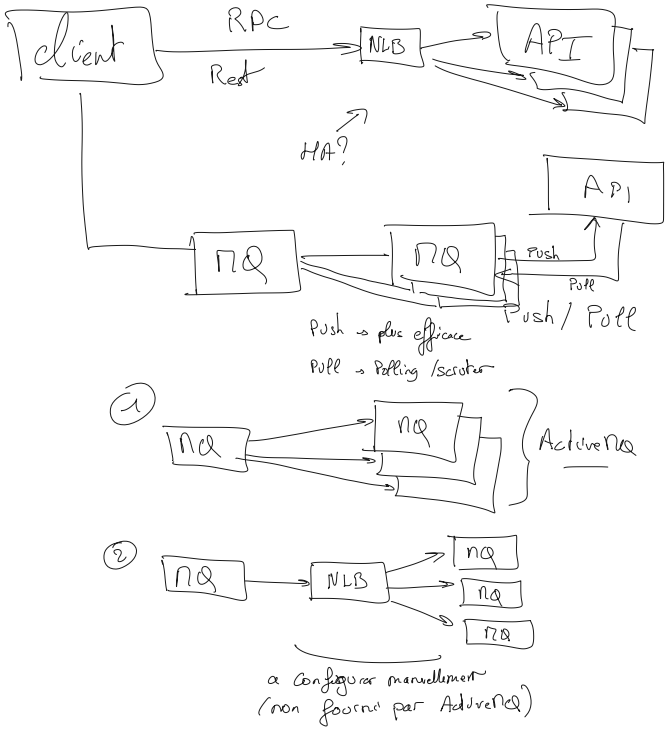
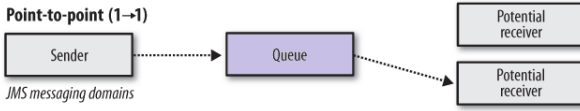
Bonjour Tout le Monde





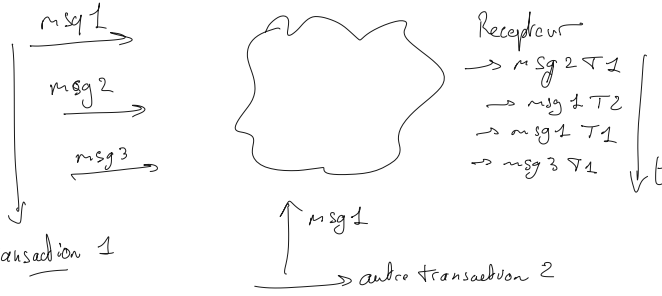
First In First Out

Last In First Out

FIFO

LIFO

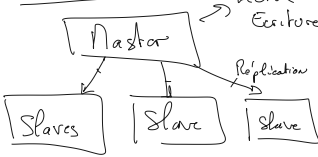
Corrélation = reconstituer des msgs ordonnés d'une transaction



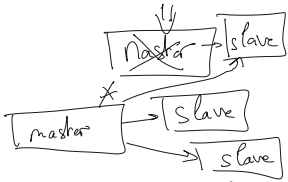
Haute Disponibilité

Monde Traditionnel / Relationnel

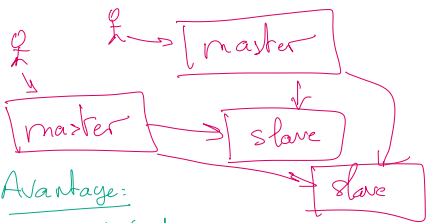
Haute Dispo



- En cas de perte du master
  → on bascule un slave en master



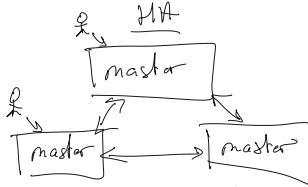
A proscrire (Split Brain) :



Avantage :
Cohérent

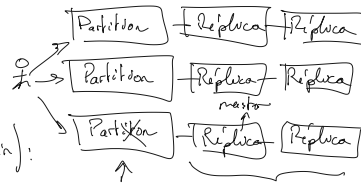
Monde "Moderne" Big Data

HA

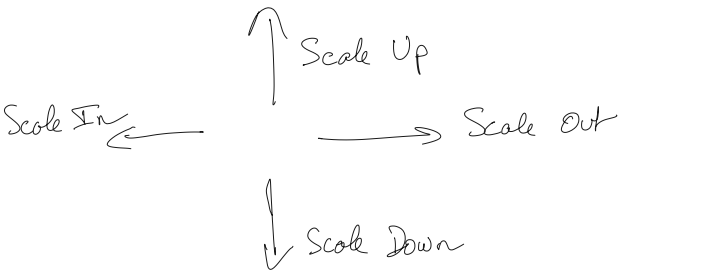


⇒ environnements multi-maîtres
→ Répartition de charge
→ Tolérance de Panne -

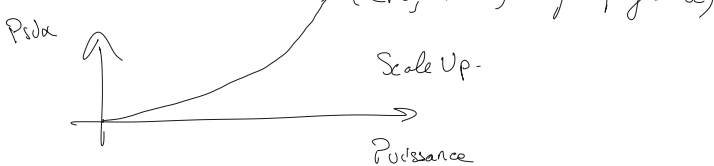
HA :



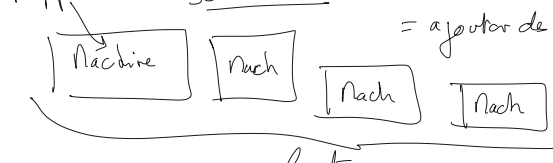
Mise à l'échelle



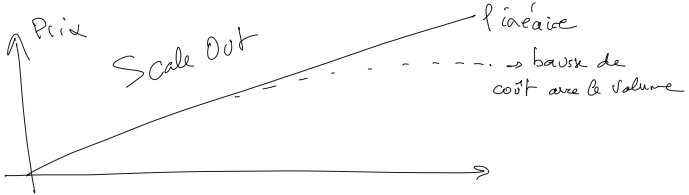
Scale Up: remplacer une machine par une plus puissante (CPU, RAM, disque performance)



Scale out Montée en charge = ajouter de machines



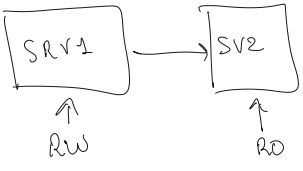
3 Machines = 3x + puissant que 1 Machine



1er avantage = rationalisation des coûts

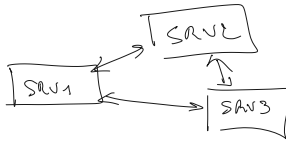
2nd avantage = Haute Dispo (plusieurs machines)

Traditionnel



Scale Up

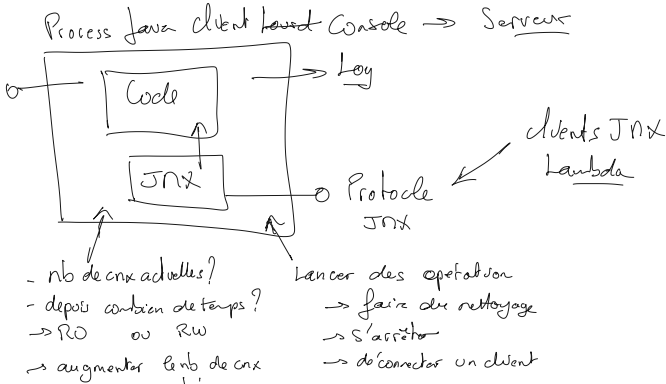
Big Data

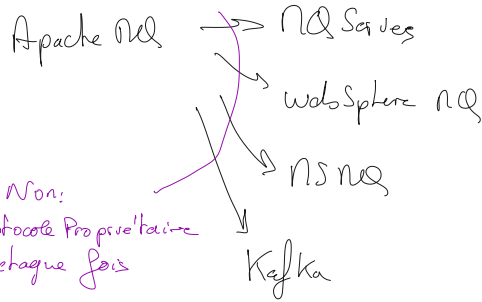


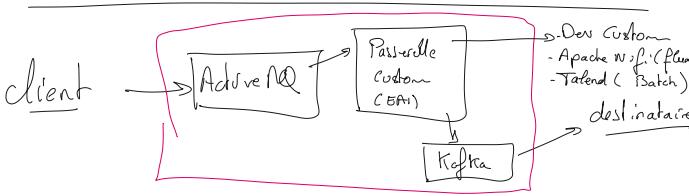
⇒ Scale Out

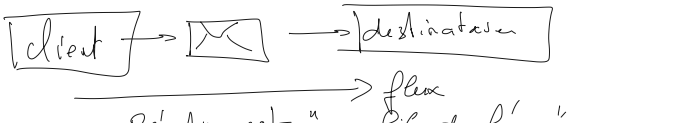
Brokers

JMX


Process Java client based Console → Serveur
Code
Log
JMX
Clients JMX
Lambda
Protocole
JMX
- nb de cnx actuelles?
- depuis combien de temps?
→ RO ou RW
→ augmenter le nb de cnx acceptées.
Lancer des opérations
→ faire du nettoyage
→ s'arrêter
→ déconnecter un client


Apache MQ
MQ Series
WebSphere MQ
MS MQ
Non:
Protocole Propriétaire
à chaque fois
Kafka


client
ActiveMQ
Passerelle
Custom
(EAI)
Kafka
- Des Custom
- Apache Nifi (Flow)
- Talend (Batch)
destinataire



Passerelle : Batch → Traditionnel → ETL
- Simple
- un lot traité régulièrement

+ Simple

flux → - en Temps réel → ELT
- Compatible gros volume
- Traite au fil de l'eau
- plus complexe à implémenter

Différé

Architecture proposée :

/opt/apache/activemq/<version>/ bin, etc, ... ← exec

data :

/var/amq/ data/, broker1/, SRV2/, ... → data

datadir :

Objectif : n Versions / instances de activemq et Updates possibles

Installer ActiveM%Q en tant que Daemon et gestion de la conf :
http://activemq.apache.org/unix-shell-script

Lancer en spécifiant un conf souhaitée :
bin/activemq console xbean:/opt/apache/amp/5.16.3_1/conf/activemq.xml

Envoyer / Recevoir des messages :
${ACTIVEMQ_HOME}/bin/activemq producer
${ACTIVEMQ_HOME}/bin/activemq consumer

java -jar activemq-all-5.x.x.jar producer
java -jar activemq-all-5.x.x.jar consumer

bin/activemq producer --message "My message" --messageCount 1

Topic durable (inscription) :
bin/activemq consumer --durable true --clientId example --destination topic://TEST

Exercices d'implémentation :
Créer un abonnement à un Topic et garder en ligne
Envoyer des messages, les recevoir

Créer un abonnement durable, se déconnecter
envoyer des messages
Les recevoir

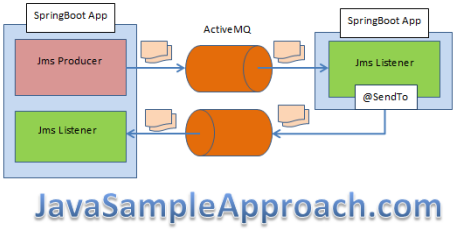
Créer un abonnement non durable, se déconnecter
envoyer des messages
Tenter de les recevoir = ?

Le **risque** pour un émetteur qui **envoie des messages non persistants** :
Que ces messages soient perdus si le serveur qui les stocke s'arrête car les messages ne seront qu'en RAM, pas sur le disque.

Est-ce que l'émetteur peut être informé si un message non persistant est reçu ou non ?

ActiveMQ ne permet pas l'accusé de réception.

Il faut donc mettre en oeuvre une autre file en callback pour en être informé.



Chaînes de connexions :

https://activemq.apache.org/uri-protocols

failover:(tcp://foo:61699,tcp://bar:61617,tcp://whatnot:61698)

Paramètres : http://activemq.apache.org/connection-configuration-uri

tcp://localhost:61616?trace=true

static:(tcp://host1:61616,tcp://host2:61616)

```
<transportConnectors>
        <transportConnector name="openwire"
uri="tcp://0.0.0.0:61626?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
        <transportConnector name="amqp"
uri="amqp://0.0.0.0:5682?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
        <transportConnector name="stomp"
uri="stomp://0.0.0.0:61623?maximumConnections=1000&;wireFormat.maxFrameSize=104857600"/>
        <transportConnector name="mqtt"
uri="mqtt://0.0.0.0:1893?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
        <transportConnector name="ws"
uri="ws://0.0.0.0:61624?maximumConnections=1000&wireFormat.maxFrameSize=104857600"/>
    </transportConnectors>
```

Protocol Description
TCP Default network protocol for most use cases.
NIO Consider NIO protocol if you need to provide better scalability for connections from producers
and consumers to the broker.
UDP Consider UDP protocol when you need to deal with the firewall between clients and the
broker.
SSL Consider SSL when you want to secure communication between clients and the broker.
HTTP(S) Consider HTTP(S) when you need to deal with the firewall between clients and the broker.
VM Although not a network protocol per se, consider VM protocol when your broker and clients
communicate with a broker that is embedded in the same Java Virtual Machine (JVM).

Infrastructures de clefs Publiques (PKI)

Exploiter du Cryptage → . crypter sur la ligne
. Authentification
- Cryptage dormant

- Encodage → changement d'un format A vers un format B. ex: Texte ascii → base64 → Texte UTF16 → ...
- Hashage → Transformation non reversible de A → B
$f(x) = y$ x → entrée f → fonction y = hash
fonctions - SHA Ø 1 2 3
- md5
différences = - taille - CRC
- fourchette de algos
- Collisions -
- Cryptage
→ Stocker une valeur de façon réversible via un ou deux "mots de passe"

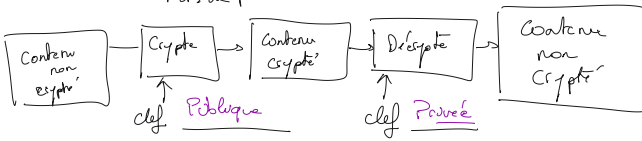
Cryptage Symétrique

→ Un Seul "mot de passe"

AES (Advanced Encryption Standard)
DES (Data Encryption Standard)
3DES (mot de passe coupé en 3 et crypté 3 fois)
IDEA (International Data Encryption Algorithm)
Blowfish (Drop-in replacement for DES or IDEA)
RC4 (Rivest Cipher 4)
RC5 (Rivest Cipher 5)
RC6 (Rivest Cipher 6)

Cryptage Asymétrique

→ Paire de "clef" → publique
"mots de passe" → privée



Algo → génération de la clef privée
élimination
clef publique → impossible d'obtenir la privée

- mot de passe
- Clef
- Secret

- pareil
- peut être mémorisé
- alphanumérique
- relativement facilement cassé par bruteforce.
- mot de passe au format binaire
- non mémorisable
- stocké dans un fichier

- mot de passe, clef
- jeton

eyJhbGciOiJIUzI1NiJ9.eyJSb2xlIjoiQWRtaW4iLCJjc3N1ZXIiOiJjc3N1ZXIiLCJVc2VybmFtZSI6Ikphd
mFjblVzZSIsImV4cCI6MTYzMTE3ODE1NSwiaWF0IjoxNjMxMTc4MTU1fQ.v3kZG8DwvcDBQUc
cuXVBd-dMkd6WVlOl_buP80LrUBs

Généralement (RSA) est utilisé en asymétrique

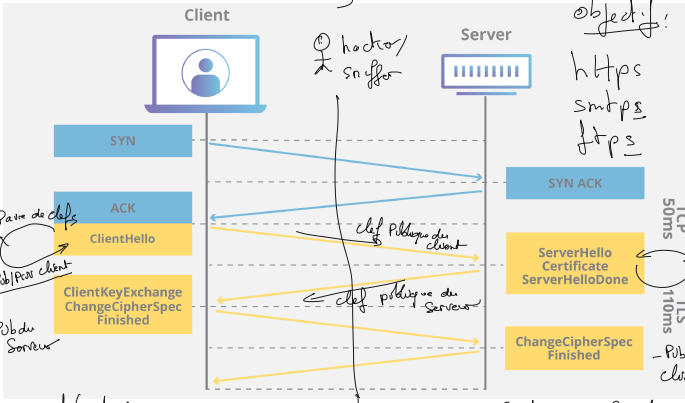
→ Clef stockée dans des certificats X.509 formats

X509 : _ Données
_ Métadonnés

- .crt
- .pdb
- .pfx
- .p7b
- etc...

"SSL" est obsolète depuis 1999
TLSv1 remplace SSL depuis

établir une session TLS

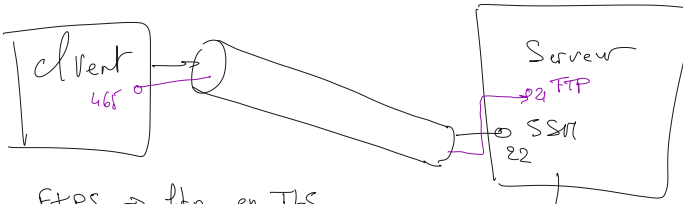


objectif :
https
smtps
ftps

http
ftp
smtp
→ sur TLS

sftp / ftps ?

smtp s
ssl ?



FTPS → ftp en TLS

SFTP → ftp dans du ssh

⇒ n' sécurité, protocoles différents -

Coffre → Firefox

→ nnc

→ "Vaults"

→ Keystore en java

→ par JRE / JDK

Keystore
- openJDK 11 ≠ Oracle JDK 11
- openJDK 14
- → Keystore explorer
- → Keytool en ligne de commande

→ Une paire de clefs doit être valide :

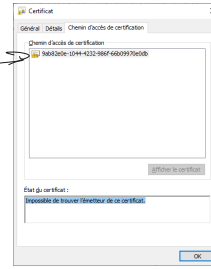
- Date de péremption
- chaîne de certification
- [domaine internet]

PKI

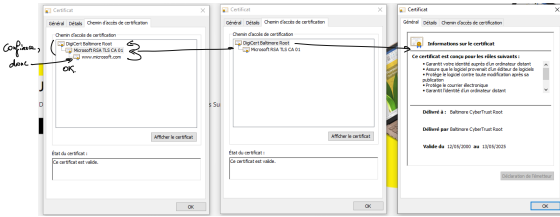
- certificats autosignés

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

→ pas reconnu par les Tiers –
→ pas de chaîne de certification



Pas de chaîne, autosigné



Obtenir de certificats conformes !

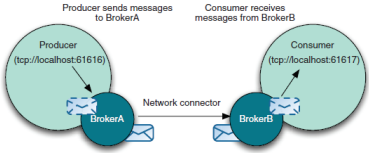
→ Soit payer pour avoir de certificats -

- Verisign, Thawte, Digicert, ...

Pour éviter de payer, on installe une 'PKI'

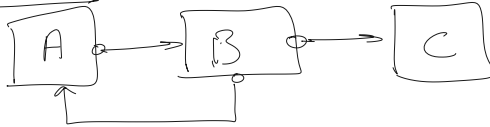
- Un "serveur", une entité "Racine"
- Qui permet de créer une "entité" intermédiaire, qui permet de créer des certificats.
- Des certificats à foison reçus depuis cette entité intermédiaire.

Sous Windows : Active Directory Certificate Services
En fichiers : avec openssl :
https://pki-tutorial.readthedocs.io/en/latest/simple/

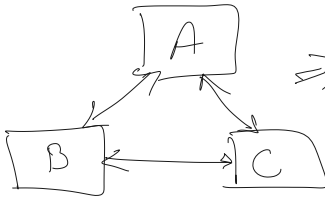



```
<broker xmlns="http://activemq.apache.org/schema/core"
brokerName="BrokerB"
dataDirectory="${activemq.base}/data">
<transportConnectors>
<transportConnector name="openwire" uri="tcp://localhost:61617" />
</transportConnectors>
</broker>

<broker xmlns="http://activemq.apache.org/schema/core"
brokerName="BrokerA"
dataDirectory="${activemq.base}/data">
<transportConnectors>
<transportConnector name="openwire" uri="tcp://localhost:61616" />
</transportConnectors>
<networkConnectors>
<networkConnector uri="static:(tcp://localhost:61617)" />
</networkConnectors>
</broker>
```

Relay:



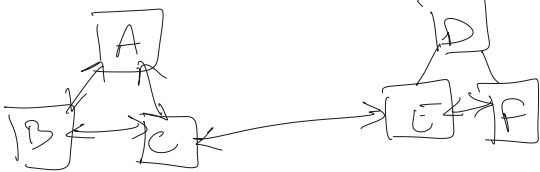
Nest (Nuclage)

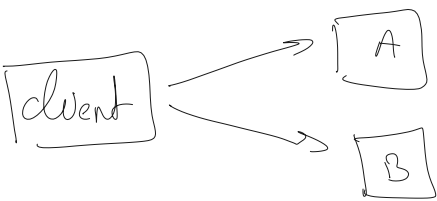


⇒ Exercice

envoyer depuis
tout Serveur,
écouter sur
chaque : — en Topic
— en Queue

hybride:





static://(tcp://remotehost1:61616,tcp://remotehost2:61616)

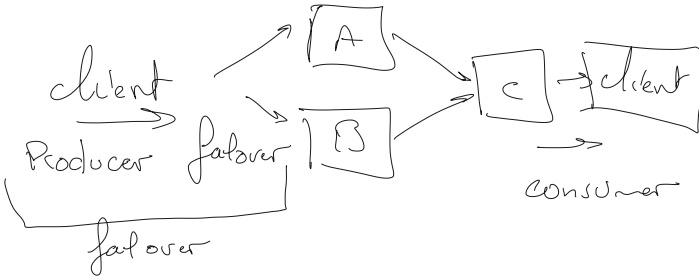
↳ plusieurs cnx actives

failover:(tcp://remotehost1:61616,tcp://remotehost2:61616)

↳ Tentative de cnx sur le premier, sinon le suivant

---

exercice :



→ lancer de l'envoi de message en continu toutes les secondes.

→ faire tomber A, B

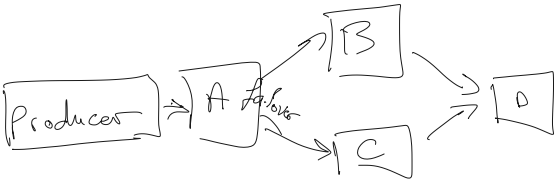
→ Consulter la réception de msg

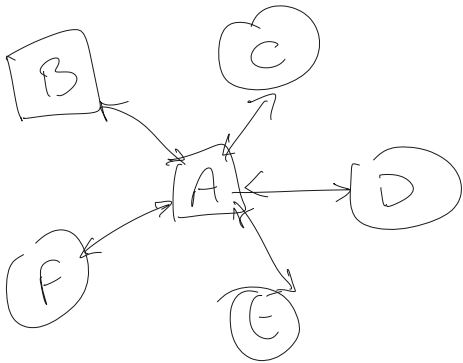
Failover recommandé même si vous n'avez qu'un broker ciblé, ne pas utiliser static

Etablir une connexion duplex initiée depuis un seul broker :

```
<networkConnector name="SYSTEM1" duplex="true" uri="static:(tcp://10.x.x.x:61616)">
 <dynamicallyIncludedDestinations>
                <topic physicalName="outgoing.System1" />
          </dynamicallyIncludedDestinations>
     </networkConnector>
```

ex2.. implémenter le failover dans le broker.





Démarrer un serveur postgres :
sudo docker container run --name postgres -p 5432:5432 -d --restart unless-stopped -e POSTGRES_PASSWORD=password postgres

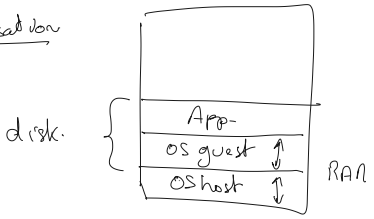
-p 5432:5432 : rendre le port 5432 accessible

S'y connecter :
docker container exec -it postgres psql -U postgres postgres
-it postgres : <nom du conteneur>
-U postgres (l'utilisateur postgres)
postgres : la base de donnée de connection par défaut

```
student@master:/opt/apache/amp/5.16.3_1/bin$ sudo docker container exec -it postgres psql -U postgres postgres
psql (13.4 (Debian 13.4-1.pgdg100+1))
Type "help" for help.

postgres=# create database amq;
CREATE DATABASE
postgres=# \q
student@master:/opt/apache/amp/5.16.3_1/bin$ sudo docker container exec -it postgres psql -U postgres amq
psql (13.4 (Debian 13.4-1.pgdg100+1))
Type "help" for help.

amq=# \dt
Did not find any relations.
amq=# \d
Did not find any relations.
amq=# \q
```

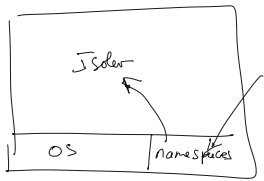
Conteneurs

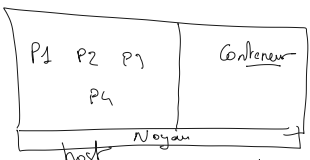
Virtualisation



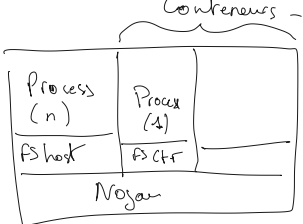
fonctionnalité de Noyau - Linux } "namespaces"
- Windows }

↳ cloisonner, isoler
- Process
- disque (FS)
- Réseau
- IPC ...



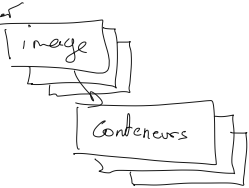


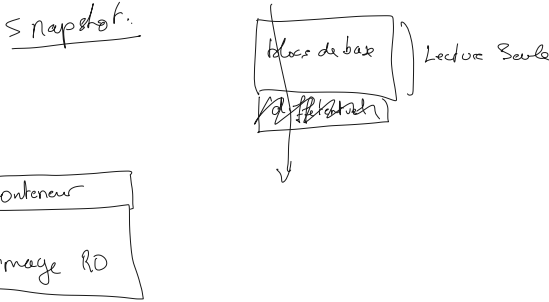
OS



Registre public
→ Stocke des images
→ officielles
→ De la Communauté

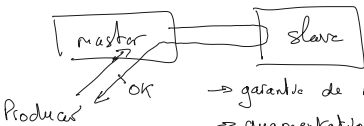
Télécharger





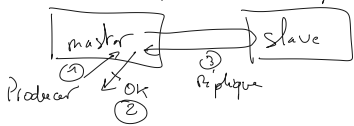

```
sudo docker network create mysql
sudo docker run --network mysql --name mysql --restart unless-stopped -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=password -d mysql
sudo docker run -it --rm --network mysql mysql mysql -hmysql -uroot -p

mysql --host=localhost -u root --password=password --protocol=tcp

<bean id="mysql-ds" class="org.apache.commons.dbcp2.BasicDataSource" destroy-
method="close">
  <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
  <property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>
  <property name="username" value="root"/>
  <property name="password" value="password"/>
  <property name="poolPreparedStatements" value="true"/>
 </bean>

<persistenceAdapter>
  <jdbcPersistenceAdapter dataSource="#mysql-ds"/>
 </persistenceAdapter>
```

Réplication Synchrone



→ garantie de persistance / aucune perte de donnée
→ augmentation de latence
→ le slave doit être disponible

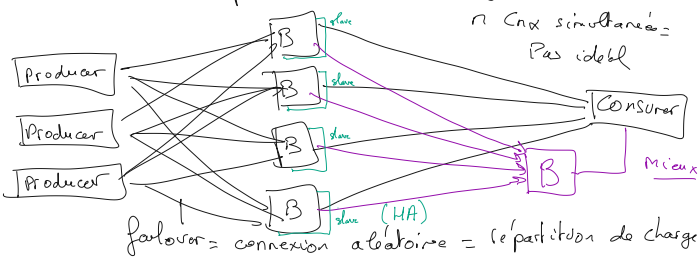
Réplication Asynchrone



→ pas de latence
→ pas dépendant du slave
→ on risque de perdre les messages déposés entre la dernière réplication et le master qui tombe.

en cas d'échec, l'esclave.
- s'arrête (bascule manuelle)
- bascule automatiquement

Répartition de Charge



n Cnx simultanées = Pas idéal

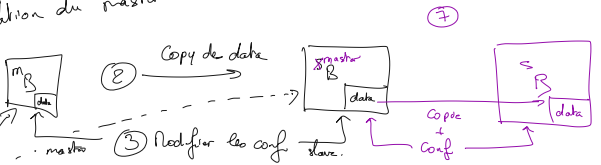
failover = connexion aléatoire = répartition de charge

failover://(tcp://masterhost:61616,tcp://slavehost:61616)?randomize=false

randomize=false permet de se connecter en priorité sur le master
waitForSlave permet de renforcer la non perte de données

Mise en oeuvre du shared Nothing

(1) création du master

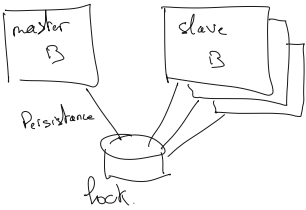


(6) SGR echec du master
→ Basculer le Slave en master
→ "Purger le master"
→ il n'y a plus de master pour l'instant

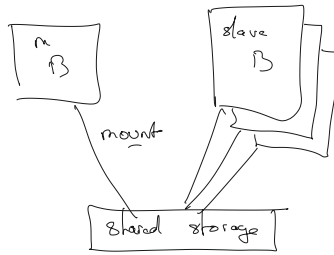
(7) Refaire l'opération copie data + conf
(2) (3) pour remonter un master/slave

Shared Storage :-

① BDD



② File System



→ mode File system : (NAS)
- \+ simple
- NFS, SMB
- tous sont up en même temps

→ mode bloc : (SAN)
- implémentation iSCSI target/initiator
- FS type glusterFS, btrfs, ... FS clustering
- ou FS ext3, ext4 si connexion unique → doit y avoir une procédure de montage pour faire la bascule.

→ Slave est down

en SAN a connexion unique

(bascule manuelle)

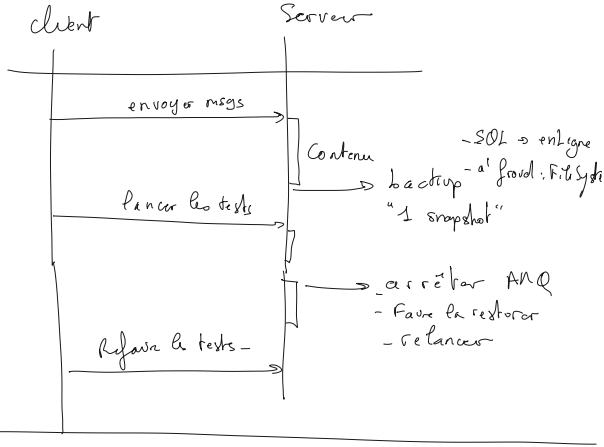
Modes de haute disponibilité à mettre en oeuvre, du plus simple, au plus complexe :

- Pour tester :
  - Le Producer doit utiliser failover:// sans randomize
  - Idem pour le Consumer

1. KahaDB partagé sur le même host
   a. Ok pour un POC (Proof Of Concept) -> On a démontré que ça marche, mais c'est pas utilisable en production.
   b. Implémentation : Créer deux amq avec deux confs qui pointent vers le même dossier data, idéalement stocké en dehors d'une instance AMQ
2. Base de données partagée
   a. Avoir un serveur Mysql
   b. Lancer deux instances qui utilisent la même configuration (sauf les ports)
3. Via un fileserver externe
   a. **Via NFS/SMB -> Voir server-world.info/en pour les tutos de montage de Fileserver et client : recommandé**
   b. Via SAN (iScsi target/initiator) : le plus performant, le meilleur, plus complexe à mettre en oeuvre : voir Server-world pour la mise en oeuvre iscsi target/initiator

mvn exec:java -Dexec.mainClass=org.apache.activemq.book.ch4.Publisher \
> -Dexec.args="failover:(tcp://localhost:61616,tcp://localhost:61617,tcp://localhost:61618)
CSCO ORCL"

Stratégies de tests :



Haute Dispo dans le cloud