Introduction to

Spring Cloud Stream

— Nandakumar Purohit

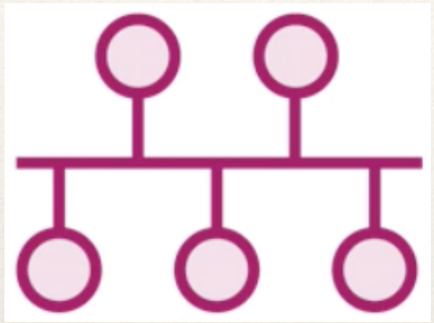
Spring Cloud Stream

Agenda

- The Role of Messaging in Microservices
- Understanding Spring Cloud Stream
- Understanding Binders and Configuring Stream Applications
- Creating Message Senders & Receivers using Spring Cloud Stream
- Creating Custom Interfaces
- Stream Listener methods based on Message Headers

Spring Cloud Stream

The Role of Messaging in Microservices



Encourage Loose Coupling

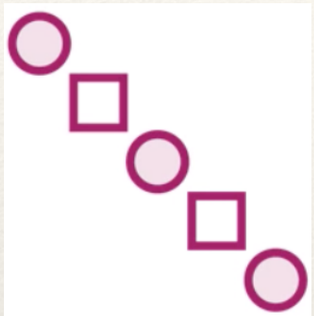
- It keeps services out from creating intentional & unintentional **linkages** between their **Endpoints**, **Business Logic & Schemas**

Improve Scalability & Reliability

- There may be needs for **abstraction** in terms of change in their **availability, evolving differently**
- Messaging **helps level** this abstraction and can **scale individual** components as necessary

Spring Cloud Stream

The Role of Messaging in Microservices



New Intake and Processing Patterns

- Introduce **new intake and processing patterns** by **Event Sourcing** OR **IoT** Message Handling
- Scenarios of **Over-engineering** of Messaging OR **moving the complexity** else where can be dealt with easily

Interact with Legacy Systems

- **Legacy systems cannot be changed**
- Gives capability to add **Messaging Facade** to Legacy systems to bring them to Microservices Architecture

Spring Cloud Stream

Understanding Spring Cloud Stream

- Framework for building message-driven Microservices Applications
- Opinionated Runtime Environment with Minimal Configuration
- Message Broker **Transparency** to connect to Endpoints
- MessageChannel & Message<T>
- Channel Adapters (Inbound & Outbound)
- **No Broker Specific code** in your Application

Spring Cloud Stream

Understanding Spring Cloud Stream

Apps communicate through channels

Pub/sub pattern

Middleware abstracted via binders

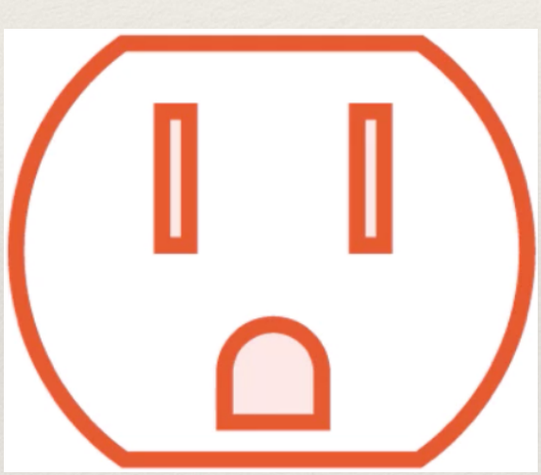
Consumer groups for competing consumer

@StreamListener to pull events

Partitioning for stateful processing

Spring Cloud Stream

Understanding Spring Cloud Stream



Binders

- Connects you to physical endpoints in the external middleware
- Spring Cloud detects binders on classpath
- Can connect to multiple brokers of same type
- Can also use different binders with same code
- Possible to write your own binder

Spring Cloud Stream

Understanding Spring Cloud Stream

@StreamListener

- Unique to Spring Cloud Stream
- Handler for inbound messages
- Does automatic content type conversion
- Dispatch to multiple methods based on conditional checks

Spring Cloud Stream

Understanding Spring Cloud Stream

```
import org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.messaging.Source;
import org.springframework.integration.annotation.InboundChannelAdapter;
```

```
@EnableBinding(Source.class)
public class OrderSource {

	// Auto-push every 1 second
	@InboundChannelAdapter(value = Source.OUTPUT)
	public String sendOrder() {

		return "Polling Demo";
	}

}
```

- Lights up class as Stream Application
- Source, Sink, Processor are built-in, basic interfaces
- One way to emit data is with Spring Integration's InboundChannelAdapter

Spring Cloud Stream

Understanding Spring Cloud Stream

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

spring.cloud.stream.bindings.output.destination = orders
```

- Properties OR YAML point to destination
- Destination name set here or defaults to name of channel
- May also set connection values

Spring Cloud Stream

Understanding Spring Cloud Stream

```
@EnableBinding(Sink.class)
@SpringBootApplication
public class StreamReceiver {

    public static void main(String args[]) {
        SpringApplication.run(StreamReceiver.class, args);
    }

    @StreamListener(Sink.INPUT)
    public void log(String msg) {
        System.out.println(msg);
    }

}
```

- Lights up class as Stream Application
- @StreamListener pulls from input channel of Sink

Spring Cloud Stream

Understanding Spring Cloud Stream

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

spring.cloud.stream.bindings.input.destination = orders
```

- Destination name needs to match value designated in source

Spring Cloud Stream

Use Case - Creating Message Sender & Receiver

- Create Message Sender in **fastpass-console** that publishes message every second
- Create new project via Spring Starter
- Add Actuator and stream-rabbit dependencies
- Create Message Receiver in this new project that processes streams of incoming messages
- Observe RabbitMQ for Exchange, Channel, Queue

Spring Cloud Stream

Use Case - Creating Custom Interfaces

- Create Custom Interface for sender application
- Use InboundChannelAdapter to automatically publish every 2 seconds

Spring Cloud Stream

Use Case - Multiple Stream Listener Methods

- Dispatch to multiple Stream Listener methods
- Set the Header value for Payload