TP 3 - Messaging

But du TP

- Manipuler RabbitMq depuis sont plugin management
- Comprendre le Clustering
- Comprendre la Haute Disponibilité dans rabbitMq
- Mettre en place spring-amqp-template
- Ecouter des évènements dans Spring-Boot-

Lancer un cluster rabbitmq

TODO 01 :

Aller dans broker/cluter/cluster-1_2

Démarrer les 2 premiers noeuds :

```
docker-compose up
```

Aller dans broker/cluster/node3

```
docker-compose up
```

Quand le cluster est démarré aller voir l'ihm fournie par le plugin management.

TOPOLOGIE du Cluster

Serveur	TCP port	Management port	Config sur Disque
rabbit1	5762	15762	OUI + stats
rabbit2	5763	15673	NON
rabbit3	5764	15764	OUT

Manipulations de base du cluster

Configuration générale :

il y a des onglets :

- `Overview` : état du cluster et traffic sur ce dernier
- `Connections` : état des connexions sur le noeud
- `Channel` : les channels sont un peu l'équivalent de la session en sql ce sont eux qui permettent de communiquer avec le broker pour recevoir ou envoyer des messages
- `Exchanges` : destinations logiques desP messages pour les publishers, permet de créer des exchanges, les configurer et envoyer du traffic
- `Queues` : réceptacles physiques des messages - permet créer des queues faire des bindings et envoyer/consommer des messages depuis la queue
- `Admin` : permet de faire de l'admin en particulier créer des utilisateurs et de gérer les mirrored queues pour faire de la Haute dispo
- `Visualiser` : nouveauté rabbitmq 3.6.x qui permet de visualiser la topologie du broker en particulier les bindings

T0D0 02 :

Aller sur http://localhost:15762/

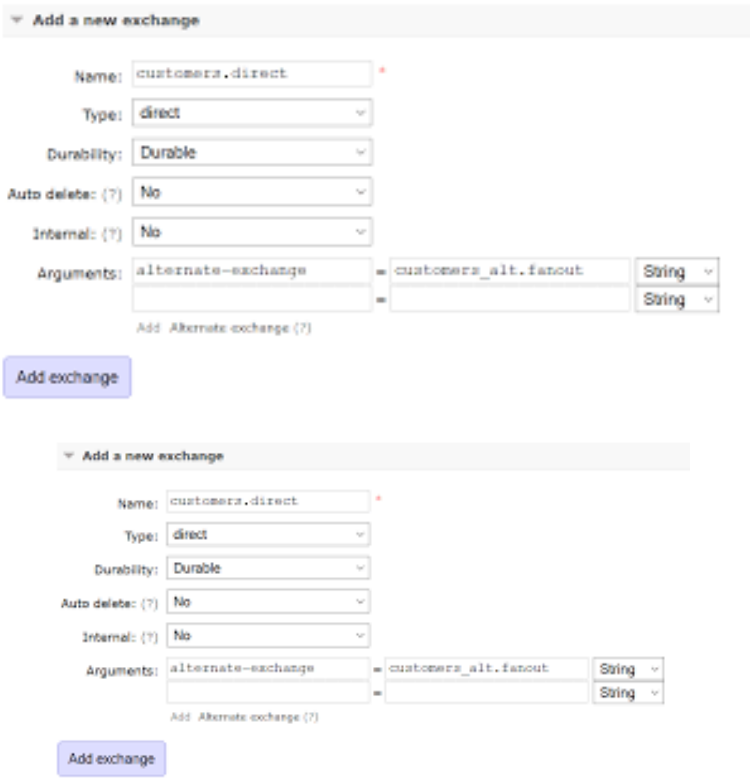
Entrer les login/pwd : *guest/guest* `(Défaut sur rabbitmq -> penser à changer cela sur de la prod)`

Regarder l'affichage.

Création d'un exchange

TODO 03

Aller dans l'onglet Exchanges

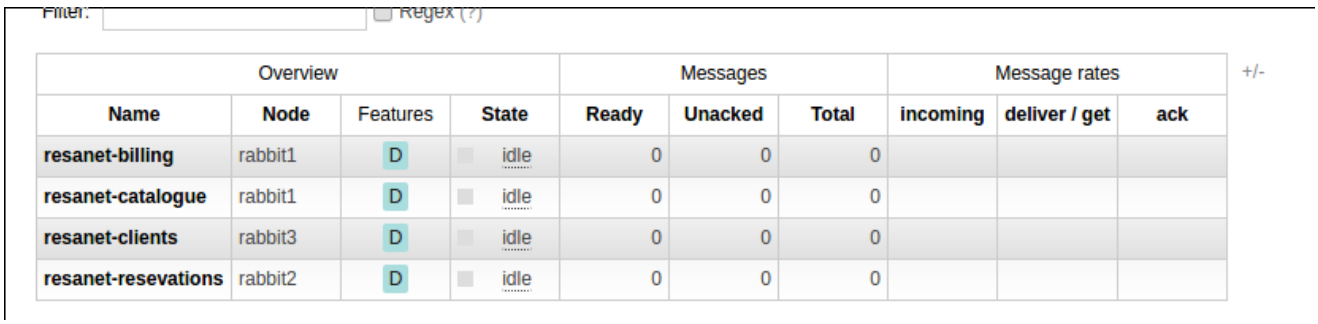


Déplier > add new exchange

Appeler l'exchange ** resanet-fanout **, il doit avoir les propriétés suivantes :

```
Name : resanet-fanout
Durability : Durable
Auto Delete : No
Internal : No
Arguments <ne rien mettre>
```

Ajout d'une queue sur un exchange de type fanout



Dans features, il y a un **D** ce qui signifie Durable c'est à dire qu'on peut éteindre le noeud, au redémarrage de celui-ci la queue existera encore mais :

- sera vide si les messages sont non persistants
- possèdera les messages déjà dessus s'ils sont persistants

TODO 04

Aller dans l'onglet Queues

On va créer une queue par application :

- `resanet-catalogue` (noeud 1)
- `resanet-resevations` (noeud 2)
- `resanet-clients` (noeud 3)
- `resanet-billing` (noeud 1)

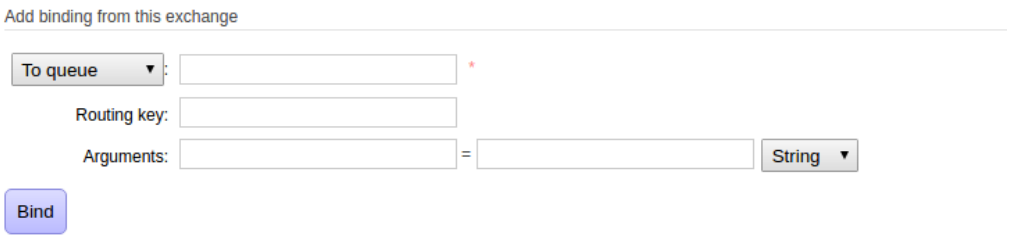
```
Name :
Durability : Durable
Node : rabbit@rabbit1
Auto Delete : No
Arguments : <vide>
```

Binding d'une queue sur un exchange de type fanout

TODO 05

Retourner sur l'onglet **Exchange**

Dans le tableau qui liste les exchanges cliquer sur **resanet-fanout** le détail de l'exchange arrive sur la page spécifique de l'exchange.



Comme il s'agit d'un **fanout** *il n'y a pas de routing key ni d'arguments à mettre*.

juste mettre le nom des 4 queues précédentes et faire Bind.

Envoi d'un message sur un fanout

TODO 06

Sur le même onglet aller plus bas dans **Publish Message**.

Vu qu'on publie sur un fanout, on n'a pas besoin ni de routing key ni de header.

Publier un message sur l'exchange.

Aller lire un message

TODO 07

Dans l'onglet **Queues**, regarder dans le tableau le nombre de messages dans chacune des queues crées.

Puis faire ce qui suit pour chacune des queue crée :

- Aller sur la queue
- aller sur le sous menu **Get Messages**

Qu'en déduire sur le fanout ?

Les exchanges de type topic

Ils sont idéaux pour écouter des évènements !

TODO 08

Créer un exchanges de type topic appelé :

```
- reservation.out
```

Faire les bindings suivants :

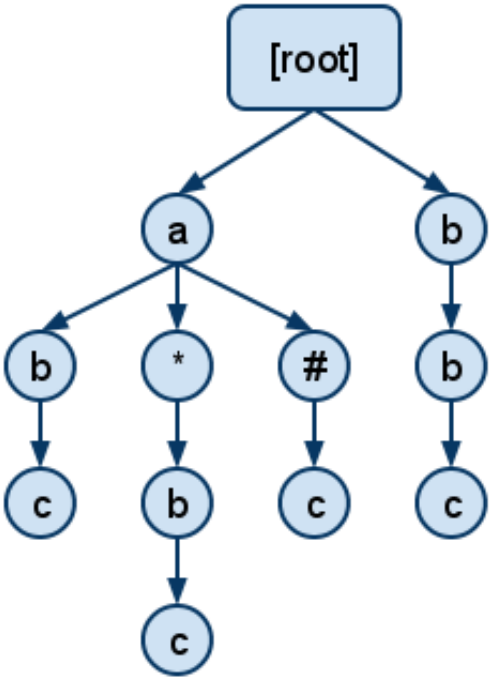
Rappel :

Un binding consiste à relier un exchange à une queue via une routing_key.

- `resanet-billing : reservation.*.*` => la facturation doit prendre en compte les annulations - modifications
- `resanet-clients : reservation.*.added` => l'appli pour les client ne s'intérressent qu'aux achats

Nota Bene : Comment fonctionnent les expressions :

- on peut découper avec des `.`
- `*` path récusif au travers des `.`
- `#` path non récursif au travers `.`



Envoyer des messages pour un topic envoyer :

- avec la routing key suivante `reservation.avion.added`

```
{
  "date" : "12/03/2016",
  "customer" : {"id":"AZ456VN","firstname":"Gérard","lastname":"Dupont",
 "gender":"Mr"},
  "company":"Air France",
  "from":"London","to":"Paris",
  "class":"A","seat":"B5",
  "price":"260","currency":"EUR",
  "billing_address":{
    "name":"Monsieur Gérard Dupont",
    "number":"10","street":"rue de Milan",
    "zip_code":"75009","city":"Paris", "country":"France"
  }
}
```

- avec la routing key suivante `reservation.train.added`

```
{
  "date" : "15/03/2016",
  "customer" : {"id":"BD456VN","firstname":"Alice","lastname":"Gensac",
"gender":"Miss"},
  "company":"Air France",
  "from":"Paris","to":"Sydney",
  "class":"A","seat":"C8",
  "price":"1260","currency":"EUR",
  "billing_address":{
    "name":"Mademoiselle Alice Gensac",
    "number":"84","street":"rue de Rivoli",
    "zip_code":"75001","city":"Paris", "country":"France"
  }
}
```

- avec la routing key suivante `reservation.train.canceled`

```
{
  "date" : "12/03/2016",
  "customer" : {"id":"AE487CF","firstname":"Pierre","lastname":"Legrand"
, "gender":"Mr"},
  "class":"A","seat":"72",
  "price":"-160","currency":"EUR",
  "reservation_id":"SNCF-75643-456"
}
```

- avec la routing key suivante `reservation.avion.modified`

```
{
  "date" : "13/03/2016",
  "customer" : {"id":"AE487CF","firstname":"Pierre","lastname":"Legrand"
, "gender":"Mr"},
  "class":"A","seat":"B5",
  "price":"+45","currency":"EUR",
  "reservation_id":"SNCF-B5643-456",
    "new_travel":{
    "company":"Thello",
    "from":"Paris","to":"Rome",
    "class":"A","bedroom":"12-5-2",
    "reservation_id":"THEL-AI7895-1252"
    "difference":"-154","currency":"EUR"
  }
}
```

Clustering comprendre le concept d'owner

Le clustering ne fait que de **la répartion de charge**, il n'assure aucunement la Haute

Disponibilité du broker.

Quand on crée une file on le fait sur un noeud donné, ainsi, le comportement est le suivant :

- on peut consulter les messages du noeud N depuis tous les autres noeuds du cluster (la requête est dirigée en tcp vers l'owner)
- si l'owner tombe, la file n'est plus accessible par les autres noeuds

TODO 09

Eteindre le noeud 3 dans `broker/cluster/node3`

```
docker-compose stop
```

Aller sur l'overview du plugin de management et vérifier que le noeud 3 soit down.

Essayer d'écrire sur la file `resanet-clients` , que se passe-t-il ?

Envoyer un message sur l'exchange `esanet-fanout`

Redémarrer le noeud 3 dans `broker/cluster/node3`

```
docker-compose start
```

Aller voir s'il y a des messages dans la queue **resanet-clients**

Qu'en déduire :

- du sort des messages reçus avant ?
- de la capacité du cluster en envoyer un message sur une file dont l'owner est down ?

Comprendre le concept de persistance des messages

L'exercice précédent a permis de voir que si les messages ne sont pas persistants, ils ne survivent pas à un arrêt relance du broker.

Nous allons ici envoyer un message persistant et vérifier qu'il survit à un arrêt relance de son noeud *owner*

TODO 10

Sur l'exchange **resanet-fanout** envoyer un message comme suit :

- propriétés
- `Routing key` :
- `Delivery Mode` : `2 /- Peristant`
- `Payload` :

```
{
  "information":"Cancelation",
  "transport_id":"AF-45687-7897",
  "date":"14/11/2015"
}
```

Faire comme précédemment pour éteindre et rallumer.

Qu'en est-il du message après le restart ?

Comprendre le concept de noeuds RAM vs DISK

La configuration est stockée dans la base **MNESIA** quand le noeud est démarré avec l'option --disk ou pas d'options (le disque est le défaut). Le premier noeud du cluster doit être de type *DISK*. Les noeuds de type *RAM* sont plus performants mais leur configuration ne survit pas à un arrêt/relance.

TODO 11

Ajouter un noeud 4 de type *RAM* au cluster.

Mettre en place de la haute disponibilité

La haute disponibilité se fait par les **mirrored-queues**. Elles se configurent par des policies via :

- rabbitmqct
- l'API REST de management
- IHM de management dans la vue ADMIN

Une policy fonctionne toujours de la même façon :

- on utlise une expression régulière pour matcher les ressources (ici queues)
- on sette un paramètre que correspond à la policy et éventuellement des configurations qui correspondent à ce dernier

Pour la haute disponibilité :

Ha Mode (ha-mode)	Valeurs de Ha Params (ha-params)
all	NEANT
exactly	**3** [Nombre de noeuds où mirorer]
nodes	**rabbit1 rabbit2** [Liste avec le nom des noeuds]

Si le noeud tombe, quand il revient il faut synchroniser toutes les files qu'il mirore, cela peut se faire via :

- rabbitmqctl
- l'API de management
- un bouton synchronise sur la vue de la file

Le noeud sur lequel est créé la file devient alors **master**

TODO 12

Appliquer la policy de HA à tous les noeuds du cluster :

- `Name` : `HA`
- `Pattern` : `.*`
- `Definition` : `ha-mode` = `all`

Aller sur la vue des queues, dans la colonne features un picto indique le nom de la police et le **+2** dans la colonne **Node** indique que les queues sont mirrorées par 2 noeuds en plus du master.

Sur l'exchage **resanet-fanout** envoyer un message comme suit :

- propriétés
- `Routing key` :
- `Delivery Mode` : `1 /- Non Peristant`
- `Payload` :

```
{
  "information":"Cancelation",
  "transport_id":"AF-45687-7897",
  "date":"14/11/2015"
}
```

Eteindre le noeud 3

Aller chercher le message sur la file **resanet-client**

Qu'en déduisez-vous ?

Regarder le nombre de mirrors sur la file du noeud éteint.

Redémarrer le noeud 3

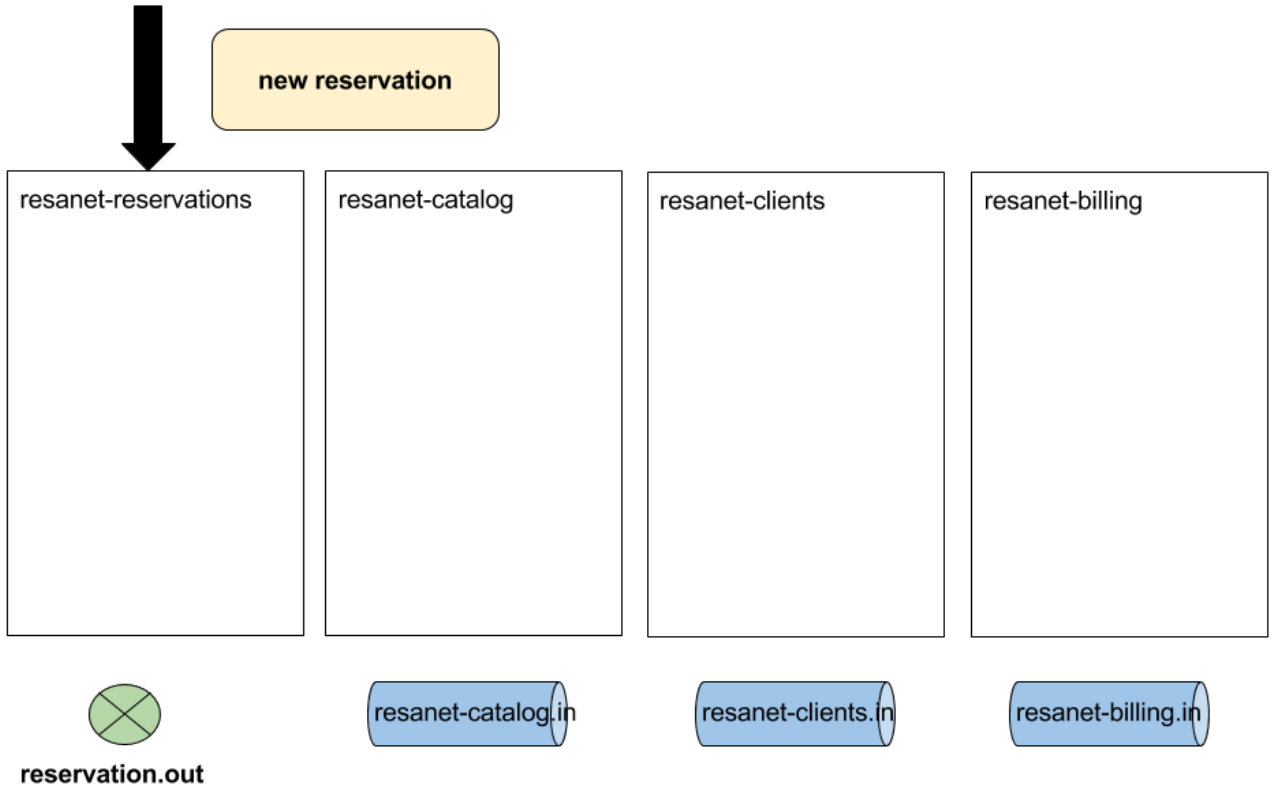
Regarder la vue de la file, dans slave, le noeud **rabbit3** est **(unsynchronised)**, pour le remettre en état appuyer sur synchronise.

Retourner sur la vue Queues, qu'en est-il du nombre de noeuds mirroirs ?

** ---------- FIN PARTIE MANIPULATION RABBITMQ ---------- **

Partie 2 : échanger entre Bounded-Context via des Bounded-Events

Le but est de montrer comment avec des topics on peut facilement échanger des messages entre microservices.



Estimated Time : **30** *minutes*

Mettre en place l'envoi

Aller dans **resanet-reservations**

TODO 01

Ajouter le support spring data rest pour avoir les Handler

dans pom.xml et ajouter **spring-data-rest-webmvc**

```
        <dependency>
          <groupId>org.springframework.data</groupId>
          <artifactId>spring-data-rest-webmvc</artifactId>
        </dependency>
```

TODO 02

Dans resanet-reservations :

dans le package listeners, il y a un eventHandler :

il faut l'annoter

```
@RepositoryEventHandler
```

Annoter les méthodes :

```
    @HandleAfterCreate
    public void objectCreateEvent(Object o){
        ...
    }
    @HandleAfterSave
    public void objectModifiedEvent(Object o){
        ...
    }
    @HandleAfterDelete
    public void objectDeletedEvent(Object o){
        ...
    }
```

TODO 03

Ajouter le repositoryHandler à la configuration

Dans Application.java

```
@Import(RepositoryConfiguration.class)
```

Dans la configuration ajouter le handler comme bean utiliser l'annotation ci-dessous

```
@Bean
```

TODO 04

Tester avec Swagger-ui

Exemple de payload :

```
{
  "billingAdress": {
    "city": "paris",
    "country": "france",
    "name": "olivier laporte",
    "number": "57",
    "street": "ouest",
    "zipCode": "75014"
  },
  "currency": "EUR",
  "customer": {
    "firstName": "Olivier",
    "gender": "Mr",
    "id": "AZER4564",
    "lastName": "Laporte"
  },
  "date": "2016-04-15T12:16:29.167Z",
  "place": {
    "classe": "A",
    "seat": "1234"
  },
  "price": "2564",
  "transportId": "SNCF-7548-115"
}
```

Vérifier la bonne insertion en base.

Utiliser get derrière

TODO 05

Ajouter les dépendences pour Spring-amqp template

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.amqp</groupId>
            <artifactId>spring-rabbit</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.amqp</groupId>
            <artifactId>spring-amqp</artifactId>
        </dependency>
```

Ajouter le driver rabbitmq

```
        <dependency>
            <groupId>com.rabbitmq</groupId>
            <artifactId>amqp-client</artifactId>
            <version>3.4.0</version>
        </dependency>
```

TODO 06

Configurer l'amqp-template

Ajouter le fichier de config au scan de l'application context

```
@ImportResource("<!-- Fichier -->")
```

Configurer l'amqp template

```
<rabbit:connection-factory id="connectionFactory"/>
    <rabbit:template id="amqpTemplate" connection-factory="connectionFac
tory"/>
    <rabbit:admin connection-factory="connectionFactory"/>
```

TODO 07

Ajouter un exchange pour envoyer les évènements

ex fanout-exchange

nom : reservations.out

```
    <rabbit:{mettre le type d'exchange ici} name="amq.fanout">
        <rabbit:bindings>
            <rabbit:binding queue="<!-- nom de la queue ici -->"/>
        </rabbit:bindings>
    </rabbit:fanout-exchange>
```

TODO 08

Ajouter une queue appelée **billing.in**

```
<rabbit:queue id="springQueue" name="<!--nom de la queue ici-->" auto-de
lete="true" durable="false"/>
```

TODO 09

Utiliser l'AmqpTemplate dans le ReservationHandler

```
    @Autowired
   private AmqpTemplate template;
```

Utiliser l'amqpTemplate

```
template.convertAndSend({exchange}, {routingKey}, {message});
```

TODO 10

Ajouter un message Handler dans la conf spring-amqp

```
    <rabbit:listener-container connection-factory="connectionFactory">
        <rabbit:listener queues="springQueue" ref="messageListener"/>
    </rabbit:listener-container>
```

TODO 11

Configurer le message Handler

implémenter la méthode onMessage

** ---------- FIN PARTIE Envoi sur récaption d'event RABBITMQ ----------**