

Pub/sub



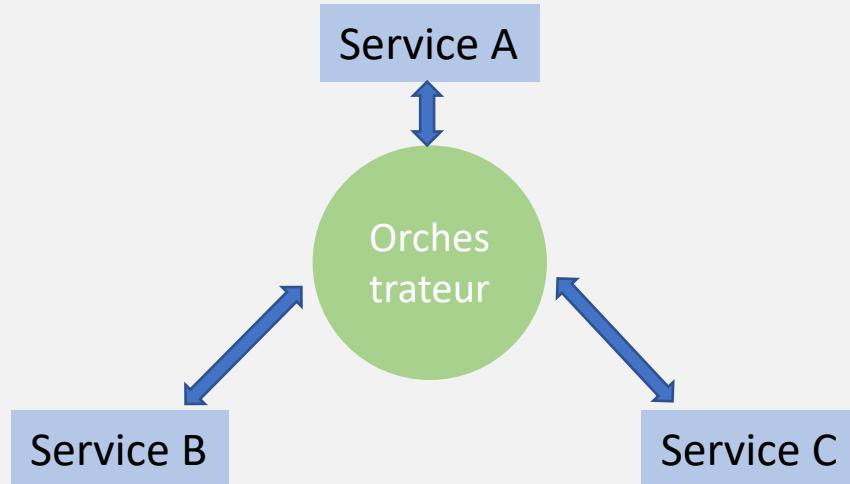
Objectifs

- Types de collaboration
- Publish / subscribe
- Mise en œuvre avec RabbitMQ

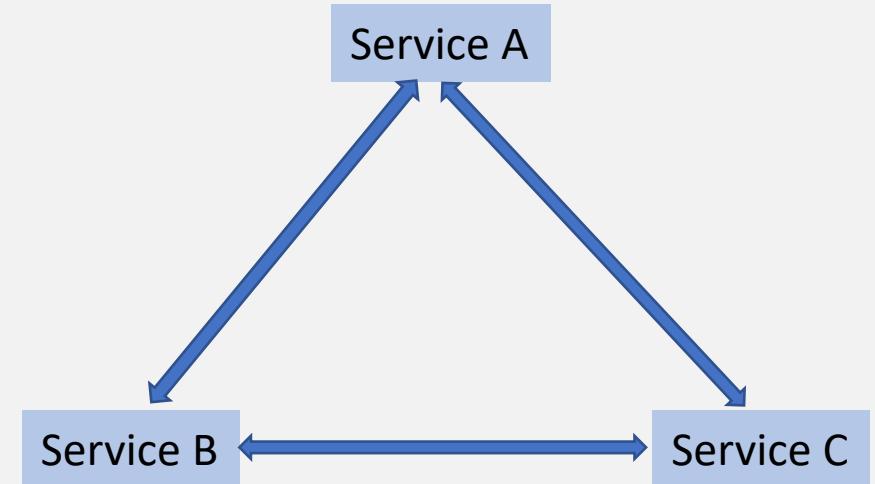
Types de collaboration

- Dans les modules précédents nous avons vu des modèles d'intégrations 1 à 1, aussi bien synchrone qu'asynchrone
- Souvent on cherche à intégrer l'information dans plusieurs systèmes
- On distingue souvent deux types de collaboration de services

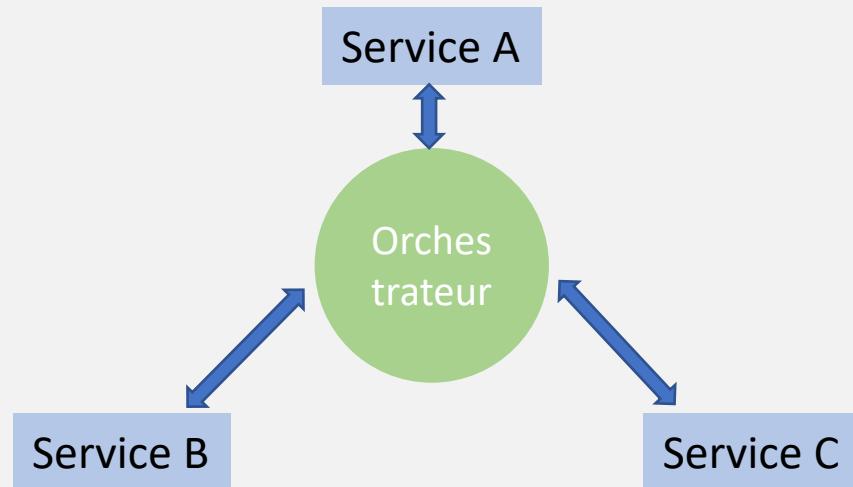
Orchestration



Chorégraphie



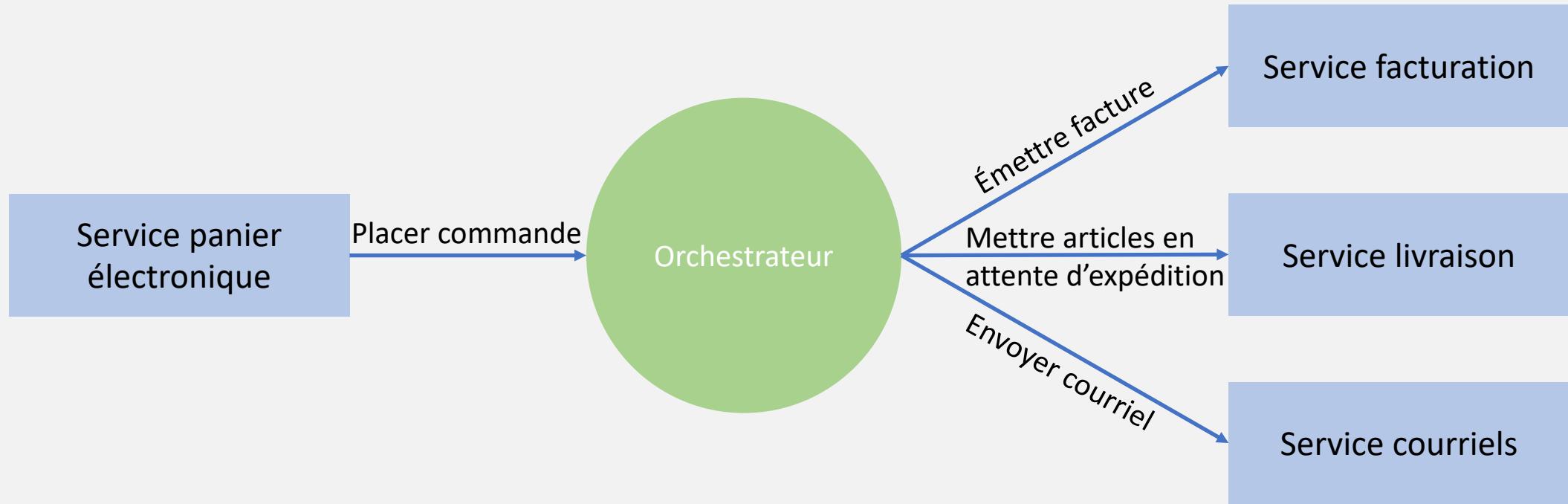
Orchestration



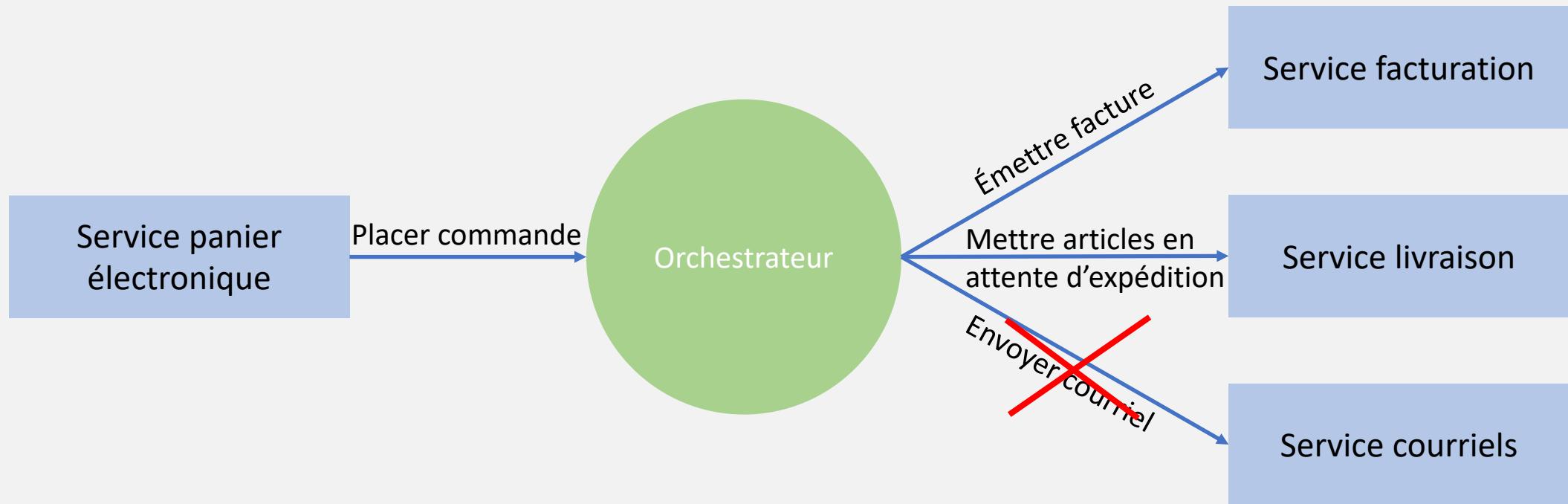
- + Efficace en mode synchrone
- + Facile à maintenir
- + Processus d'affaire unifié (Workflow)
- + Processus d'affaire explicite
- + Peut effectuer des traitements
- Les services sont couplés
- Point de défaillance unique
- Difficultés de mise à jour d'un service

Exemples d'orchestrateurs : MS Biztalk / Logic app service, Mulesoft ESB

Orchestration – Exemple

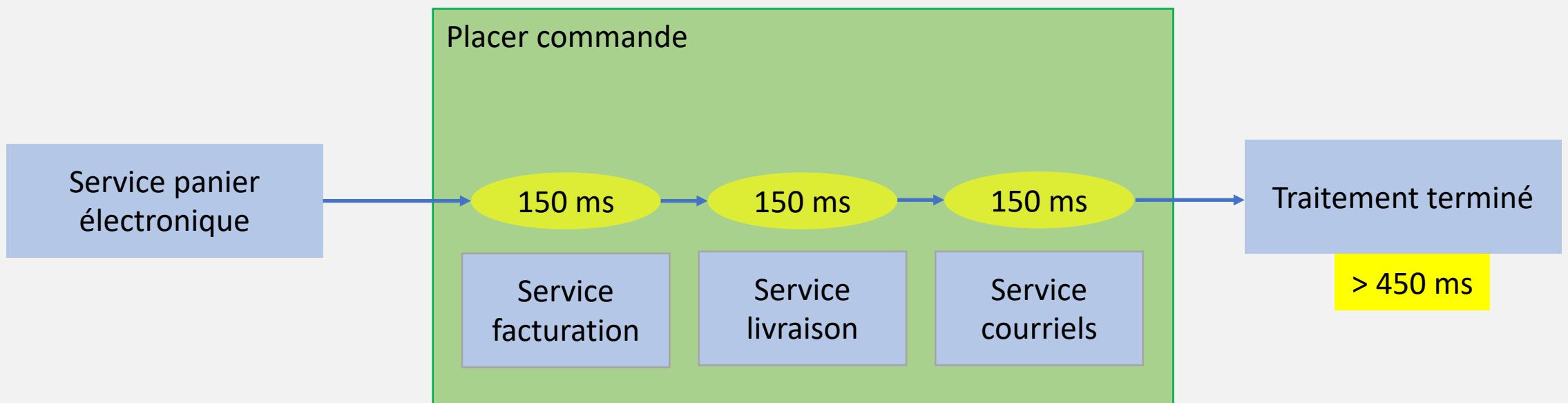


Orchestration – Exemple

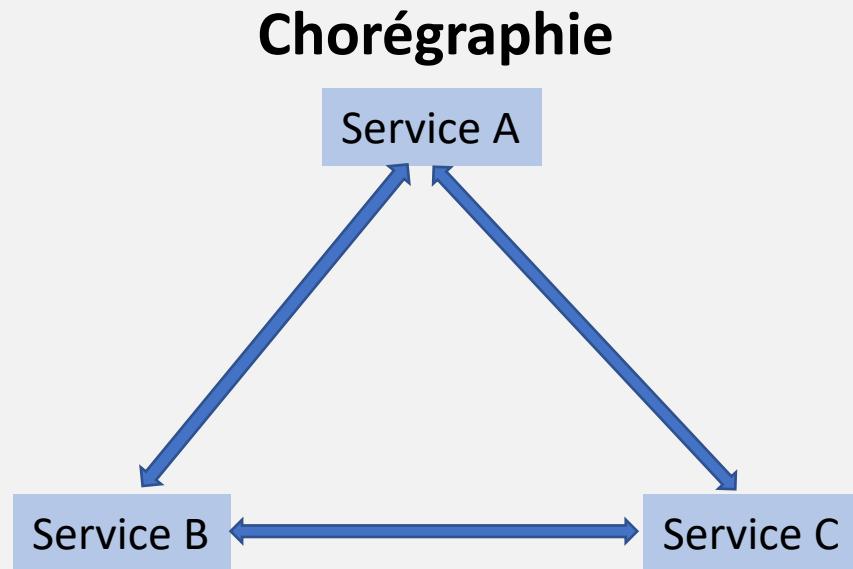


Il faut gérer des cas de reprise.
Les services doivent être idempotents.

Orchestration – Exemple



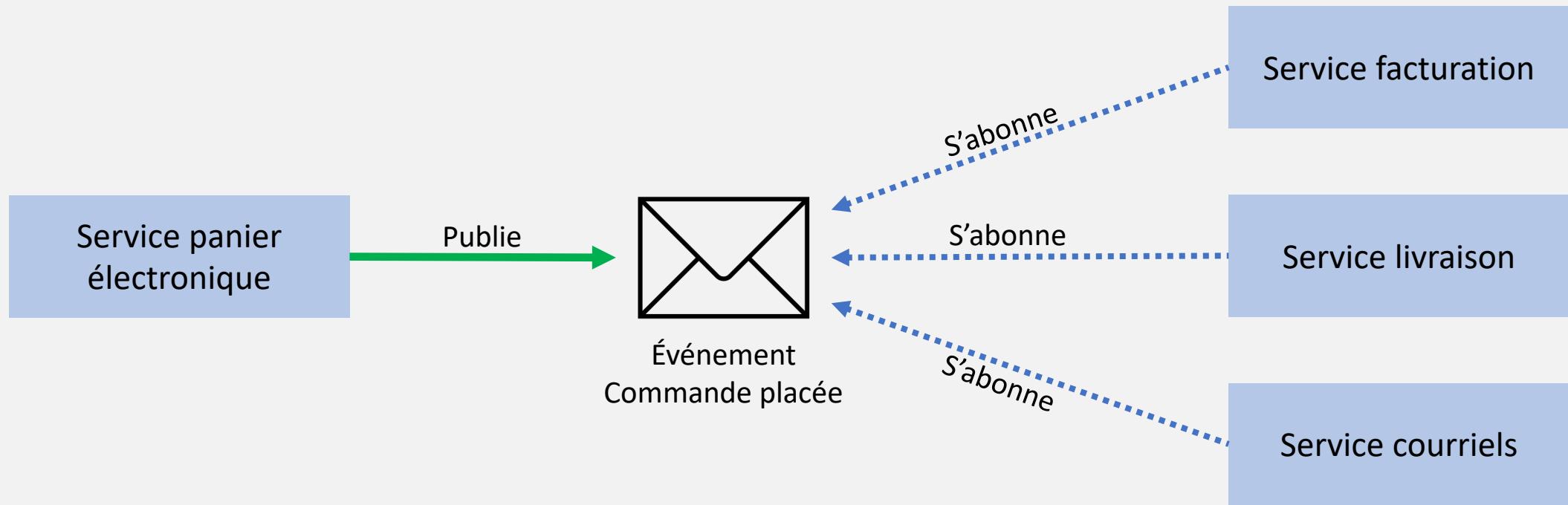
Chorégraphie



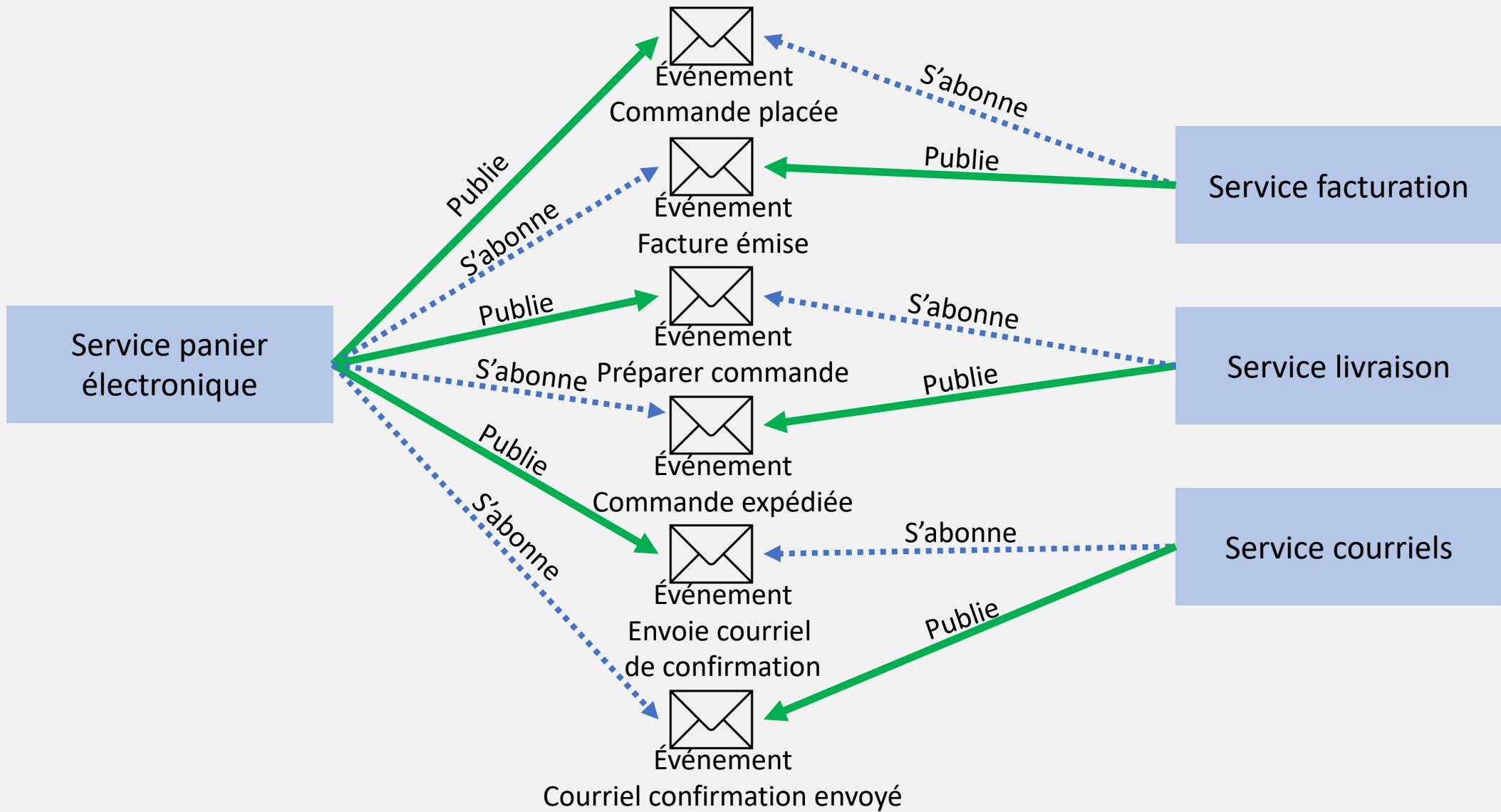
- + Les services envoient des événements
- + Pas de dépendance avec un orchestrateur
- + Pas de point de défaillance unique
- + S'intègre bien dans un mode de développement agile
- + Mise à l'échelle facile
 - Utilisation d'intégrations asynchrones
 - Le processus d'affaire n'est pas visible
 - Difficile à debugger

Exemples de types d'intégration : event sourcing, pub/sub

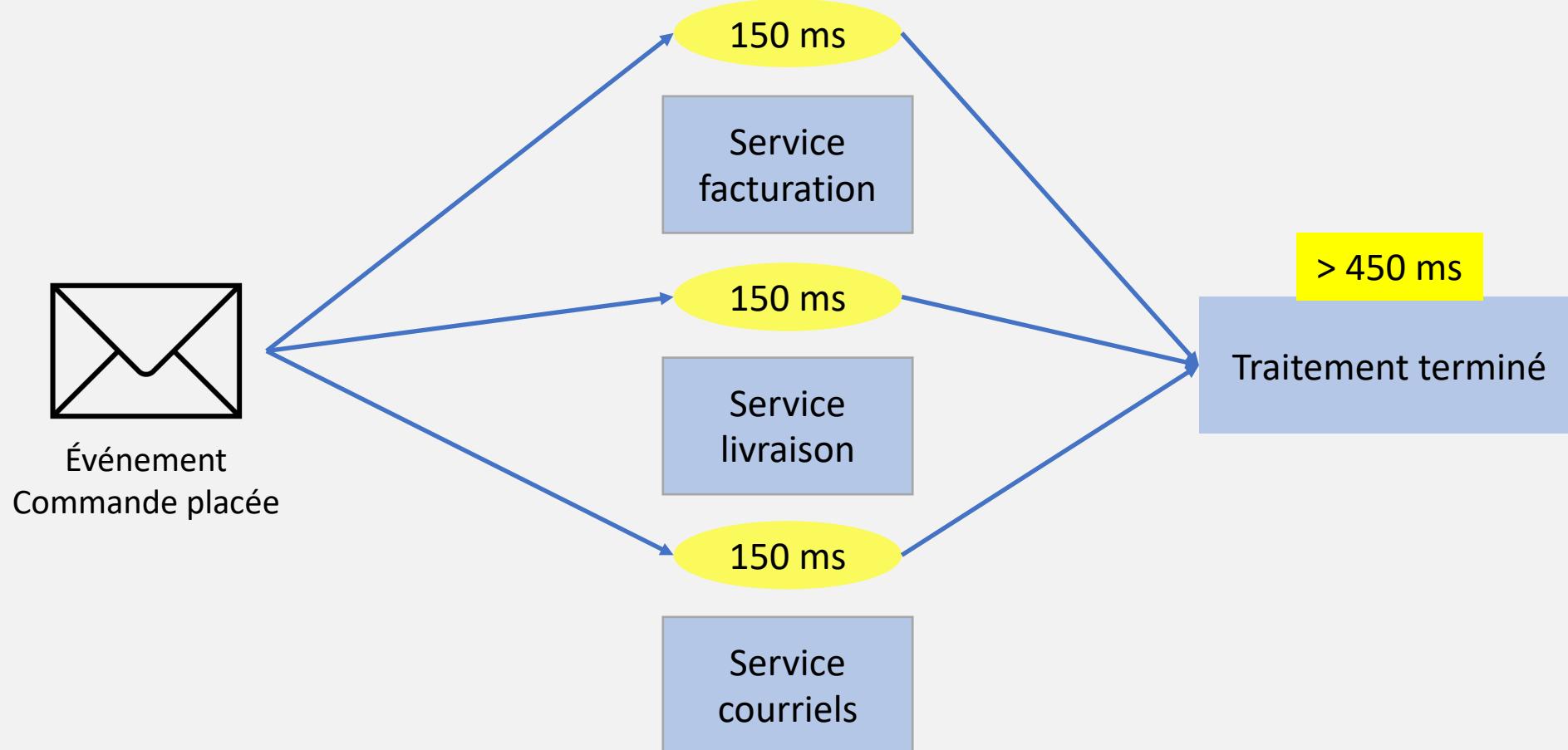
Chorégraphie – Exemple – Pub/sub



Chorégraphie – Exemple – Pub/sub



Chorégraphie – Exemple – Pub/sub



Publish / subscrib (pub/sub)

- Publication de messages dans un courtier de messages (message broker) ou un gestionnaire d'événements (Event bus)
- Abonnement à ces derniers
- Deux types communs de filtrage de messages :
 - **Basé sur le sujet (topic-based)** : séparation logique des messages par rapport à un sujet
 - Basé sur le contenu (content-based) : messages distribué à un destinataire en se basant sur des attributs ou le contenu du message

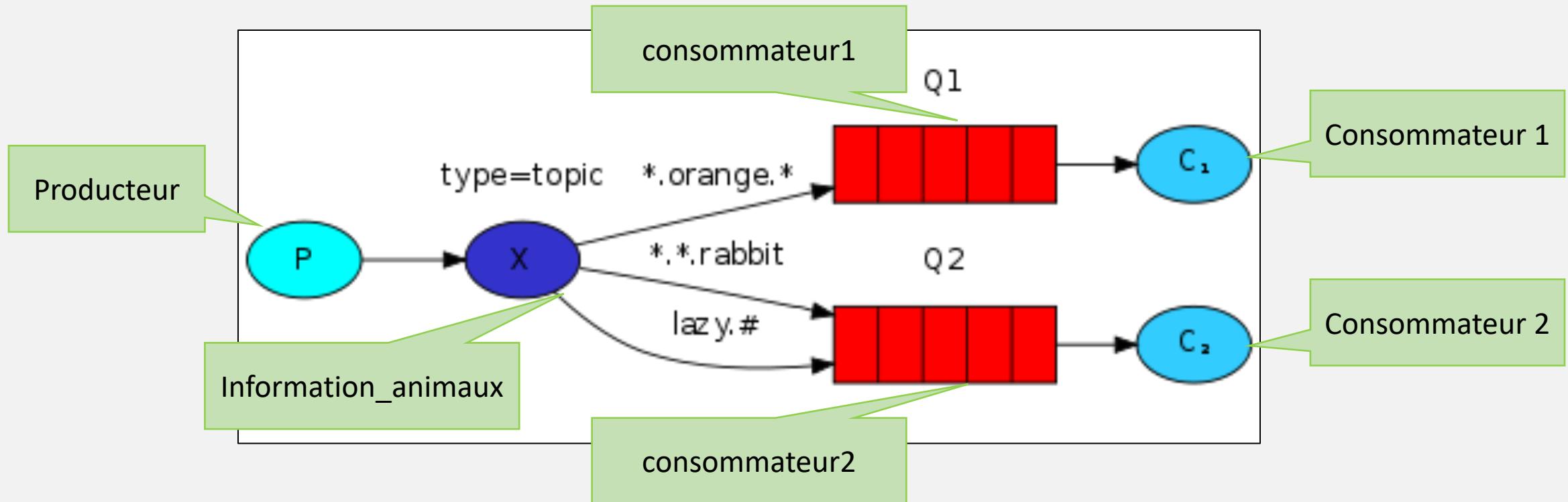
https://en.wikipedia.org/wiki/Publish–subscribe_pattern

Pub/sub avec topic - Rabbitmq

- Publication dans un sujet
- Sujet : liste de mots séparés par un « . »
- Abonnement
 - À un ou plusieurs sujets
 - Possibilité d'utiliser des jetons :
 - « * » : remplace un mot
 - « # » : remplace zéro ou plusieurs mots

Pub/sub avec topic - Rabbitmq

Exemple de la documentation de Rabbitmq : <speed>.<colour>.<species>



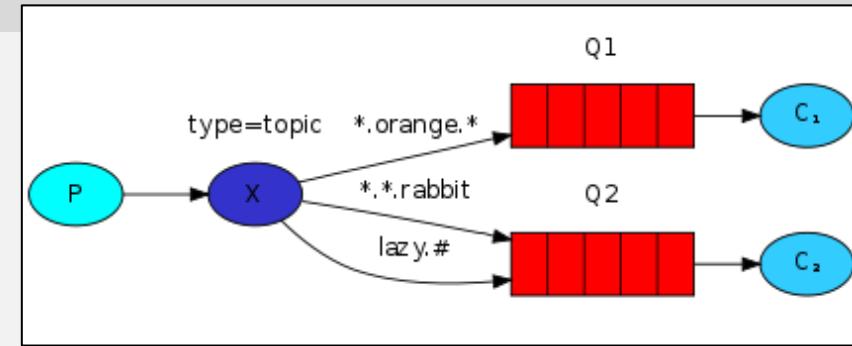
Q1 s'intéresse à tous les animaux oranges, quelque soit la vitesse ou son type

Q2 s'intéresse à tous les lapins et à tout les animaux lent quelque soit sa couleur ou son type

RabbitMQ – Publisher – C#

```
ConnectionFactory factory = new ConnectionFactory() { HostName = "localhost" };

using (IConnection connection = factory.CreateConnection())
{
    using (IModel channel = connection.CreateModel())
    {
        channel.ExchangeDeclare(
            exchange: "information_animaux",
            type: "topic",
            durable: true,
            autoDelete: false
        );
    }
}
```



```
for (int i = 0; i < nombreMessages; i++)
{
    string vitesse = vitesses[rnd.Next(vitesses.Length)];
    string couleur = couleurs[rnd.Next(couleurs.Length)];
    string type = types[rnd.Next(types.Length)];

    string sujet = $"{vitesse}.{couleur}.{type}";
    string message = $"L'animal {type} est {couleur} et est {vitesse}";

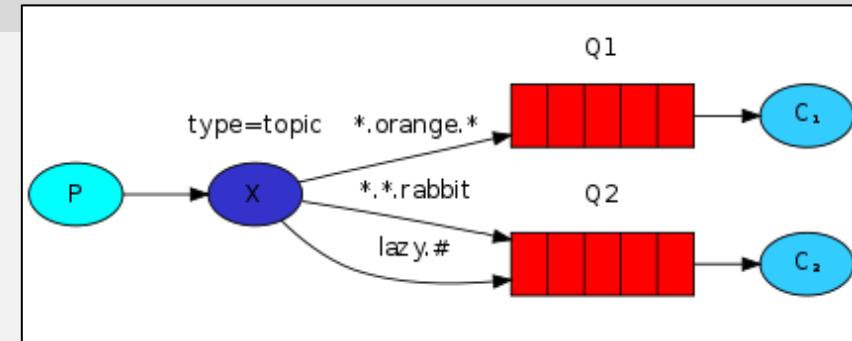
    var body = Encoding.UTF8.GetBytes(message);
    channel.BasicPublish(exchange: "information_animaux",
                         routingKey: sujet,
                         basicProperties: null,
                         body: body);

    Console.Out.WriteLine($"Message \'{message}\' dans le sujet {sujet}");
}
```

RabbitMQ – Subscriber – C#

```
string[] requetesSujets = { "*.orange.*" };
ConnectionFactory factory = new ConnectionFactory() { HostName = "localhost" };

using (IConnection connection = factory.CreateConnection())
{
    using (IModel channel = connection.CreateModel())
    {
        channel.ExchangeDeclare(
            exchange: "information_animaux",
            type: "topic",
            durable: true,
            autoDelete: false
        );
        channel.QueueDeclare(
            "consommateur1",
            durable: false,
            exclusive: false,
            autoDelete: false,
            arguments: null
        );
    }
}
```



```
foreach (var requeteSujet in requetesSujets)
{
    channel.QueueBind(queue: "consommateur1", exchange: "information_animaux",
                      routingKey: requeteSujet);
}
EventingBasicConsumer consommateur = new EventingBasicConsumer(channel);
consommateur.Received += (model, ea) =>
{
    byte[] body = ea.Body.ToArray();
    string message = Encoding.UTF8.GetString(body);
    string sujet = ea.RoutingKey;
    Console.WriteLine($"Message reçu \\"{message}\\\" avec le sujet : {sujet}");
};
channel.BasicConsume(queue: "consommateur1",
                      autoAck: true,
                      consumer: consommateur);

Console.WriteLine(" Press [enter] to exit.");
Console.ReadLine();
}
```

RabbitMQ – Commandes

- Ajoutez « C:\Program Files\RabbitMQ Server\rabbitmq_server-3.8.9\sbin » dans votre PATH
- Reset de votre service RabbitMQ :
 - rabbitmqctl stop_app
 - rabbitmqctl reset
 - rabbitmqctl start_app
- Lister les files de messages : rabbitmqctl.bat list_queues
- Lister les échanges : rabbitmqctl.bat list_exchanges

Références

- <https://fr.wikipedia.org/wiki/Idempotence> : Définition de l'idempotence
- [https://en.wikipedia.org/wiki/Publish–subscribe pattern](https://en.wikipedia.org/wiki/Publish–subscribe_pattern) : Patron publier / s'abonner
- <https://www.rabbitmq.com/tutorials/tutorial-five-dotnet.html> : Tutoriel sur les topics avec RabbitMQ

