CS544 EA

Integration

Messaging: RabbitMQ Exchanges

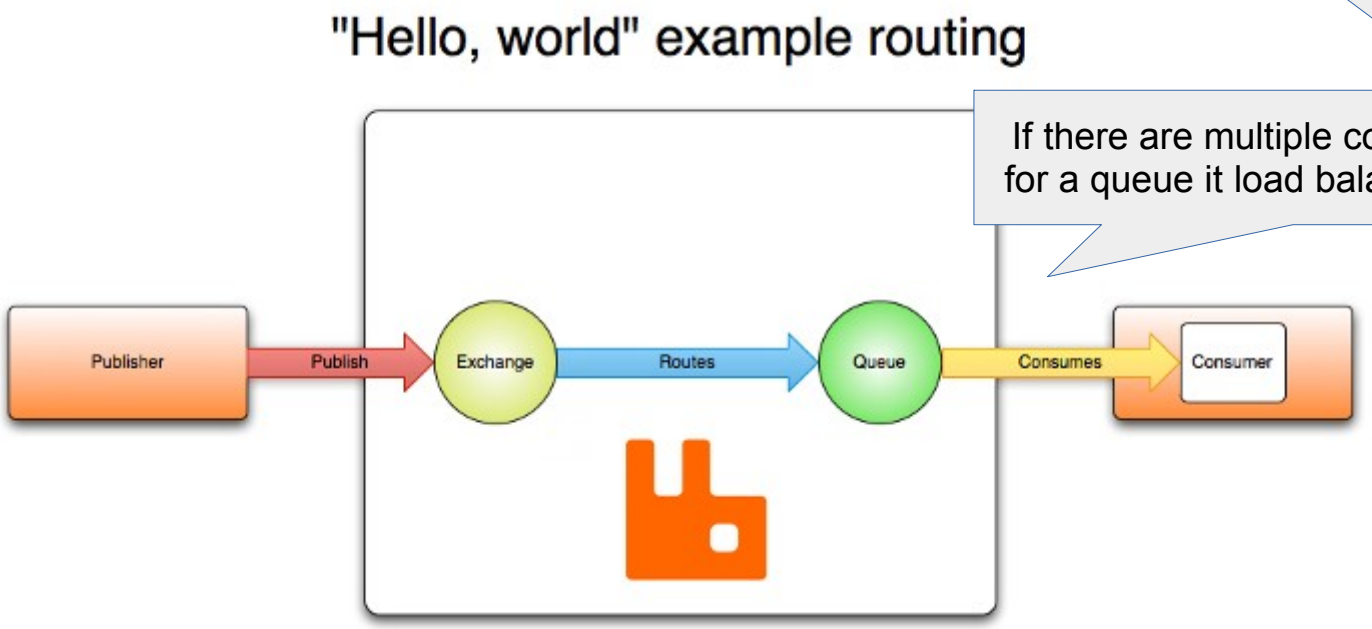
Exchange Types

- There are different exchange types
  - The routing algorithm used depends on the exchange type and rules called bindings

Name	Description
Direct Exchange	Routing == queue name (simply delivers to named queue)
Fanout Exchange	Routing ignored (a copy is delivered to all queues bound to it)
Topic Exchange	Matches routing key to bind pattern (can deliver to one or many)
Headers Exchange	Matches (one or more) msg headers instead of routing key

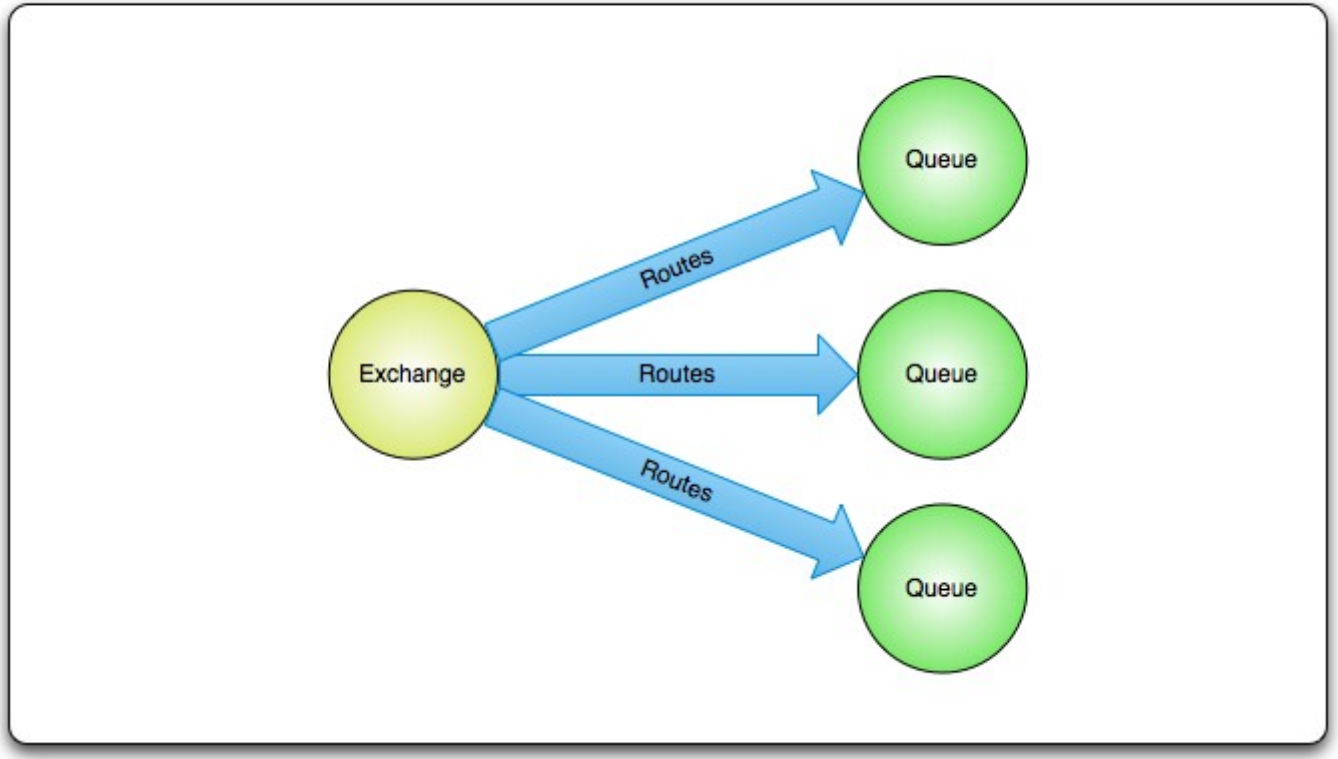
Direct Exchange

This is often used to split tasks among many workers



From: https://www.rabbitmq.com/tutorials/amqp-concepts.html

Fanout exchange routing



From: https://www.rabbitmq.com/tutorials/amqp-concepts.html

TopicExchange Example

```
@SpringBootApplication
public class Application {
	private static final String topicExchangeName = "spring-boot-exchange";
	private static final String queueName = "spring-boot";
	@Bean
	public Queue queue() {
		return new Queue(queueName, false);
	}
	@Bean
	public TopicExchange exchange() {
		return new TopicExchange(topicExchangeName);
	}
	@Bean
	public Binding binding(Queue queue, TopicExchange exchange) {
		return BindingBuilder.bind(queue).to(exchange).with("foo.bar.#");
	}
	@Bean
	public MessageListenerAdapter listenerAdapter(Receiver receiver) {
		return new MessageListenerAdapter(receiver, "receiveMessage");
	}
	@Bean
	public SimpleMessageListenerContainer container(ConnectionFactory connectionFactory,
			MessageListenerAdapter listenerAdapter ) {
		SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
		container.setConnectionFactory(connectionFactory);
		container.setQueueNames(queueName);
		container.setMessageListener(listenerAdapter);
		return container;
	}
	public static void main(String[] args) {
		SpringApplication.run(Application.class, args);
	}
}
```

Create the queue

Create the TopicExchange

Bind this queue to the pattern: Anything starting with foo.bar.

Specify the details object and method that will receive the message (see Receiver on next slide)

Setup Spring to listen for the message and then call the adapter when it receives

Receiver Bean

- This time no @Rabbit annotations
  - The previous config has connected it instead

```
package edu.mum.cs544.message;

import org.springframework.stereotype.Component;

@Component
public class Receiver {

	public void receiveMessage(String message) {
		System.out.println("Received message: " + message);
	}
}
```

Sender

```java
package edu.mum.cs544.message;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
	public static void main(String[] args) {
		SpringApplication.run(Application.class, args).close();;
	}
}
```

```java
package edu.mum.cs544.message;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class Sender implements CommandLineRunner {
	private static final String topicExchangeName = "spring-boot-exchange";

	@Autowired
	private RabbitTemplate template;

	@Override
	public void run(String... args) throws Exception {
		template.convertAndSend(topicExchangeName, "foo.bar.baz", "Hello from Sender");
	}
}
```

Specify the exchange

Specify the route
(will match pattern)

Sending Objects

- In these slides we've only sent Strings
  - You can send Objects that implement **Serializable**
  - As long as it is the **exact same class** on both sides
  - Including it being in the same package
    - Fully Qualified Class Name should be the same