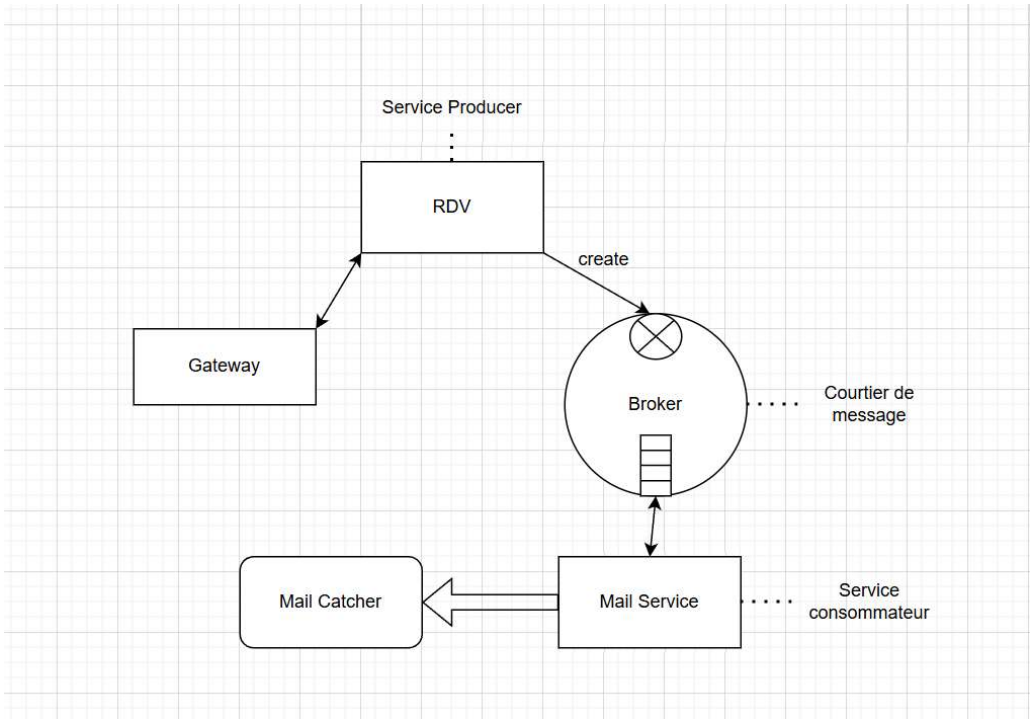
## TD2.3 – Exercice 1

- Faire un schéma identifiant les services concernés sur un plan fonctionnel (service producteur et consommateur) et technique (courtier de message).




- Proposer une configuration du courtier de message : quel type d'exchange, combien de queues, bindings à prévoir. Vous devrez tenir compte pour cela des contraintes indiquées ci-dessus et des évolutions possibles sur l'envoi de messages.

Type d'exchange : on a pensé à choisir un Fanout car c'est flexible et ça permet que le message soit dupliqué vers toutes les queues qui y sont reliées, sans tenir compte d'une clé de routage. Dans ce cas-ci, ça n'a pas d'importance que le message soit redirigé vers toutes les queues car on en a qu'une seule. A l'avenir il y aura plusieurs queues (2, un pour le service de mail et un pour le service de messagerie) donc faire un Fanout peut fonctionner car les deux devront reçevoir le même message.

Queue : 1 seule du coup car un seul service est relié au Broker. Il y en aura d'autres dans le futur avec l'ajout des envois d'SMS.

- Dans le producteur d'événements (microservice rdvs), identifier quel composant ou partie de l'architecture sera en charge de la détection des événements et de leur transmission vers le service d'envoi. Comment faire en sorte que l'on puisse faire évoluer le mode de transmission des évènements (i.e. en utilisant un autre protocole/serveur que AMQP avec RabbitMQ) sans impacter ce producteur d'évènements ?

Dans le microservice rdvs, la responsabilité est partagée selon l'architecture hexagonale. La détection de l'événement se fait dans la couche Application (le Use Case ServiceRdv), juste après la confirmation de la modification en base de données.

La transmission technique est déléguée à la couche Infrastructure via un adaptateur spécifique.

Pour garantir l'évolution du protocole (ex: quitter AMQP pour Kafka) sans impacter le producteur, on utilise le principe d'inversion de dépendance. On définit une interface agnostique (ex: EventDispatcherInterface) dans le "Cœur" de l'application (Ports). Le code métier (ServiceRdv) appelle uniquement cette interface pour notifier un événement. L'implémentation réelle contenant le code RabbitMQ est isolée dans un Adaptateur d'Infrastructure qui implémente cette interface. Ainsi, changer de technologie de transport demande uniquement de créer un nouvel adaptateur et de modifier l'injection de dépendances, sans jamais avoir à modifier le code métier du ServiceRdv.