ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Spring Workshop

5 - Monoliths & MicroServices

Agenda

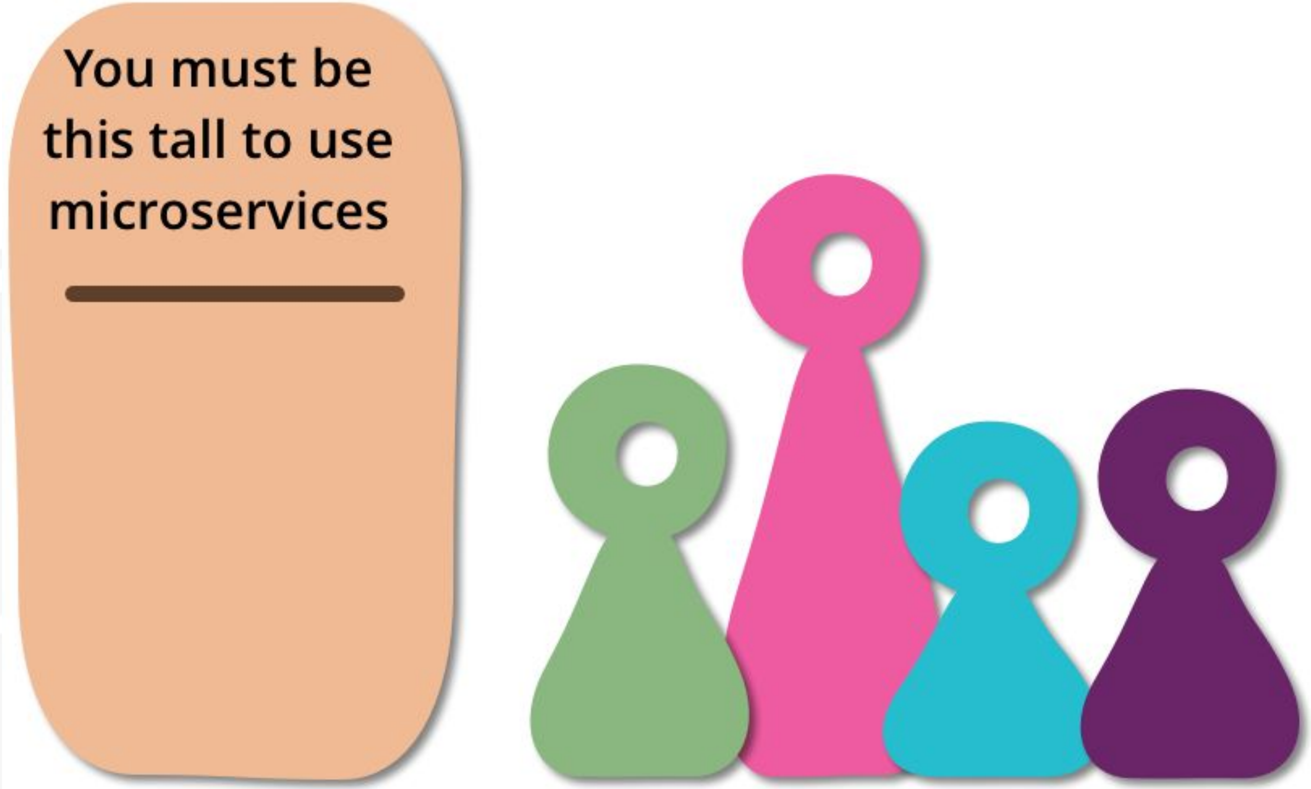
- Basics
- Synchronous Communication
    - WebClient
    - (Open)Feign
- Asynchronous Communication
    - AMQP with RabbitMQ
- Alternatives

Basics

Basics

- Common Use Cases
  - Legacy Systems / Integrating Systems
  - Scaling Systems (Load)
  - Scaling of Teams
- **MicroServices don't guarantee modularity**
- More Information
  - https://martinfowler.com/microservices/
  - https://semaphoreci.com/blog/bad-microservices

Pre-Requisites



WebClient (Synchronous)

WebClient Approach

```
@Service
public class ReviewClient {

  public boolean reviewClient(WebClient.Builder webClientBuilder, Object data) {
    var webClient = webClientBuilder.baseUrl("http://localhost:8081").build();
  }
}
```

WebClient Approach

```java
@Service
public class ReviewClient {

  public boolean reviewClient(WebClient.Builder webClientBuilder, Object data) {
    var webClient = webClientBuilder.baseUrl("http://localhost:8081").build();

    webClient
      .post().uri("/api/review")
      .contentType(MediaType.APPLICATION_JSON).bodyValue(data)
      .retrieve()
  }
}
```

WebClient Approach

```
@Service
public class ReviewClient {

  public boolean reviewClient(WebClient.Builder webClientBuilder, Object data) {
    var webClient = webClientBuilder.baseUrl("http://localhost:8081").build();

    ResponseEntity<Void> returner = webClient
      .post().uri("/api/review")
      .contentType(MediaType.APPLICATION_JSON).bodyValue(data)
      .retrieve()
      .toBodilessEntity()
      .block();
  }
}
```

WebClient Approach

```
@Service
public class ReviewClient {

  public boolean reviewClient(WebClient.Builder webClientBuilder, Object data) {
    var webClient = webClientBuilder.baseUrl("http://localhost:8081").build();

    ResponseEntity<Void> returner = webClient
      .post().uri("/api/review")
      .contentType(MediaType.APPLICATION_JSON).bodyValue(data)
      .retrieve()
      .toBodilessEntity()
      .block();

    if (returner.getStatusCode().is2xxSuccessful()) {
      return BrochureStatus.FAILED;
    } else {
      return BrochureStatus.CONFIRMED;
    }
  }
}
```

Feign (Synchronous)

Feign

- Declarative approach
- Uses HTTP client internally
- Originally from Netflix, but deprecated
  - OpenFeign as successor
- Perfect use case for gateway
- Supports OAuth2, Caching, Error Handling, Integration into Spring Cloud (for example Eureka or CircuitBreakder)

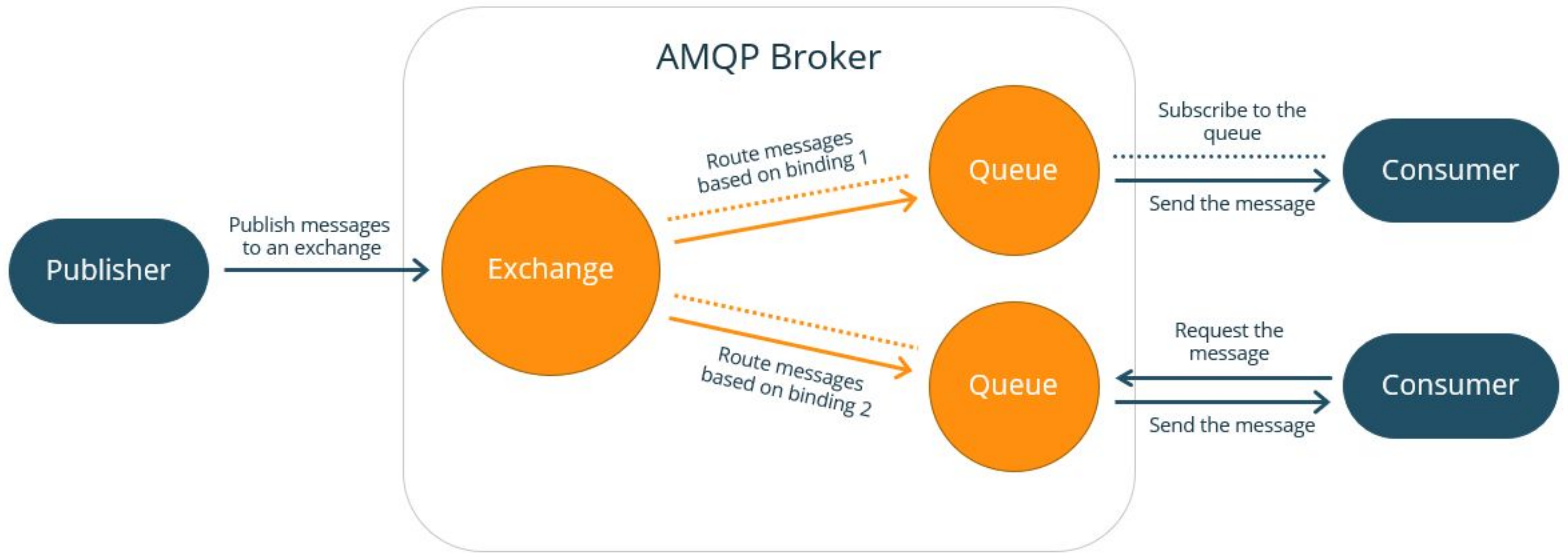
Code Example

```
@FeignClient(name = "printing", url = "http://localhost:8081")

public interface PrintingClient {

  @PostMapping(value = "/api/review")

  boolean addPrintingJob(CreateReviewData createReviewData);

}
```

AMQP

AMQP

- Asynchronous
- Technology Agnostic
- Implementation: RabbitMQ, StormMQ, OpenAMQ
- Main Elements:
  - Exchange
  - Queue
  - Binding
  - Message



https://support.smartbear.com/readyapi/docs/testing/amqp.html

Code Example

```java
@Configuration public class MessagingConfiguration {
  public static final String queueName = "printing-events-queue";

  @Bean Queue getQueue() {
    return new Queue(queueName, false);
  }
}
```

Code Example

```
@Configuration public class MessagingConfiguration {

  public static final String exchangeName = "printing-events";
  public static final String queueName = "printing-events-queue";

  @Bean Queue getQueue() {
    return new Queue(queueName, false);
  }

  @Bean TopicExchange getExchange() {
    return new TopicExchange(exchangeName);
  }
}
```

Code Example

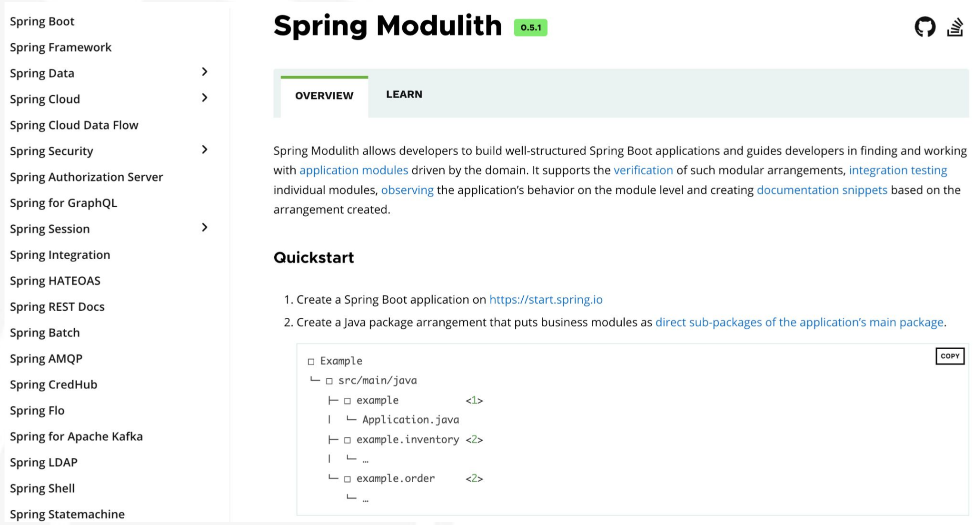
```java
@Configuration public class MessagingConfiguration {

  public static final String exchangeName = "printing-events";
  public static final String queueName = "printing-events-queue";
  public static final String routingKey = "printing.routing";

  @Bean Queue getQueue() {
    return new Queue(queueName, false);
  }

  @Bean TopicExchange getExchange() {
    return new TopicExchange(exchangeName);
  }

  @Bean Binding getBinding(Queue queue, TopicExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with(routingKey);
  }
}
```

Code Example

```
@Configuration public class MessagingConfiguration {

  // ...

  @Bean MessageListenerAdapter listenerAdapter(PrintedJobReceiver printedJobReceiver) {
    return new MessageListenerAdapter(printedJobReceiver, "processMessage");
  }

  @Bean SimpleMessageListenerContainer getContainer(
    ConnectionFactory connectionFactory, MessageListenerAdapter listenerAdapter) {

    SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
    container.setConnectionFactory(connectionFactory);
    container.setQueueNames(queueName);
    container.setMessageListener(listenerAdapter);
    return container;
  }
}
```

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Alternative: Monolithic Systems



Spring Modulith 0.5.1

Spring Modulith allows developers to build well-structured Spring Boot applications and guides developers in finding and working with application modules driven by the domain. It supports the verification of such modular arrangements, integration testing individual modules, observing the application's behavior on the module level and creating documentation snippets based on the arrangement created.

Quickstart

1. Create a Spring Boot application on https://start.spring.io
2. Create a Java package arrangement that puts business modules as direct sub-packages of the application's main package.

```
□ Example
└─ □ src/main/java
   ├─ □ example          <1>
   |  └─ Application.java
   ├─ □ example.inventory <2>
   |  └─ …
   └─ □ example.order     <2>
      └─ …
```

```
private void executeLoad(long timeout, int usersCount) {
    showDebugInfo(timeout);
    Load.setPages(URL, parsingTimeout);
    Load.setTimeout(timeout);
    List<Load> threads = new ArrayList<>();
    for (int i = 0; i < usersCount; i++) {
        threads.add(new Load(this.URL));
    }
    logger.info( s: usersCount +
    for (Load thread : threads)
        thread.start();
    }
    logger.info( s: "All threads are started");
    progressInfo(timeout);
    System.out.print("........DONE\nProcessing with data...\n");
}

private void executeAvailability(long timeout, int
}

private void
```

Demo

Lab Time