(..)

Requirements

- A cloned version of the code repository (https://github.com/platform-acceleration-lab/apps-cloud-native-evolution-code) and finished steps 1, 2 and 3 in the cloud native evolution.

What you will learn/review

- Use a message queue based interface to implement a subscription model

Implement message queue based interface via RabbitMQ

There are times when you do not want to use HTTP as the interface for your microservices to communicate.

One example is if one of your microservices performs expensive work. Communicating via a synchronous interface like HTTP would slow the system down so using an asynchronous mechanism like RabbitMQ would allow you to scale the slower services in response to the number of messages coming in.

See the Spring Boot Messaging documentation (http://docs.spring.io/autorepo/docs/spring-boot/current/reference/htmlsingle/#boot-features-amqp) for more details.

1. Make sure that you have RabbitMQ installed and running. See the RabbitMQ website (https://www.rabbitmq.com/download.html) for details on your operating system.
2. Create an interface called `BillingClient` in `components/billing`. This interface will have one method, `billUser`, that has the same interface as the existing `billUser` method. Rename the existing class to `HttpBillingClient`.

3. Create a new client class, `RabbitBillingClient`, that also implements this interface. This client will use a `RabbitTemplate`, a queue name, and a newly created `BillingMessage` class to send messages to RabbitMQ. The queue name and `RabbitTemplate` will be injected into the `RabbitClient` constructor. Make sure that that `BillingMessage` is `Serializable`.

4. In `applications/ums`, change the billing client `@Bean` definition to return a `RabbitClient` instead of the existing client class.

5. Confirm that `applications/ums` compiles with the `build` command.

6. Build a class in `applications/billing` to consume `BillingMessages` from RabbitMQ on the same queue defined in the producer above using `@RabbitListener`. This class will have one method called `process` that uses a payment gateway to create a recurring payment, very similar to the existing controller. This method should report both successes and failures to `stdout` for debugging purposes.

7. Modify the `applications/billing` application class to create a `Queue` `@Bean` that is not durable and matches the name defined in `applications/ums`.

8. Modify the `applications/billing` application class with an `@Bean` that creates an instance of the `BillingMessage` consumer class created above.

9. Start both applications with the `bootRun` command. Make sure you start `applications/billing` so that it can create the RabbitMQ queue.

10. Use `curl` to create subscriptions and test your applications. Confirm that you see the output in the billing logs that you expect for successful message processing.

Troubleshooting: Check to make sure that messages are ending up in the correct RabbitMQ queue using the admin console at http://localhost:15672/ (http://localhost:15672/).

Pushing to cf

1. Create a `p.rabbitmq` instance and bind it to both applications.

2. Re-build both applications and deploy to PCF, deploying `applications/billing` first to make sure the RabbitMQ queue gets created properly.

3. Use `curl` to create subscriptions and test your applications. Confirm that you see the output in the billing logs that you expect for successful message processing.

Troubleshooting: Check to make sure that messages are ending up in the correct RabbitMQ queue using the admin console. You can access the service instance RabbitMQ console from Apps Manager on PCF.

Assignment

Once you are done with this section and the application is deployed and working on PCF, you can submit the assignment using the `submitSubscriptions` gradle task. It requires you to provide the `umsUrl`, `username` and `password` project properties. The default user is `user`. Your password can be found in your `ums` logs. For example:

```
cd ~/workspace/assignment-submission
./gradlew submitSubscriptions -PumsUrl=http://my-ums.cfapps.io -Pusern
ame=user -Ppassword=secret
```

(https://pivotal.io)