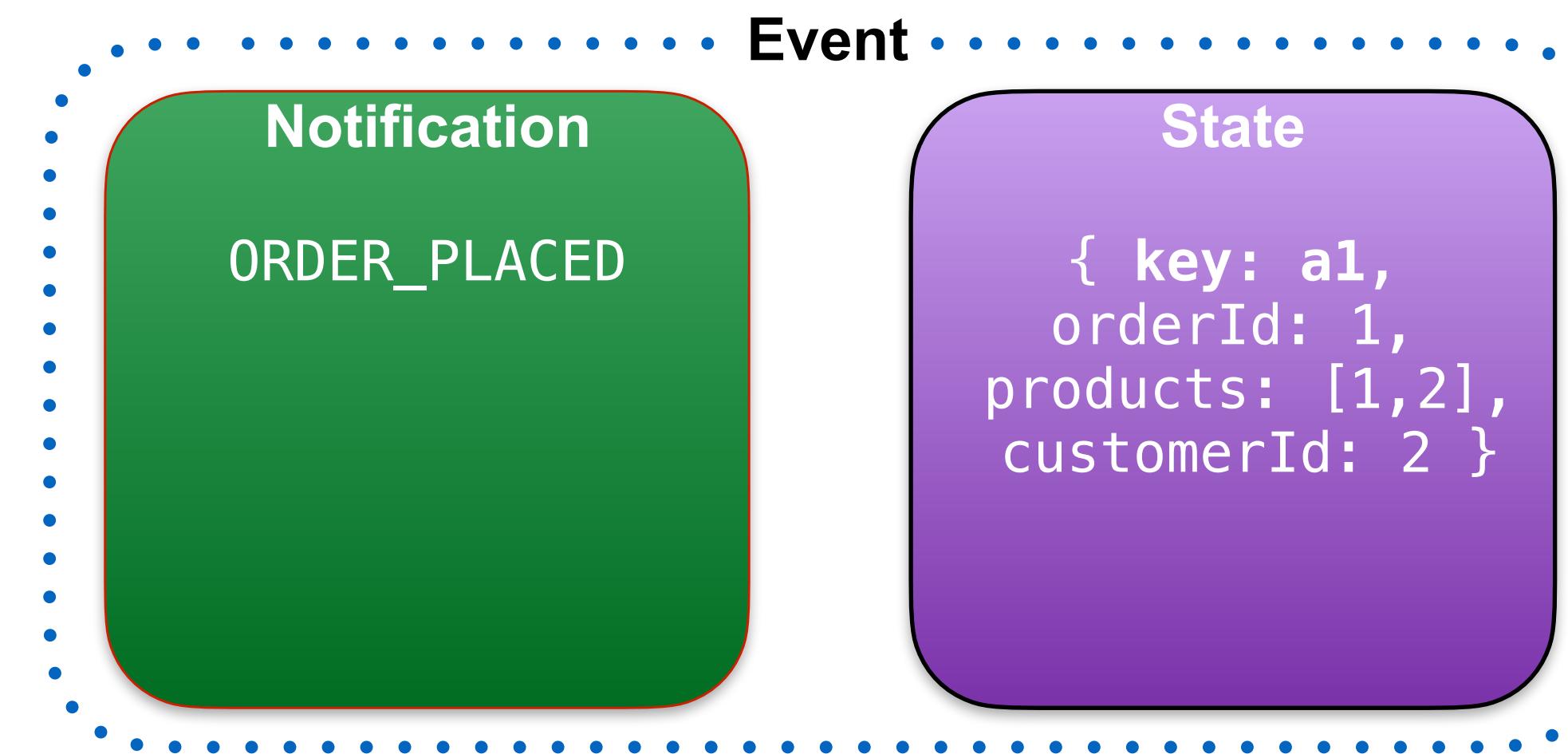


- Event-Streaming Platform
- Based on the concept of a **commit log**
 - Immutable
 - Persistent
 - Divided by “topics”
 - Partitioned
 - Events (log entries) are optionally keyed

What is an Event?



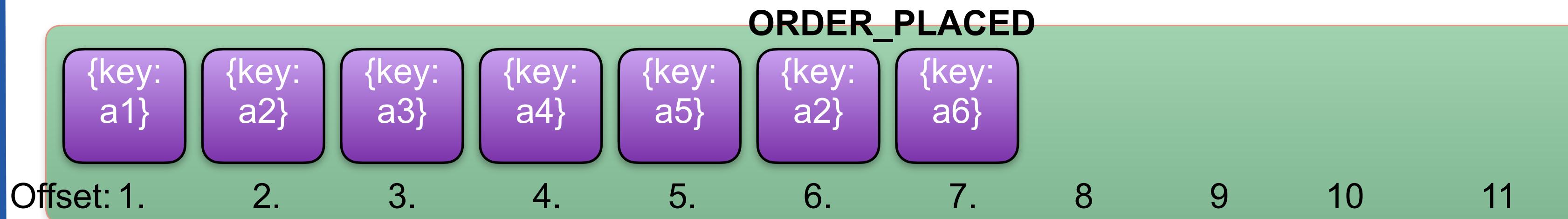
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



What is a Topic?



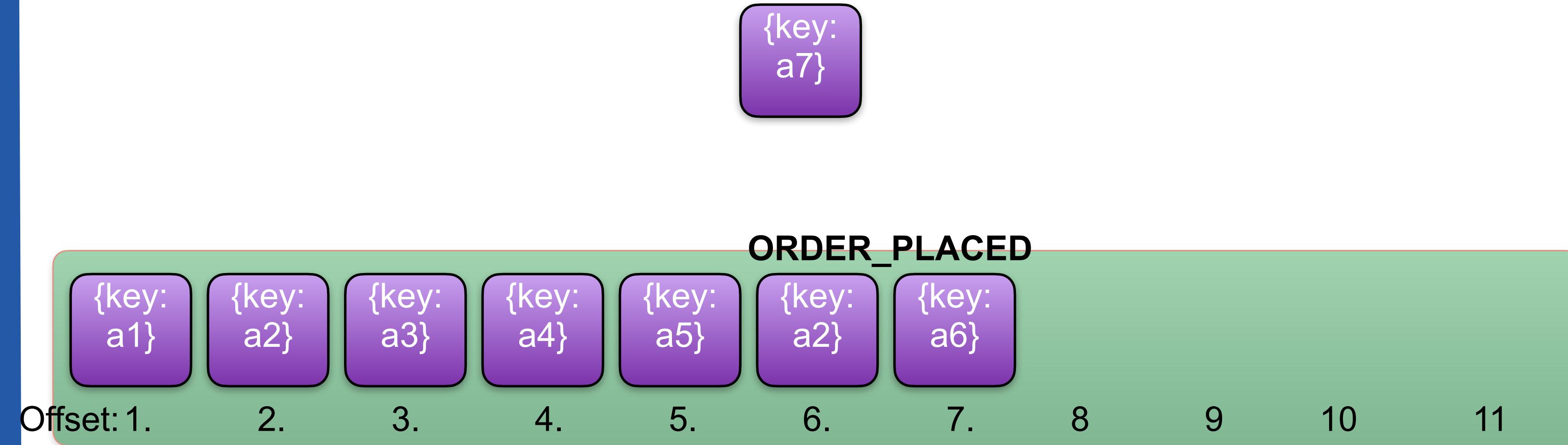
- Topics are persistent queues
- Events in the topic have an **offset**
- Events can be read from the topic in multiple modes - latest, all events from the beginning, batches, and even queried (e.g, using ksqlDB)



Adding an event to a topic



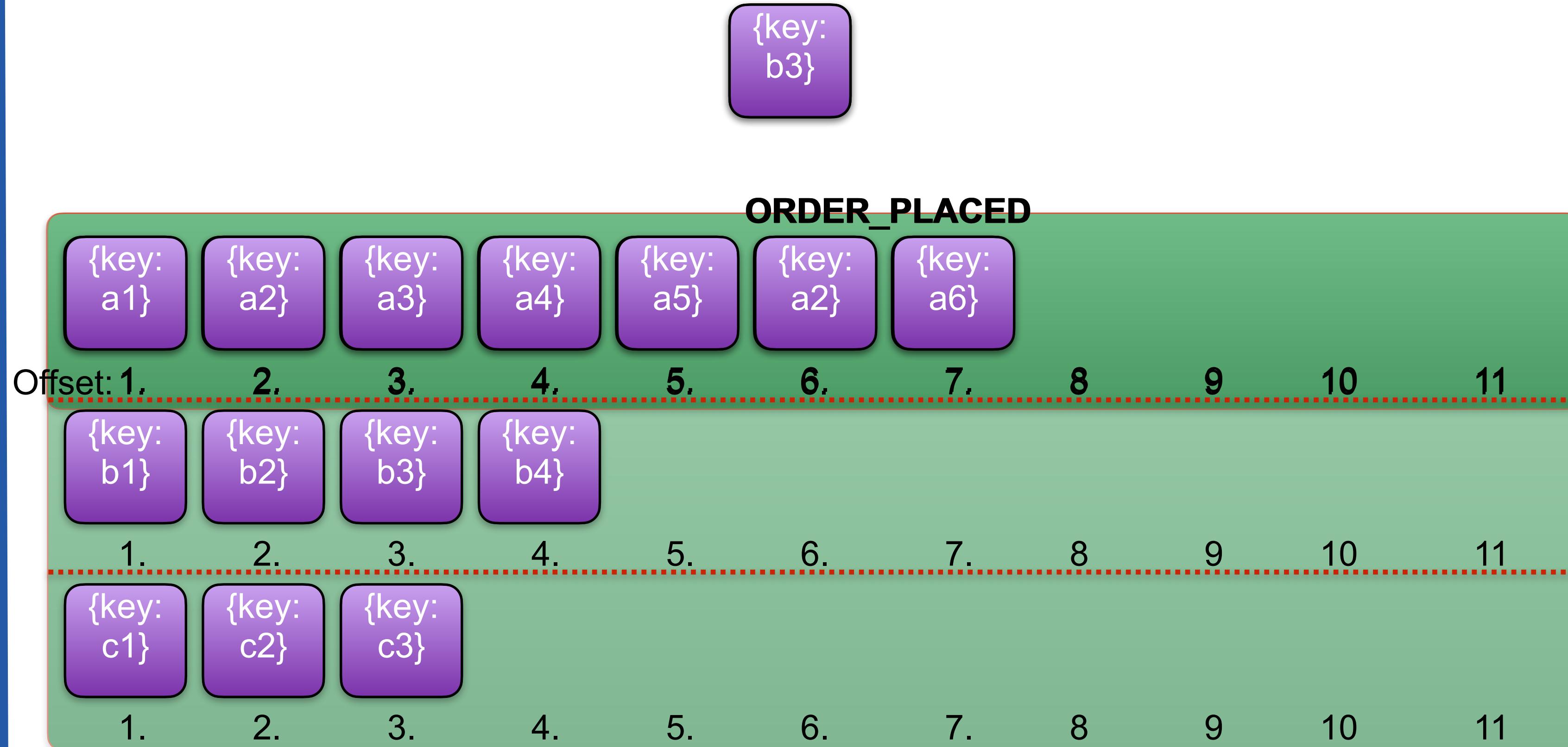
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



Topic partitions



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

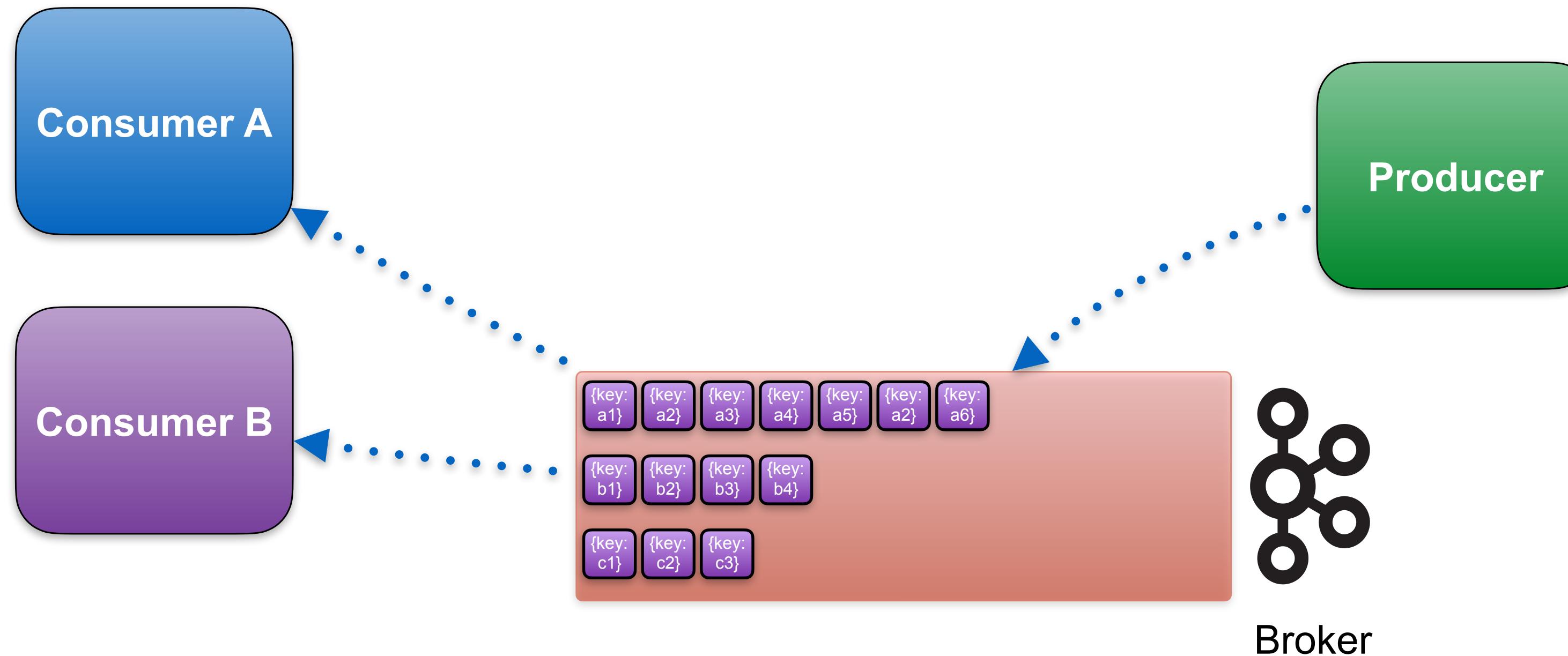


Producers and Consumers

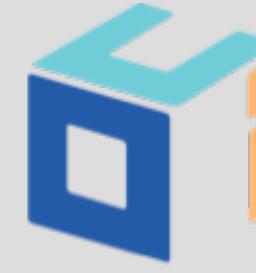


OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

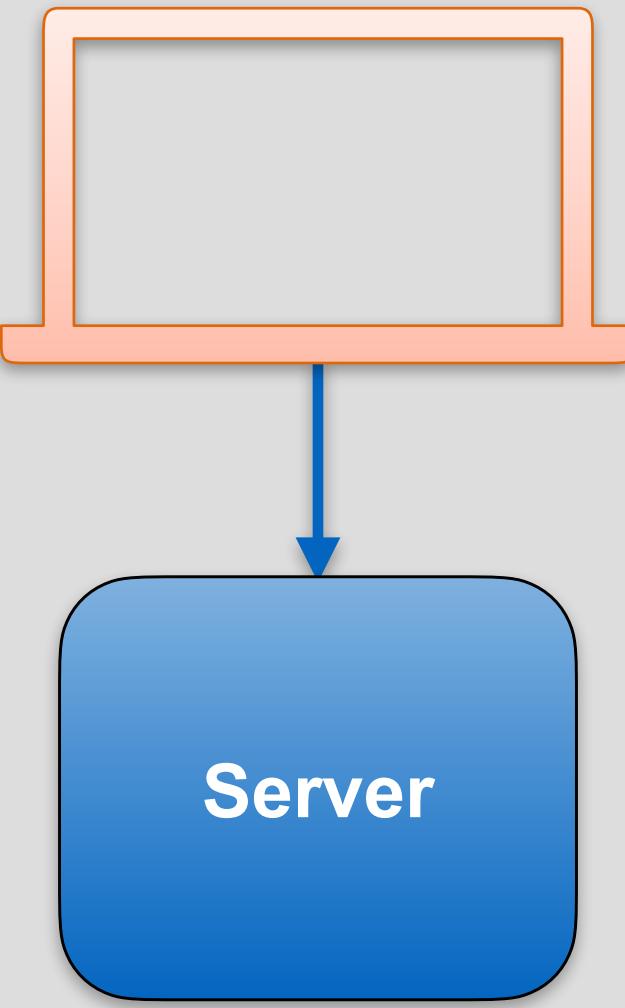
- Producers write events **to** topic/s
- Consumers read events **from** topics
- Events are read sequentially



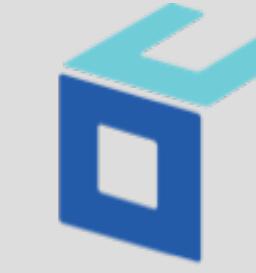
Inter-Service Communication



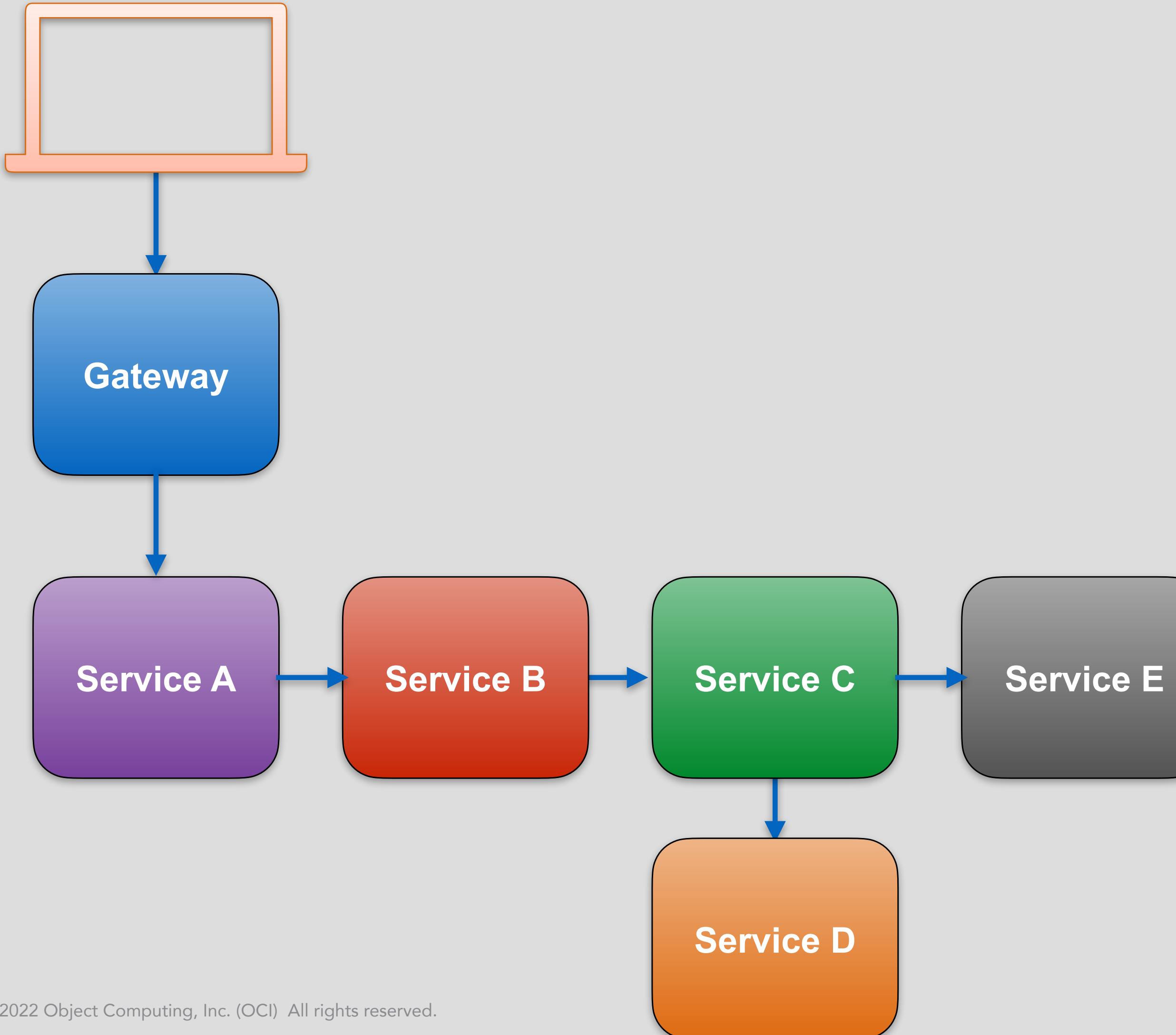
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



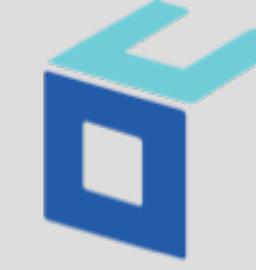
Inter-Service Communication



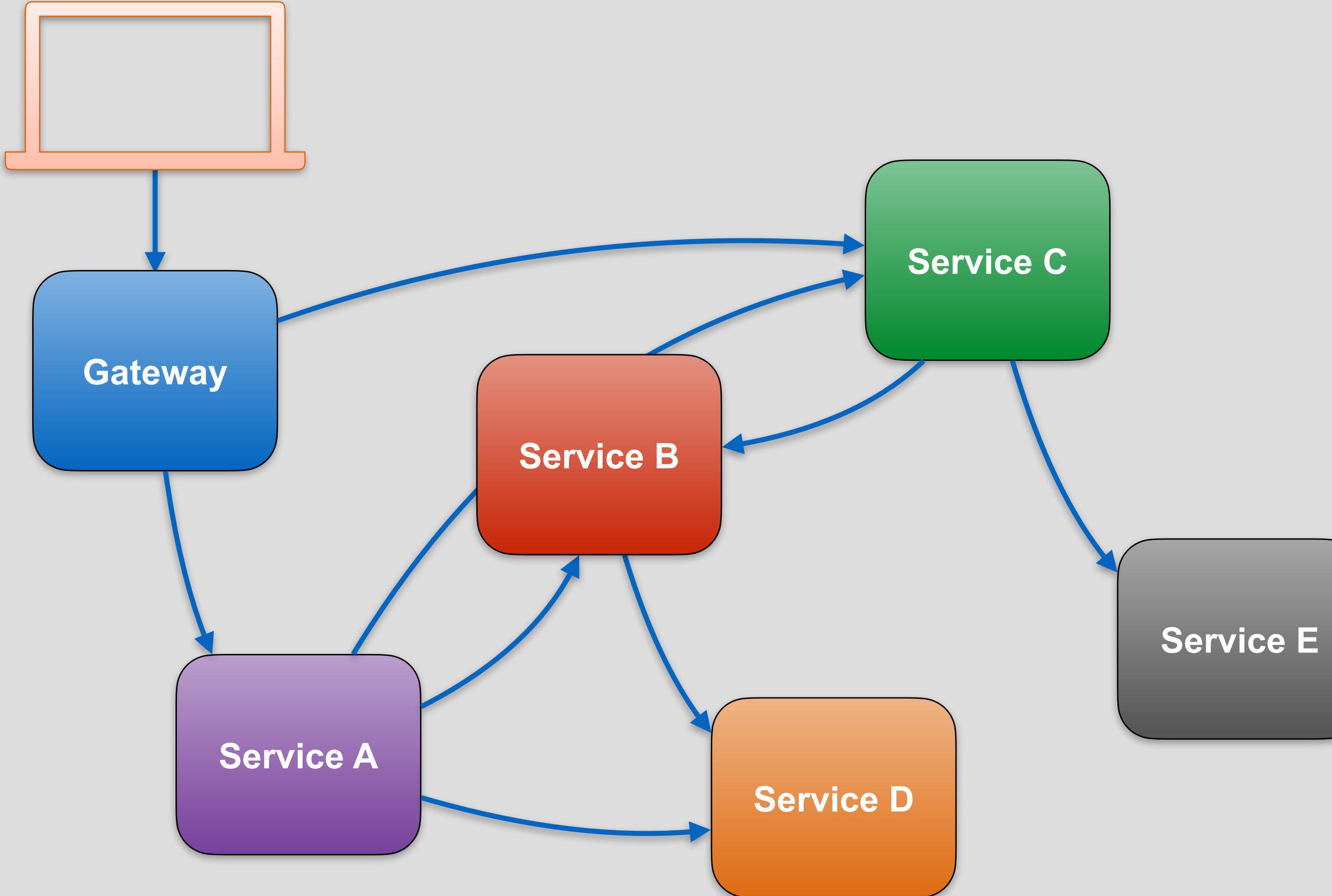
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



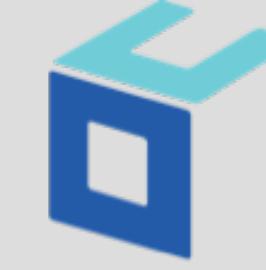
Inter-Service Communication



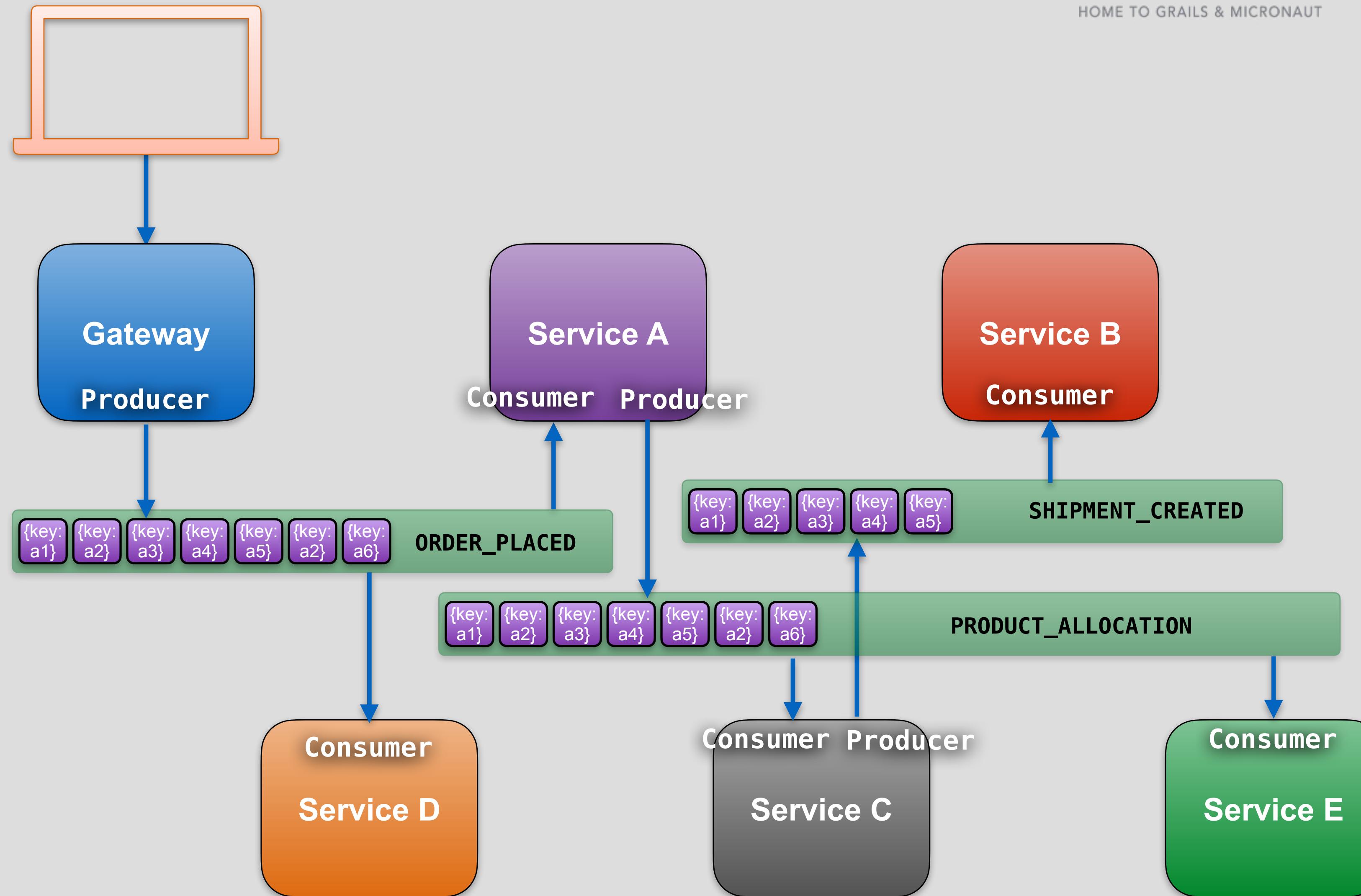
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



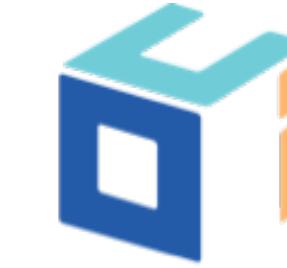
Event-Driven Microservices



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

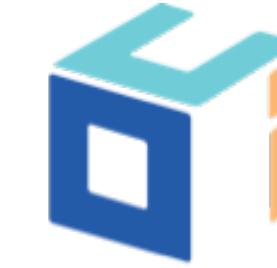


Event-Driven Microservices



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Event-driven architecture plays to microservices' strengths
 - Bounded context
 - Loose coupling
- Multiple services can consume events from same topic/s
- Events are persistent (replayable)
- Events can be materialized as tables for traditional querying (ksqlDB)
- Events replace REST calls for inter-service communication



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

O'REILLY®

Building Event-Driven Microservices

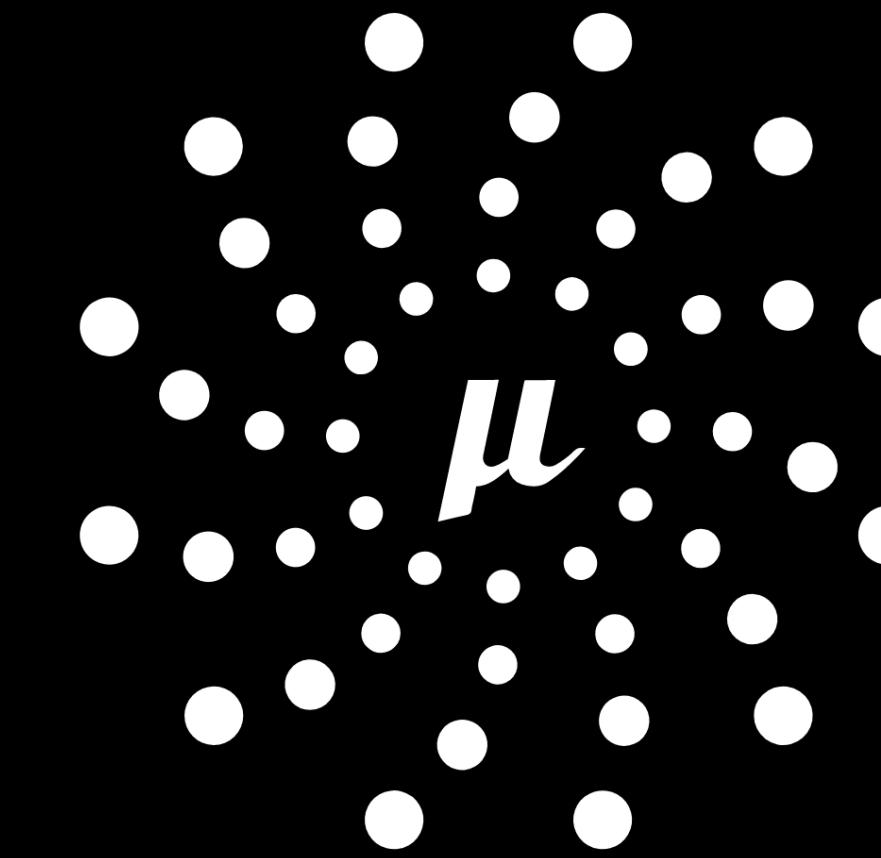
Leveraging Organizational Data at Scale



Adam Bellemare

- Communication structures
- Event-modeling
- Scalability
- Stream processing
- Event scheduling
- Integration with request/response

<https://www.oreilly.com/library/view/building-event-driven-microservices/9781492057888/>



M I C R O N A U T[®]

A modern, JVM-based, full-stack framework for building modular,
easily testable microservice and serverless applications.

What is Micronaut?



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- A modern framework for microservice & serverless applications
- Ahead Of Time (AOT) Compilation for DI, AOP, and configuration injection
- Reactive HTTP layer built on Netty
- Declarative HTTP Client
- Natively Cloud Native
- A suite of official and community-contributed integrations and libraries

The screenshot shows the Micronaut Launch interface on a dark-themed browser window. At the top, there's a navigation bar with icons for back, forward, search, and other browser functions. The URL bar shows micronaut.io. Below the header, the Micronaut logo (a stylized Greek letter μ) is displayed next to the text "MICRONAUT® LAUNCH". On the right side of the header, there are social media sharing icons for GitHub, Twitter, LinkedIn, and Email.

The main form area contains several dropdowns and input fields:

- Application Type:** A dropdown menu set to "Micronaut Application".
- Java Version:** A dropdown menu set to "11".
- Name:** An input field containing "demo".
- Base Package:** An input field containing "com.example".

Below these, there are three rows of configuration options:

- Micronaut Version:** Radio buttons for "3.7.1" (selected), "3.8.0-SNAPSHOT", and "2.5.13".
- Language:** Radio buttons for "Java" (selected), "Groovy", and "Kotlin".
- Build Tool:** Radio buttons for "Gradle" (selected), "Gradle Kotlin", and "Maven".
- Test Framework:** Radio buttons for "JUnit" (selected), "Spock", and "Kotest".

At the bottom of the configuration area are four buttons:

- FEATURES**: A button with a plus sign and the word "FEATURES".
- DIFF**: A button with a left-right arrow icon.
- PREVIEW**: A button with a magnifying glass icon.
- GENERATE PROJECT**: A button with a magnifying glass icon.

Below the configuration area, the text "Included Features (0)" is displayed. At the very bottom of the page, the Micronaut Foundation logo is shown.

<https://micronaut.io/launch>

```
1 package com.example;
2
3 import io.micronaut.http.HttpResponse;
4 import io.micronaut.http.MediaType;
5 import io.micronaut.http.annotation.Controller;
6 import io.micronaut.http.annotation.Get;
7 import io.micronaut.http.annotation.Produces;
8
9 @Controller
10 public class HelloController {
11
12     @Get("/{name}")
13     @Produces(MediaType.TEXT_PLAIN)
14     HttpResponse<String> greet(String name) {
15         return HttpResponse.ok("Hello, " + name);
16     }
17
18 }
19
```



NG

OpenJDK 14 on 2019 iMac Pro Xeon 8 Core. Winner in Red.

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	??? (3)
Request Per Second -Xmx18m	50k req/sec	46k req/sec	??? (3)
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

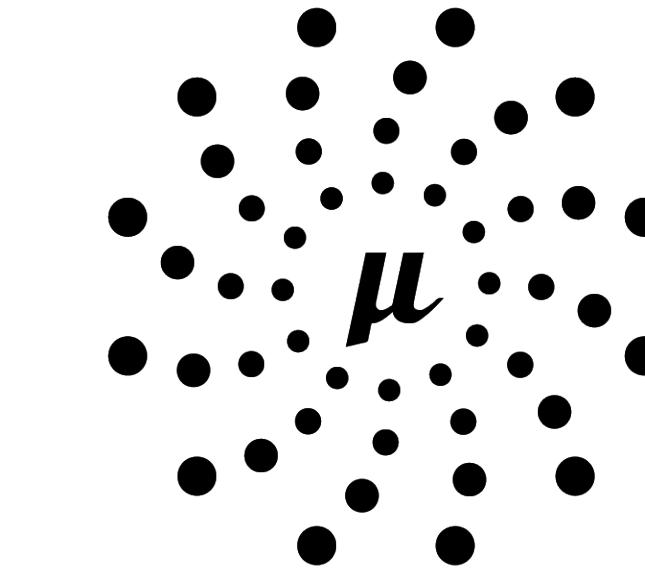
(1) Verifier Disabled
(2) Measured with: ab -k -c 20 -n 10000 <http://localhost:8080/hello/John>
(3) Spring WebFlux doesn't seem to support keep alive?
(4) Measured with: ps x -o rss,vsz,command | grep java

<https://micronaut.io/2020/04/07/micronaut-vs-quarkus-vs-spring-boot-performance-on-jdk-14/>

Micronaut Kafka



- <https://micronaut-projects.github.io/micronaut-kafka/>
- Provides declarative API for publishing/consuming events in Kafka
- Reflection-free serialization/deserialization of events
- Create messaging apps without a HTTP server



M I C R O N A U T ®



Getting started



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Create an app (from the CLI or Micronaut Launch) with the **kafka** feature
- Or add the dependency and minimum required configuration
- Requires a working Kafka installation

Getting started



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Add dependency:



Gradle

maven

Maven

```
implementation("io.micronaut.kafka:micronaut-kafka")
```

Minimum configuration:

Configuring Kafka

```
kafka:  
  bootstrap:  
    servers: localhost:9092
```

Advanced Configuration



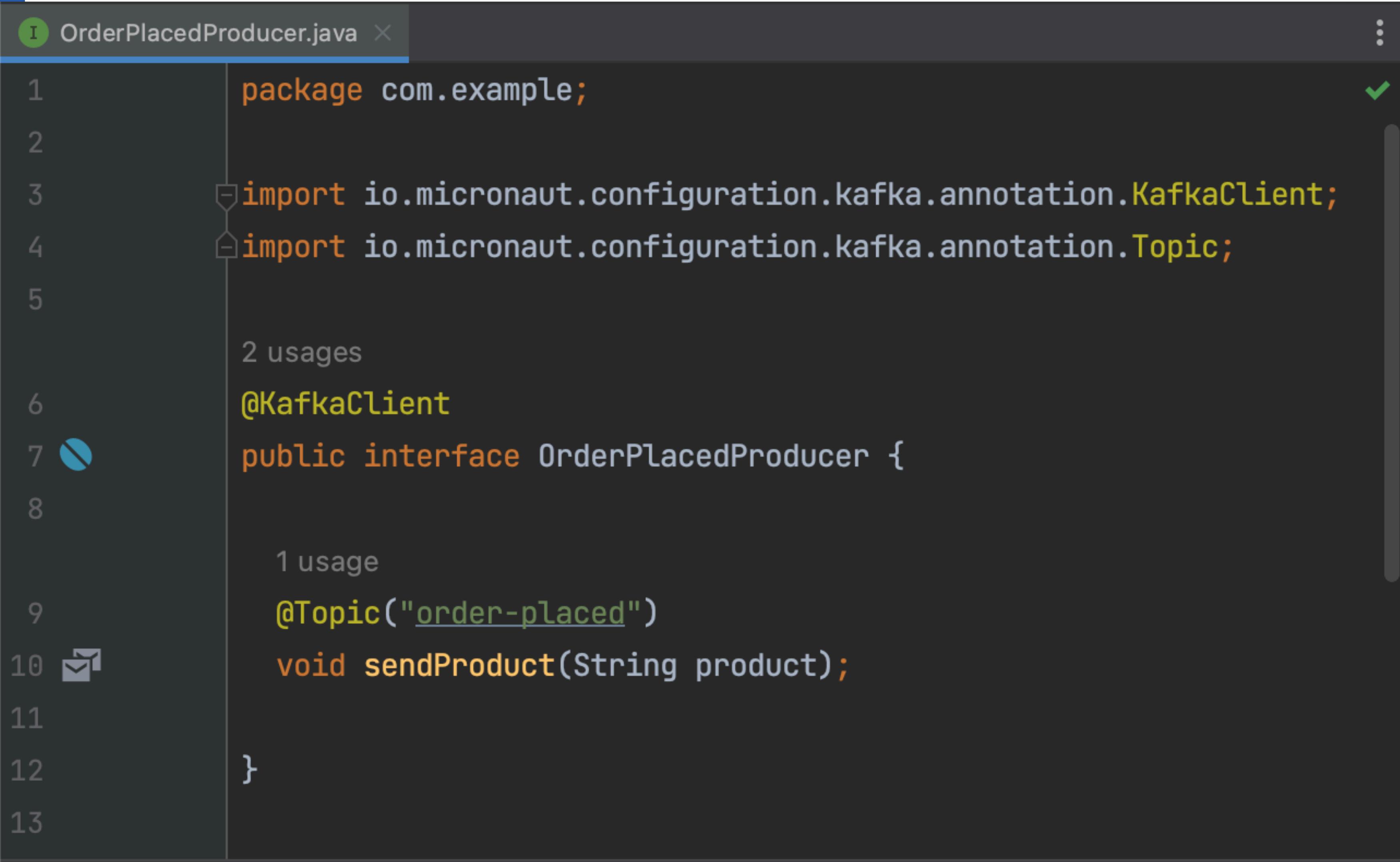
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
kafka:  
    bootstrap:  
        servers: localhost:9092  
    ssl:  
        keystore:  
            location: /path/to/client.keystore.p12  
            password: secret  
        truststore:  
            location: /path/to/client.truststore.jks  
            password: secret  
            type: PKCS12  
    security:  
        protocol: ssl  
    producers:  
        default:  
            retries: 5  
    consumers:  
        default:  
            session:  
                timeout:  
                    ms: 30000
```

Producer: @KafkaClient



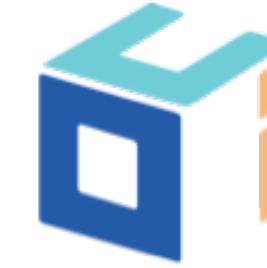
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



A screenshot of a code editor showing the file `OrderPlacedProducer.java`. The code defines a public interface `OrderPlacedProducer` with a single method `sendProduct` annotated with `@Topic("order-placed")`. The code editor interface includes a toolbar at the top with icons for file operations, a status bar at the bottom, and a vertical scrollbar on the right.

```
1 package com.example;
2
3 import io.micronaut.configuration.kafka.annotation.KafkaClient;
4 import io.micronaut.configuration.kafka.annotation.Topic;
5
6 @KafkaClient
7 public interface OrderPlacedProducer {
8
9     @Topic("order-placed")
10    void sendProduct(String product);
11
12 }
13
```

Specify Topic with @Topic



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
@Topic("my-products")
void sendProduct(@KafkaKey String brand, String name);
```

Topics can also be specified as arrays, Regular Expressions, and configuration properties:

```
@Topic({"fun-products", "awesome-products"})
```

JAVA

```
@Topic(patterns="products-\w+")
```

```
@Topic("${custom.topic.name}")
```

Add Headers with `@MessageHeader`



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Method-level:

```
@Topic("my-products")
void sendProduct(
    @KafkaKey String brand,
    String name,
    @MessageHeader("My-Header") String myHeader);
```

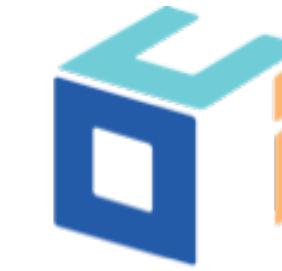
Interface-level (all methods):

JAVA

```
import io.micronaut.configuration.kafka.annotation.KafkaClient;
import io.micronaut.messaging.annotation.MessageHeader;

@KafkaClient(id="product-client")
@MessageHeader(name = "X-Token", value = "${my.application.token}")
public interface ProductClient {
    ...
}
```

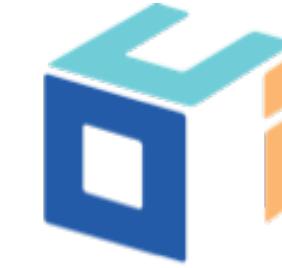
Producer: @KafkaClient



OBJECT COMPUTING

```
I OrderPlacedProducer.java × : ✓  
1 package com.example;  
2  
3 import io.micronaut.configuration.kafka.annotation.KafkaClient;  
4 import io.micronaut.configuration.kafka.annotation.Topic;  
5  
6 2 usages  
7 @KafkaClient  
8  
9 public interface OrderPlacedProducer {  
10 1 usage  
11     @Topic("order-placed")  
12     void sendProduct(String product);  
13 }
```

Inject and use Producer



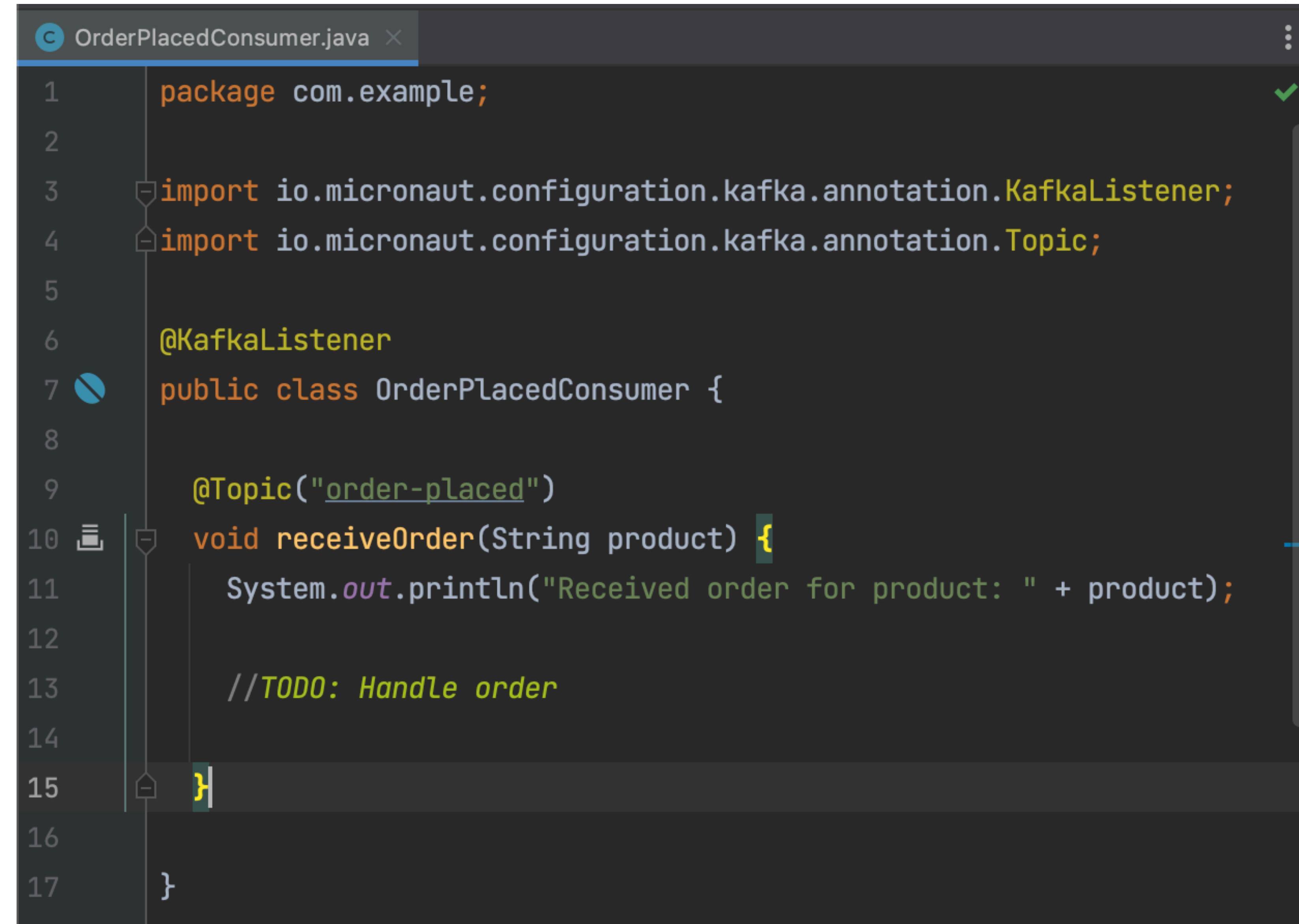
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
11
12     2 usages
13
14 14  public HelloController(OrderPlacedProducer producer) { this.producer = producer; }
15
16
17
18 18  @Post("/{product}")
19 19  @Produces(MediaType.TEXT_PLAIN)
20 20  HttpResponse<?> placeOrder(String product) {
21
22 22  producer.sendProduct(product);
23
24 24  return HttpResponse.ok();
25 25 }
```

Consumer: @KafkaListener



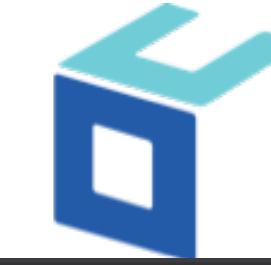
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



A screenshot of a code editor showing a Java file named `OrderPlacedConsumer.java`. The code defines a consumer for a Kafka topic named "order-placed". The code is as follows:

```
1 package com.example;
2
3 import io.micronaut.configuration.kafka.annotation.KafkaListener;
4 import io.micronaut.configuration.kafka.annotation.Topic;
5
6 @KafkaListener
7 public class OrderPlacedConsumer {
8
9     @Topic("order-placed")
10    void receiveOrder(String product) {
11        System.out.println("Received order for product: " + product);
12
13        //TODO: Handle order
14
15    }
16
17 }
```

Use POJO as Event Payload



```
c OrderEvent.java ×
1 package com.example;
2
3 +import ...
7
8 1 usage
9 @Introspected
10 public class OrderEvent {
11     2 usages
12     private final Long id;
13     2 usages
14     private final String customer;
15     2 usages
16     private final Date dateCreated;
17     2 usages
18     private final List<String> products;
19
20     public OrderEvent(Long id, String customer, Date dateCreated, List<String> products) {
21         this.id = id;
22         this.customer = customer;
23     }
24 }
```

The screenshot shows the Kafka Magic interface for Apache Kafka®. The top navigation bar includes standard browser controls (refresh, back, forward) and a search bar set to "localhost". Below the header, there's a purple navigation bar with links for "Pro Features" and "Feedback". The main content area has tabs for "Run Search" (selected), "Move messages", "Publish message", "Create Topic", "Automate Tasks", and "Delete Topic". On the left, a sidebar titled "Cluster Explorer" lists "Kafka Clusters" and "Localhost". Under "Localhost", it shows "Topics" (with a "find..." input field), including "_consumer_offsets", "order-metrics", "Partitions", and "order-placed". The "order-placed" topic is selected and highlighted in green. Under "order-placed", it shows "Partitions" and "Partition 0". The central panel displays search results under "Results JSON". A single message is shown as a JSON object:

```
[{"Timestamp": "2022-10-03T15:29:45.151+00:00", "Topic": "order-placed", "Partition": 0, "Offset": 0, "SchemaId": null, "SchemaType": null, "Key": [99, 117, 115, 116, 111, 109, 101, 114, 32, 65], "Headers": null, "Message": { "id": "1", "customer": "customer A", "dateCreated": 1288180702973, "products": [ "productA" ] }, "Error": null}]
```

At the bottom right, a URL is displayed: <https://www.kafkamagic.com/start/>

@KafkaKey to specify Event key



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

The screenshot shows a code editor window for a file named `OrderPlacedProducer.java`. The code defines a public interface `OrderPlacedProducer` with a single method `sendProduct`. The `sendProduct` method is annotated with `@Topic("order-placed")` and `@KafkaKey String orderId,`. The code editor has syntax highlighting and a dark theme. A yellow lightbulb icon is visible next to the `sendProduct` method, indicating potential code analysis or suggestions.

```
1 package com.example;
2
3 import io.micronaut.configuration.kafka.annotation.KafkaClient;
4 import io.micronaut.configuration.kafka.annotation.KafkaKey;
5 import io.micronaut.configuration.kafka.annotation.Topic;
6
7 2 usages
8
9
10 @KafkaClient
11
12 public interface OrderPlacedProducer {
13
14     1 usage
15
16     @Topic("order-placed")
17
18     void sendProduct(@KafkaKey String orderId,
19                      OrderEvent orderEvent);
20 }
```

@KafkaKey to specify Event key

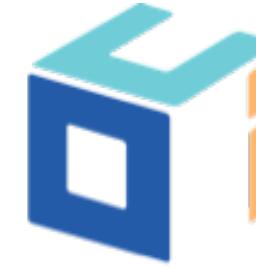


OBJECT COMPUTING

The screenshot shows a Java code editor with the file `OrderPlacedConsumer.java` open. The code defines a Kafka consumer for the topic "order-placed". The `receiveOrder` method is annotated with `@KafkaKey String orderId,`, which is highlighted with a yellow rectangle. The code prints the order ID and product count to `System.out`. A TODO comment is present to handle the order. The code editor interface includes a toolbar at the top and a status bar at the bottom.

```
6
7 @KafkaListener
8 public class OrderPlacedConsumer {
9
10    @Topic("order-placed")
11    void receiveOrder(@KafkaKey String orderId,
12                      OrderEvent orderEvent) {
13        System.out.println("Received order with id: " +
14            orderId +
15            " for products: " +
16            orderEvent.getProducts().size());
17
18        //TODO: Handle order
19
20    }
21
22 }
```

Specify Consumer Groups

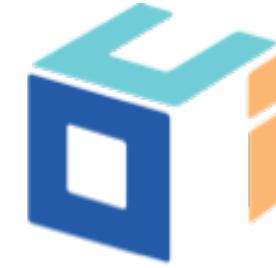


OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

A screenshot of a code editor showing a Java file named `OrderPlacedConsumer.java`. The code defines a class `OrderPlacedConsumer` with a `@KafkaListener` annotation. The `groupId` attribute is highlighted with a yellow box. The code also includes annotations `@Topic` and `@KafkaKey`, and a method `receiveOrder` that prints a message to `System.out`. A `TODO` comment is present in the code.

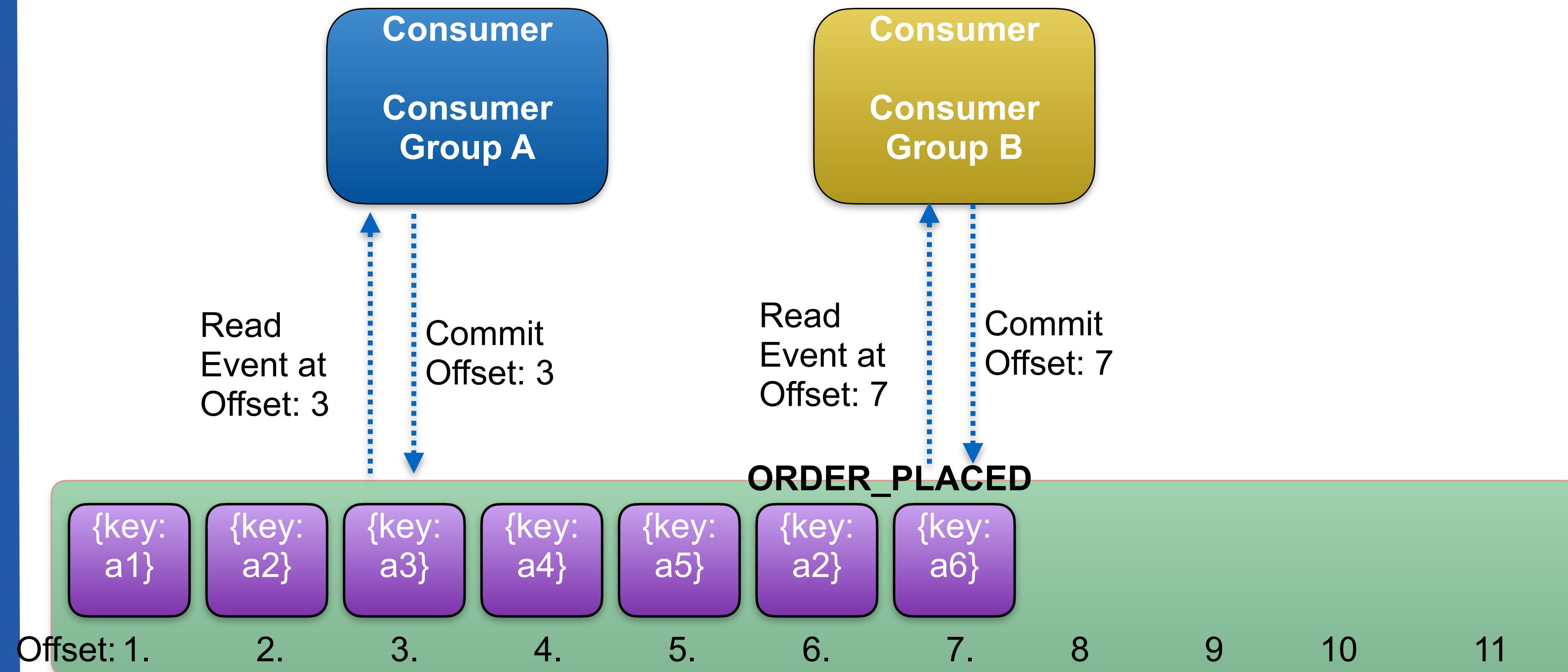
```
6
7     @KafkaListener(
8         groupId="order-placed-group",
9         threads=10)
10    public class OrderPlacedConsumer {
11
12        @Topic("order-placed")
13        void receiveOrder(@KafkaKey String customer,
14                           OrderEvent orderEvent) {
15            System.out.println("Received order from customer: " +
16                               customer +
17                               " for products: " +
18                               orderEvent.getProducts().size());
19
20            //TODO: Handle order
21        }
22    }
```

Configure OffsetStrategy



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Default: Auto-commit



Configure OffsetStrategy



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Example: Disabled (Manual Acknowledgement)

```
c OrderPlacedConsumer.java ×

9  @KafkaListener(
10     offsetStrategy = OffsetStrategy.DISABLED,
11     groupId="order-placed-group",
12     threads=10)
13 public class OrderPlacedConsumer {
14
15     @Topic("order-placed")
16     void receiveOrder(@KafkaKey String customer,
17                         OrderEvent orderEvent,
18                         Acknowledgement acknowledgement) {
19         System.out.println("Received order from customer: " +
20                           customer +
21                           " for products: " +
22                           orderEvent.getProducts().size());
23
24         //TODO: Handle order
25
26         acknowledgement.ack();
```

Route Events with @SendTo



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

The screenshot shows a code editor window for a Java file named `OrderPlacedConsumer.java`. The code defines a class `OrderPlacedConsumer` with a `@KafkaListener` annotation. Inside the listener method, there is a `@Topic` annotation on the parameter `orderEvent`, and a `@SendTo` annotation on the return statement. The `@SendTo` annotation is highlighted with a yellow box. The code also includes a `receiveOrder` method that prints a message to `System.out` and then acknowledges it.

```
10     * @KafkaListener(
11     *     offsetStrategy = OffsetStrategy.DISABLED,
12     *     groupId="order-placed-group",
13     *     threads=10)
14
15     public class OrderPlacedConsumer {
16
17         * @Topic("order-placed")
18         * @SendTo("order-metrics")
19         * Integer receiveOrder(@KafkaKey String customer,
20         *                      OrderEvent orderEvent,
21         *                      Acknowledgement acknowledgement) {
22
23             System.out.println(
24                 "Received order from customer: " + customer
25             );
26
27             acknowledgement.ack();
28
29     }
30 }
```

Exception handling: ErrorStrategy



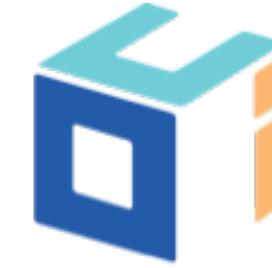
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

The screenshot shows a code editor window for `OrderPlacedConsumer.java` in a project named `demo`. The code uses the `@KafkaListener` annotation to handle messages from the `"order-placed"` topic and send them to the `"order-metrics"` topic. It demonstrates exception handling with the `@ErrorStrategy` annotation, which specifies a retry strategy of `RETRY_ON_ERROR` with a `retryDelay` of `"50ms"` and a `retryCount` of `3`.

```
OrderPlacedConsumer.java [demo.main]
demo > src > main > java > com > example > OrderPlacedConsumer
OrderPlacedConsumer.java ×

8
9  @KafkaListener(
10 +     offsetStrategy = OffsetStrategy.DISABLED,
11   ⚡ errorStrategy = @ErrorStrategy(value = RETRY_ON_ERROR, retryDelay = "50ms", retryCount=3),
12   groupId="order-placed-group",
13   threads=10)
14
15  public class OrderPlacedConsumer {
16
17    @Topic("order-placed")
18    @SendTo("order-metrics")
19  ⏷  @  Integer receiveOrder(@KafkaKey String customer,
20                           OrderEvent orderEvent,
21                           Acknowledgement acknowledgement) {
22    System.out.println(
23      "Received order from customer: " + customer
24  }
```

KafkaListenerExceptionHandler



OBJECT COMPUTING

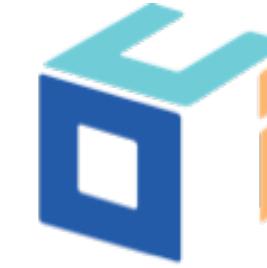
HOME TO SPRINGS & MICROSERVANT

Reader Mode



```
@Singleton  
@Primary  
public class DefaultKafkaListenerExceptionHandler implements KafkaListenerExceptionHandler {  
    6 usages  
    private static final Logger LOG = LoggerFactory.getLogger(KafkaListenerExceptionHandler.class);  
    1 usage  
    private static final Pattern SERIALIZATION_EXCEPTION_MESSAGE_PATTERN = Pattern.compile( regex: ".+ for partiti  
  
    2 usages  
    private boolean skipRecordOnDeserializationFailure = true;  
  
    @Override  
    public void handle(KafkaListenerException exception) {  
        final Throwable cause = exception.getCause();  
        final Object consumerBean = exception.getKafkaListener();  
        if (cause instanceof SerializationException) {...} else {  
            if (LOG.isErrorEnabled()) {  
                Optional<ConsumerRecord<?, ?>> consumerRecord = exception.getConsumerRecord();  
                if (consumerRecord.isPresent()) {...} else {  
                    LOG.error("Kafka consumer [{}] produced error: {}", consumerBean, cause.getMessage(), cause);  
                }  
            }  
        }  
    }  
}
```

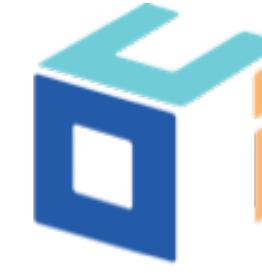
KafkaListenerExceptionHandler



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

```
c CustomKafkaListenerExceptionHandler.java × : A 1 ▲ ▼
11
12     @Singleton
13     @Replaces(DefaultKafkaListenerExceptionHandler.class)
14     public class CustomKafkaListenerExceptionHandler implements KafkaListenerExceptionHandler {
15         @Override
16         public void handle(KafkaListenerException exception) {
17
18             Optional<ConsumerRecord<?, ?>> consumerRecord = exception.getConsumerRecord();
19             if (consumerRecord.isPresent()) {
20                 //TODO Handle the consumer record
21             }
22         }
23     }
24
25 }
```

Health Checks, Metrics, & Tracing



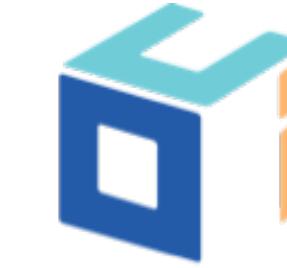
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Micronaut Kafka provides Kafka-specific health information to Micronaut's **management** endpoints (if enabled)
 - <https://docs.micronaut.io/latest/guide/#management>

```
{  
    "status": "DOWN",  
    "details": {  
        "kafka": {  
            "status": "DOWN",  
            "details": {  
                "error": "java.util.concurrent.ExecutionException:  
org.apache.kafka.common.errors.TimeoutException: Timed out waiting for a node  
assignment."  
            }  
        }  
    }  
}
```

`http://localhost:8080/health`

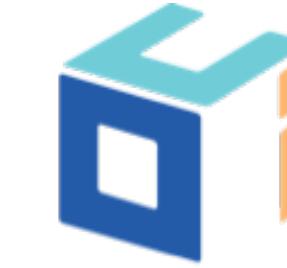
Health Checks, Metrics, & Tracing



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Micronaut Kafka provides metrics to Micronaut's **metrics** endpoint via Micrometer (if enabled)
 - <https://micronaut-projects.github.io/micronaut-micrometer/latest/guide/>
- Micronaut Kafka support **distributed tracing** via Micronaut Tracing, using Jaeger or Zipkin
 - <https://micronaut-projects.github.io/micronaut-tracing/latest/guide/>

Kafka Streams



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Kafka Streams is an additional library that provides advanced processing capabilities on top of Kafka topics
- Streams accept one/more input topics, define one or more operators/transformations, and publish results to an output topic
- Micronaut Kafka supports creation of **KStreams** that can be injected as **@Singleton** beans
- Kafka Stream's **InteractiveQueryService** is also exposed as an injectable bean
- More: <https://micronaut-projects.github.io/micronaut-kafka/latest/guide/#kafkaStreams>

The screenshot shows the Micronaut Launch interface on a dark-themed browser window. At the top, there's a navigation bar with icons for back, forward, search, and other browser functions. The URL bar shows micronaut.io. Below the header, the Micronaut logo (a stylized Greek letter μ) is displayed next to the text "MICRONAUT® LAUNCH". On the right side of the header, there are social media sharing icons for GitHub, Twitter, LinkedIn, and Email.

The main form area contains several dropdowns and input fields:

- Application Type:** A dropdown menu set to "Micronaut Application".
- Java Version:** A dropdown menu set to "11".
- Name:** An input field containing "demo".
- Base Package:** An input field containing "com.example".

Below these, there are three rows of configuration options:

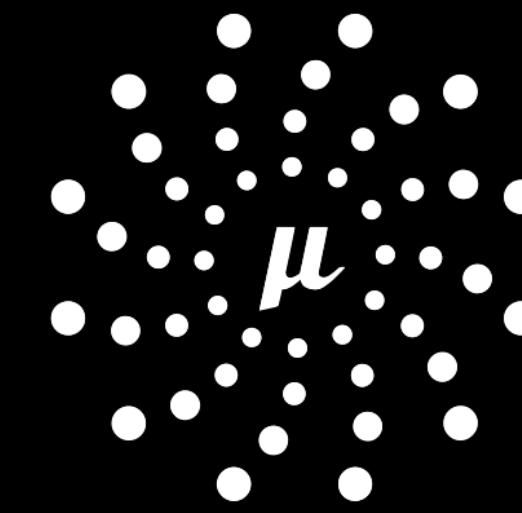
- Micronaut Version:** Radio buttons for "3.7.1" (selected), "3.8.0-SNAPSHOT", and "2.5.13".
- Language:** Radio buttons for "Java" (selected), "Groovy", and "Kotlin".
- Build Tool:** Radio buttons for "Gradle" (selected), "Gradle Kotlin", and "Maven".
- Test Framework:** Radio buttons for "JUnit" (selected), "Spock", and "Kotest".

At the bottom of the configuration area are four buttons:

- FEATURES**: A button with a plus sign and the word "FEATURES".
- DIFF**: A button with a left-right arrow icon.
- PREVIEW**: A button with a magnifying glass icon.
- GENERATE PROJECT**: A button with a magnifying glass icon.

Below the configuration area, the text "Included Features (0)" is displayed. At the very bottom of the page, the Micronaut Foundation logo is shown.

<https://micronaut.io/launch>



MICRONAUT FOUNDATION™

TECHNICAL ADVISORY BOARD

YURIY ARTAMONOV

JetBrains, Microservices Fellow

BRUNO BORGES

Microsoft, Principal Product Manager for Java

ZHAMAK DEHGHANI

ThoughtWorks, Principal Consultant

NEAL FORD

ThoughtWorks, Director, Cloud Architect

JAMES KLEEH

Amazon, Software Development Engineer

GUILLAUME LAFORGE

Google, Developer Advocate for Google Cloud Platform

GRAEME ROCHER

Micronaut Foundation, Co-founder and Director, Architect at Oracle

MARK SAILES

Amazon, Specialist Solution Architect for Serverless at AWS Cloud

JASON SCHINDLER

Object Computing Inc., Partner and Groovy, Grails, and Micronaut Team Manager

KEN SIPE

Edward Jones, Department Leader – Application and Technology Architecture

VENKAT SUBRAMANIAM

Agile Developer Inc., Founder



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Micronaut Resources



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

The screenshot shows a web browser displaying the guides.micronaut.io website. The page title is "Micronaut Guides". The main navigation bar includes links for DOWNLOAD, LEARN (which is underlined), RESOURCES, FOUNDATION, and LAUNCH. The LEARN link is currently active. Below the navigation, a breadcrumb trail shows the user has navigated from HOME to GUIDES to KAFKA AND THE MICRONAUT FRAMEWORK - EVENT-DRIVEN APPLICATIONS, specifically the GRADLE | JAVA version. The main content area features a "TABLE OF CONTENTS" sidebar on the left with a blue background, listing chapters from 1. GETTING STARTED to 8.1. NATIVE EXECUTABLE GENERATION. The main content on the right is titled "KAFKA AND THE MICRONAUT FRAMEWORK - EVENT-DRIVEN APPLICATIONS" and contains a sub-section titled "Use Kafka to communicate between your Micronaut applications." It lists authors Burt Beckwith and Micronaut Version 3.7.0. The "1. Getting Started" section describes creating a Java application, and the "2. What you will need" section lists required resources.

Micronaut Guides

KAFKA AND THE MICRONAUT FRAMEWORK - EVENT-DRIVEN APPLICATIONS

Use Kafka to communicate between your Micronaut applications.

Authors: Burt Beckwith
Micronaut Version: 3.7.0

1. Getting Started

In this guide, we will create a Micronaut application written in Java.

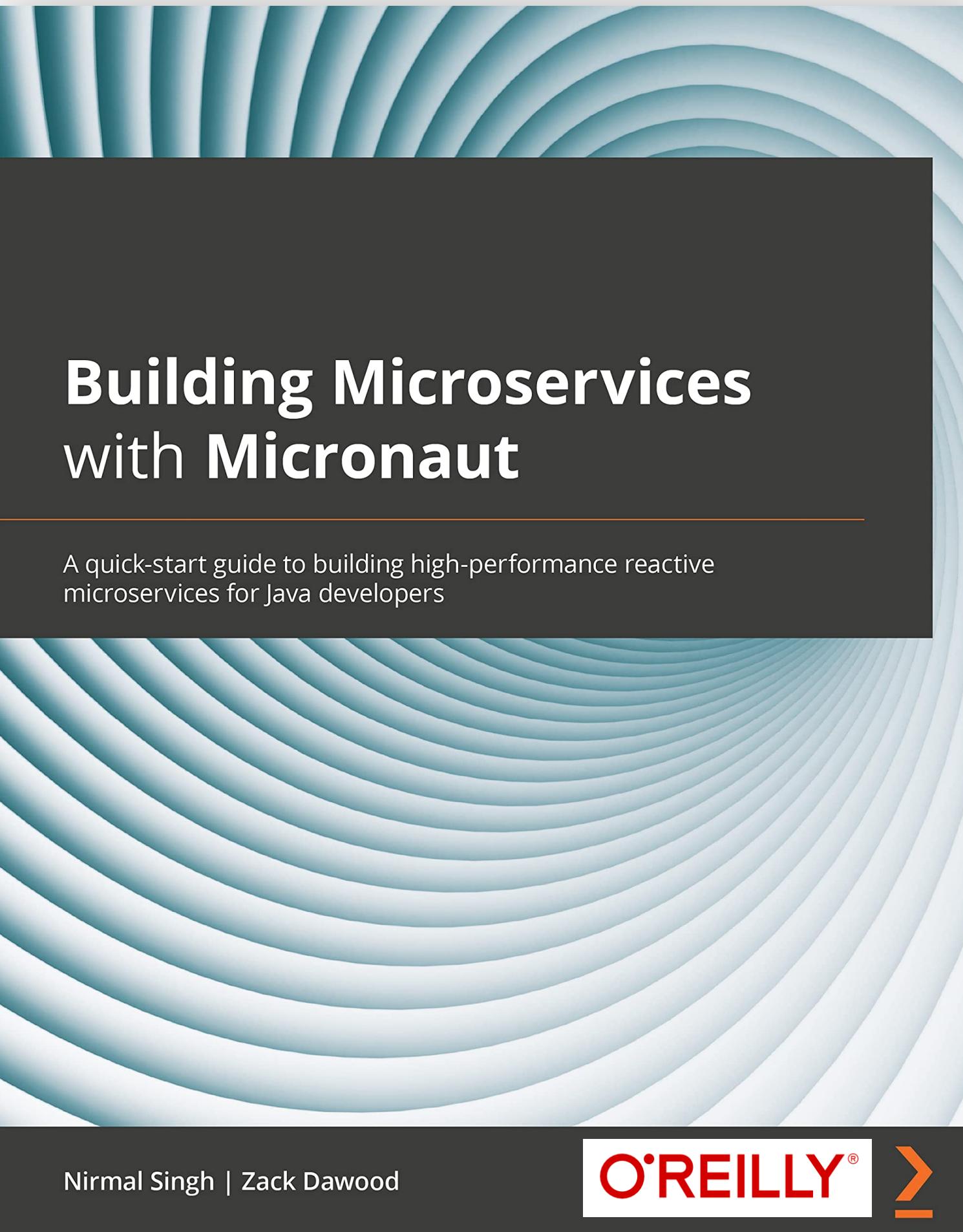
In this guide, we will create two microservices that will use [Kafka](#) to communicate with each other in an asynchronous and decoupled way.

2. What you will need

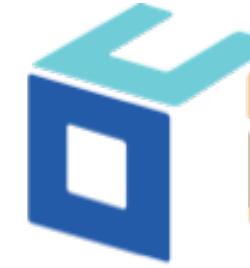
To complete this guide, you will need the following:

Micronaut Resources

Books:



<https://www.packtpub.com/product/building-microservices-with-micronaut/9781800564237>



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Podcast:



<https://micronautpodcast.com/>

Micronaut Resources



167 lines (149 sloc) | 26.3 KB

all contributors 2

A hand-picked and curated list of resources for the [Micronaut framework](#).

Contributing

Pull requests are highly appreciated. Please take note of the [guidelines](#) before contributing. Thanks to every [contributor](#).

Micronaut

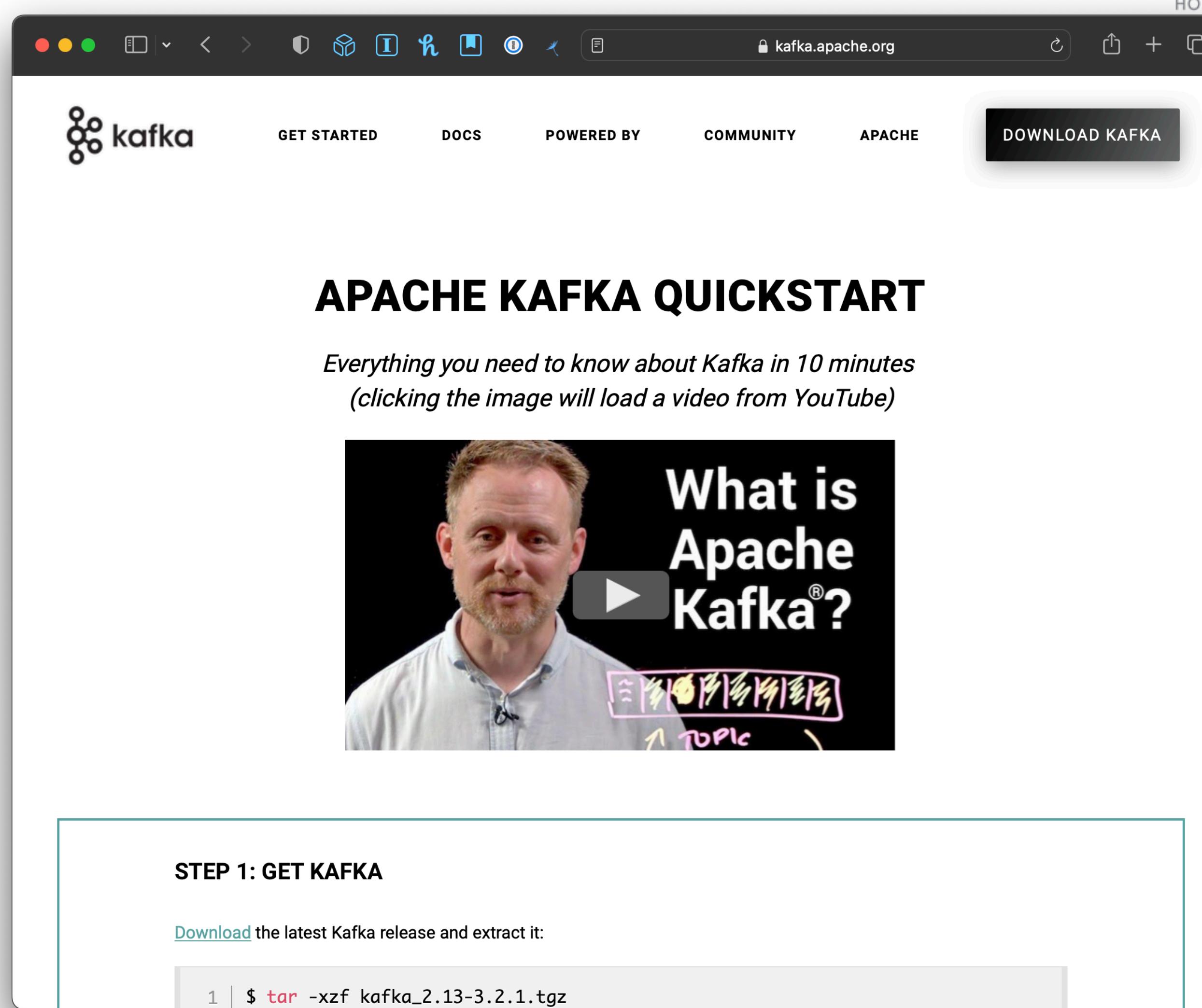
Micronaut is a modern, JVM-based, full stack microservices framework designed for building modular, easily testable microservice applications.

- Official Website: <http://micronaut.io/>
- Documentation: <https://micronaut.io/documentation.html>
- Micronaut Guides: <https://guides.micronaut.io/index.html>
- FAQ: <https://micronaut.io/faq.html>
- Github: <https://github.com/micronaut-projects/>
- Examples: <https://github.com/micronaut-projects/micronaut-examples/>
- Gitter Community: <https://gitter.im/micronautfw/>
- Twitter: <https://twitter.com/micronautfw/>
- Twitter Hashtag: #micronautfw



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Kafka/Event-Driven Resources



The screenshot shows a web browser displaying the Apache Kafka quickstart page. The URL in the address bar is kafka.apache.org/quickstart. The page features the Apache Kafka logo and navigation links for GET STARTED, DOCS, POWERED BY, COMMUNITY, and APACHE. A prominent "DOWNLOAD KAFKA" button is visible. The main content area is titled "APACHE KAFKA QUICKSTART" and includes a video thumbnail for a "What is Apache Kafka?" video. Below the video, a section titled "STEP 1: GET KAFKA" provides instructions to download the latest Kafka release and extract it, with a terminal command example shown.

APACHE KAFKA QUICKSTART

*Everything you need to know about Kafka in 10 minutes
(clicking the image will load a video from YouTube)*

What is Apache Kafka®?

STEP 1: GET KAFKA

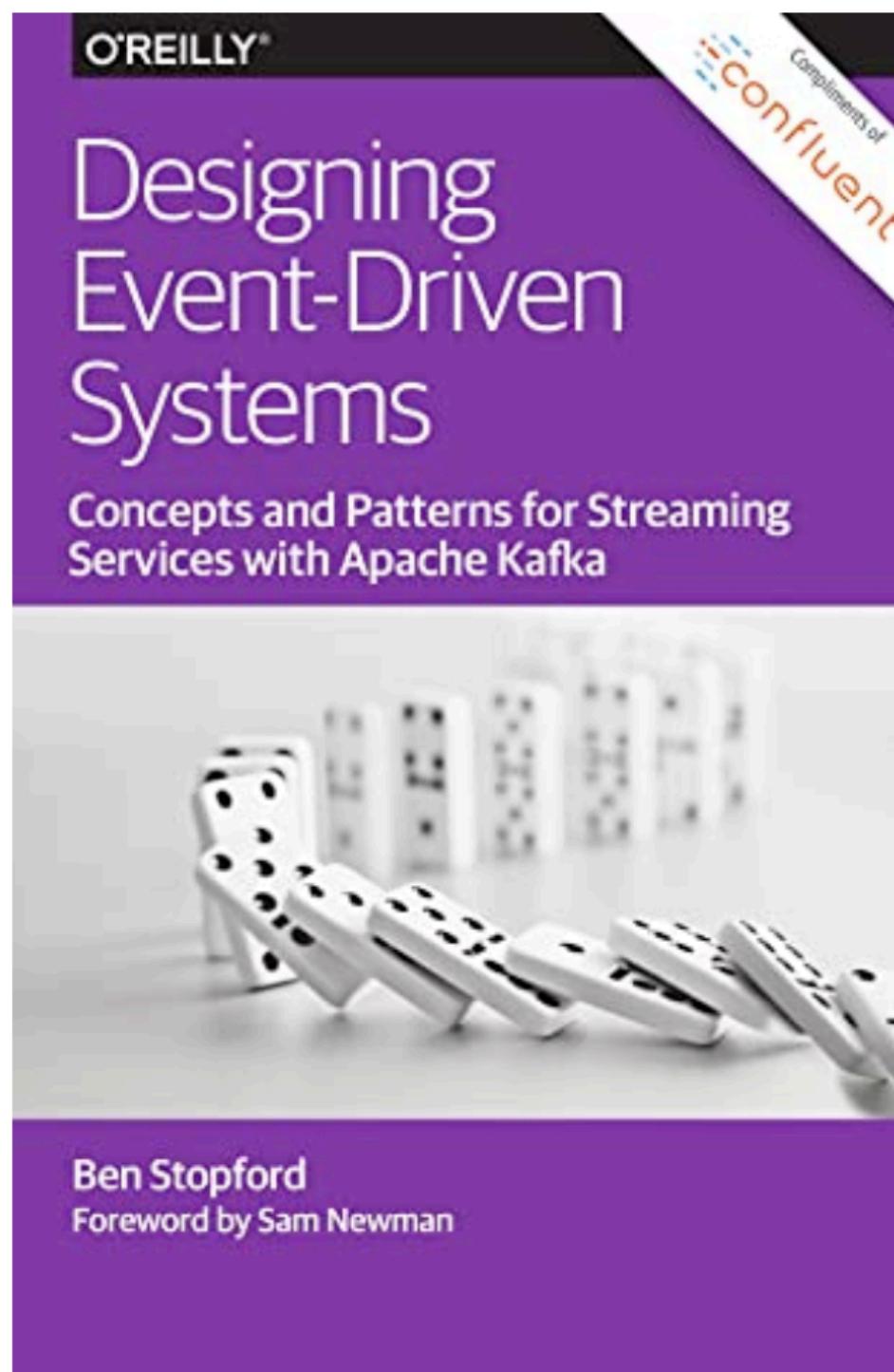
[Download](#) the latest Kafka release and extract it:

```
1 | $ tar -xzf kafka_2.13-3.2.1.tgz
```

Kafka/Event-Driven Resources



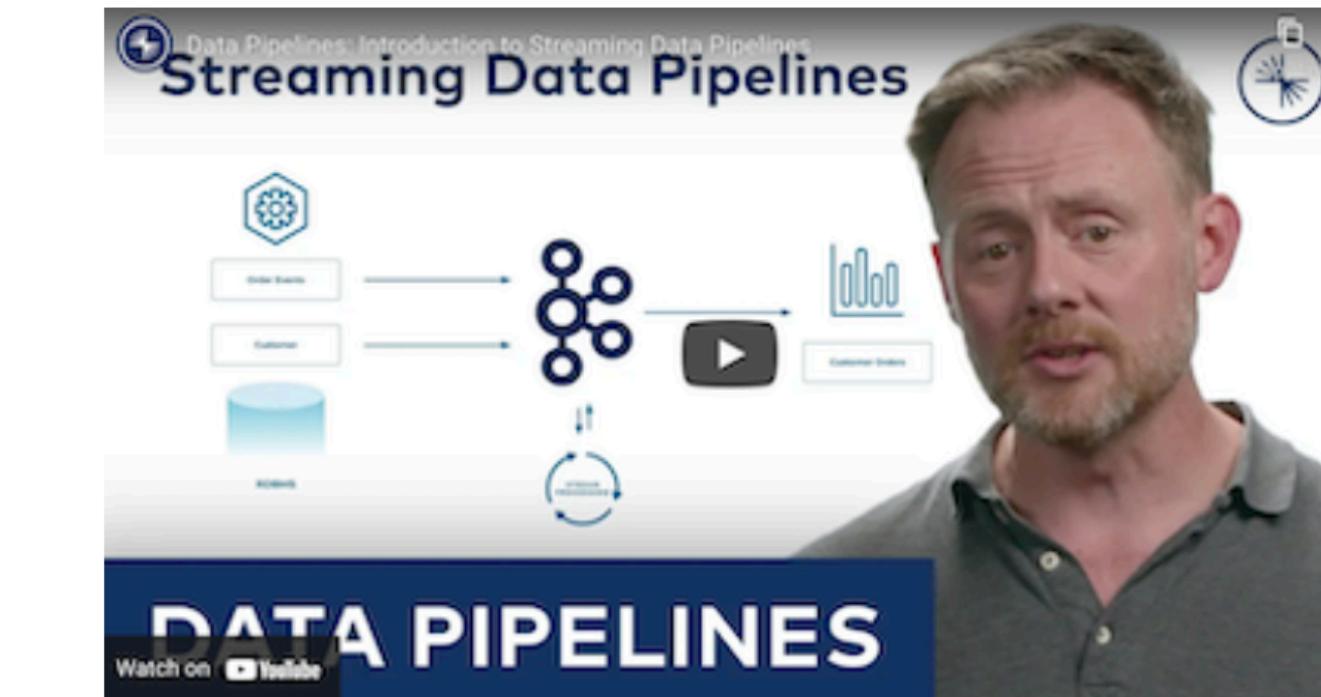
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



<https://cnfl.io/book-bundle>



<https://developer.confluent.io/learn-kafka/event-sourcing>



<https://developer.confluent.io/learn-kafka/data-pipelines>

Learn More: OCI Training Event



EVENT-DRIVEN MICROSERVICES WITH MICRONAUT

This 12-hour workshop is a great fit for those who want to gain a productive understanding of the Micronaut® framework, Apache Kafka, and event-driven architecture in cloud-based microservices. Engage in hands-on exercises to gain experience and working examples of event-driven microservices using the Micronaut framework and Apache Kafka.

NEXT OPEN-ENROLLMENT CLASS

Dates: November 14 to November 17

Time: 9:00 a.m. to 12:00 p.m. CST

Duration: 12 hours (3 hours per day for 4 days)

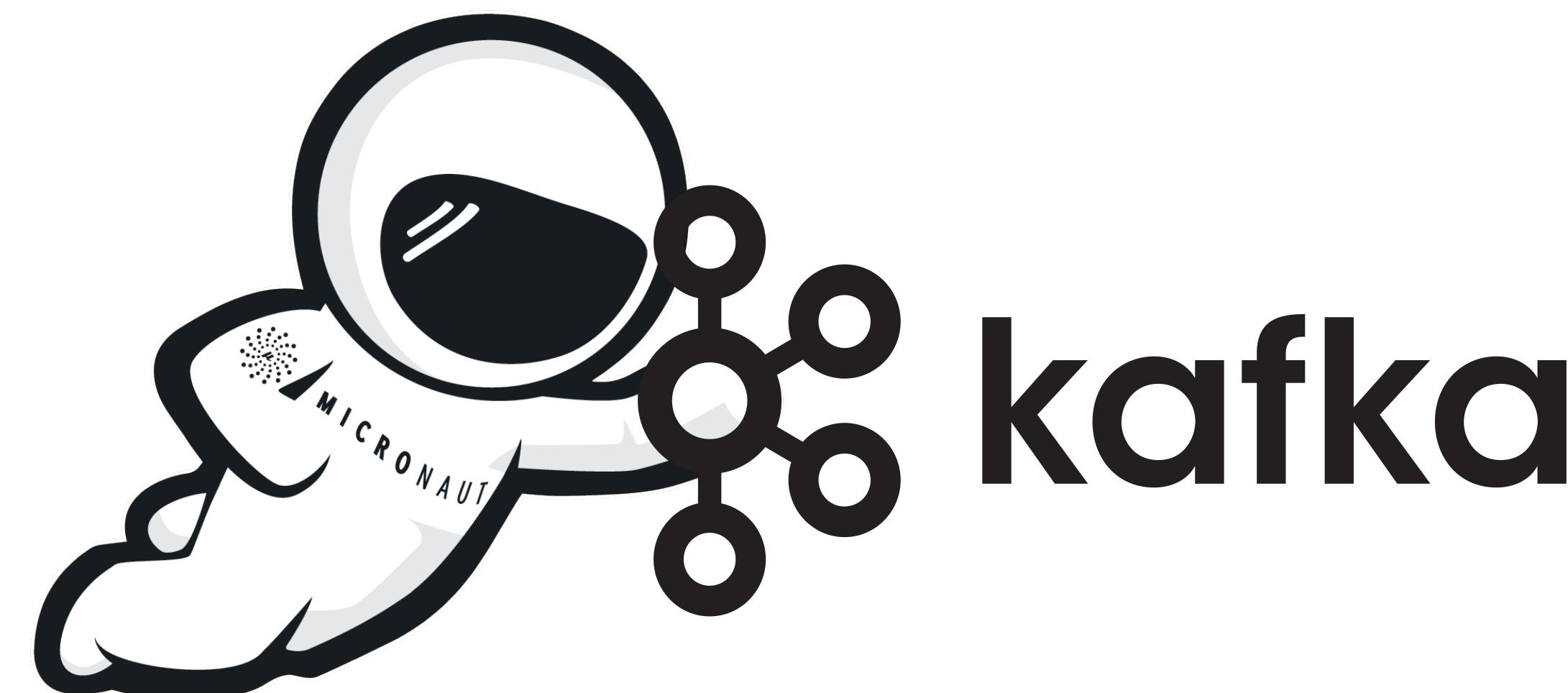


[https://micronaut.io/professional-training/
event-driven-microservices-with-micronaut/](https://micronaut.io/professional-training/event-driven-microservices-with-micronaut/)



Thank you!

Zachary Klein, Principal Software Engineer, 2GM Team



APACHECON



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Scan with your device.
Retweet for a chance at free T-shirt!

