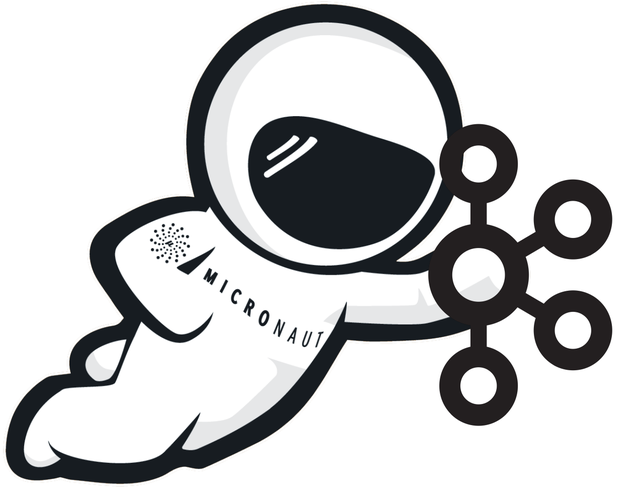
APACHECON

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Event-Driven Microservices With Apache Kafka and Micronaut

Zachary Klein, Principal Software Engineer, 2GM Team

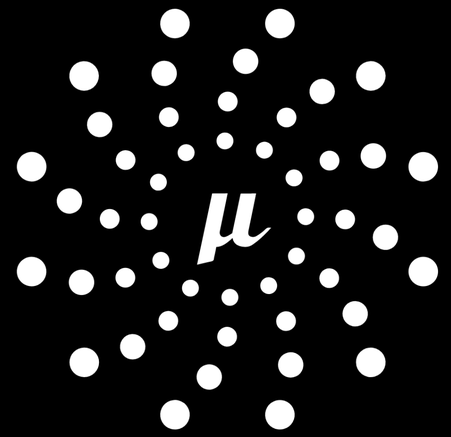


kafka

About me

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

OBJECT COMPUTING

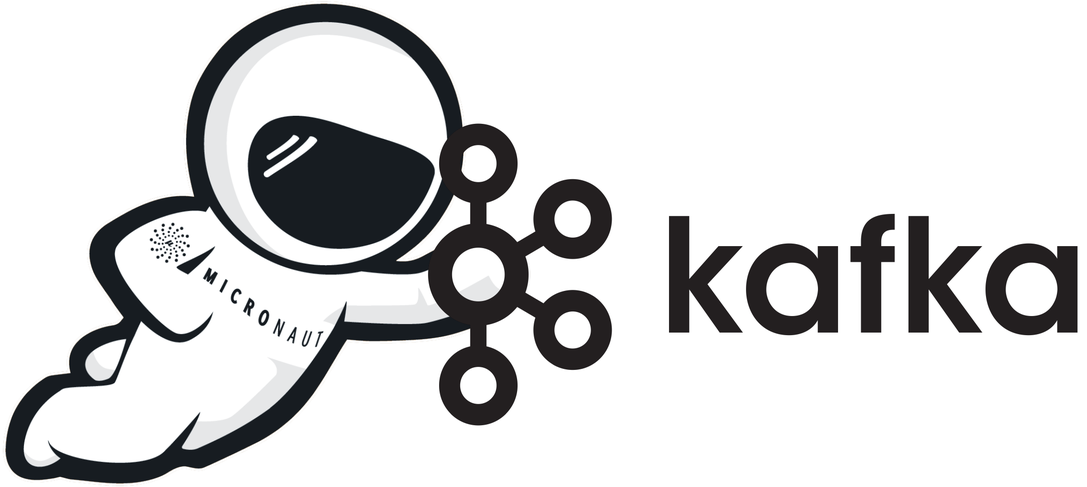


MICRONAUT®

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

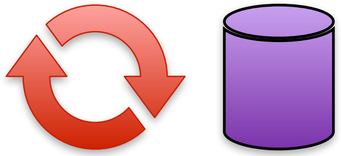


What is **Event-Driven** Architecture?

1. Event Notification

2. State Transfer 

3. Event Sourcing 

4. CQRS* 

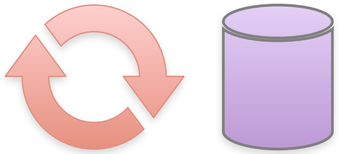
* Command & Query Responsibility Segregation

What is **Event-Driven** Architecture?

1. Event Notification

2. State Transfer 

3. Event Sourcing 

4. CQRS* 

* Command & Query Responsibility Segregation

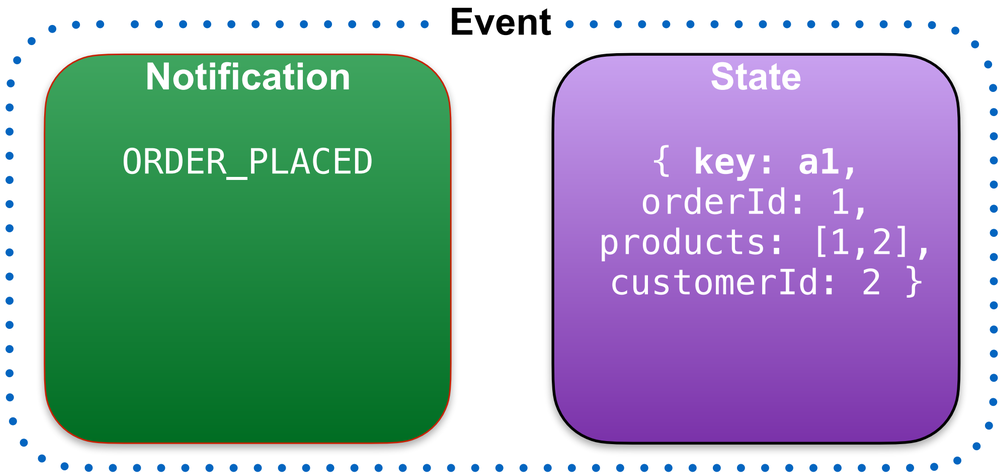
Event-Driven architecture: Key points

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

What is Apache Kafka?

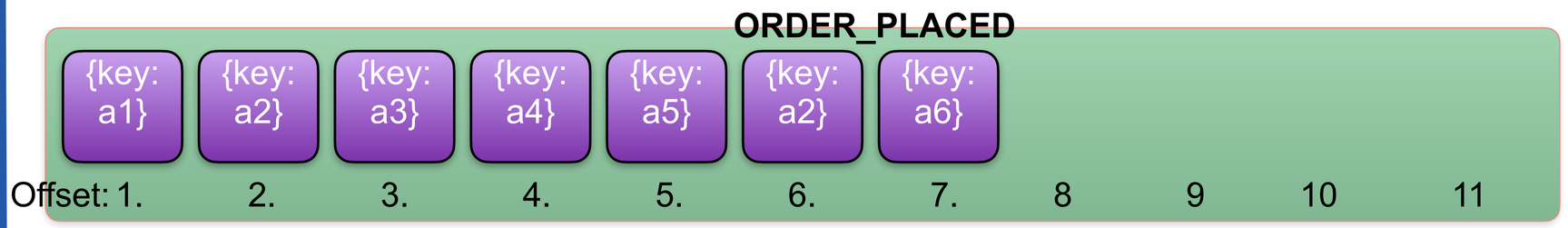
- Event-Streaming Platform
- Based on the concept of a **commit log**
  - Immutable
  - Persistent
  - Divided by “topics”
  - Partitioned
  - Events (log entries) are optionally keyed

What is an Event?

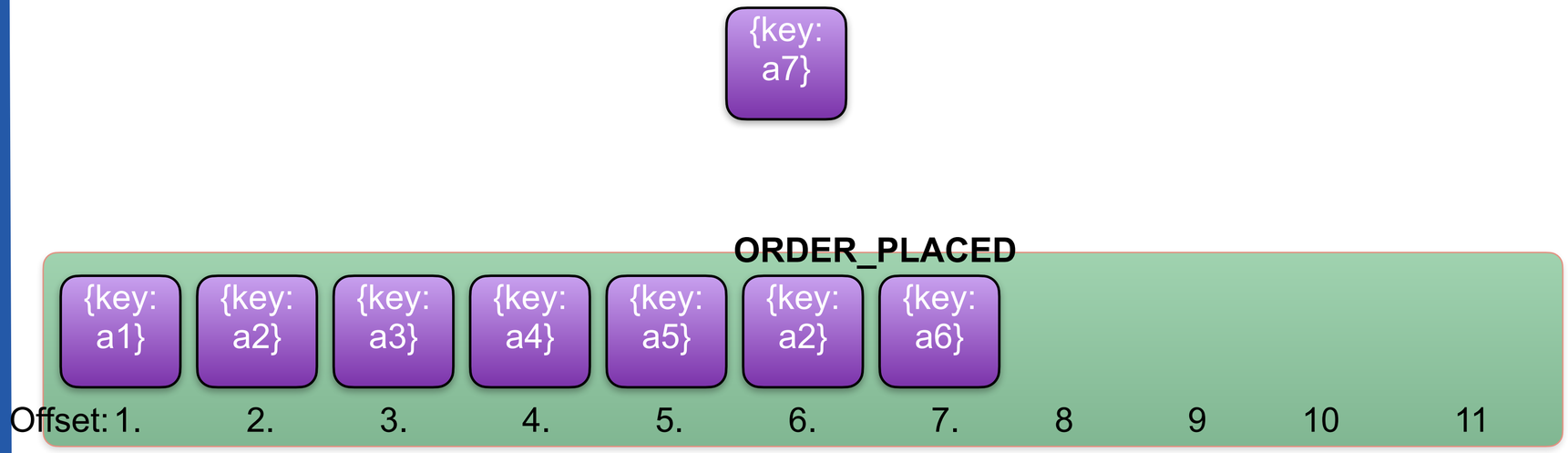


What is a Topic?

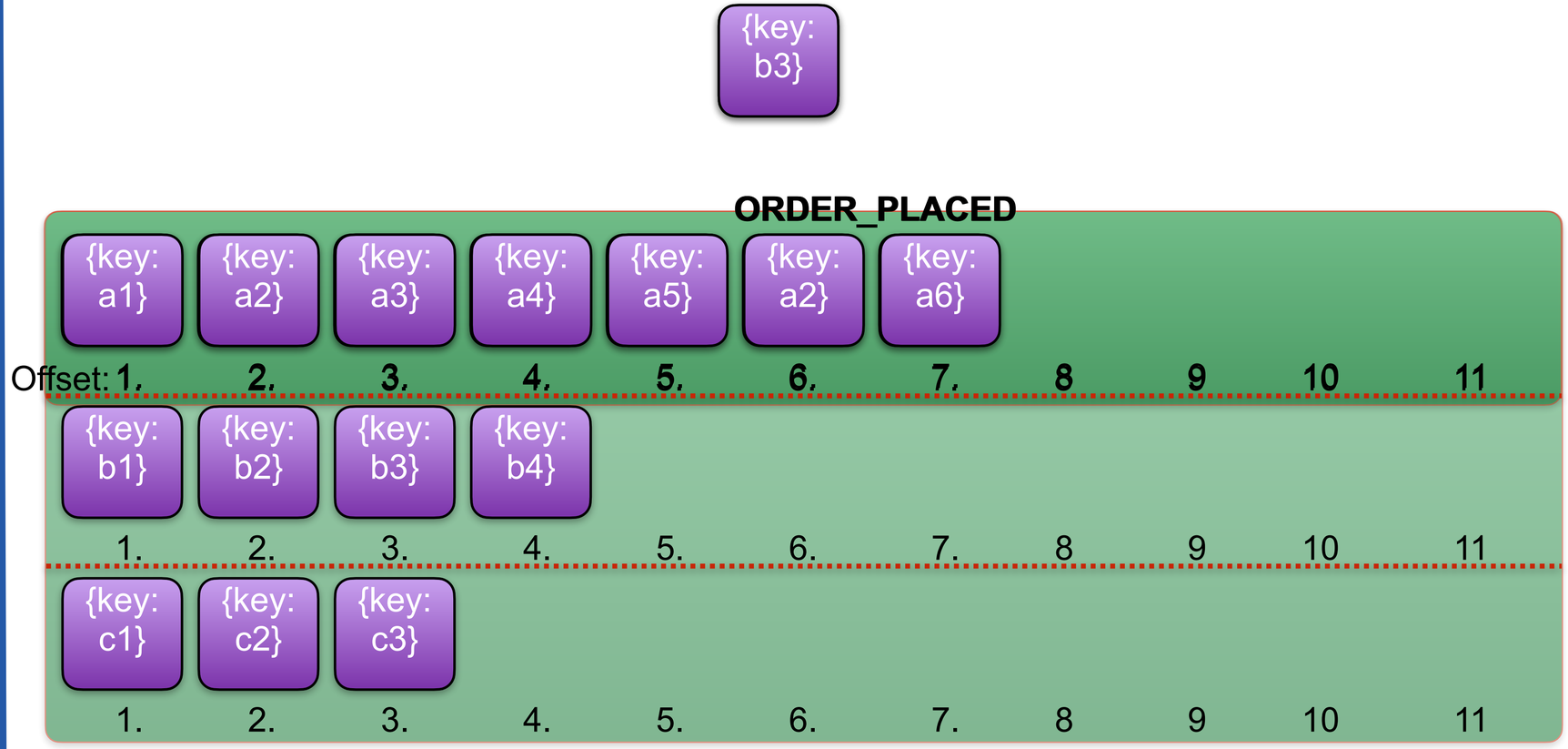
- Topics are persistent queues
- Events in the topic have an **offset**
- Events can be read from the topic in multiple modes - latest, all events from the beginning, batches, and even queried (e.g, using ksqlDB)



Adding an event to a topic

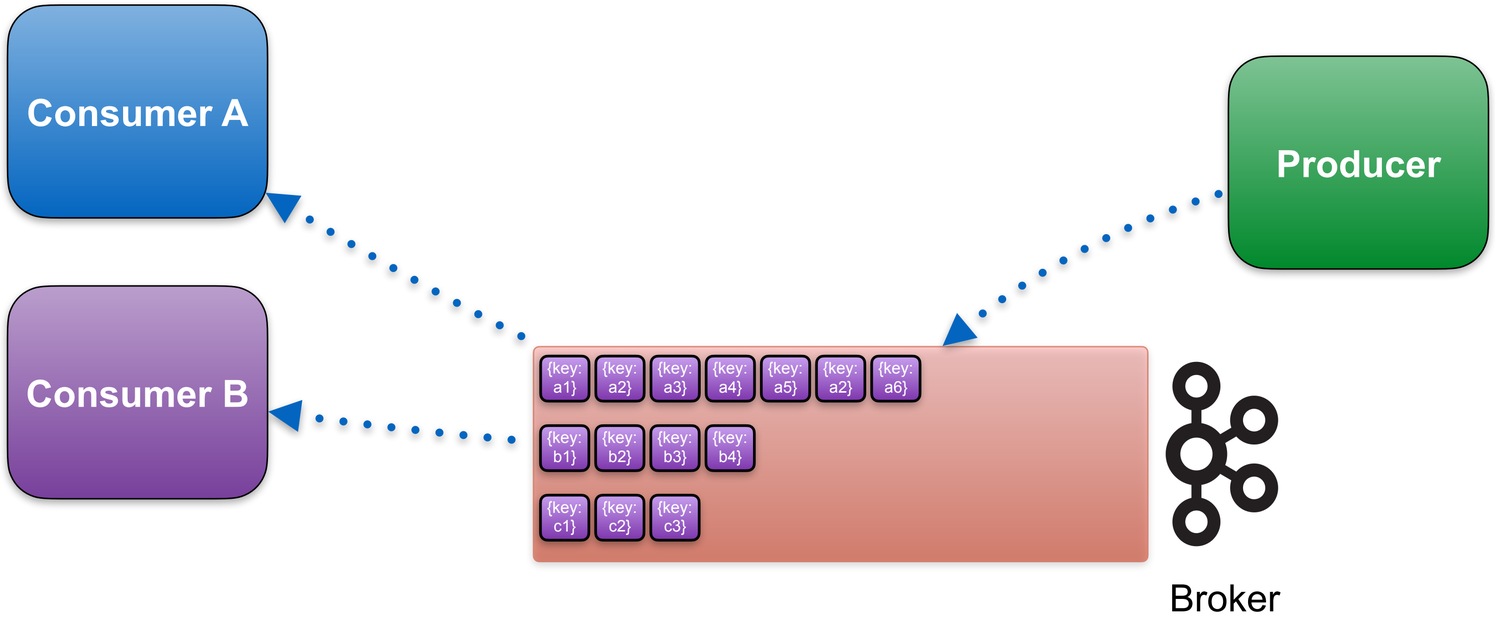


Topic partitions

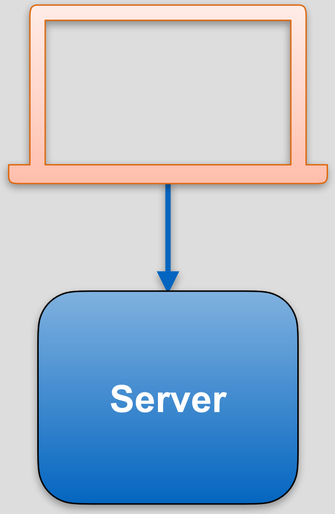


Producers and Consumers

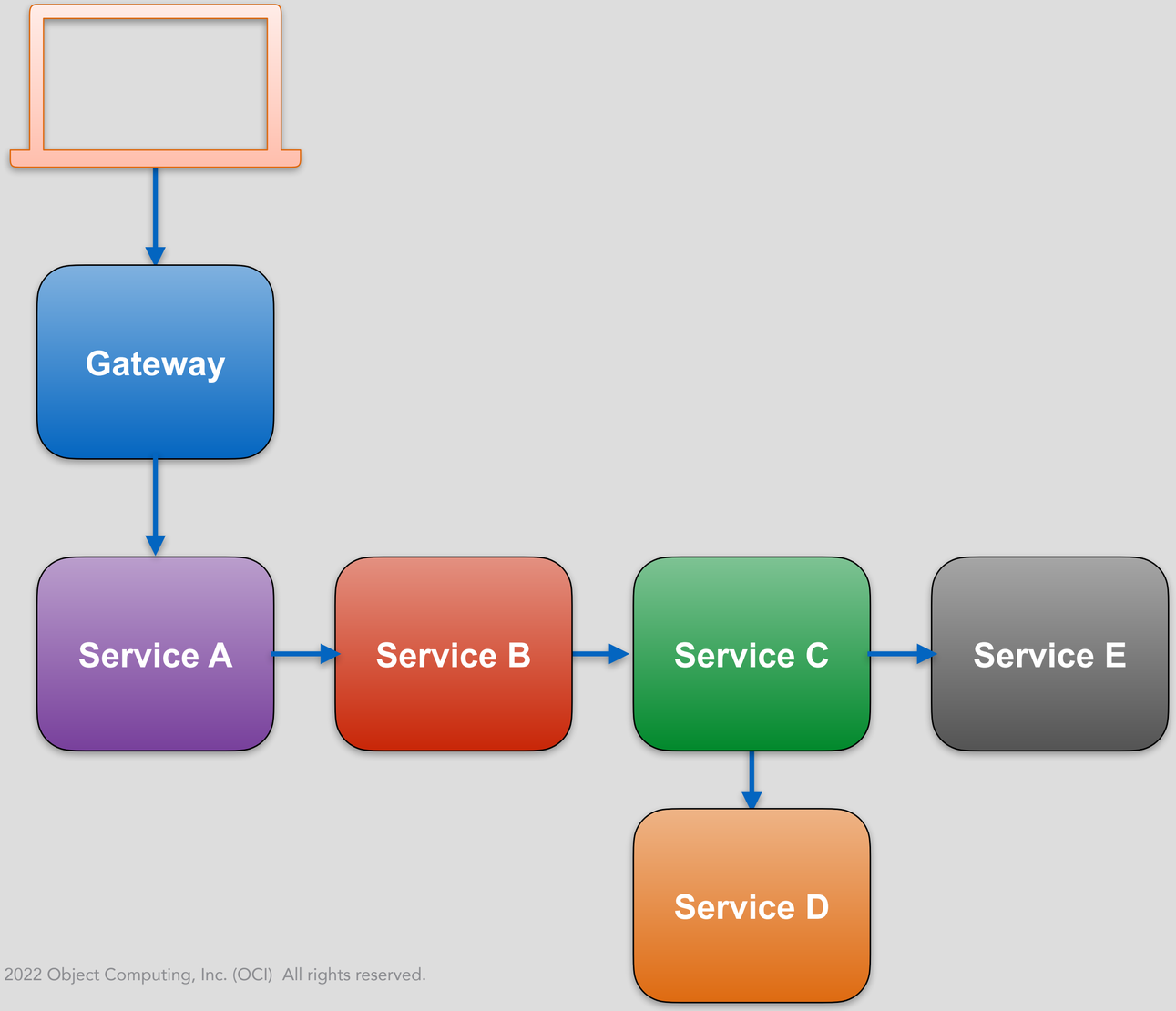
- Producers write events **to** topic/s
- Consumers read events **from** topics
- Events are read sequentially



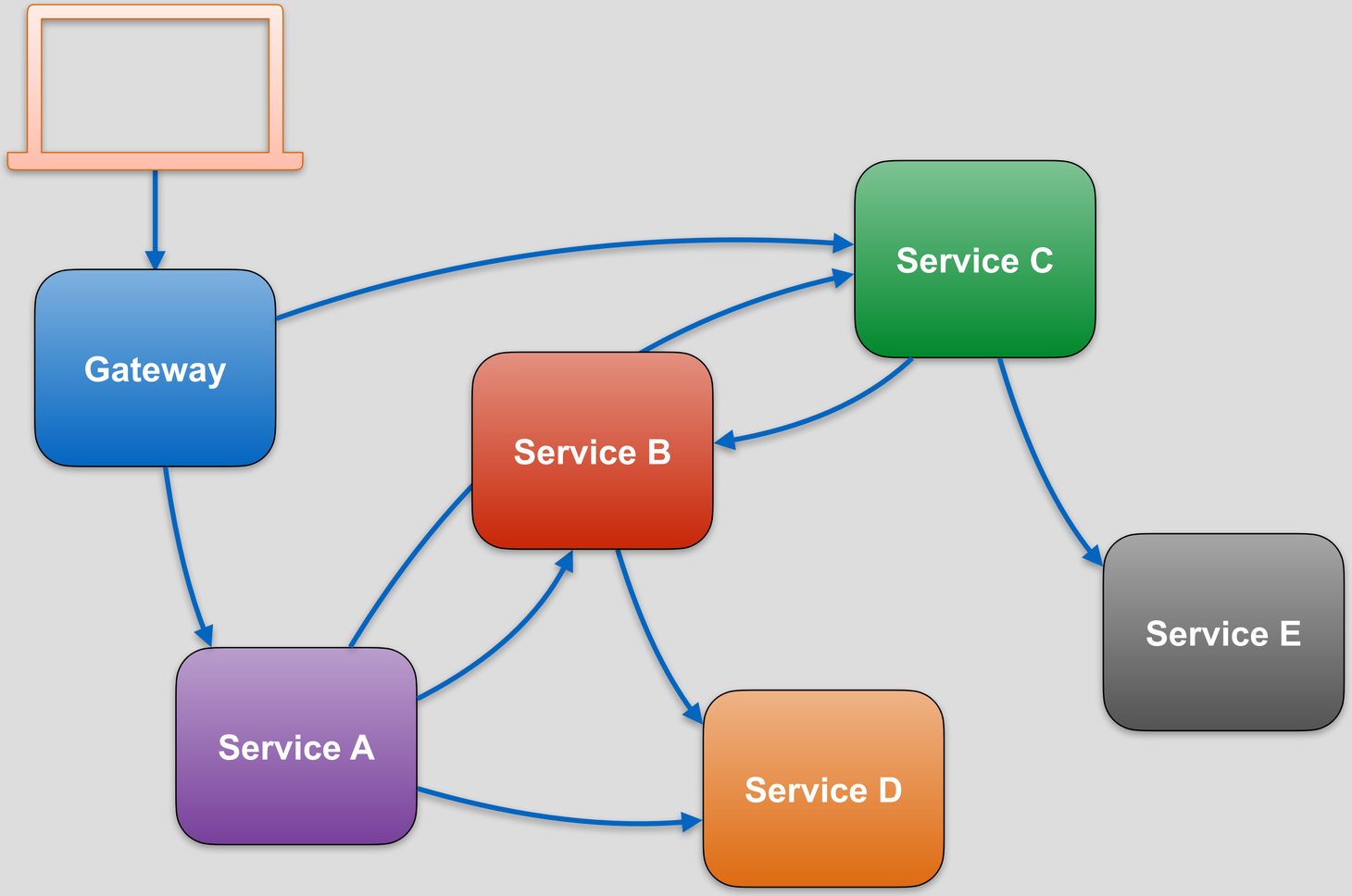
Inter-Service Communication



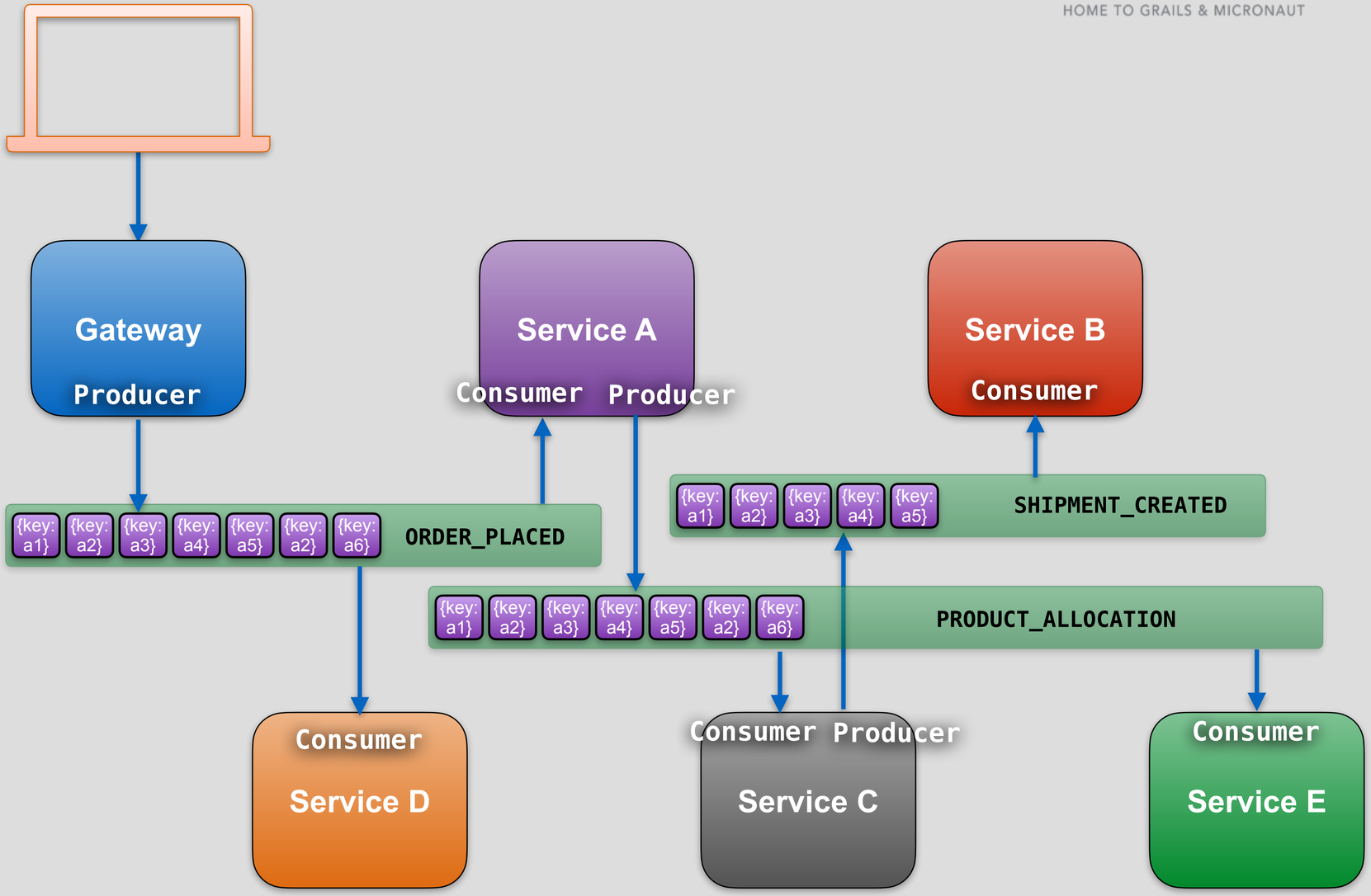
Inter-Service Communication



Inter-Service Communication

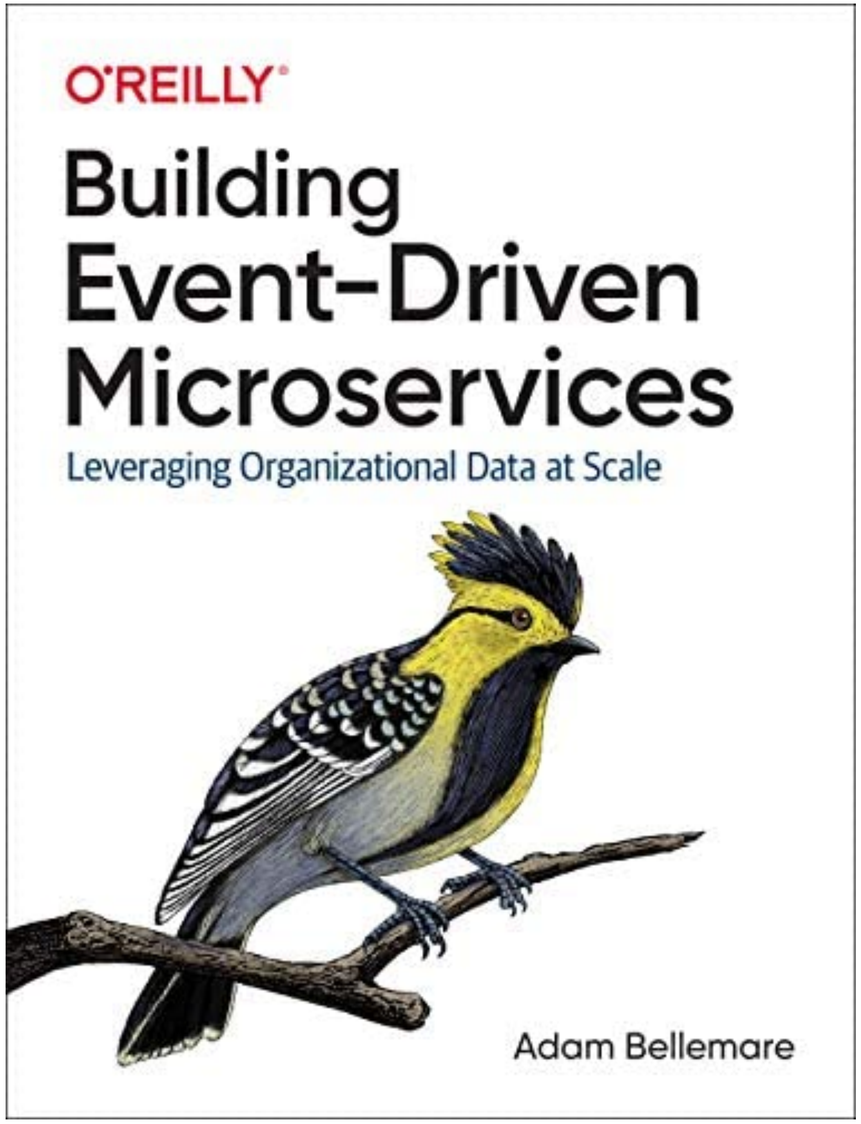


Event-Driven Microservices



Event-Driven Microservices

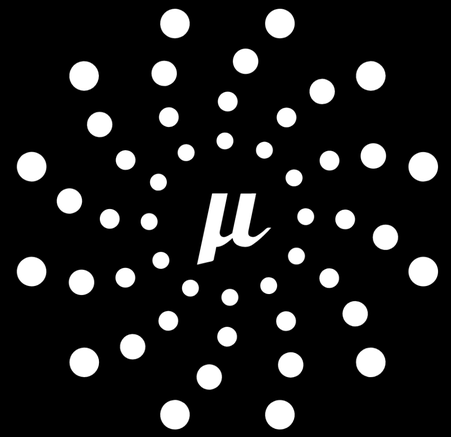
- Event-driven architecture plays to microservices' strengths
  - Bounded context
  - Loose coupling
- Multiple services can consume events from same topic/s
- Events are persistent (replayable)
- Events can be materialized as tables for traditional querying (ksqlDB)
- Events replace REST calls for inter-service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



OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

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

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

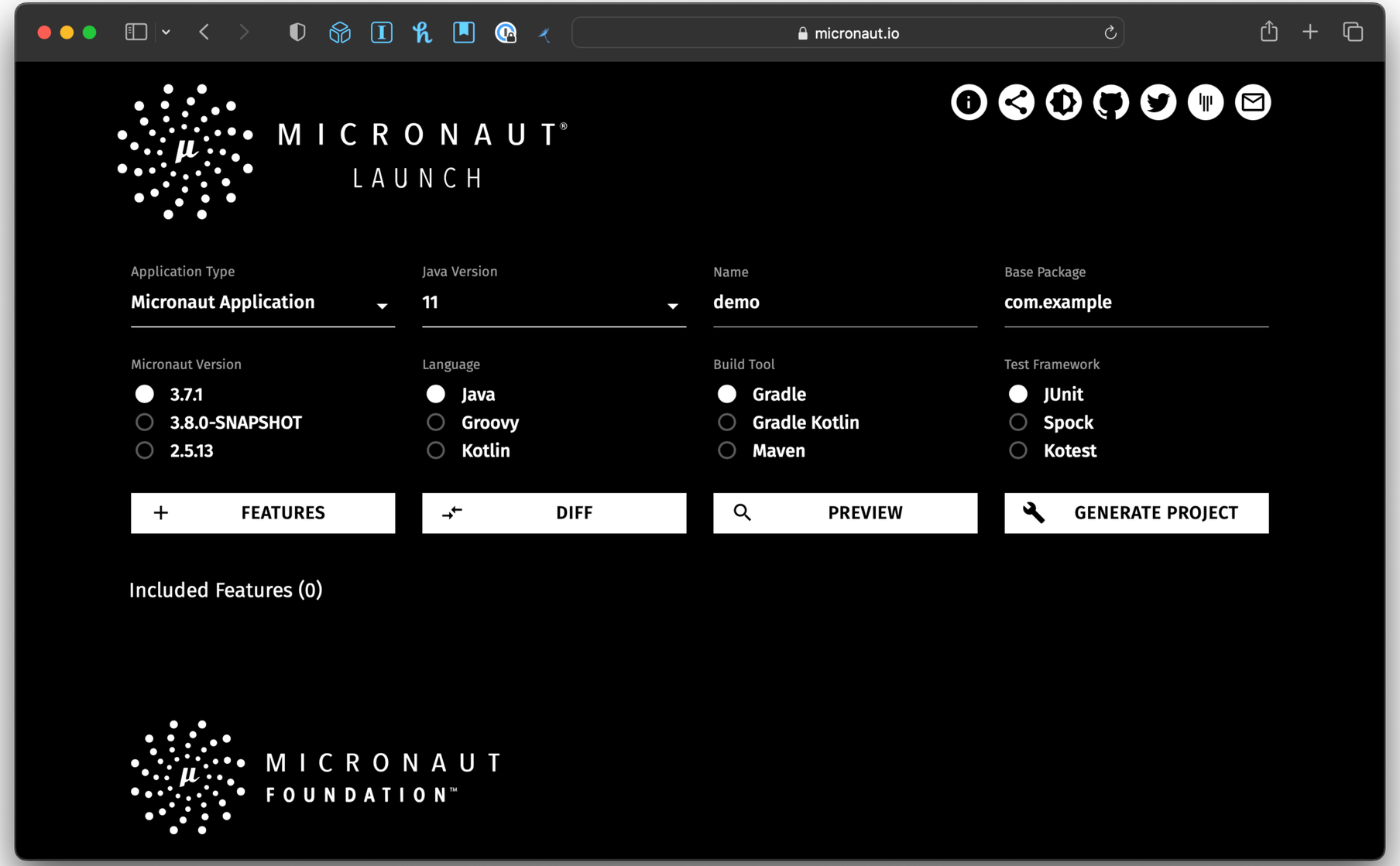


MICRONAUT®

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

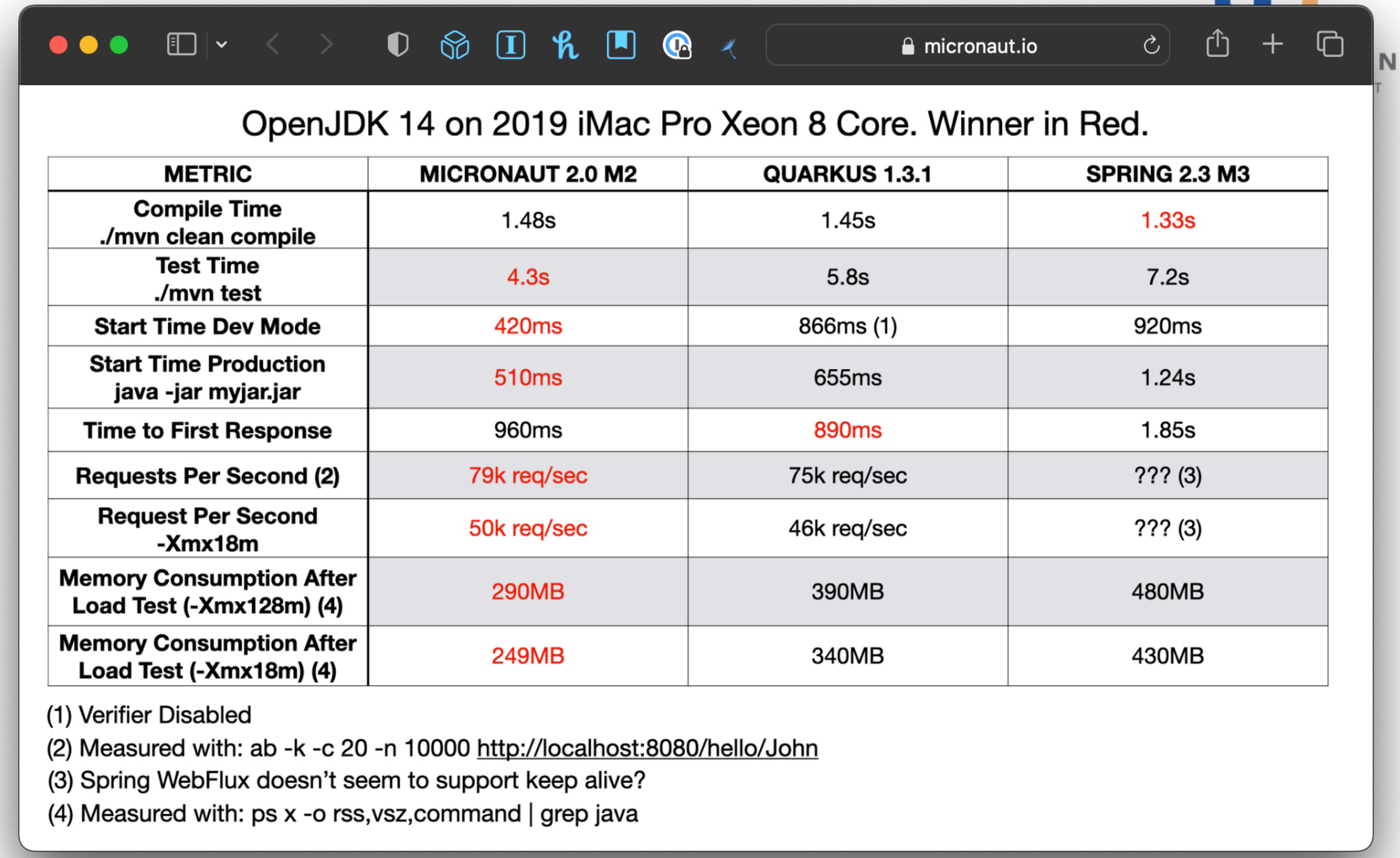
What is Micronaut?

- 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



https://micronaut.io/launch

```java
package com.example;

import io.micronaut.http.HttpResponse;
import io.micronaut.http.MediaType;
import io.micronaut.http.annotation.Controller;
import io.micronaut.http.annotation.Get;
import io.micronaut.http.annotation.Produces;

@Controller
public class HelloController {

  @Get("/{name}")
  @Produces(MediaType.TEXT_PLAIN)
  HttpResponse<String> greet(String name) {
    return HttpResponse.ok( body: "Hello, " + name);
  }

}

```



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

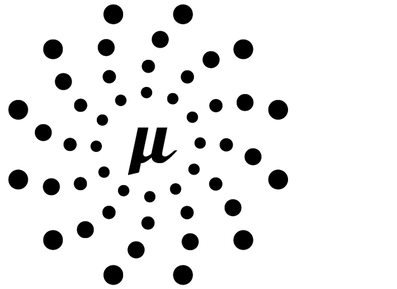
METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (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



Getting started

- 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

Add dependency:

Gradle 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**

OrderPlacedProducer.java

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

Specify Topic with **@Topic**


```
@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**

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**

OrderPlacedProducer.java

```java
package com.example;

import io.micronaut.configuration.kafka.annotation.KafkaClient;
import io.micronaut.configuration.kafka.annotation.Topic;

2 usages
@KafkaClient
public interface OrderPlacedProducer {

    1 usage
    @Topic("order-placed")
    void sendProduct(String product);

}
```

Inject and use Producer


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

Consumer: **@KafkaListener**

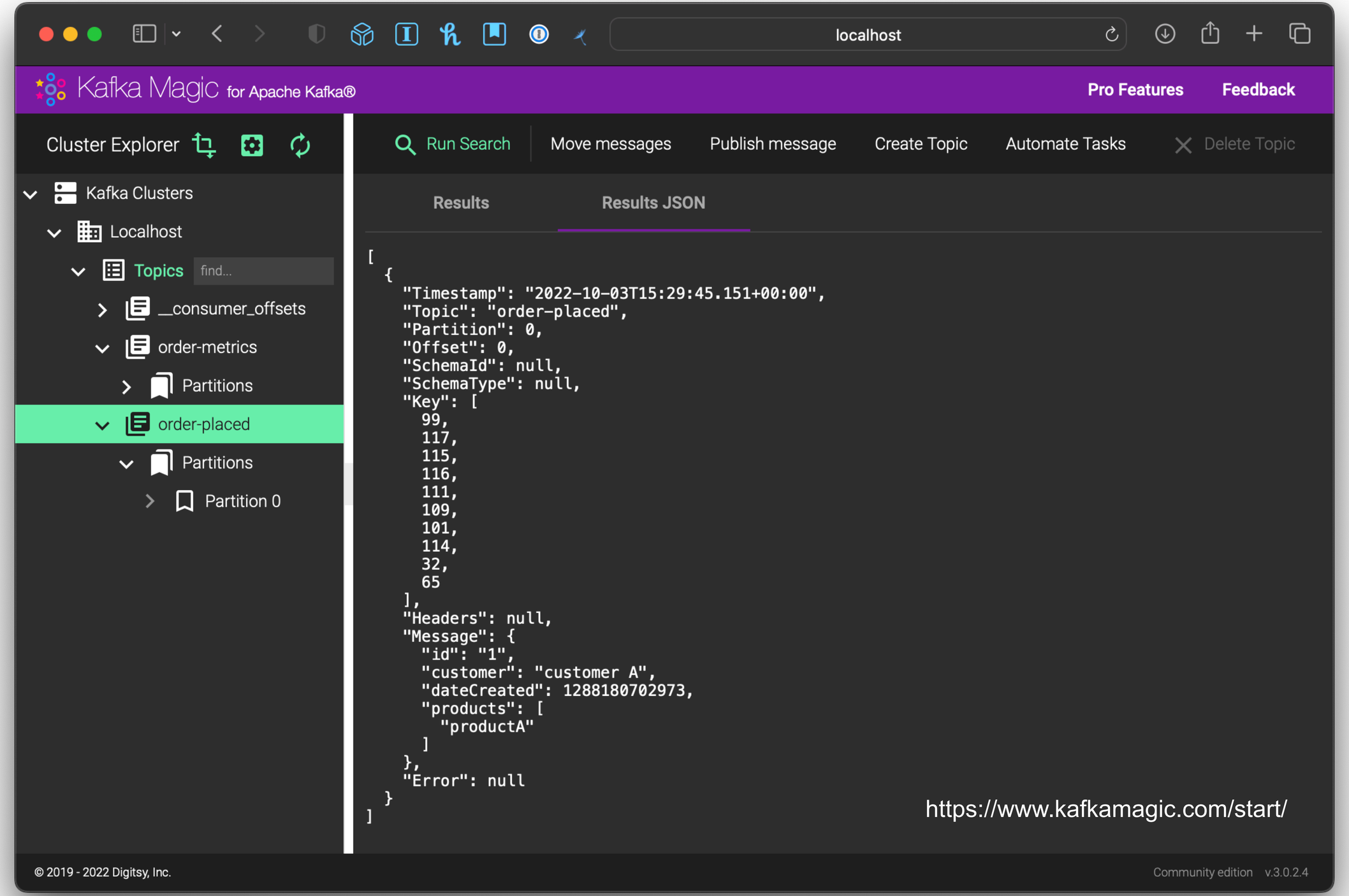
OrderPlacedConsumer.java

```java
package com.example;

import io.micronaut.configuration.kafka.annotation.KafkaListener;
import io.micronaut.configuration.kafka.annotation.Topic;

@KafkaListener
public class OrderPlacedConsumer {

  @Topic("order-placed")
  void receiveOrder(String product) {
    System.out.println("Received order for product: " + product);

    //TODO: Handle order

  }

}
```

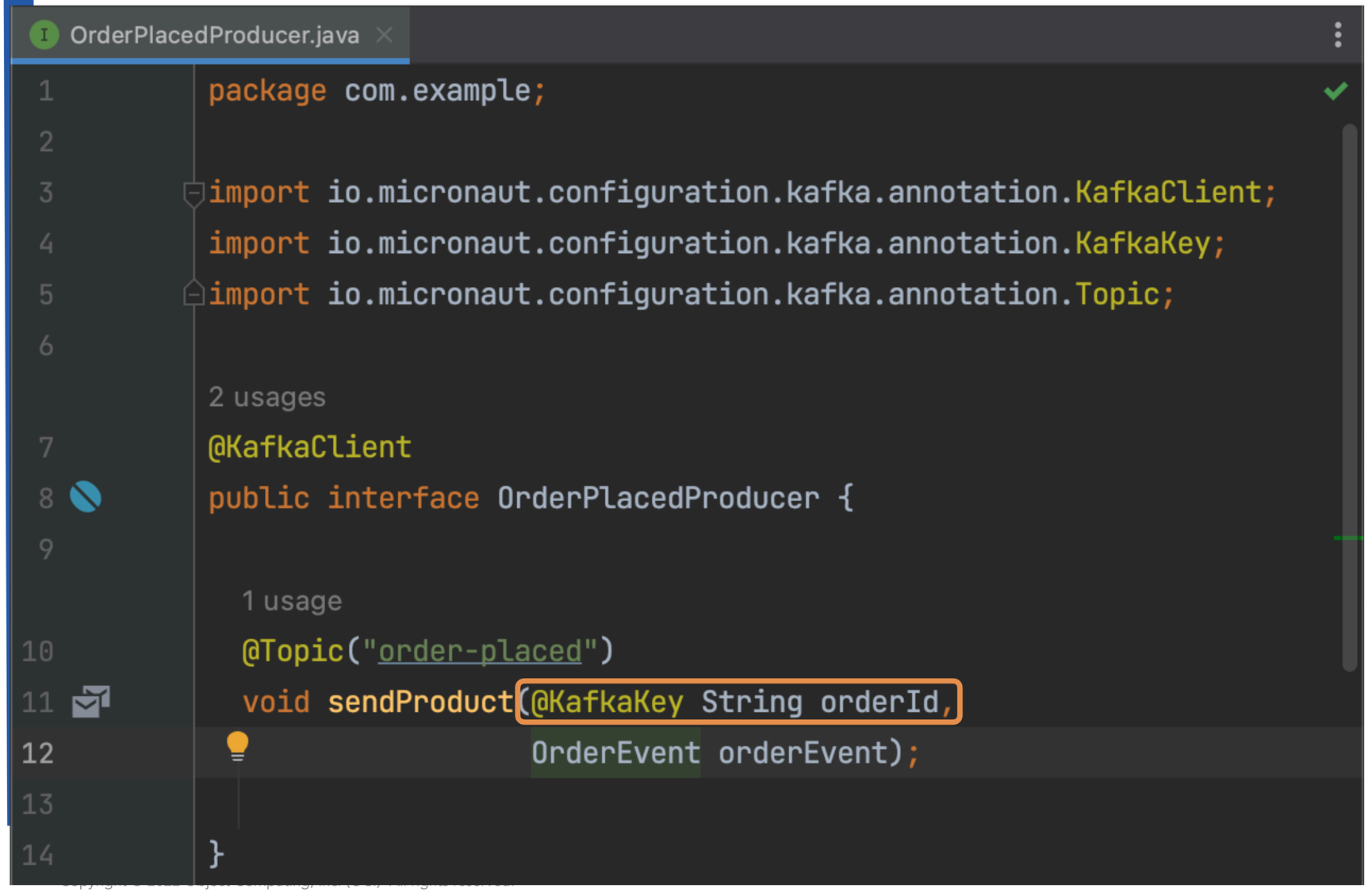
Use POJO as Event Payload

OrderEvent.java

```
1   package com.example;                                                    ⚠ 3
2
3   import ...
7
    1 usage
8   @Introspected
9   public class OrderEvent {
      2 usages
10      private final Long id;
        2 usages
11      private final String customer;
        2 usages
12      private final Date dateCreated;
        2 usages
13      private final List<String> products;
14
15      public OrderEvent(Long id, String customer, Date dateCreated, List<String> products) {
16          this.id = id;
17          this.customer = customer;
```


localhost
Kafka Magic for Apache Kafka®
Pro Features
Feedback
Cluster Explorer
Run Search
Move messages
Publish message
Create Topic
Automate Tasks
Delete Topic
Kafka Clusters
Localhost
Topics
find...
__consumer_offsets
order-metrics
Partitions
order-placed
Partitions
Partition 0
Results
Results JSON
[
{
"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
}
]
https://www.kafkamagic.com/start/
© 2019 - 2022 Digitsy, Inc.
Community edition v.3.0.2.4

@KafkaKey to specify Event key



@KafkaKey to specify Event key

OrderPlacedConsumer.java

```
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
```

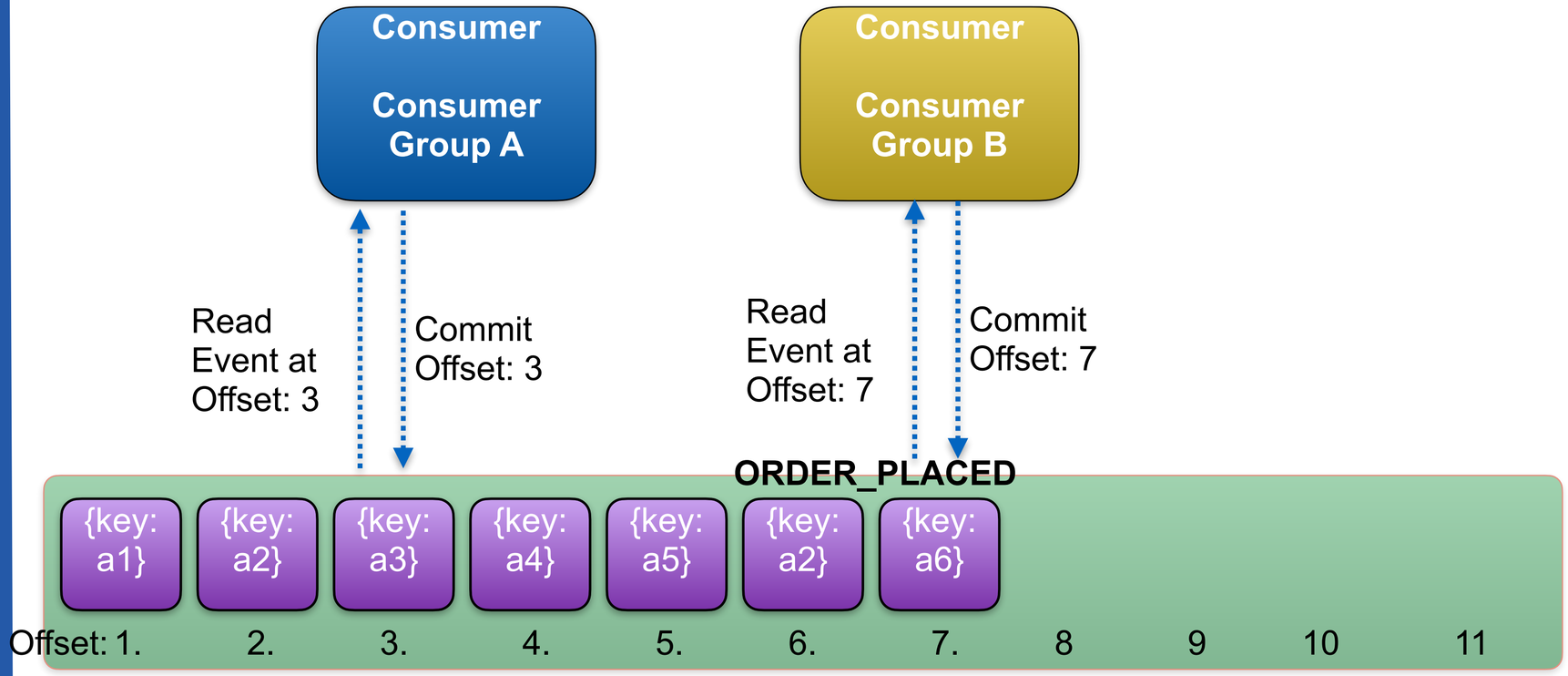
Specify **Consumer Groups**

OrderPlacedConsumer.java

```
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        }
23      }
```

Configure **OffsetStrategy**

Default: Auto-commit



Configure **OffsetStrategy**

Example: Disabled (Manual Acknowledgement

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**

OrderPlacedConsumer.java

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

Exception handling: **ErrorStrategy**

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
```

KafkaListenerExceptionHandler

Reader Mode

```java
@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

CustomKafkaListenerExceptionHandler.java

```
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

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

http:localhost:8080/health

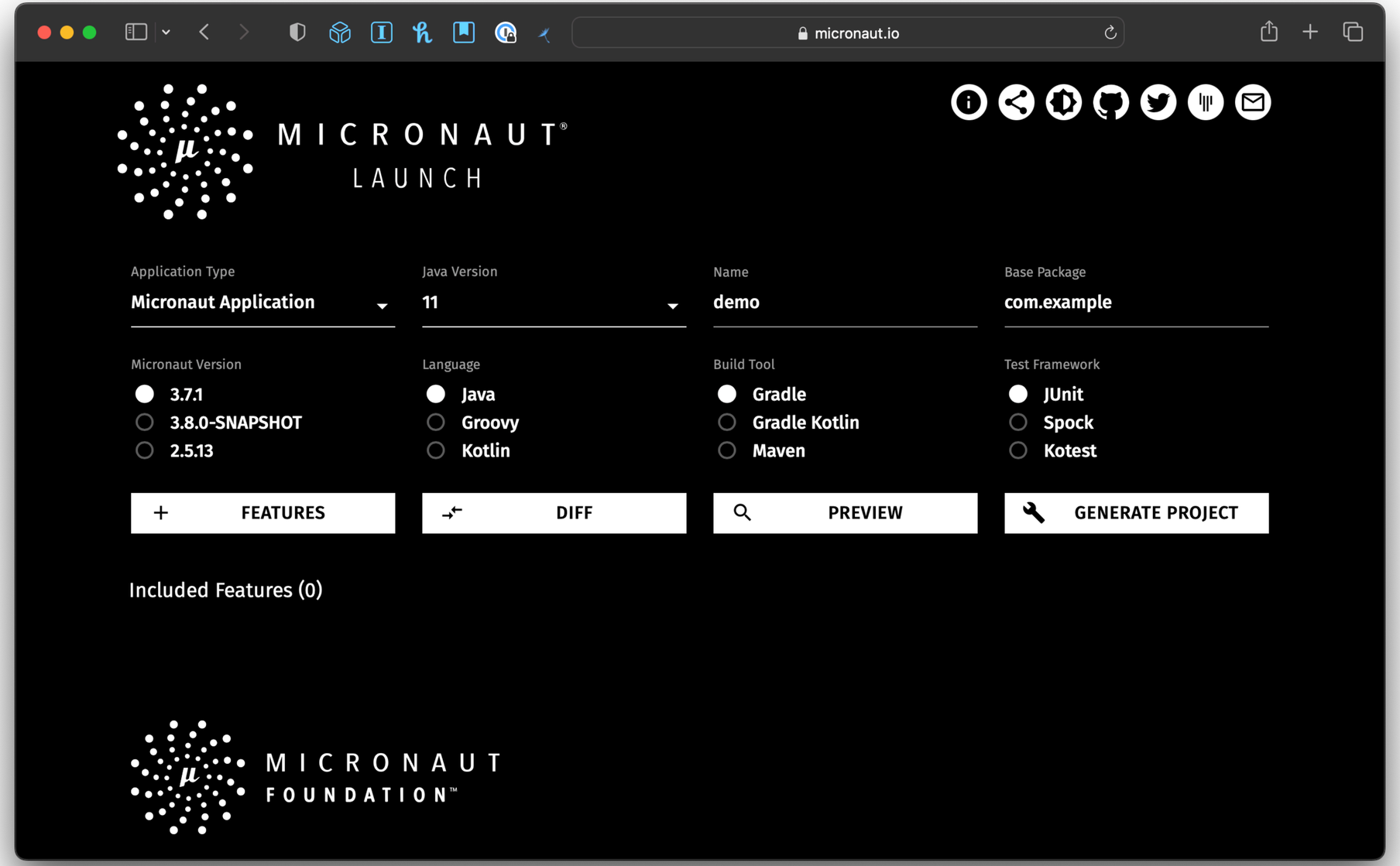
```
{
    "status": "DOWN",
    "details": {
        "kafka": {
            "status": "DOWN",
            "details": {
  “error": "java.util.concurrent.ExecutionException:
org.apache.kafka.common.errors.TimeoutException: Timed out waiting for a node
assignment."
            }
        }
    }
}
```

Health Checks, Metrics, & Tracing

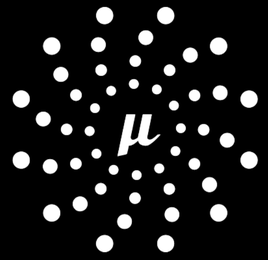
- Micronaut Kafka provides metrics to Micronaut's **metrics** endpoint via Micrometer (if enabled)
  - https://micronaut-projects.github.io/micronaut-micrometer/latest/guide/
- Micronaut Kafka support **distributed tracing** via Micronaut Tracing, using Jaeger or Zipkin
  - https://micronaut-projects.github.io/micronaut-tracing/latest/guide/

Kafka Streams

- 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



https://micronaut.io/launch



MICRONAUT
FOUNDATION™

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

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

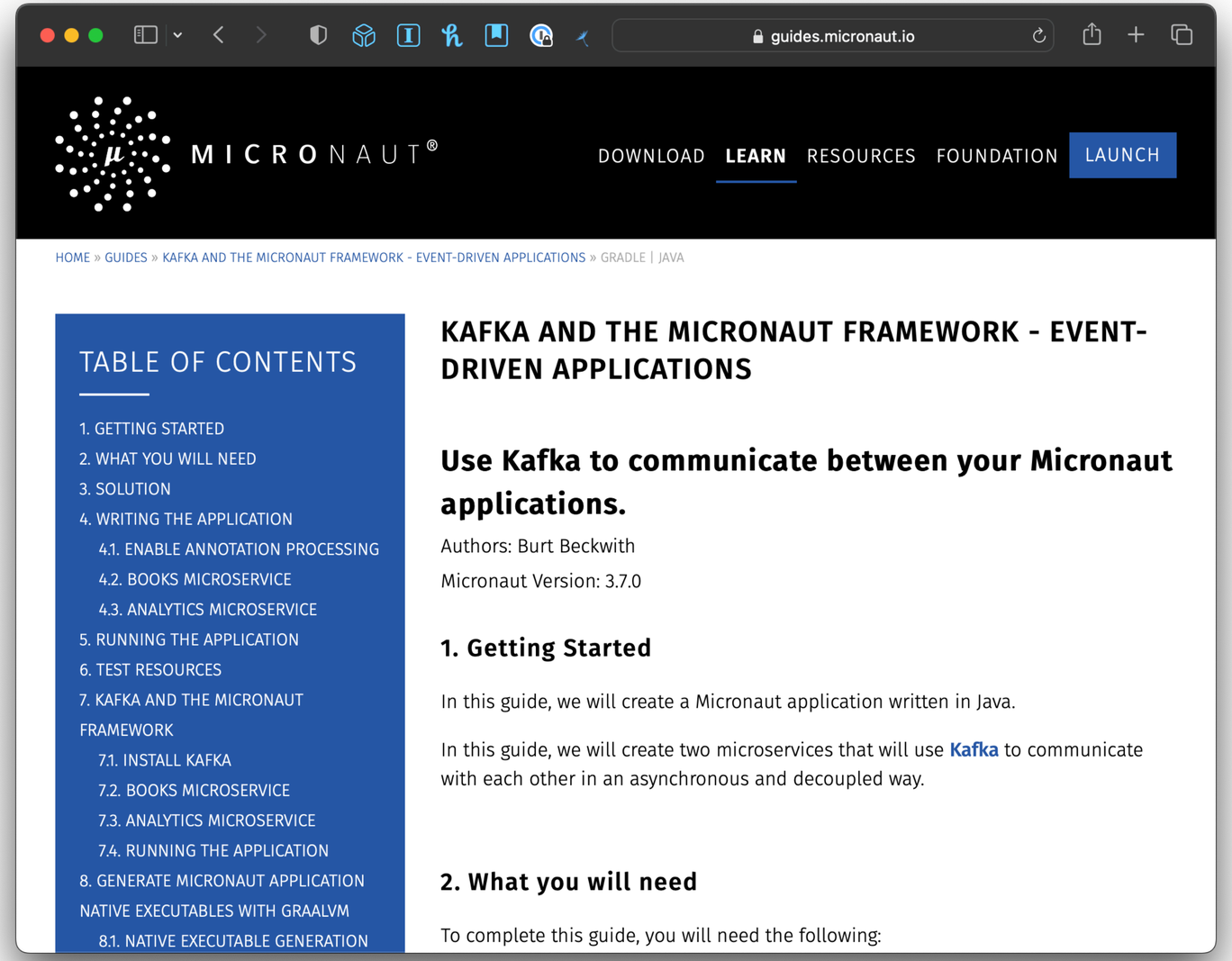
KEN SIPE
Edward Jones, Department Leader – Application and Technology Architecture

VENKAT SUBRAMANIAM
Agile Developer Inc., Founder

https://micronaut.io/foundation

Micronaut Resources

Micronaut Guides



https://guides.micronaut.io/latest/micronaut-kafka-gradle-java.html

Micronaut Resources

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

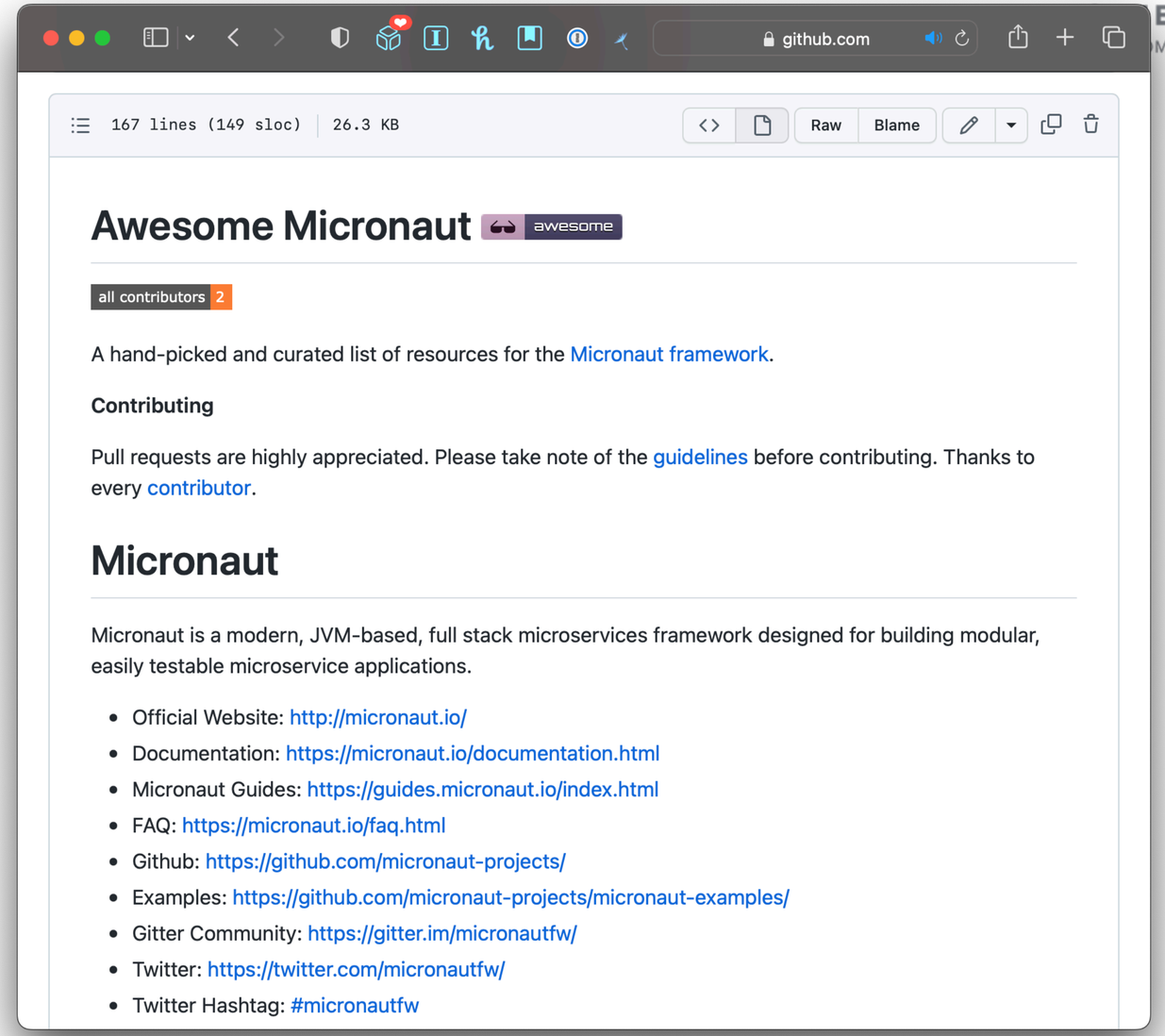
Books:

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

Podcast:

https://micronautpodcast.com/

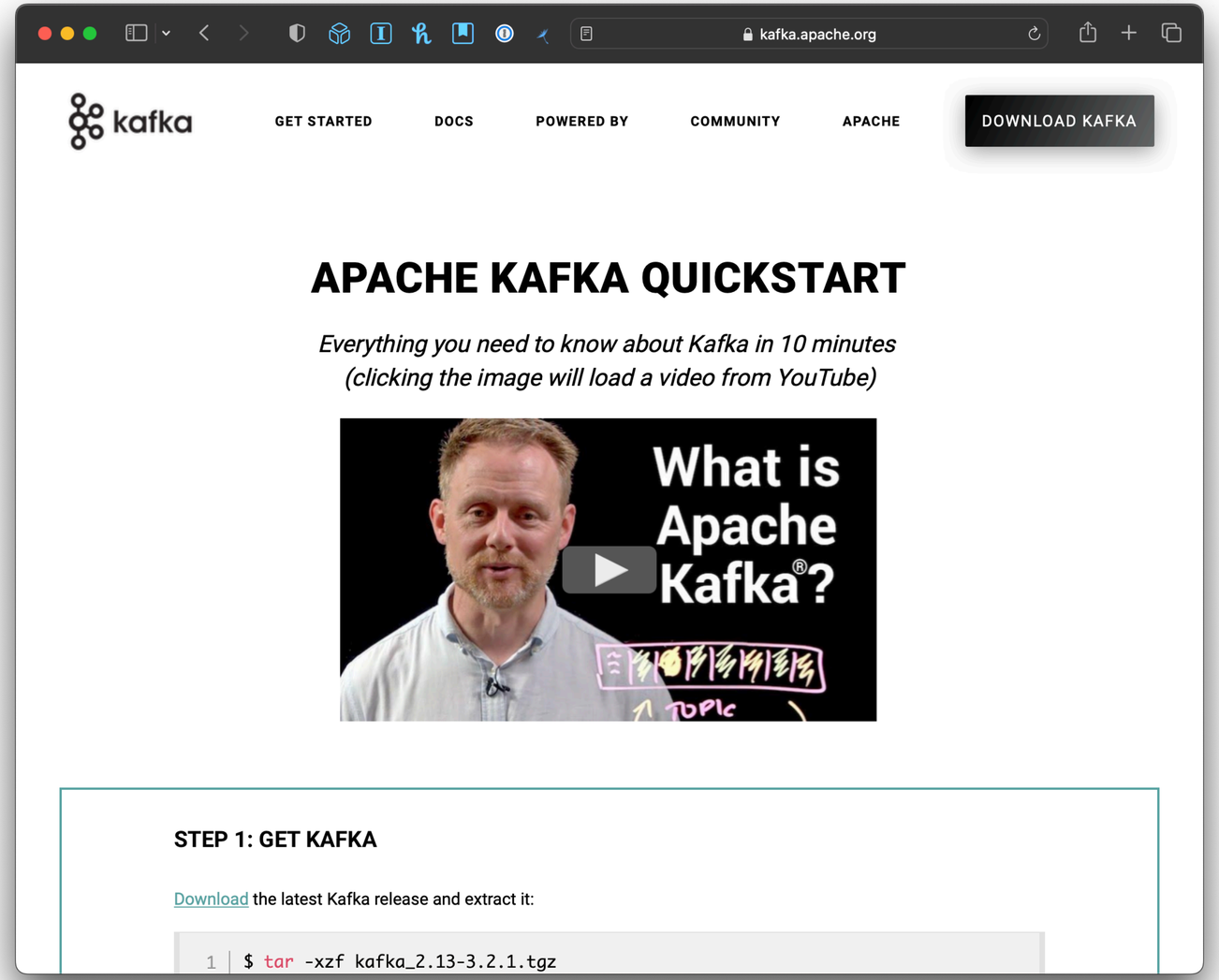
Micronaut Resources



https://github.com/JonasHavers/awesome-micronaut

Kafka/Event-Driven Resources

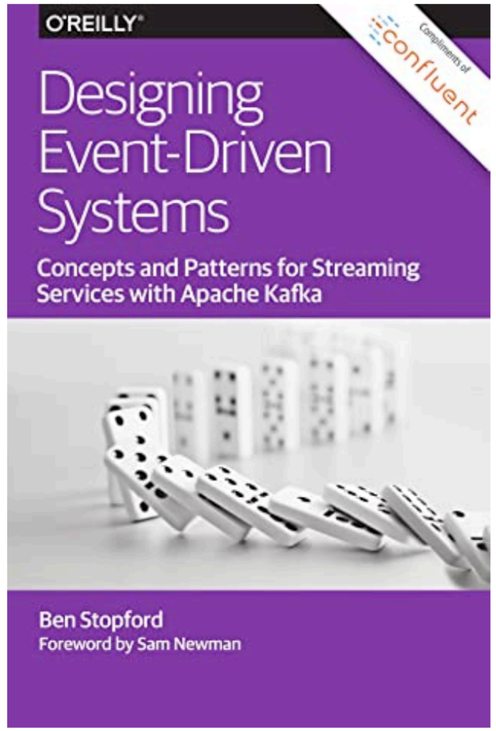
OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT



https://kafka.apache.org/quickstart

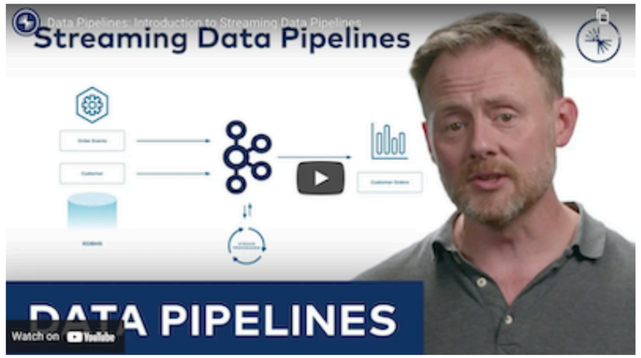
Kafka/Event-Driven Resources

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

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

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)

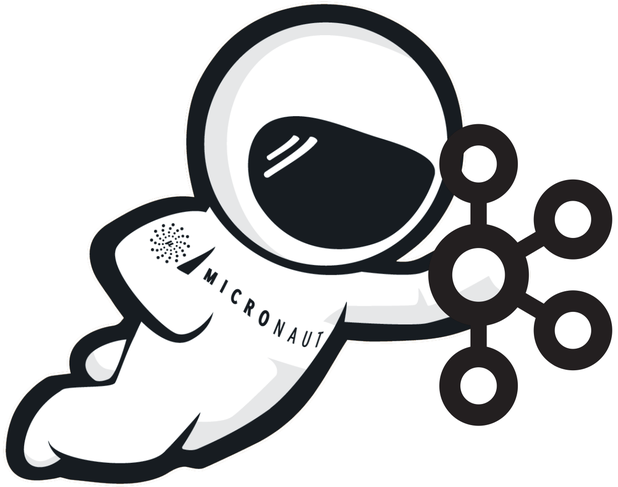
https://micronaut.io/professional-training/event-driven-microservices-with-micronaut/

APACHECON

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

Thank you!

Zachary Klein, Principal Software Engineer, 2GM Team



kafka

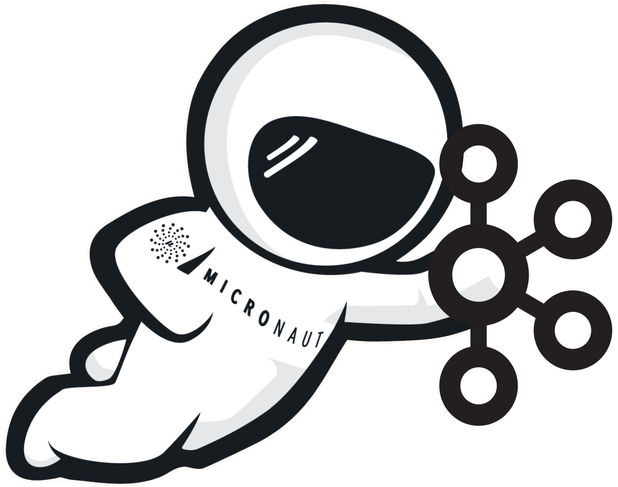
APACHECON

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

Scan with your device.

Retweet for a chance at free T-shirt!



kafka