AMQP avec RabbitMQ

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

ENSEIRB PG306

Interconnexion faiblement couplée

- **MOM : Message-Oriented Middleware**
- Passage de messages entre :
  - Organisations
  - Technologies, plate-formes
  - Temps (asynchronisme)
  - Espace

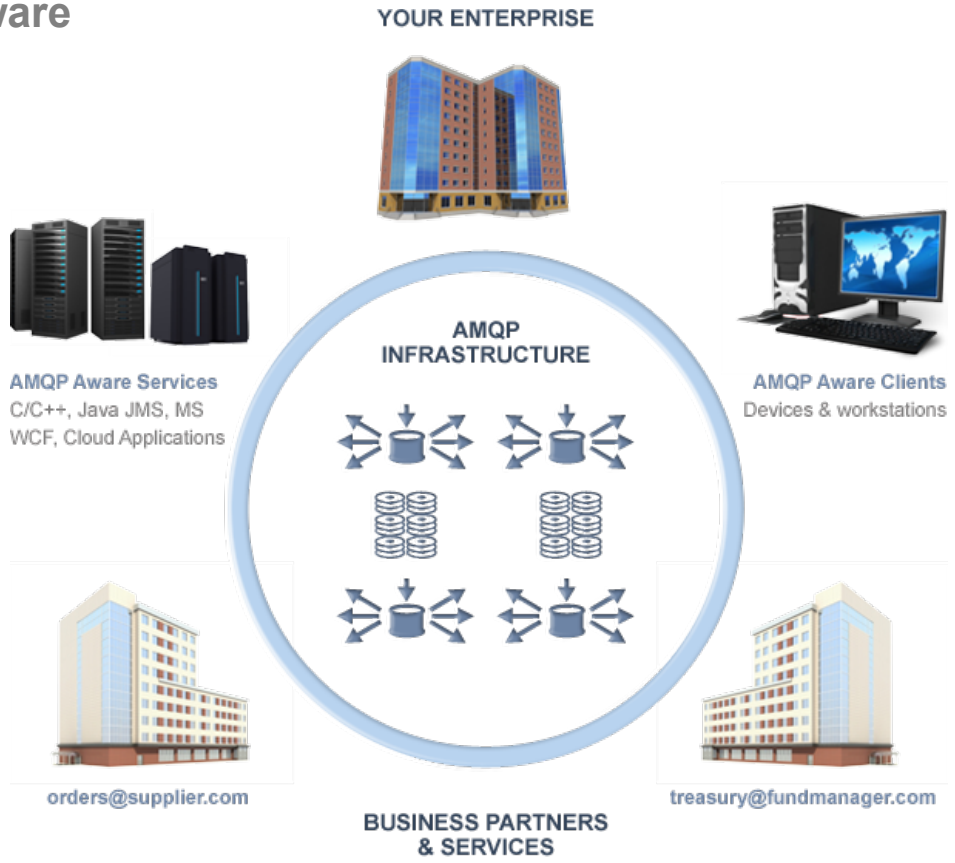


Figure AMQP.org

Le standard AMQP
Advanced Message Queuing Protocol

- À l'initiative de JPMorgan (2004)
  - À l'origine, orientation « business »
  - Cible élargie depuis (finance, logs, contrôle de cloud, IoT, MMO, ...)
    - Utilisateurs : bourse de Francfort, JPMorgan, NASA, VMWare, Mozilla, OpenStack, IBM, etc.
- Workgroup de 23 entreprises
  - Plusieurs drafts successifs, incompatibles entre eux
  - Standard AMQP 1.0 (2011)
  - Normalisation OASIS (2012)
  - Normalisation ISO (2014)

AMQP
Passage de message faiblement couplé

- Standard de passage de message **faiblement couplé**
  - Sécurité
  - Interopérabilité
  - Fiabilité
  - Standard
- Standardisation du protocole sur le fil
  - **Pas de standard d'API**
  - Un serveur (***broker***) et un ***client***
- Implémentations
  - Apache Qpid
  - **RabbitMQ**
  - iMatix OpenAMQP
  - OW2 JORAM
  - Microsoft Azure Service Bus
  - Apache ActiveMQ
  - HornetQ
  - Jboss A-MQ
  - IBM MQ light

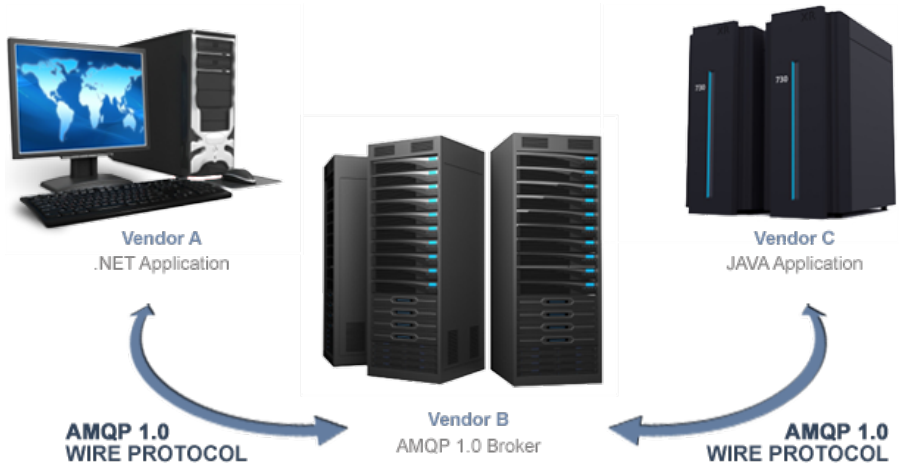


Figure AMQP.org

Architecture AMQP

- Architecture centralisée
  - Un ***broker*** route les messages
  - Les ***clients*** se connectent au broker
- Envoi de messages sur une ***queue***
  - Le broker s'occupe du dispatch de messages
  - Les queues peuvent être ***durables***, i.e. persistantes
  - Modèle producteur/consommateur
  - Modèle publish/subscribe

Exchange, queue, route

- Message publiés sur un ***exchange***
  - Différents types d'exchange pour différentes fonctionnalités
- Routage vers les queues en fonction de règles (***binding***)
- Consommateur connecté à une ***queue***
- Queues habituellement FIFO
  - Mais pas toujours
  - Possibilité de QoS

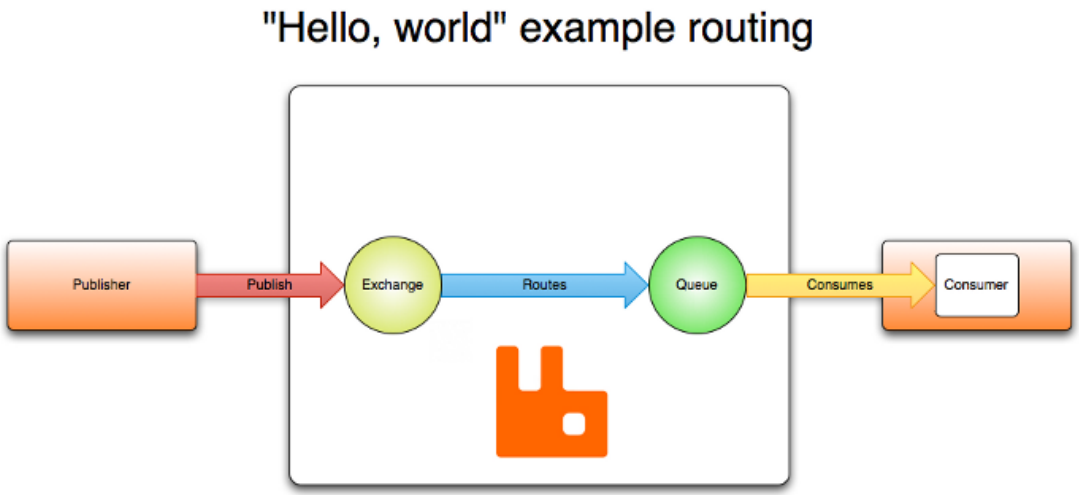


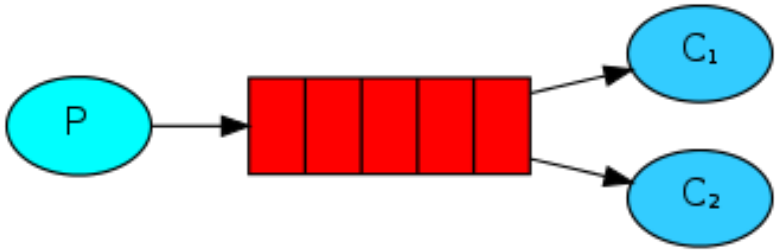
Figure RabbitMQ.com

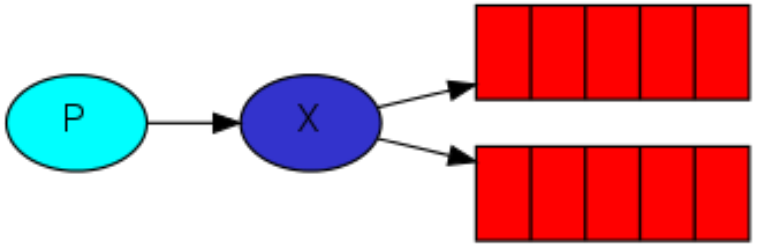
Queues

- Stockent les messages en attente de consommation
- Propriétés :
  - Nom
  - Durable / temporaire : persistante lors d'un reboot
  - Auto-delete : suppression automatique à la déconexion du dernier consommateur
- Déclarées par les clients
  - Erreur de re-déclarer une queue avec un nom existant et des propriétés différentes

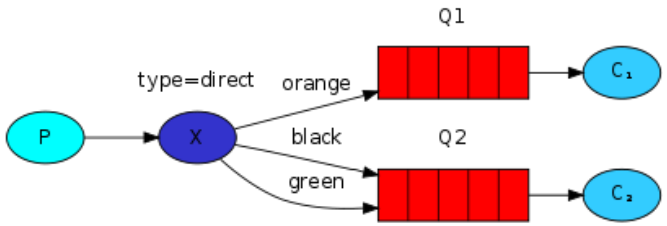
Default Exchange

- Routage par défaut avec un exchange non-spécifié
- Le producteur envoie les paquets directement vers la queue demandée
- Politique round-robin si plusieurs consommateurs se connectent à la queue

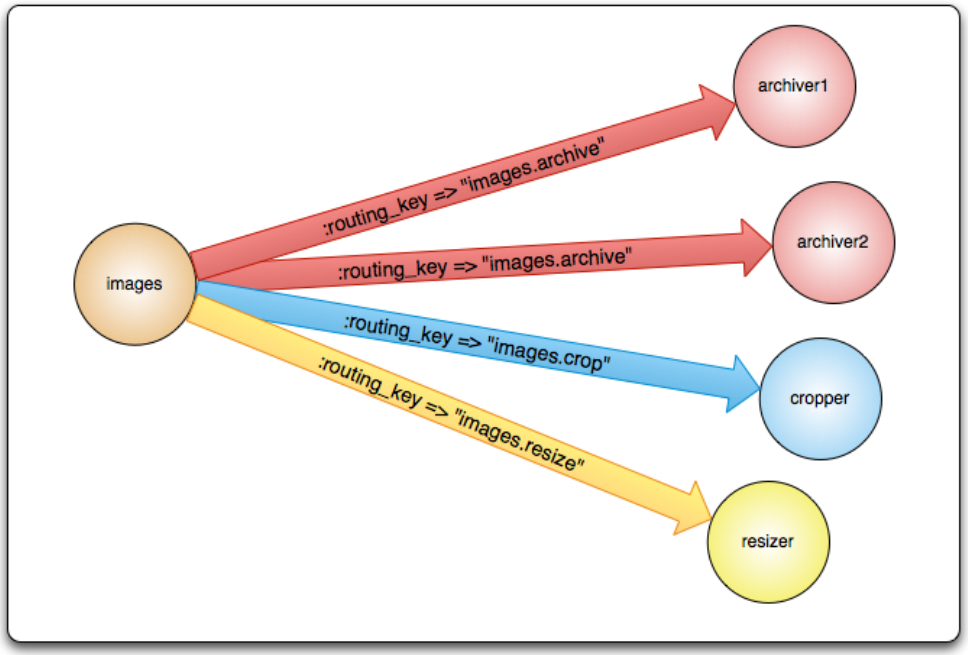




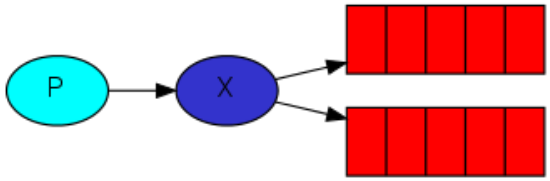
Direct Exchange



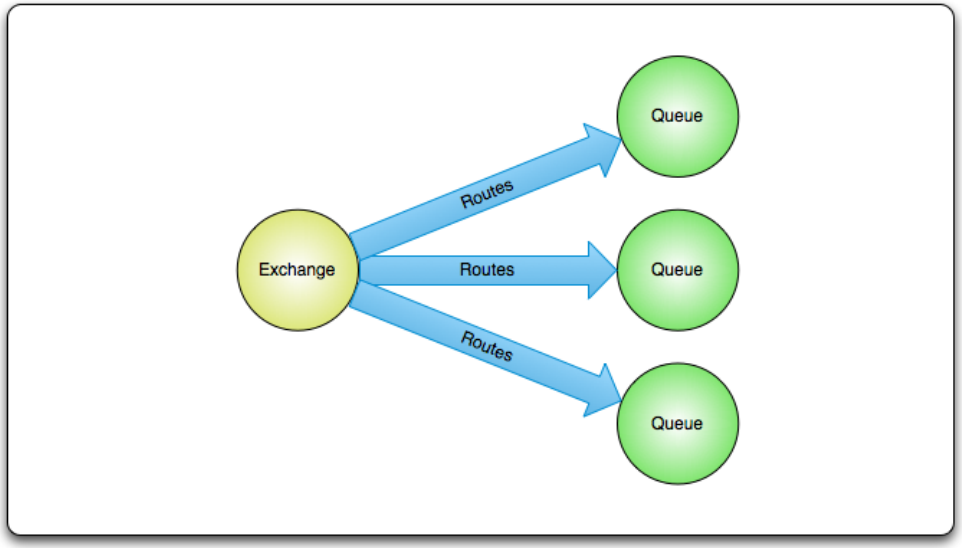
- Chaque queue est liée par un **binding** (contenant une **clef**) à l'exchange
- Un message soumis à l'exchange est diffusé vers toutes les queues dont le binding contient la même clef que le message



Fanout Exchange

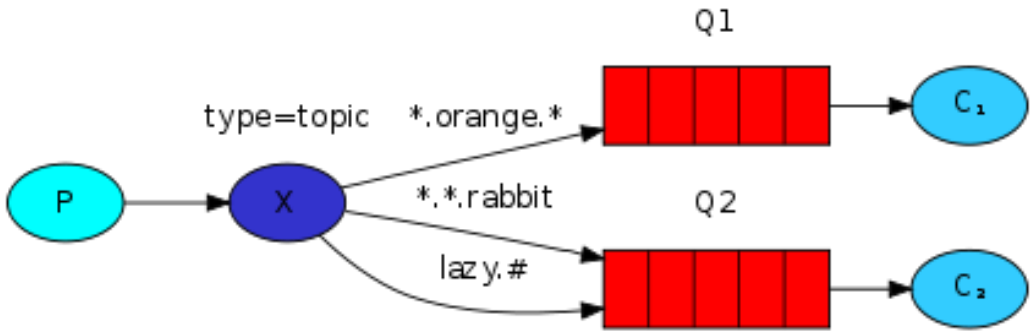


- Chaque queue est liée par un **binding** (sans **clef**) à l'exchange
- Un message soumis à l'exchange est diffusé vers toutes les queues liées à l'exchange
  - Broadcast pour logs, MMO, actualités, etc.



Topic Exchange

- Clef de routage = liste de mots séparés par des points
- Les bindings routent vers les queues en suivant un filtrage
  - (presque) une expression régulière
    - * pour un mot, # pour 0 ou plusieurs mots
- Pour une diffusion contrôlée
  - Données financières, actualités par catégorie, orchestration de cloud, etc.



Robustesse

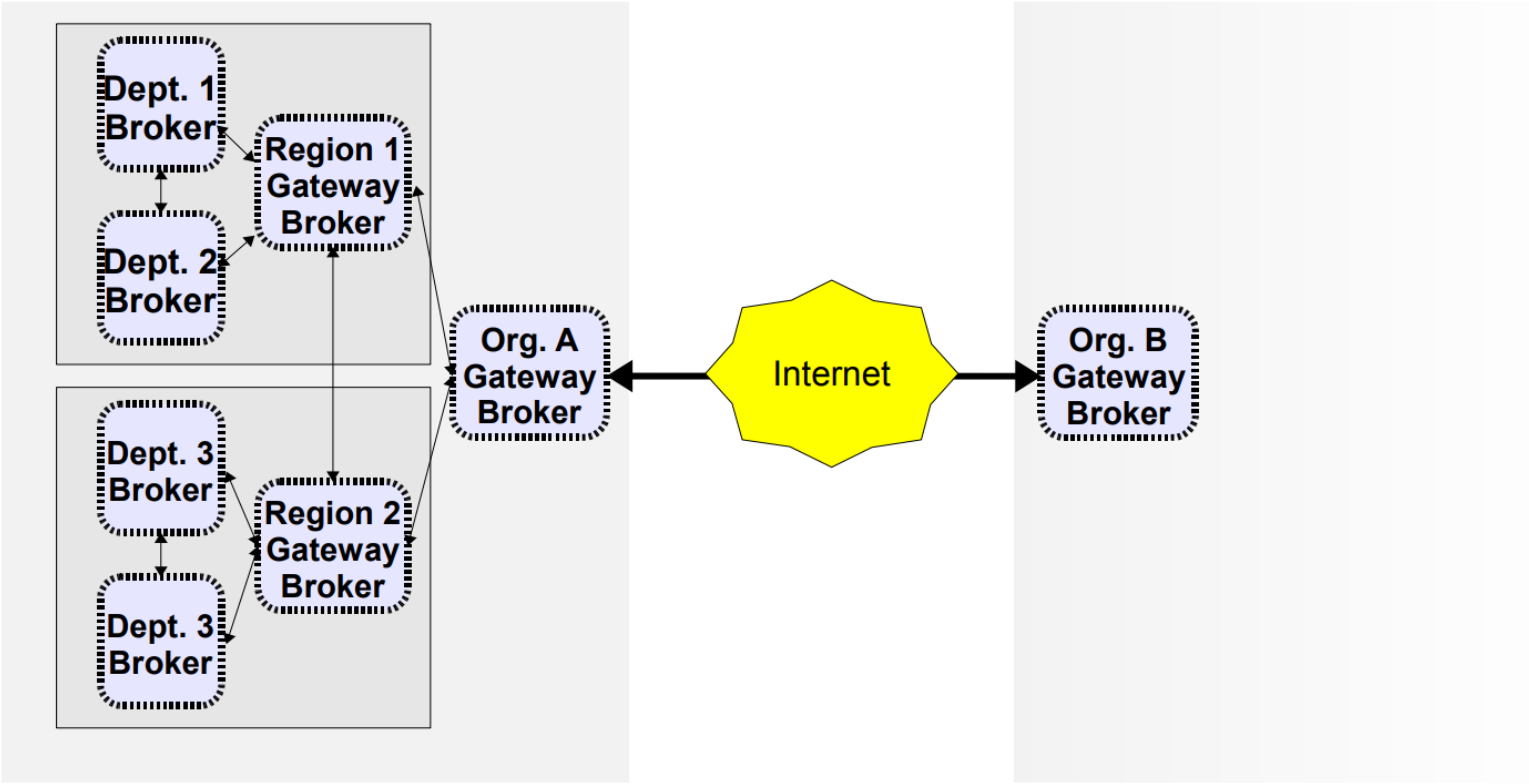
- Possibilité d'accusés de réception
  - Envoi uniquement après traitement de la requête
  - Pas de timeout
  - Message envoyé à un autre consommateur si le consomateur initial se déconnecte sans envoyer d'accusé de réception
- Le broker sauvegarde son état sur disque
  - Uniquement pour les messages persistants dans une queue durable
- Messages stockés dans le broker si pas de clients consommateurs connectés
  -

Sécurité

- User
  - Authentification, chiffrement TLS
  - Souvent : compte guest sans mot de passe, uniquement local
- Vhost : virtual host
  - ~vhost des serveurs web
  - Environnement isolé des autres vhosts

Routage Global

- Interconnexion de brokers



RabbitMQ

- Pivotal software, spin-out de VMWare
  - Introduction en bourse en avril 2018
- À l'origine uniquement AMQP
  - Supporte actuellement en plus STOMP (Streaming Text Oriented Messaging Protocol) et MQTT (Message Queuing Telemetry Transport)
  - Implémentation AMQP-0-9-1 et 1.0
- Broker écrit en Erlang
  - Repose sur Erlang/OTP (Open Telecom Platform)
- Bibliothèque pour clients :
  - Erlang, Java, .NET, Python, Ruby, Javascript, PHP, Go, Objective-C, Swift, etc.

Code producteur Java

```
import com.rabbitmq.client.*;

public class hello_sender
{
  private final static String QUEUE_NAME = "hello";
  public static void main(String[] argv) throws Exception
  {
    /* create connection & channel */
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();
    /* declare queue */
    boolean durable = true;
    channel.queueDeclare(QUEUE_NAME, durable, false, false, null);
    /* send to server */
    String message = "Hello World!";
    channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
    System.out.println(" [x] Sent '" + message + "'");

    /* close channel */
    channel.close();
    connection.close();
  }
}
```

Code consommateur Java

```
import com.rabbitmq.client.*;
import java.io.IOException;

public class hello_recv
{
  private final static String QUEUE_NAME = "hello";
  public static void main(String[] argv) throws Exception
  {
    /* create connection & channel */
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();
    /* declare queue */
    boolean durable = true;
    channel.queueDeclare(QUEUE_NAME, durable, false, false, null);
    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
    /* consume messages */
    Consumer consumer = new DefaultConsumer(channel)
      {
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body)throws IOException {
          String message = new String(body, "UTF-8");
          System.out.println(" [x] Received '" + message + "'");
        }
      };
    channel.basicConsume(QUEUE_NAME, true, consumer);
  }
}
```

Code consommateur avec ACK

```
import com.rabbitmq.client.*;
import java.io.IOException;

public class hello_recv
{
  private final static String QUEUE_NAME = "hello";
  public static void main(String[] argv) throws Exception
  {
    /* create connection & channel */
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();
    /* declare queue */
    boolean durable = true;
    channel.queueDeclare(QUEUE_NAME, durable, false, false, null);
    System.out.println(" [*] Waiting for messages. To exit press CTRL+C");
    /* consume messages */
    Consumer consumer = new DefaultConsumer(channel)
      {
        @Override
        public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body)throws IOException {
          String message = new String(body, "UTF-8");
          System.out.println(" [x] Received '" + message + "'");
          channel.basicAck(envelope.getDeliveryTag(), false);
        }
      };
    boolean autoAck = false;
    channel.basicConsume(QUEUE_NAME, autoAck, consumer);
  }
}
```

Déploiement

- Lancer votre propre broker AMQP
  - Utiliser le `rabbitmq-server` déjà compilé par mes soins

```
export PATH=/net/autre/aldenis/soft/x86_64/bin:${PATH}
export LD_LIBRARY_PATH=/net/autre/aldenis/soft/x86_64/lib:${LD_LIBRARY_PATH}
/net/autre/aldenis/soft/rabbitmq-server/rabbitmq-server/scripts/rabbitmq-server
```

- Pour piloter le broker, utiliser `rabbitmqctl`

```
/net/autre/aldenis/soft/rabbitmq-server/rabbitmq-server/scripts/rabbitmqctl
```

- Pour effacer son état, avec `rabbitmqctl` réaliser la séquence :

```
stop_app
reset
start_app
```

  - Ou plus simplement :
    - Arrêter le broker
    - Supprimer ${HOME}/var/lib/rabbitmq
    - Redémarrer le broker

Travail à faire

Échauffement

- Compiler et faire tourner les exemples
  - En local
  - À distance
  - Avec plusieurs producteurs / consommateurs
  - Avec queue durable / temporaire

Retour sur le calcul distribué

- Reprendre le calcul de pi (c.f. TP1 Java RMI)
  - Adapter le code pour distribuer le travail avec RabbitMQ
- Quel modèle est le plus simple entre AMQP et Java RMI
  - pour l'asynchronisme ?
  - pour le typage de données ?

Retour sur l'application bancaire

- Reprendre le code de l'application bancaire (c.f. TP2 CORBA)
- Remplacer les échanges CORBA par de l'AMQP
  - n'oubliez pas
    - l'asynchronisme
    - les accusés de réception
    - les transactions
  - Conclusion ?

À vous de jouer !

http://dept-info.labri.fr/~denis/