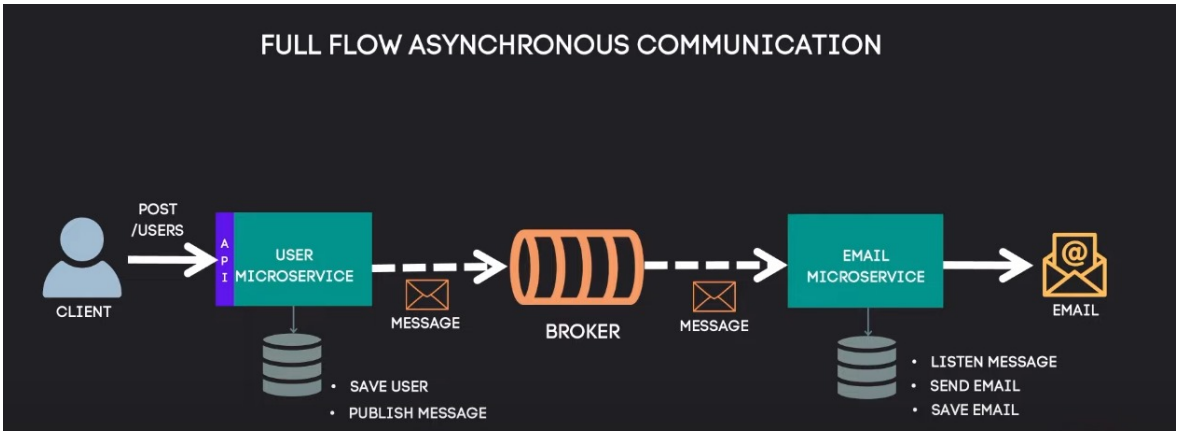
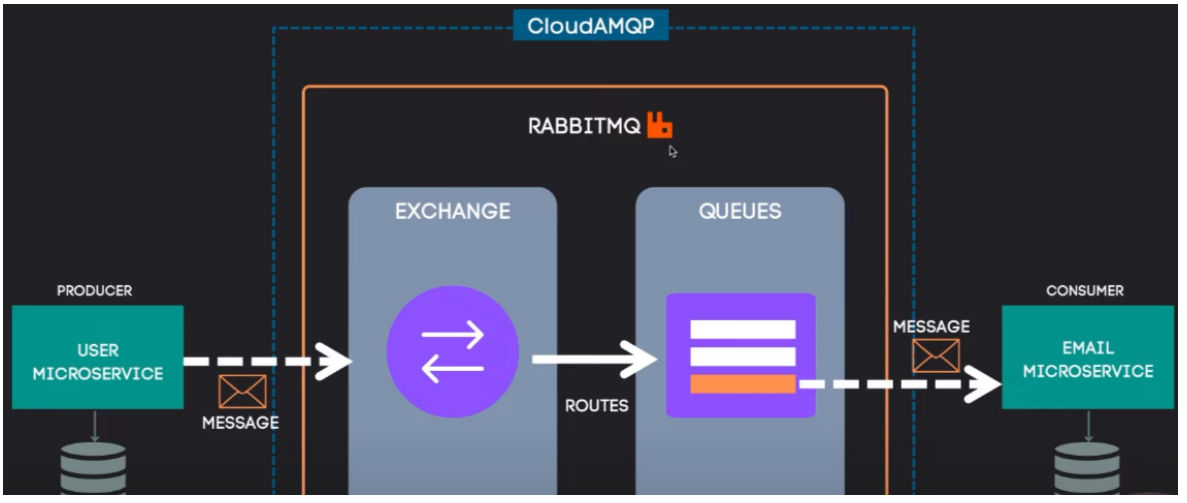
Microservices

Minhas anotações sobre **microservices** usando Spring e RabitMQ

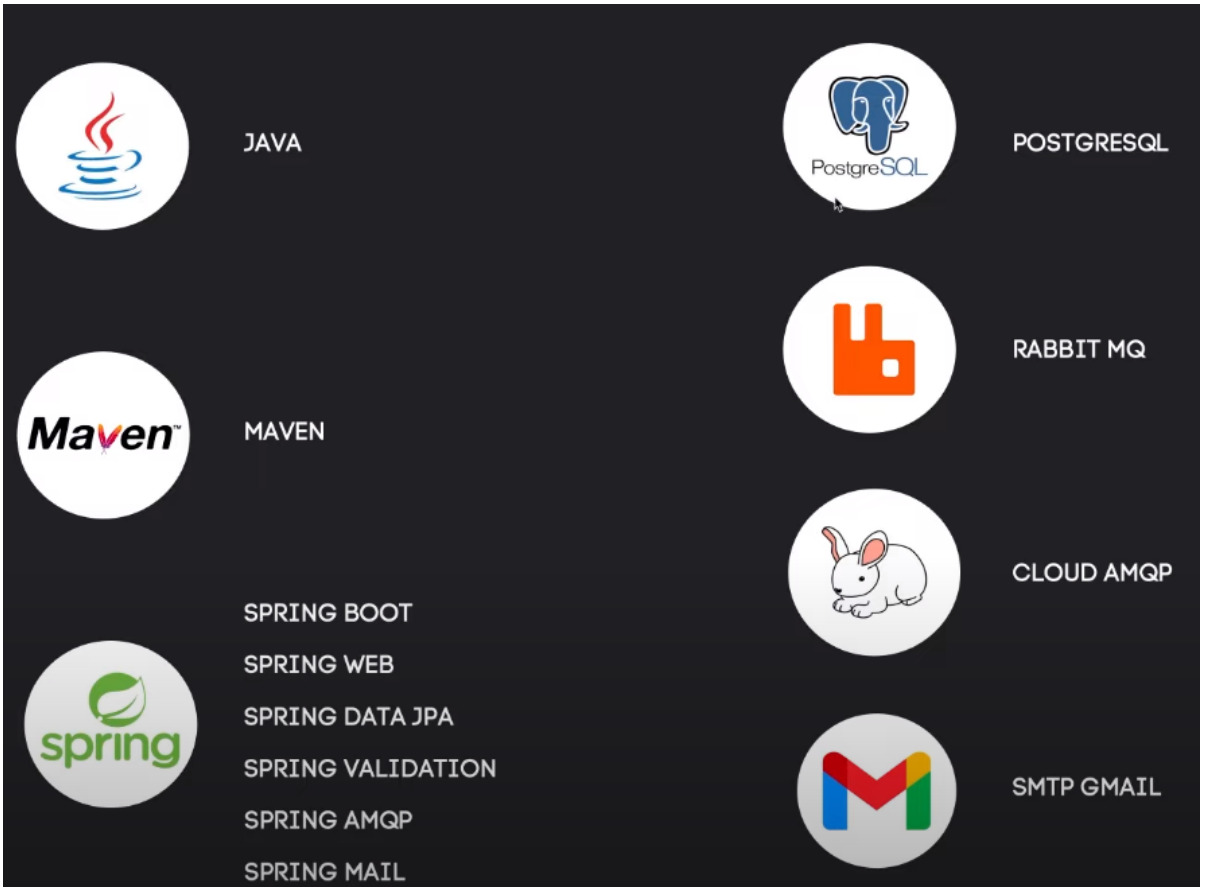
Como o projeto está estruturado:
Será um microservice que a pessoa manda um novo usuário (cadastro) ele é salvo no banco de dados do **user**, é enviado a mensagem para o **broker** o **broker** manda a mensagem para o **email** que escuta a mensagem envia um e-mail de boas vindas para aquele usuário e salva o e-mail no banco de dados do **email**.



O **broker** no caso será o **RABBITMQ** que nós usaremos o **CloudAMQP** para monitora-lo e configura-lo.



Tecnologias que iremos utilizar:



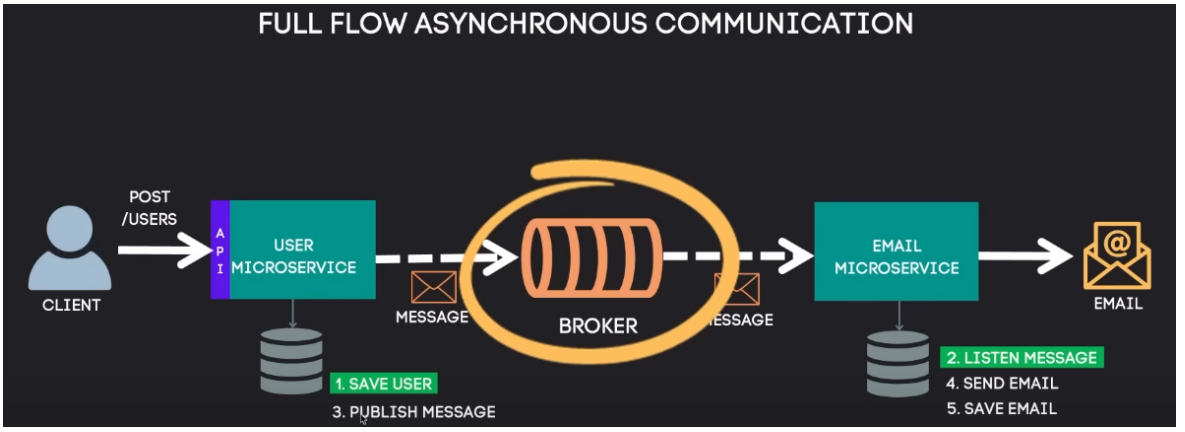
Após criarmos as estruturas bases dos micro serviços, precisamos criar o arquivo de configuração do Rabbit, iremos começar no ms-email:

```
@Configuration
public class RabbitMQConfig {
@Value("${broker.queue.email.name}")
private String queue;
# Dessa maneira criamos uma fila ao iniciar nosso projeto com o nome do valor da
variável
broker.queue.email.name que está no nosso application.properties
@Bean
public Queue queue() {
return new Queue(queue, true);
}
/*
* Precisamos desse método, pois estamos recebendo do rabbit no consumer que temos de
Email
* um JSON e para convertermos em um EmailtRecordDto sem termos problemas
* usamos essa abordagem de termos um método que converta.
*/
@Bean
public Jackson2JsonMessageConverter messageConverter() {
ObjectMapper objectMapper = new ObjectMapper();
return new Jackson2JsonMessageConverter(objectMapper);
}
}
```

Também precisamos de uma classe para o consumer para ouvir o que chega nessa fila que criamos

```
@Component
public class EmailConsumer {
@RabbitListener(queues = "${broker.queue.email.name}")
public void listenEmails(@Payload EmailRecordDto emailRecordDto) {
System.out.println(emailRecordDto.emailTo());
}
}
```

Com isso já temos prontos 2 pontos da nossa arquitetura desejada:



Agora iremos prosseguir para a criação do **publish message**.
Para fazer o producer precisamos o Dto igual ao que temos no ms-email, também precisamos criar uma classe configuração do Rabbit MQ:

```
@Configuration
public class RabbitMQConfig {
/*
* Ao contrário do converter de um cossumer que serve para traduzir o que chegou
* esse método em um producer faz com que ele transforme um objeto java
* em um objeto json entendível pelo brocker.
*/
@Bean
public Jackson2JsonMessageConverter messageConverter() {
ObjectMapper objectMapper = new ObjectMapper();
return new Jackson2JsonMessageConverter(objectMapper);
}
}
```

Colocar as propriedades do endereço e nome da fila no application properties, assim como temos no **ms-email**.
Criar uma classe do **Producer**:

```
@Component
public class UserProducer {
@Autowired
RabbitTemplate rabbitTemplate;
```

```
/* Quando estamos usando o exchange default temos que passar como Routing Key o mesmo
nome da fila */
@Value(value = "${broker.queue.email.name}")
private String routingKey;
public void publishMessageEmail(UserModel userModel) {
var emailDto = new EmailDto();
emailDto.setUserId(userModel.getUserId());
emailDto.setEmailTo(userModel.getEmail());
emailDto.setSubject("Cadastro realizado");
emailDto.setText(userModel.getName() + ", seja bem vindo(a)! \nAgradecemos o seu
cadastro," +
"aproveite agora o melhor folgão de promoções!");
/*
* Passando uma string vazia no primeiro parâmetro ele entende que usaremos o exchange
default e não outro.
* Como o exchange é dafault a routingKey tem que ter o mesmo nome do que a fila.
*/
rabbitTemplate.convertAndSend("", routingKey, emailDto);
}
}
```

Por fim para enviarmos a mensagem nós iremos chamar esse método de publicas mensagem no save do **Service**:

```
@Service
public class UserService {
@Autowired
UserRepository userRepository;
@Autowired
UserProducer userProducer;
/*
* Garante o rollback, ou seja: se uma das operações falha ou da erro a gente realiza o
rollback
* na outra operação que já foi realizada para tudo voltar ao normal.
*/
@Transactional
public UserModel save(UserModel user) {
user = userRepository.save(user);
userProducer.publishMessageEmail(user);
return user;
}
}
```