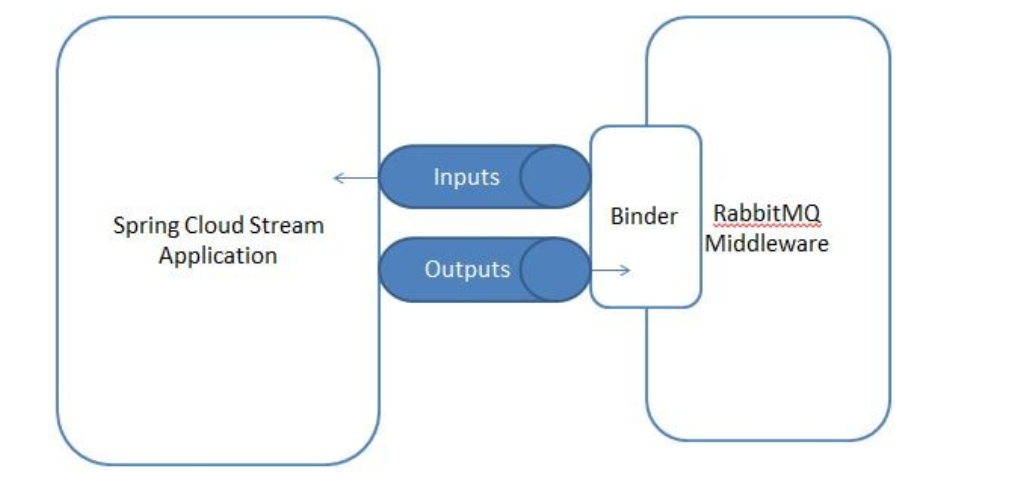
Spring Cloud Stream - Publish Message to RabbitMQ Simple Example

Using Spring Cloud Stream we can develop applications where we do not need to specify the implementation details of the messaging system we want to use. We just need to specify the required binding dependencies and Spring Cloud Stream will the integrate the messaging systems to Spring Boot Application.



Spring Cloud Concepts-

* **Binder -** Depending upon the messaging system we will have to specify a the messaging platform dependency, which in this case is RabbitMQ
* <dependency> <groupId>org.springframework.cloud</groupId> <artifactId>spring-cloud-starter-stream-rabbit</artifactId> </dependency>
* **Source -** When a message is needed to be published it is done using Source. The Source is an interface having a method annotated with @Output. The @Output annotation is used to identify output channels. The Source takes a POJO object, serializes it and then publishes it to the output channel.

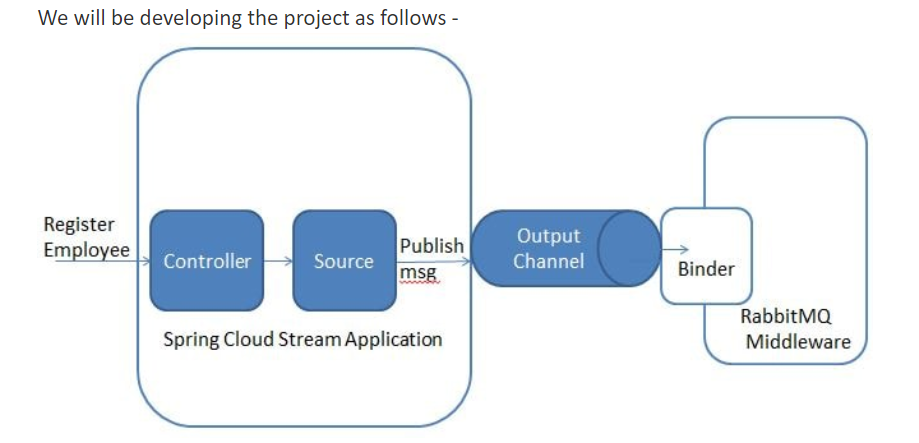
public interface EmployeeRegistrationSource {

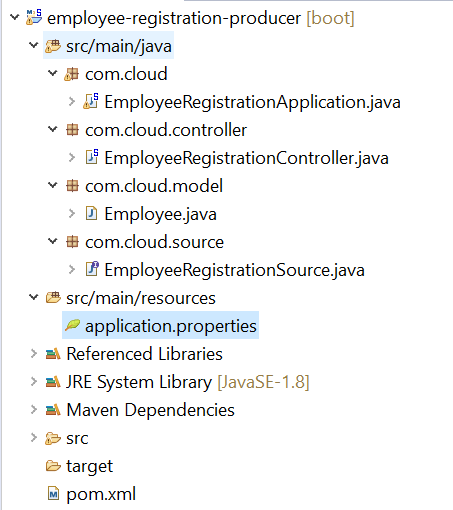
@Output("employeeRegistrationChannel")

MessageChannel employeeRegistration();

}

* **Channel -** A channel represents an input and output pipe between the Spring Cloud Stream Application and the Middleware Platform. A channel abstracts the queue that will either publish or consume the message. A channel is always associated with a queue. With this approach, we do not need to use the queue name in the application code. So if tomorrow the queue needs to be changed, we dont need to change the application code.   
  For example in the EmployeeRegistrationSource we have specified the channel name as employeeRegistrationChannel. In application.properties we have associated this channel with a RabbitMQ Exchange.





The pom.xml will be as follows with the binder dependency of RabbitMQ as follows-

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-stream-rabbit</artifactId>**

**</dependency>**

Create the Spring Boot Bootstrap class with the SpringBootApplication annotation as follows-

**package** com.cloud;

**import** org.springframework.boot.SpringApplication;

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

@SpringBootApplication

**public** **class** EmployeeRegistrationApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EmployeeRegistrationApplication.**class**, args);

}

}

**package** com.cloud.model;

**import** com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties(ignoreUnknown = **true**)

**public** **class** Employee {

**private** String empName;

**private** String empID;

**public** String getEmpName() {

**return** empName;

}

**public** **void** setEmpName(String empName) {

**this**.empName = empName;

}

**public** String getEmpID() {

**return** empID;

}

**public** **void** setEmpID(String empID) {

**this**.empID = empID;

}

@Override

**public** String toString() {

**return** "Employee [empName=" + empName + ", empID=" + empID + "]";

}

}

Next define the Source class. This will simply be an interface that defines ways of obtaining the MessageChannel object needed to send the message. Here we define the output channel named as employeeRegistrationChannel.

**package** com.cloud.source;

**import** org.springframework.cloud.stream.annotation.Output;

**import** org.springframework.messaging.MessageChannel;

**public** **interface** EmployeeRegistrationSource {

@Output("employeeRegistrationChannel")

MessageChannel employeeRegistration();

}

Next we define a simple controller that will make use of the above defined classes to publish the message upon receiving the employeeRegistration request. Here the **@EnableBinding** annotation tells Spring Cloud Stream that you want to bind the Controller to a message broker.

**package** com.cloud.controller;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.cloud.stream.annotation.EnableBinding;

**import** org.springframework.integration.support.MessageBuilder;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.cloud.model.Employee;

**import** com.cloud.source.EmployeeRegistrationSource;

@RestController

@EnableBinding(EmployeeRegistrationSource.**class**)

**public** **class** EmployeeRegistrationController {

@Autowired

EmployeeRegistrationSource employeeRegistrationSource;

@RequestMapping("/register")

@ResponseBody

**public** String orderFood(@RequestBody Employee employee) {

employeeRegistrationSource.employeeRegistration().send(MessageBuilder.*withPayload*(employee).build());

System.***out***.println(employee.toString());

**return** "Employee Registered";

}

}

Finally we specify the properties. Here we specify the RabbitMQ properties. Also we associate the channel to the queue to be used

server.port=8080

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

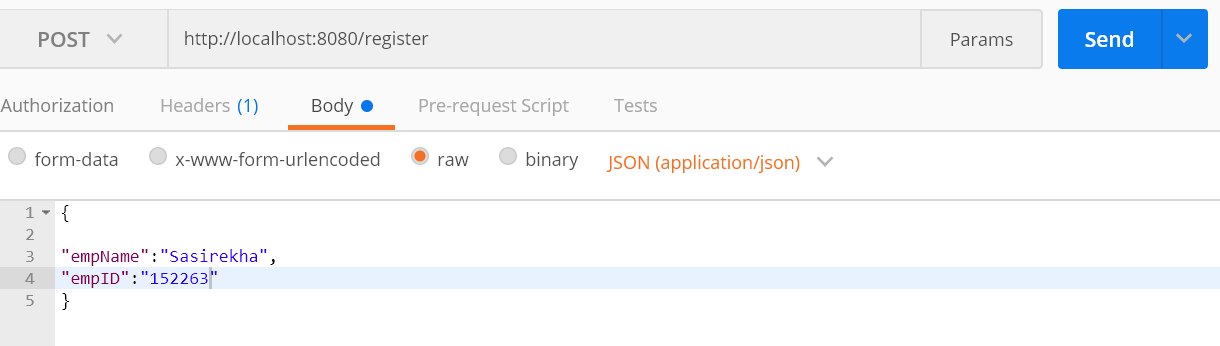
spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

spring.cloud.stream.bindings.**employeeRegistrationChannel**.destination=employeeRegistrations

spring.cloud.stream.default.contentType=application/json

We are done with the required Java code. Now lets start RabbitMQ. As we had explained in detail in the [Getting started with RabbitMQ](https://www.javainuse.com/misc/rabbitmq-hello-world) perform the steps to start the RabbitMQ.  
Next start the Spring Boot Application by running it as a Java Application. Create a POST request with payload as follows



Send the above request, it trigger the message to be sent to the RabbitMQ.  
Next go to the RabbitMQ console-**http://localhost:15672/**. We can see in the Exchange section, an exchange named employeeRegistration gets created and it has one message.

