Event Driven Architecture w/ Apache Kafka and Spring Cloud Stream

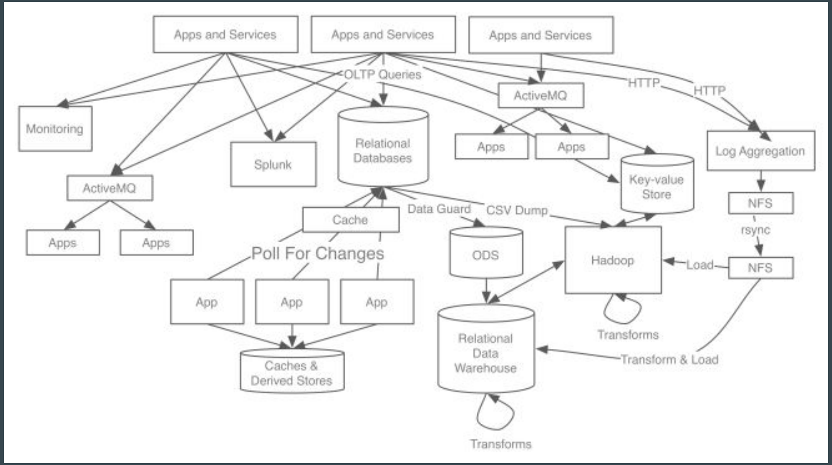
•••

Indy Java User Group Feb 2019

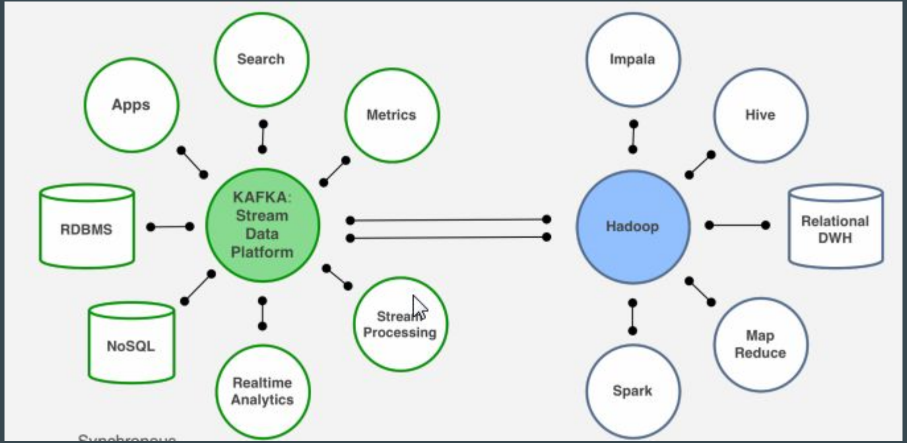
About Me

Dan Gradl
Principal Engineer @ Finish Line (JD Sports)
e: dangradl@gmail.com

Streaming Data Architecture



TO



Key Patterns

Source - Stream data in from some location (file, database, etc.)

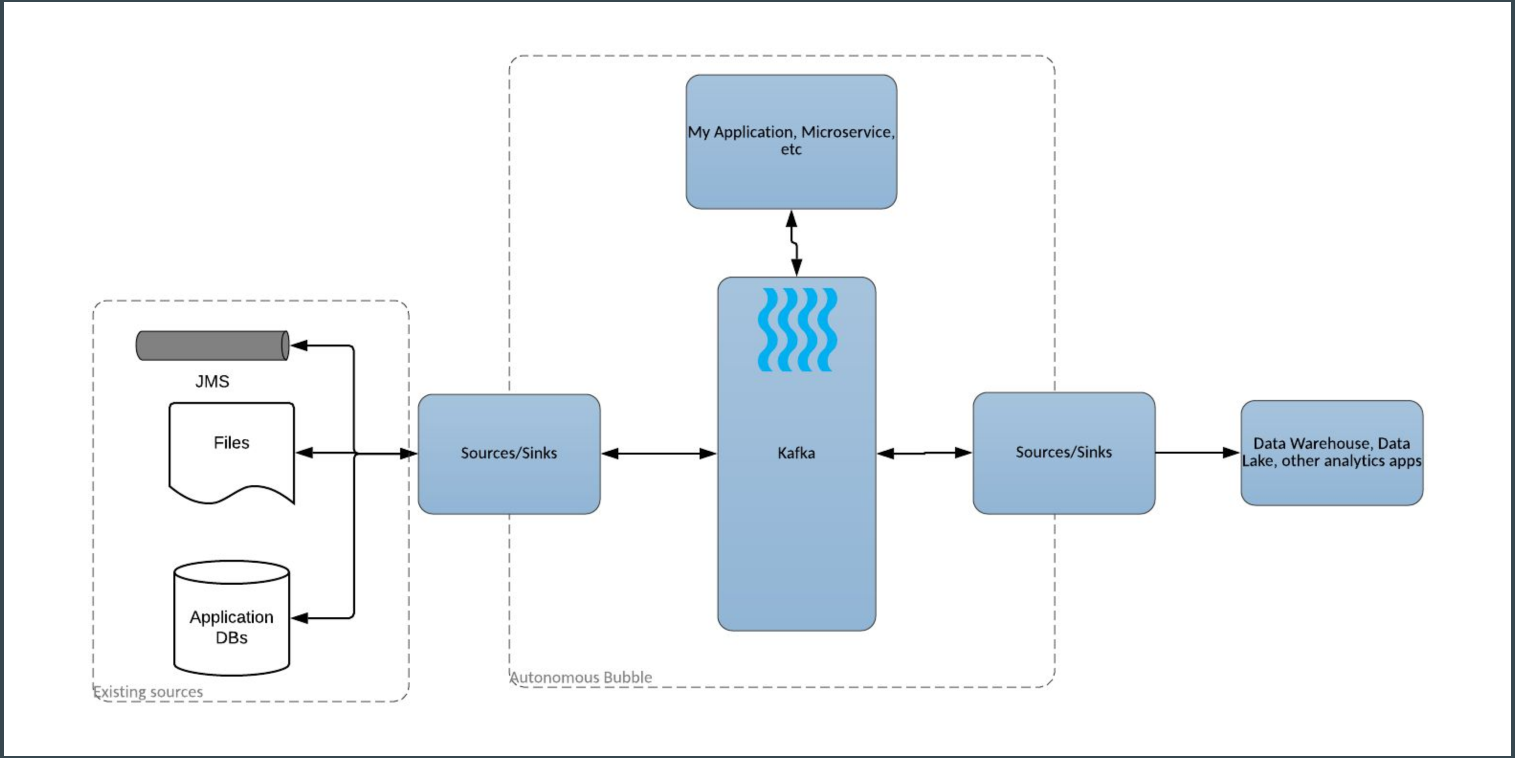
Sink - Stream data out to some location (file, database, etc.)

Processor - Process stream data from one stream and output to another including:

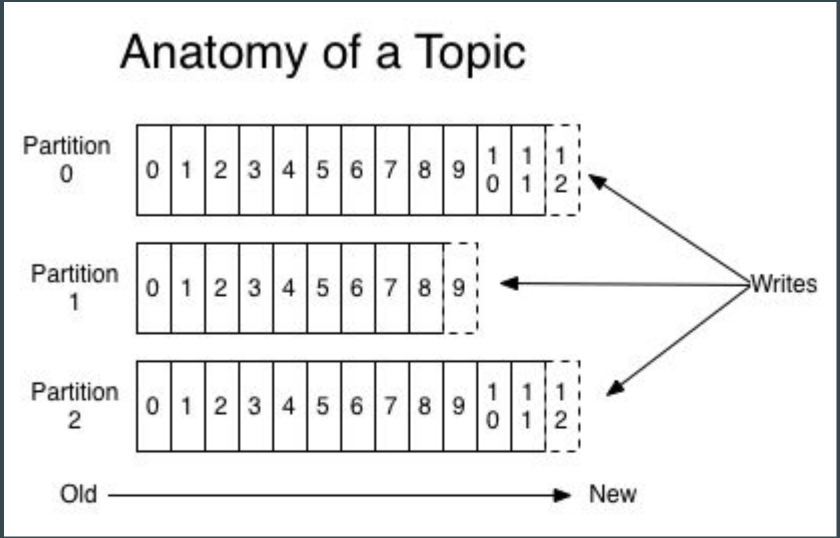
Aggregation, filtering, and transforming of data

Joining two streams and outputting to another

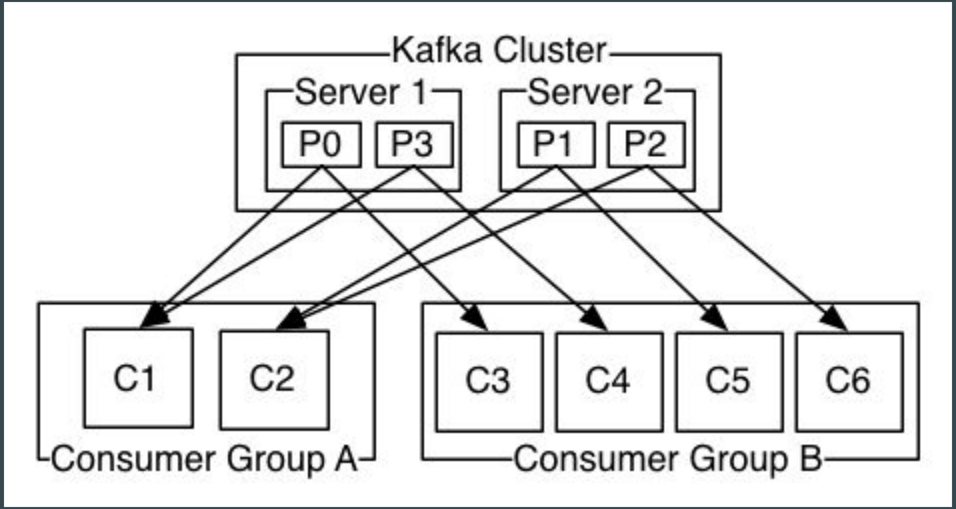
Autonomous Bubble



Kafka Topics



Kafka Consumers



Kafka features that support streaming architecture

- Low latency, high throughput
- Scalability - in particular partitioning enables parallelization of the stream processing for speed
- Flexible topic data retention
  - Size
  - Time
  - Compaction
- Support for different data payloads: e.g. Avro, JSON, Plain Text
- Replication for resilience and high availability
- Consumer controlled offset
- Kafka Connect and Kafka Streams

Concrete Pattern Implementations

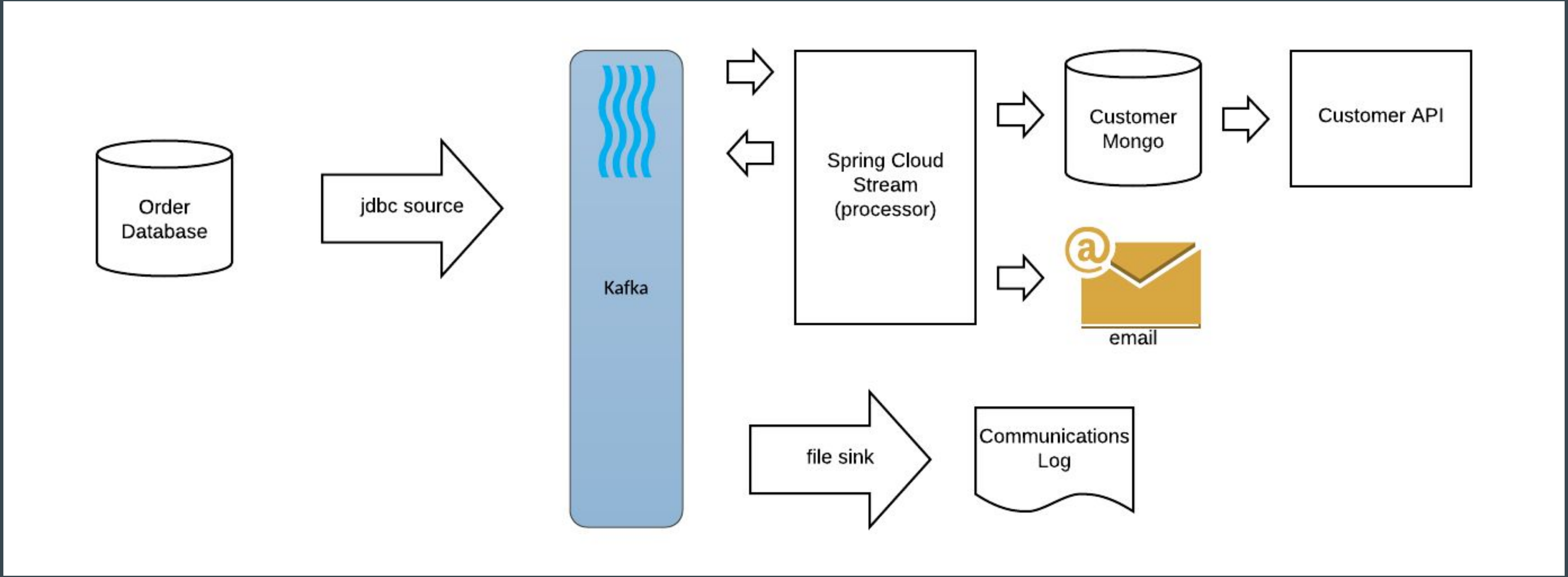
Sources and Sinks

- Kafka Connect and Confluent connectors
- Spring Cloud Stream and associated App Starters
- Open source and commercial tools such as Striim and Streamsets
- Kafka console commands (poor man’s)

Processors

- Kafka Streams
- Spring Cloud Stream

Patterns and Demo Use Case

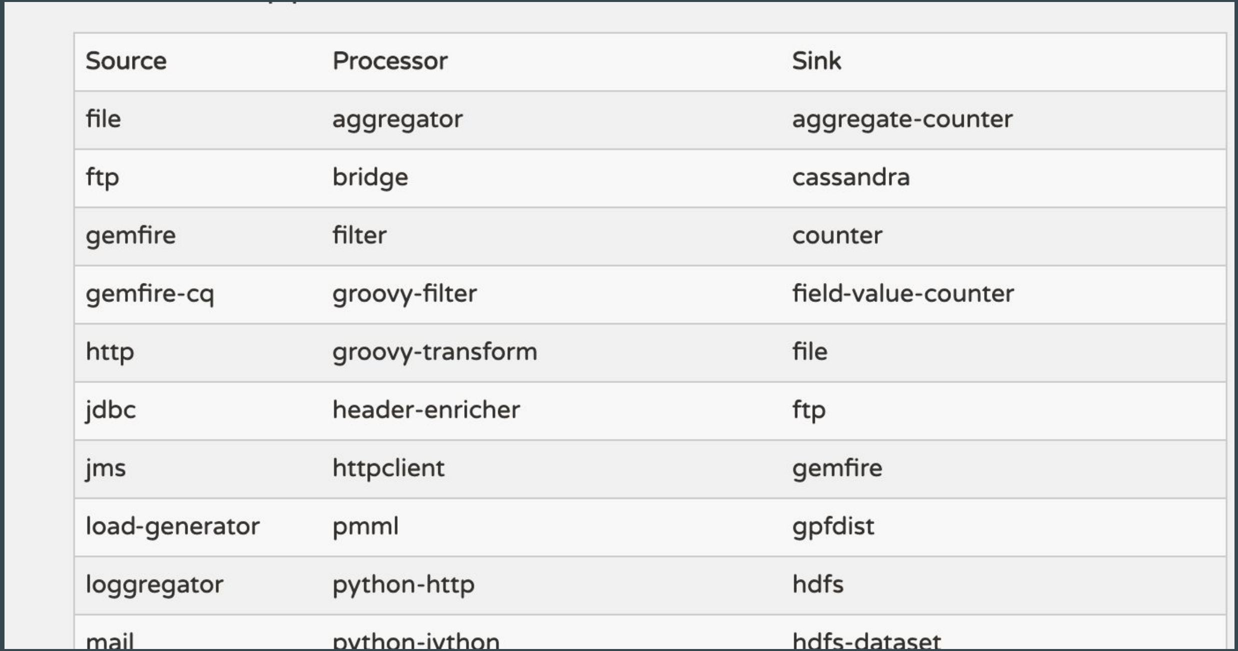


Order Generator and JDBC Source

Created a simple project that simulates a website or other application creating new orders in a postgres database (every 30 seconds).

Spring Boot App Starters

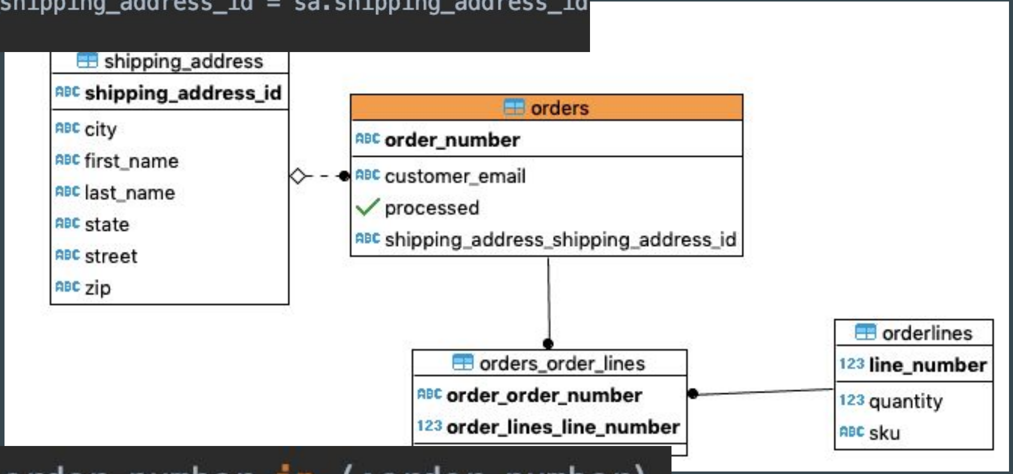
- Prepackaged sources, sinks and processors
- Configurable
- Simple to use



Source	Processor	Sink
file	aggregator	aggregate-counter
ftp	bridge	cassandra
gemfire	filter	counter
gemfire-cq	groovy-filter	field-value-counter
http	groovy-transform	file
jdbc	header-enricher	ftp
jms	httpclient	gemfire
load-generator	pmml	gpfdist
loggregator	python-http	hdfs
mail	python-jython	hdfs-dataset

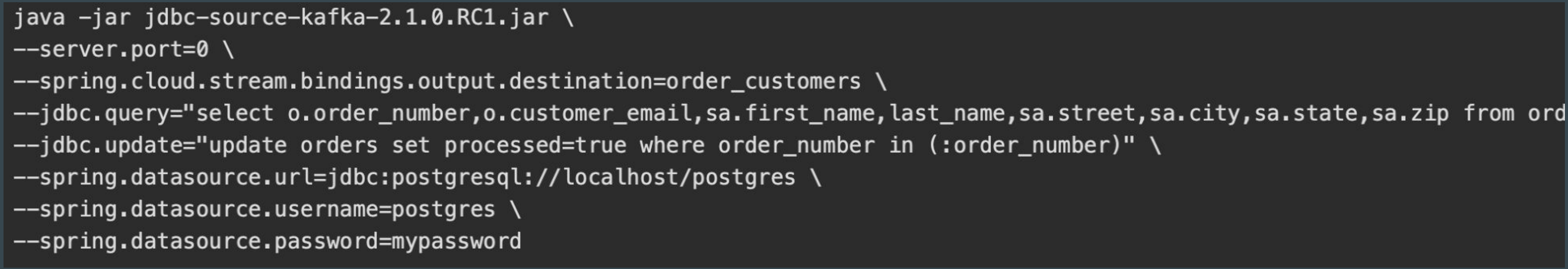
Orders - Polling

```
select o.order_number,
       o.customer_email,
       sa.first_name,
       last_name,
       sa.street,
       sa.city,
       sa.state,
       sa.zip
from orders o
       inner join shipping_address sa on o.shipping_address_shipping_address_id = sa.shipping_address_id
where o.processed = false
```



```
update orders set processed=true where order_number in (:order_number)
```

Running the App Starter



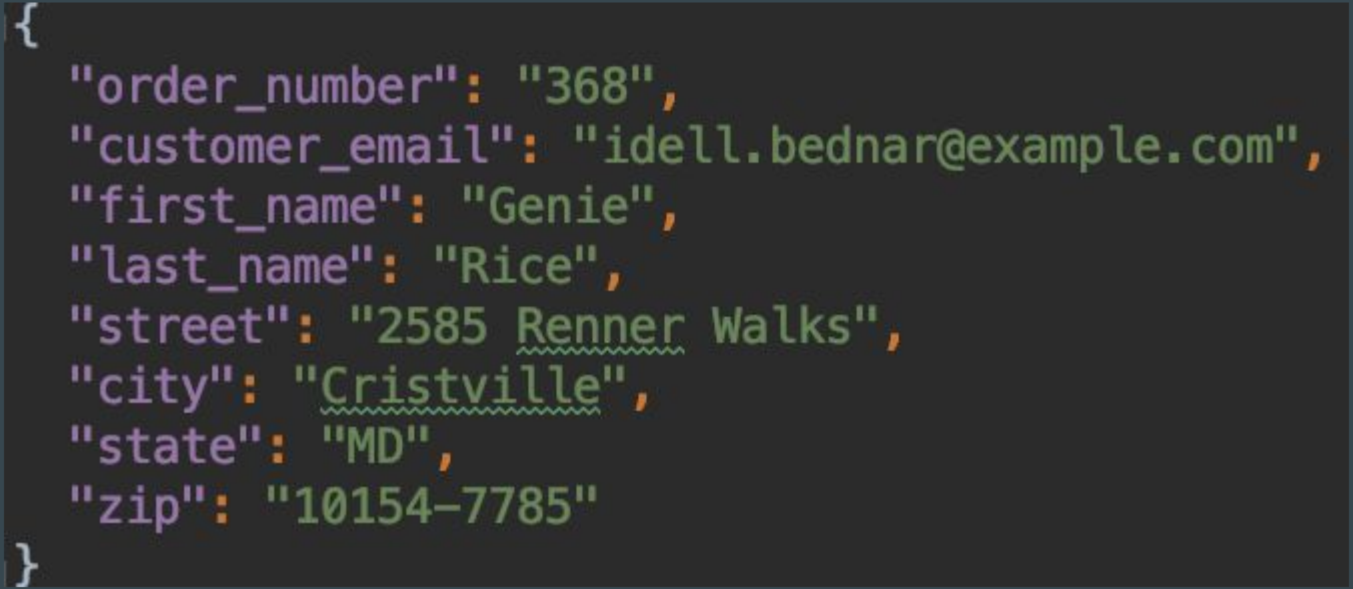
```
java -jar jdbc-source-kafka-2.1.0.RC1.jar \
--server.port=0 \
--spring.cloud.stream.bindings.output.destination=order_customers \
--jdbc.query="select o.order_number,o.customer_email,sa.first_name,last_name,sa.street,sa.city,sa.state,sa.zip from ord
--jdbc.update="update orders set processed=true where order_number in (:order_number)" \
--spring.datasource.url=jdbc:postgresql://localhost/postgres \
--spring.datasource.username=postgres \
--spring.datasource.password=mypassword
```

Database connection info

Query to retrieve data from DB and another to mark what has been polled

Destination binding for Kafka

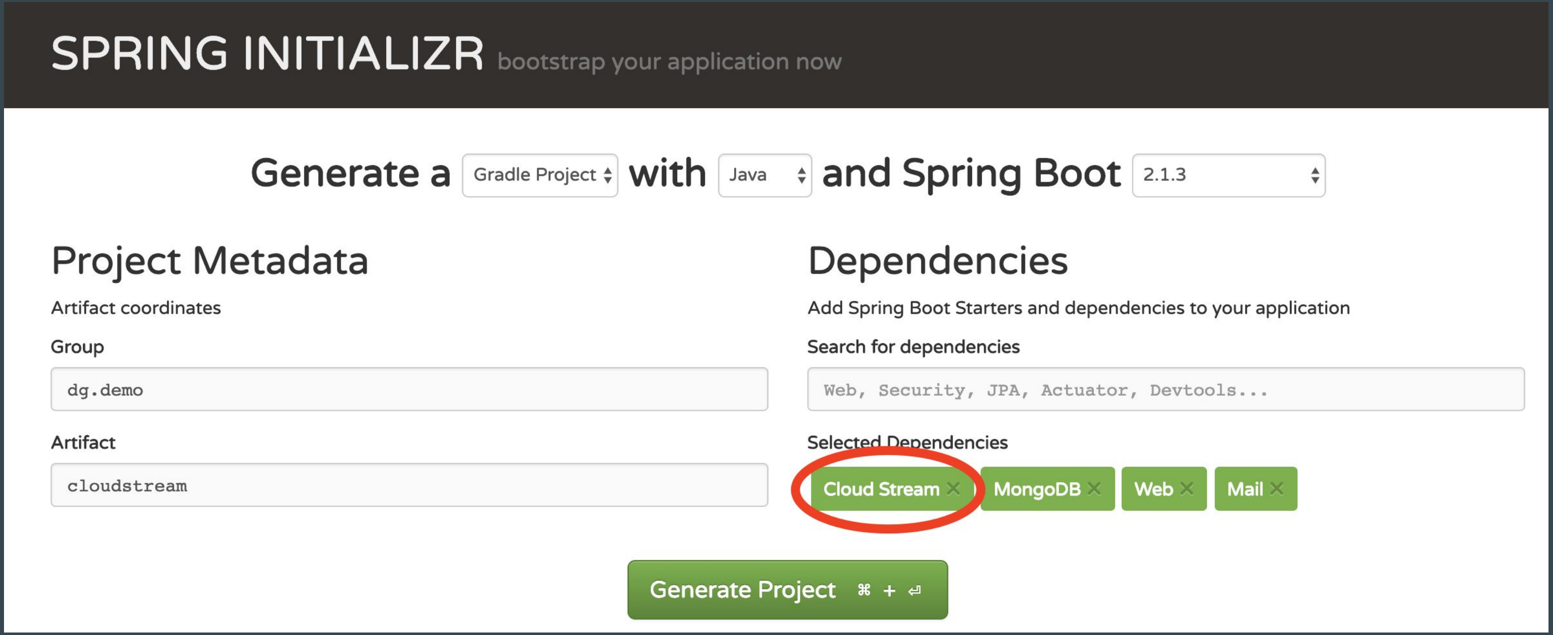
JDBC Source Output



```
{
  "order_number": "368",
  "customer_email": "idell.bednar@example.com",
  "first_name": "Genie",
  "last_name": "Rice",
  "street": "2585 Renner Walks",
  "city": "Cristville",
  "state": "MD",
  "zip": "10154-7785"
}
```

Initialize Your Project

https://start.spring.io/



SPRING INITIALIZR bootstrap your application now

Generate a Gradle Project with Java and Spring Boot 2.1.3

Project Metadata

Artifact coordinates

Group

dg.demo

Artifact

cloudstream

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

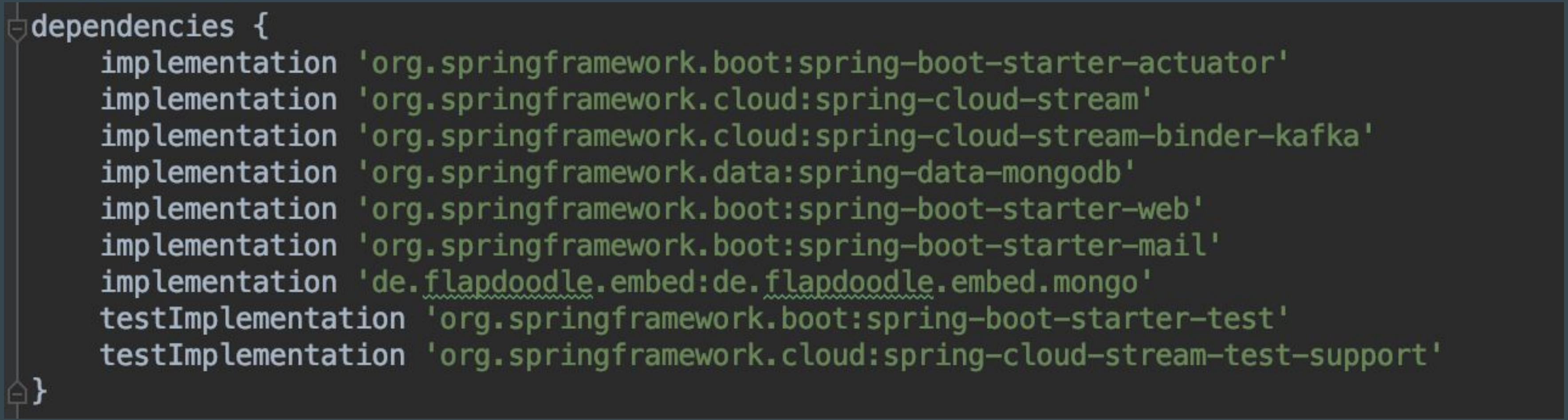
Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Cloud Stream ✕ MongoDB ✕ Web ✕ Mail ✕

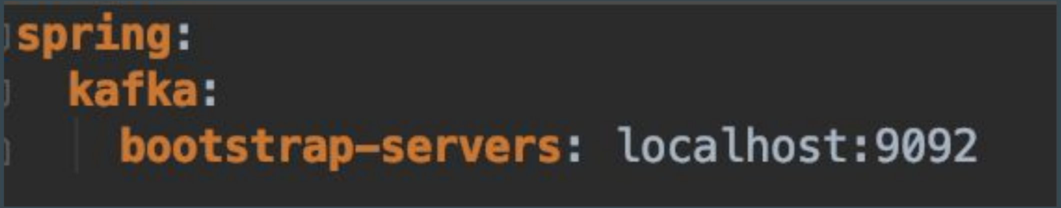
Additional Setup

Add the kafka binder to the build.gradle



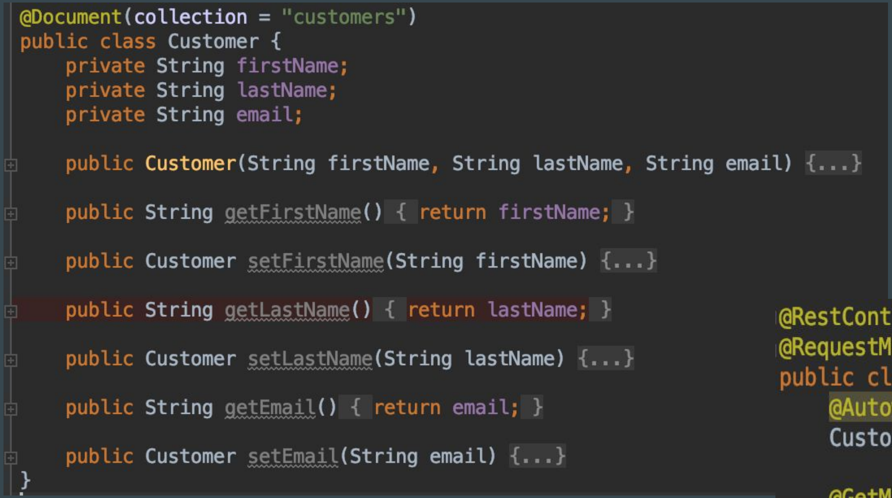
```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springframework.cloud:spring-cloud-stream'
    implementation 'org.springframework.cloud:spring-cloud-stream-binder-kafka'
    implementation 'org.springframework.data:spring-data-mongodb'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-mail'
    implementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.cloud:spring-cloud-stream-test-support'
}
```

Point application.yml to our local (docker) kafka

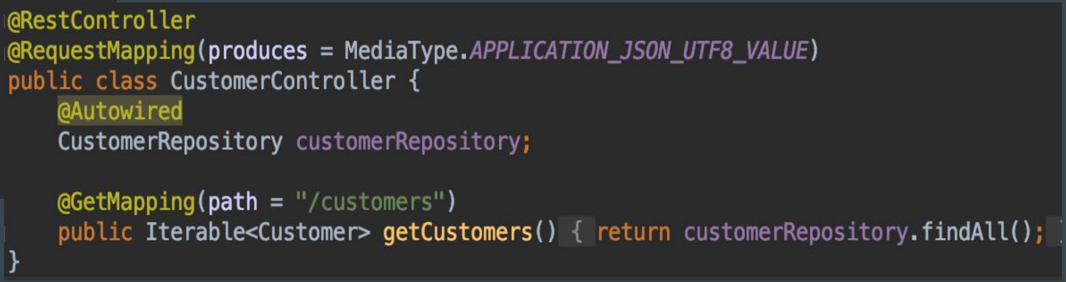


```
spring:
  kafka:
    bootstrap-servers: localhost:9092
```

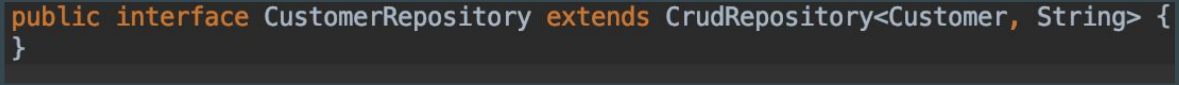
Customer Repository and API



```java
@Document(collection = "customers")
public class Customer {
    private String firstName;
    private String lastName;
    private String email;

    public Customer(String firstName, String lastName, String email) {...}

    public String getFirstName() { return firstName; }

    public Customer setFirstName(String firstName) {...}

    public String getLastName() { return lastName; }

    public Customer setLastName(String lastName) {...}

    public String getEmail() { return email; }

    public Customer setEmail(String email) {...}
}
```

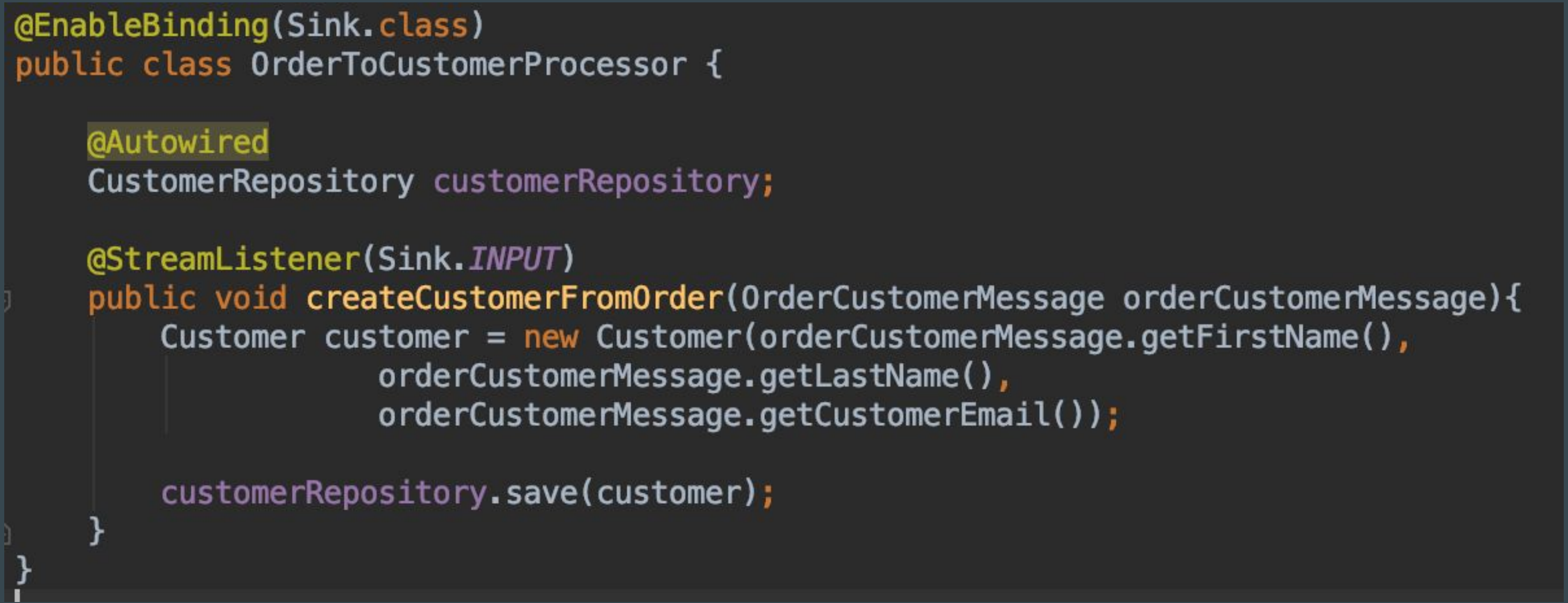


```java
@RestController
@RequestMapping(produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
public class CustomerController {
    @Autowired
    CustomerRepository customerRepository;

    @GetMapping(path = "/customers")
    public Iterable<Customer> getCustomers() { return customerRepository.findAll();
}
```



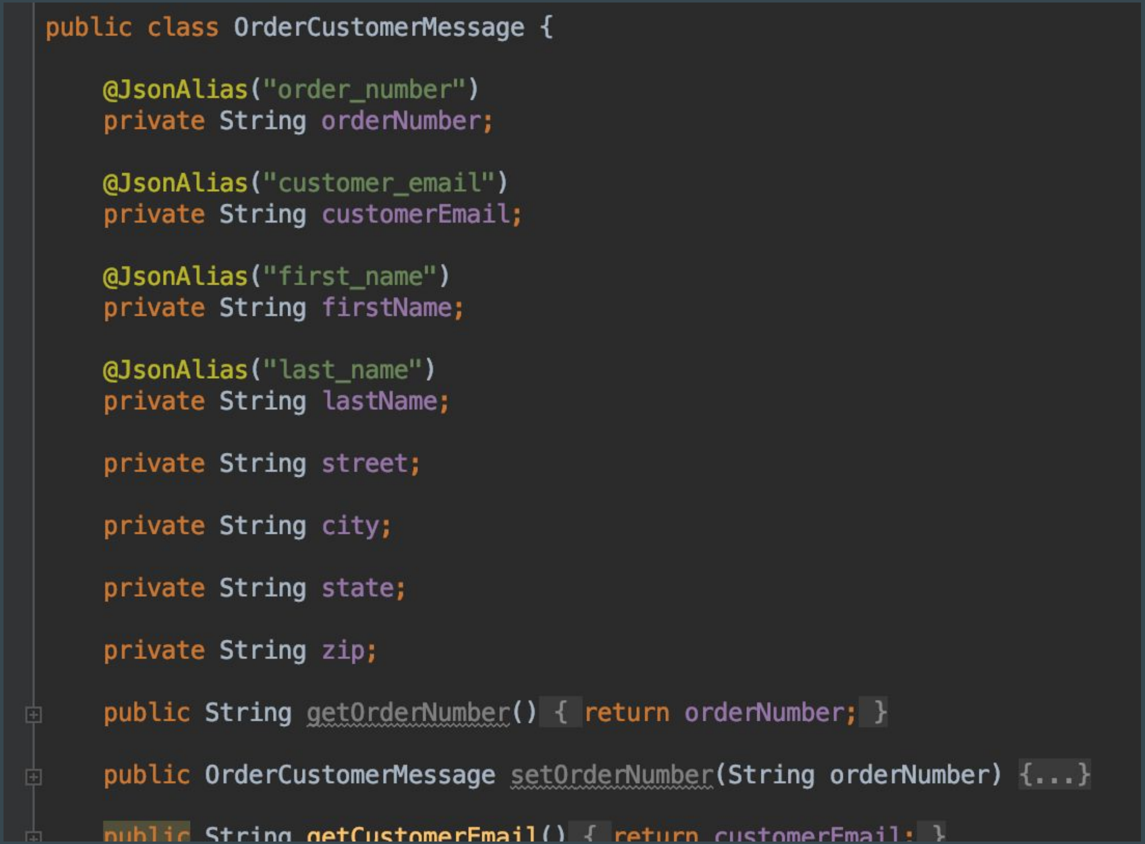
```java
public interface CustomerRepository extends CrudRepository<Customer, String> {
}
```

Our First Stream - Sink



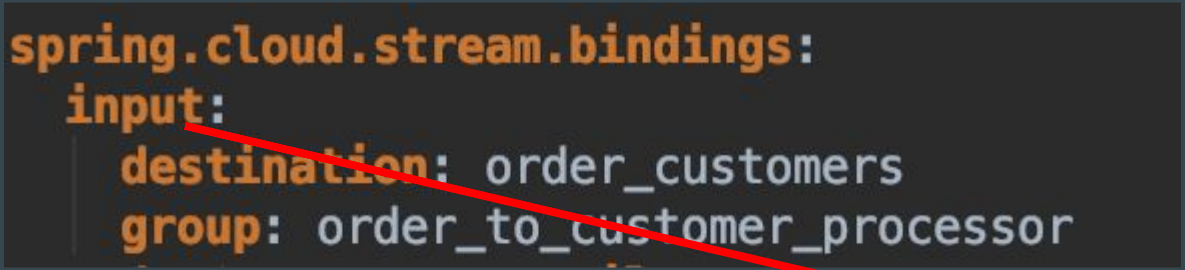
```java
@EnableBinding(Sink.class)
public class OrderToCustomerProcessor {

    @Autowired
    CustomerRepository customerRepository;

    @StreamListener(Sink.INPUT)
    public void createCustomerFromOrder(OrderCustomerMessage orderCustomerMessage){
        Customer customer = new Customer(orderCustomerMessage.getFirstName(),
                    orderCustomerMessage.getLastName(),
                    orderCustomerMessage.getCustomerEmail());

        customerRepository.save(customer);
    }
}
```

OrderCustomerMessage

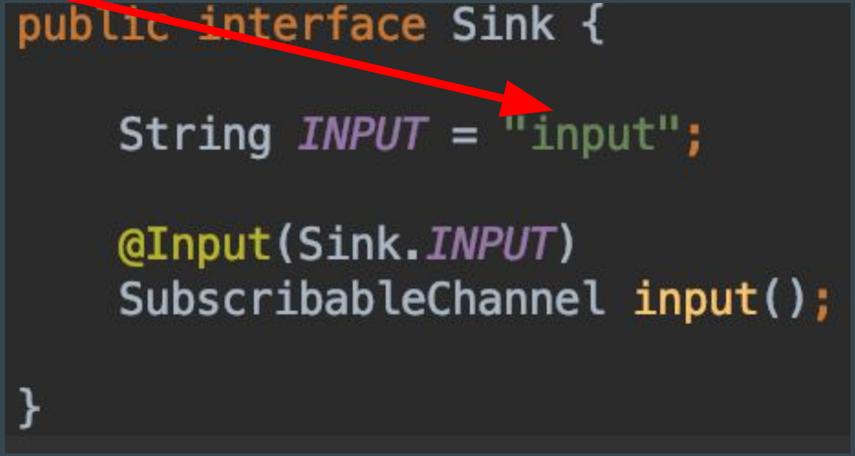


```java
public class OrderCustomerMessage {

    @JsonAlias("order_number")
    private String orderNumber;

    @JsonAlias("customer_email")
    private String customerEmail;

    @JsonAlias("first_name")
    private String firstName;

    @JsonAlias("last_name")
    private String lastName;

    private String street;

    private String city;

    private String state;

    private String zip;

    public String getOrderNumber() { return orderNumber; }

    public OrderCustomerMessage setOrderNumber(String orderNumber) {...}

    public String getCustomerEmail() { return customerEmail; }
```

Sink Binding

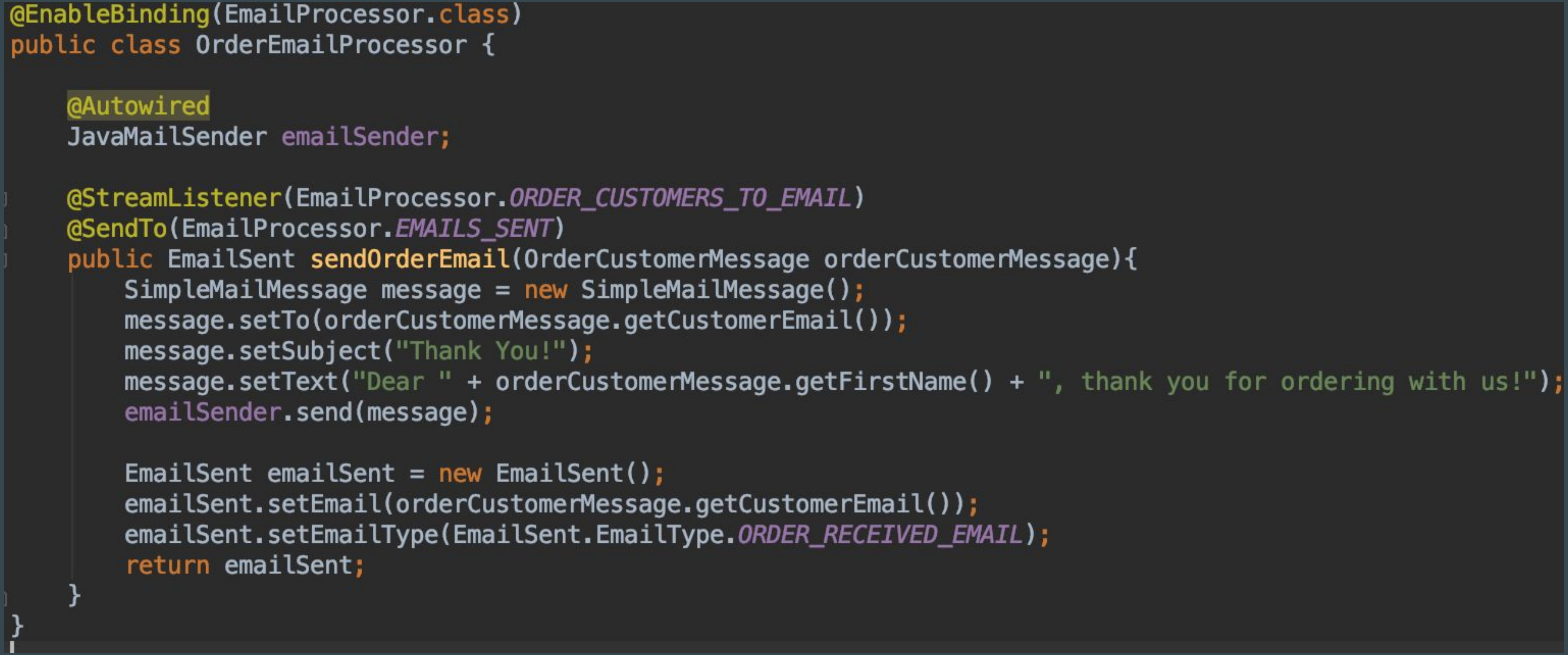


```
spring.cloud.stream.bindings:
  input:
    destination: order_customers
    group: order_to_customer_processor
```



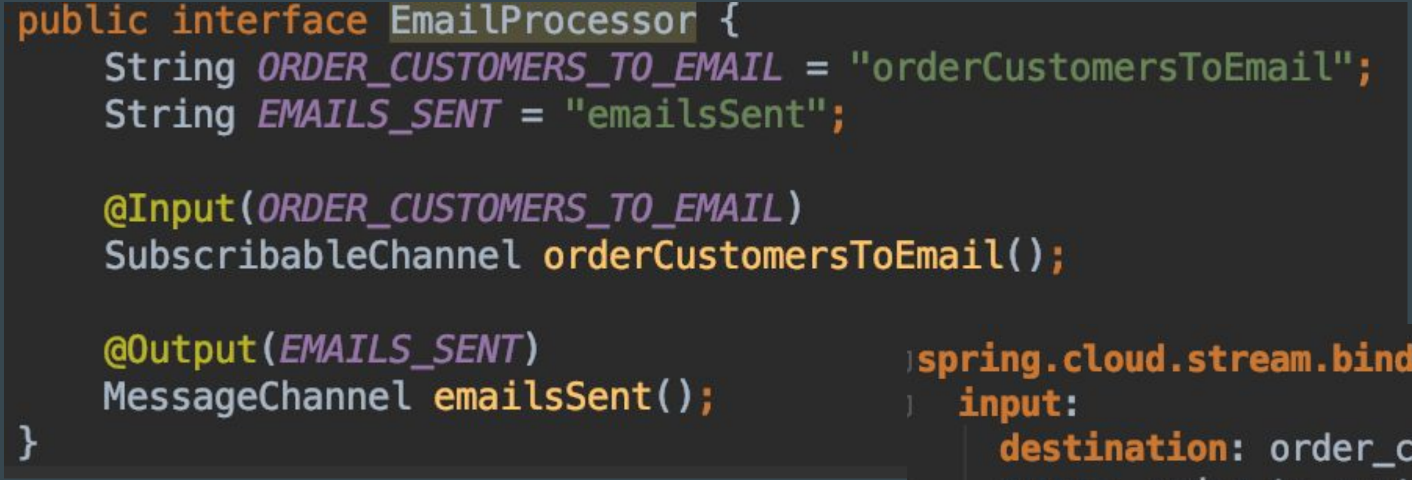
```
public interface Sink {

    String INPUT = "input";

    @Input(Sink.INPUT)
    SubscribableChannel input();

}
```

Sink and Source - a Processor

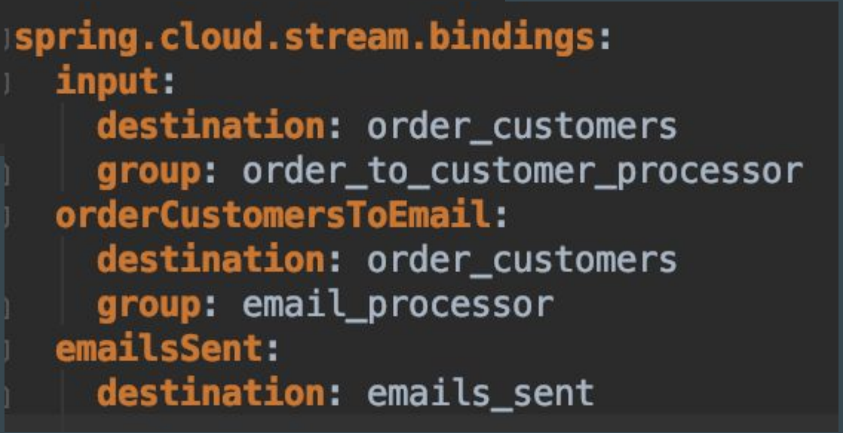


```java
@EnableBinding(EmailProcessor.class)
public class OrderEmailProcessor {

    @Autowired
    JavaMailSender emailSender;

    @StreamListener(EmailProcessor.ORDER_CUSTOMERS_TO_EMAIL)
    @SendTo(EmailProcessor.EMAILS_SENT)
    public EmailSent sendOrderEmail(OrderCustomerMessage orderCustomerMessage){
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(orderCustomerMessage.getCustomerEmail());
        message.setSubject("Thank You!");
        message.setText("Dear " + orderCustomerMessage.getFirstName() + ", thank you for ordering with us!");
        emailSender.send(message);

        EmailSent emailSent = new EmailSent();
        emailSent.setEmail(orderCustomerMessage.getCustomerEmail());
        emailSent.setEmailType(EmailSent.EmailType.ORDER_RECEIVED_EMAIL);
        return emailSent;
    }
}
```

EmailProcessor Binding

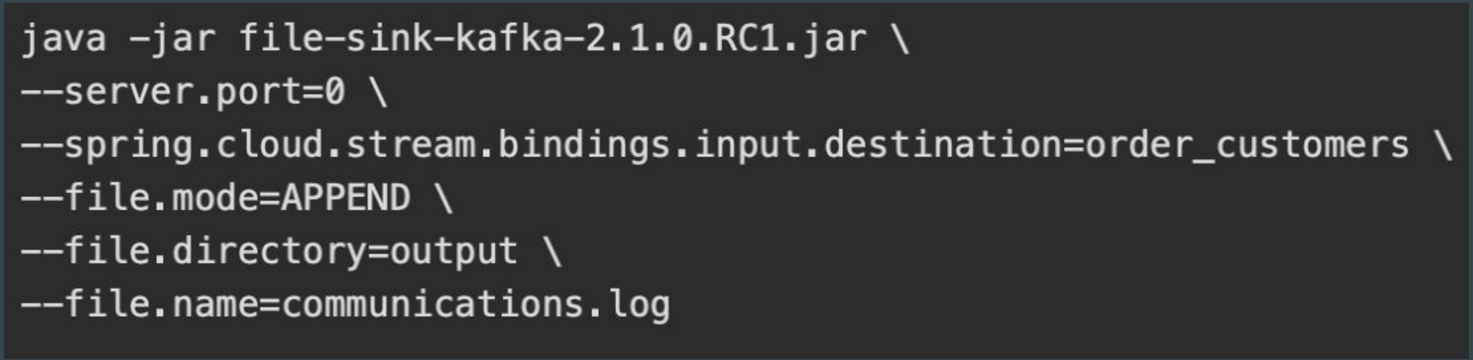


```java
public interface EmailProcessor {
    String ORDER_CUSTOMERS_TO_EMAIL = "orderCustomersToEmail";
    String EMAILS_SENT = "emailsSent";

    @Input(ORDER_CUSTOMERS_TO_EMAIL)
    SubscribableChannel orderCustomersToEmail();

    @Output(EMAILS_SENT)
    MessageChannel emailsSent();
}
```



```yaml
spring.cloud.stream.bindings:
  input:
    destination: order_customers
    group: order_to_customer_processor
  orderCustomersToEmail:
    destination: order_customers
    group: email_processor
  emailsSent:
    destination: emails_sent
```

Lastly a file Sink



```
java -jar file-sink-kafka-2.1.0.RC1.jar \
--server.port=0 \
--spring.cloud.stream.bindings.input.destination=order_customers \
--file.mode=APPEND \
--file.directory=output \
--file.name=communications.log
```

File location and configuration

Destination binding for Kafka

Demo Time

Resources

Kafka Tool