Haute disponibilité d'un service Web dynamique

Activité 3 - Configuration de la réplication des bases de données

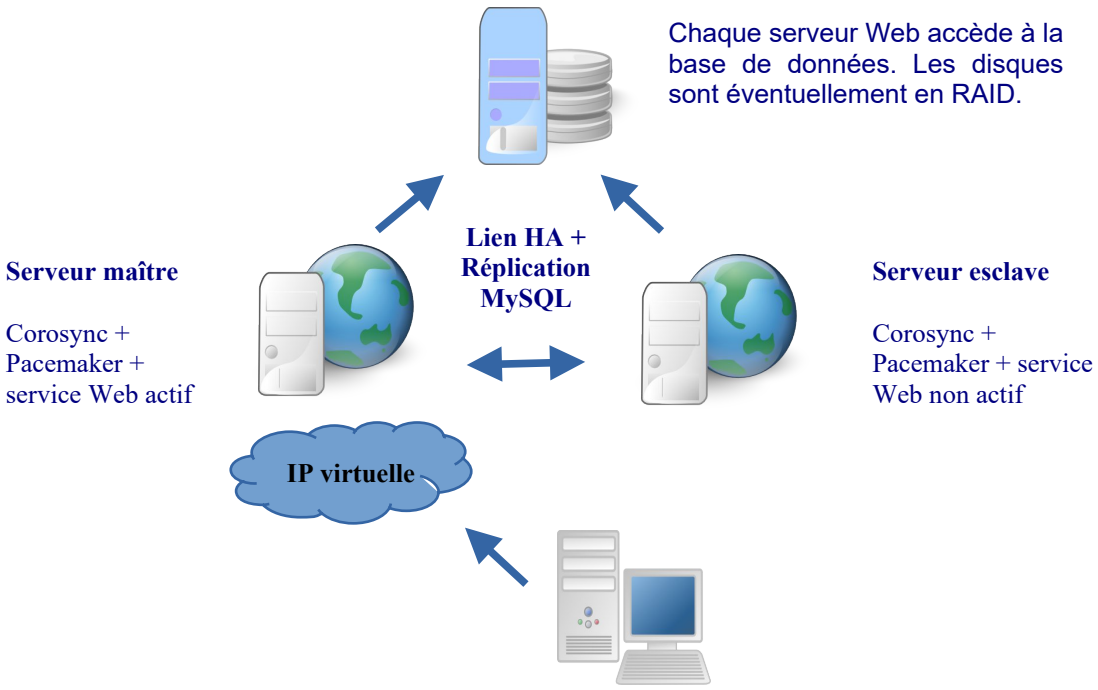
L'architecture mise en place dans les activités précédentes nécessite aussi une solution qui va permettre d'accéder à tout moment aux données actuelles, en l'occurrence les données présentes dans la base de données de l'application *appliFrais*.

La plupart des services inclus dans un système de haute disponibilité utilise des données, parmi eux :

- **le service Web** utilise quasi systématiquement des données stockées dans des bases de données SQL (et/ou XML) qu'il est donc nécessaire de répliquer également comme nous allons le faire dans ce coté labo ;
- **le service d'authentification** (parfois lui-même également utilisé par le service Web) stocke des données dans des annuaires (comme LDAP) qu'il faut également répliquer ;
- **Le service de fichier** (comme Active Directory ou Samba sur Linux, un serveur NFS, un serveur FTP, etc.) utilise bien évidemment les données des utilisateurs stockées sur une partition locale ou distante. Il est alors nécessaire de disposer d'un programme qui synchronise en temps réel une partition entre deux machines (une sorte de RAID 1 réseau) : la solution la plus utilisée pour créer ce type de serveur est d'utiliser Distributed Replicated Block Device (DRBD) comme nous allons l'étudier dans le Côté Labo suivant (http://www.reseaucerta.org/hd-serveur-ftp).

Plusieurs architectures sont possibles.

En règle générale, les données sont stockées sur un serveur « à part » (un serveur de base de données, par exemple) :

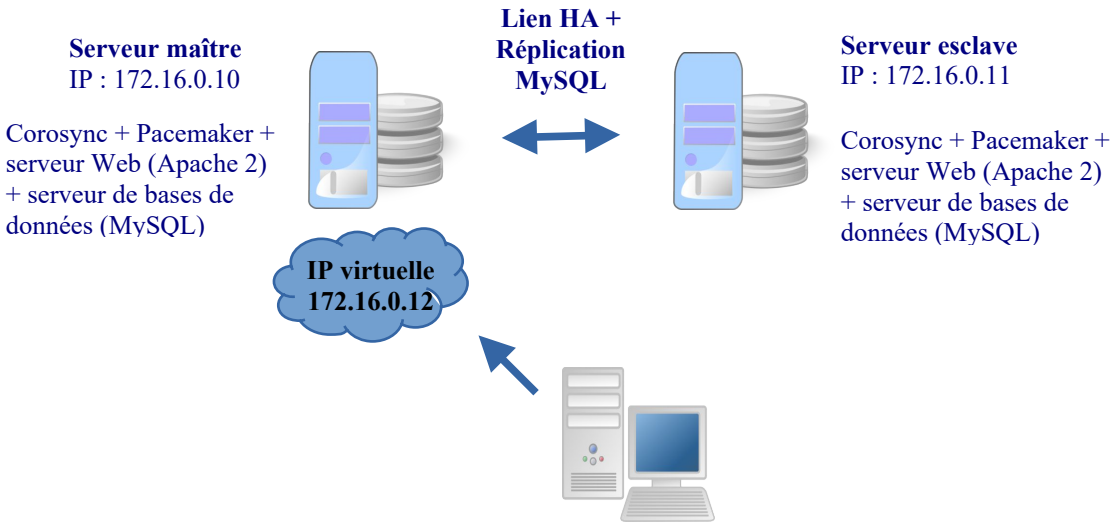


Si le serveur maître n'est plus disponible, tout ce qui est écrit dans la base de données sera retrouvé par le serveur esclave puisqu'il s'agit de la même base de données mais nous introduisons ici un point de faiblesse (Single Point Of Failure – SPOF en anglais) car en cas de défaillance physique du serveur de bases de données lui-même (et pas seulement d'un disque), les données ne sont plus disponibles.

Il faut donc assurer aussi la haute disponibilité du serveur de bases de données en répliquant les bases contenant les données utiles en temps réel sur une autre machine.

Par mesure de simplification, les bases de données seront, dans ce Coté labo, implantées sur le même serveur que le serveur Web mais le principe de réplication est strictement identique.

L'architecture que nous allons mettre en place est la suivante :



La réplication d'un système de gestion de base de données

L'architecture native de la réplication MySQL est basée sur une architecture maître-esclave ou **réplication unidirectionnelle.**

Le principe est simple : un seul serveur (le maître) assure les opérations d'écriture (commandes SQL INSERT, UPDATE et DELETE) et éventuellement de lecture (commande SQL SELECT), tandis que les esclaves se contentent d'enregistrer les modifications opérées sur le maître. Dans certains cas (lorsque l'application cliente est développée à cet effet) les esclaves assurent aussi les opérations de lecture. La synchronisation entre le maître et les esclaves est quasi instantanée.

Une solution existe en cas de défaillance du maître : il suffit d'activer les opérations d'écriture sur un esclave pour disposer d'un nouveau maître ; cette solution peut être mise en œuvre manuellement ou automatiquement si l'on s'adjoint d'autres outils comme un outil capable de dire si le serveur est tombé et/ou des scripts pour basculer un serveur d'esclave vers maître et vice versa**.**

Le type de réplication idéal lorsque deux nœuds sont en jeu reste quand même **la réplication bidirectionnelle** : tout ce qui est réalisé sur un serveur est recopié sur l'autre en temps réel et vice-versa : **relation maître-maître** ou pour être plus précis (maître/esclave et esclave/maître). Ce type de réplication est obligatoire lorsqu'on a pour but d'équilibrer la charge d'une base de données en lecture et écriture. Mais une base de données devant se mettre à jour en écriture simultanément sur plusieurs serveurs pose des problèmes techniques et il est nécessaire d'utiliser des technologies de Clustering comme le fait MySQL Cluster qui est la base de données distribuée de MySQL. Cette dernière solution est, sommes toutes, difficile à mettre en œuvre, aussi des configurations plus simples et fiables permettant de transformer une réplication maître-esclave en maître-maître (ou multi maître) ont vu le jour et correspondent à la majeure partie des contextes de production**.**

C'est cette dernière que nous mettrons en œuvre dans ce Coté labo.

L'objectif opérationnel de cette troisième activité est de configurer la réplication entre le serveur maître et le serveur esclave selon une architecture de type maître-esclave isolé des problématiques de Corosync et de Pacemaker

Vous disposez de la documentation suivante :

- **document 1** : principe de la réplication sur MySQL ;
- **document 2** : dépannage et maintenance.

Travail à faire

Q1. Écrivez la procédure détaillée que vous allez mettre en œuvre pour configurer la réplication de la base de données. Cette procédure fera l'objet d'un échange avec votre professeur avant son implémentation.

Q2. Mettez en œuvre la réplication selon la procédure.

Q3. Complétez le tableau ci-dessous permettant de vérifier l'opérationnalité de la solution.

Actions à effectuer	Résultats attendus	Résultats obtenus	Statut (OK ou non OK)
Saisie d'une fiche de frais à partir de l'application (on s'assure que l'adresse IP virtuelle et le service Web sont bien lancés sur le maître).			

Document 1 : principes de la réplication sur MySQL

Comment ça marche ?

La réplication MySQL est basée sur le fichier log binaire du serveur maître qui garde une trace de toutes les requêtes SQL (update, delete, etc.) et un fichier d'index des modifications à prendre en compte. L'esclave, au moment de la connexion, informe le maître de l'endroit où il s'est arrêté, rattrape les modifications qui ont eu lieu depuis (lecture du fichier log binaire du serveur maître pour qu'il puisse exécuter les requêtes SQL stockées dans ce fichier), puis se bloque en attente des prochaines modifications.

Le fichier de log binaire (ou *binlogs*) est simplement un enregistrement des modifications depuis un point fixe dans le temps (le moment où vous activez le log binaire). Tous les esclaves qui seront activés auront besoin de la copie des données qui existaient au moment du démarrage du log. *Ainsi, si l'on désire mettre une réplication en place, alors que la base de données, concernée par la réplication, du maître est déjà fournie en données, il faut tout d'abord effectuer une copie complète de celle-ci avant de commencer la réplication.*

Ensuite, l'esclave va simplement se connecter au maître et attendre des requêtes de modifications. Il lira le fichier journal binaire du maître, en copiera les requêtes dans un fichier tampon, ou fichier « relais » (relay) en terminologie MySQL, et il les « jouera » sur son serveur.
Si le maître est indisponible ou que l'esclave perd la connexion avec le maître, il va essayer de se reconnecter selon une période fixée en secondes par le paramètre « master-retry-count » jusqu'à ce qu'il soit capable d'établir la communication, et de recommencer à appliquer les modifications.

Chaque esclave garde la trace du point où il en était rendu. Le serveur maître n'a pas de notion du nombre d'esclaves qui se connectent, ou qui sont à jour à un moment donné.

Remarque : un esclave peut aussi servir de maître à son tour pour réaliser une chaîne de réplication.

Étapes avant modification des fichiers de configuration

- La réplication de bases de données entre plusieurs serveurs présuppose l'existence **d'un utilisateur MySQL ayant des droits suffisants sur chaque serveur maître** ; chaque serveur esclave doit pouvoir se connecter au maître avec cet utilisateur de manière à ce qu'il puisse consulter l'état du log binaire et répliquer les données si des modifications ont été apportées : **le droit spécial « REPLICATION SLAVE » suffit**.
- Les bases de données à répliquer **doivent être identiques** sur tous les serveurs avant de commencer la réplication : on va devoir s'assurer qu'aucune nouvelle donnée ne soit enregistrée pendant le laps de temps nécessaire à la configuration. À ce moment, **il s'agit de bloquer l'écriture sur la (les) base(s)** que l'on souhaite répliquer de manière à s'assurer qu'aucune nouvelle donnée ne soit enregistrée : on doit pour cela saisir la commande suivante sur le serveur « maître » *« FLUSH TABLES WITH READ LOCK; »* : cette dernière va alors fermer toutes les tables ouvertes, et verrouiller en lecture toutes les tables et bases. À ce moment là, les requêtes qui voudront faire une écriture seront mises en file d'attente, jusqu'au déblocage par la commande « UNLOCK TABLES; ».
- Il est nécessaire de configurer la réplication via le fichier */etc/mysql/mariadb.conf.d/50-server.cnf* : la réplication de bases de donnés implique une configuration différente sur chaque serveur selon le rôle de chacun et nécessite (entre autres) un identifiant unique dans la chaîne de réplication.

Quelques directives utilisées dans */etc/mysql/mariadb.conf.d/50-server.cnf* sur des versions de MySQL inférieures à 5.5 ne fonctionnent plus (et empêchent MySQL de démarrer avec une erreur dans les logs « unknown variable »), même si ces directives sont encore présentes dans le fichier exemple !
Aussi, il ne faut pas se fier complètement aux configurations proposées sur Internet et TOUJOURS exploiter les logs.

Fichier de configuration : /etc/mysql/maridb.conf.d/50-server.cnf

Les fichiers de configuration déterminent uniquement un statut « maître » ou « esclave ». Tous les paramétrages permettant d'établir un lien entre le maître et l'esclave (comme l'adresse IP du maître, le nom de l'utilisateur et le mot de passe, etc) doivent être réalisés « à chaud » directement dans la console SQL, à l'aide de la commande SQL ***CHANGE MASTER TO***. À noter que la plupart des directives sont déjà écrites dans le fichier de configuration, il suffit alors de les "décommenter" et/ou de les adapter.

Sur le serveur « maître » :

```
[mysqld]
#bind-address     = 127.0.0.1
log_error          =/var/log/mysql/error.log
server-id     =100
log_bin       = /var/log/mysql/mysql-bin.log
expire_logs_days       = 10
max_binlog_size        = 100M
binlog_do_db  = include_database_name
```

On commente la ligne pour que mysql écoute sur toutes ses interfaces réseaux et non uniquement sur 127.0.0.1. Car l'esclave va se connecter au maître pour récupérer les données.

Isolation des logs de démarrage de MySQl dans un fichier (sinon les logs s'écrivent dans /var/log/syslog).

Identificateur du serveur maître (libre mais forcément différent de celui des autres serveurs participant à la réplication).

Activation du log binaire nécessaire pour la réplication : un fichier journal binaire des requêtes SQL sera ainsi créé sur le serveur maître et sera utilisé par l'esclave pour « rejouer » les requêtes.

Gestion des fichiers de log binaire qui peuvent devenir très lourds : on gère ici le délai d'expiration et leur taille maximale.

Spécifie le nom de la base à répliquer (on écrit autant de lignes qu'il y a de bases à répliquer).

On peut maintenant redémarrer MySQL et on doit bloquer IMMÉDIATEMENT l'écriture (voir « étapes après modifications des fichiers de configuration ») ==> N'hésitez pas à lire en parallèle le fichier « /var/log/mysql/error.log » notamment si MySQL ne se lance plus...

Sur le serveur « esclave » :

```
[mysqld]
log_error          =/var/log/mysql/error.log
server-id          = 104
expire_logs_days       = 10
max_binlog_size        = 100M
master-retry-count = 20
replicate-do-db = include_database_name
```

Identificateur du serveur esclave différent de celui du maître.

L'esclave va essayer de se reconnecter toutes les 20 secondes en cas d'inaccessibilité du maître.

Spécifie le nom de la base qui sera répliquée (on écrit autant de lignes qu'il y a de bases à répliquer).


Les deux dernières directives doivent être ajoutées.

On peut maintenant redémarrer MySQL sur l'esclave ==> N'hésitez pas à lire en parallèle le fichier « /var/log/mysql/error.log » notamment si MySQL ne se lance plus...

Lorsque la configuration est terminée, et si on ne l'a pas encore fait, il faut :

- s'assurer que les bases de données soient bien synchronisées, dans le cas contraire il faut le faire (voir procédure dans la partie « Dépannages et maintenance ») ;
- bloquer au préalable l'écriture sur le maître (si ce n'est pas déjà fait) ;
- récupérer des informations sur le maître (voir ci-après) ;
- indiquer à l'esclave via la commande SQL ***CHANGE MASTER TO*** l'ensemble des opérations à réaliser (voir ci-après) ;
- permettre l'écriture sur le maître ;
- lancer l'esclave.

Étapes après modifications des fichiers de configuration

Sur le serveur maître, il est nécessaire, après s'être assuré que les deux bases de données sont identiques (voir comment faire dans la partie « dépannage » si vous avez un doute) :

- **De bloquer l'écriture** :

mysql> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.09 sec)

- **De repérer le log binaire et la position du jeu de commandes dans le log** :

mysql> show master status;

```
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |     3921 | databasesName  |                  |
+------------------+----------+--------------+------------------+
```

1 row in set (0.00 sec)

Les bases à répliquer doivent apparaître, séparées d'une virgule.

Du résultat de cette requête, ce sont les champs File et Position qu'il faut noter.

C'est à partir de cette position que tout ce qui sera écrit sur le maître sera répliqué sur l'esclave. Une fois que les données ont été répliquées, l'esclave garde la trace du point où il en était rendu (une nouvelle *Position*, voire un nouveau *File*). Le serveur maître n'a pas de notions du nombre d'esclaves qui se connectent, ou qui sont à jour à un moment donné.

Il faut donc ensuite indiquer au serveur esclave :

- à partir de quel serveur maître il doit répliquer ses bases de données (on utilisera l'adresse IP du serveur maître),
- en lisant quel fichier journal de requêtes (dans l'exemple, mysql-bin.00001),
- à partir de quelle position dans ce fichier journal (dans l'exemple, *3921*),
- et enfin, avec quel utilisateur (il s'agit du compte utilisateur créé précédemment – Voir « Étapes avant modification des fichiers de configuration »).

Tout ceci est effectué en une seule ligne avec la syntaxe spéciale CHANGE MASTER TO.

Le processus esclave, s'il existe (ce que l'on peut voir avec la commande « show slave status; », doit être arrêté auparavant.
Pour ce faire, dans la console SQL du serveur esclave, exécutez : STOP SLAVE ;.
Ce processus doit être redémarré ensuite.

```
mysql> stop slave;
mysql> change master to
    -> master_host='172.16.0.10',
    -> master_user='replicateur',
    -> master_password='mdpreplicateur',
    -> master_log_file='mysql-bin.000001',
    -> master_log_pos=3921;
mysql>start slave ;
```

Ces informations sont sauvées automatiquement dans le fichier

Il est temps maintenant de déverrouiller les bases de données sur le serveur maître :

```
mysql> UNLOCK TABLES;
```

Pour connaître le statut de l'esclave

Sur l'esclave

Permet d'avoir un affichage ligne par ligne pour

```
mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 172.16.0.10
                  Master_User: replicateur
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 3921
               Relay_Log_File: mysqld-relay-bin.000008
        Relay_Log_Pos: 63459
        Relay_Master_Log_File: mysql-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB: gsb_valide
          Replicate_Ignore_DB:
           Replicate_Do_Table:
    Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 444808
              Relay_Log_Space: 65440
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 100
```

Le résultat retourné est relativement complexe mais il contient notamment les informations à vérifier (master_log_file, read_master_log_pos, fichier journal lu

Document 2 : Dépannage et maintenance

Une des erreurs les plus fréquentes est de ne pas démarrer avec les mêmes données dans la base (voire se trouver avec des données en conflit). Vous pourrez alors lire ce genre de choses dans les logs :

```
Slave: Duplicate entry '75267' for key 'PRIMARY' Error_code: 1062
121210  9:12:54 [ERROR] Error running query, slave SQL thread aborted. Fix the problem, and restart the slave SQL thread with "SLAVE START". We stopped at log 'mysql-bin.000002' position 501
```

Dans ce cas il est nécessaire de sauvegarder sur le maître la bonne base et de la restaurer sur l'esclave :
Cette sauvegarde peut se faire directement à partir de l'esclave si le maître accepte les connexions à partir de l'utilisateur qui va effectuer cette sauvegarde (en production, on crée généralement un utilisateur pour cela).

Exemple, à partir du serveur esclave (et après avoir stoppé l'esclave avec la commande mysql « stop slave; » :

Stopper l'esclave :

```
mysql> stop slave ;
Query OK, 0 rows affected (0.06 sec)
```

Sauvegarder dans un fichier « sauvBases.sql »

```
mysqldump -h 172.16.0.11 -u root --databases nom_base1,nom_basen --add-drop-table -p<motDePasse> sauvBases.sql
```

L'option -- add-drop-table ajoute une commande « drop table » avant chaque requête de création de table : il n'est ainsi pas nécessaire de supprimer la table avant.

Restaurer les bases sur l'esclave :

```
mysql -p<motDePasse < sauvBases.sql
```

Une fois le *dump* effectué et avant de restaurer les écritures sur le maître, il faut récupérer la position du maître dans ses binlogs :

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000002 |    48044 | DatabasesName |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Il est nécessaire de :

Lancer la commande « change master to » correspondante (seules les données qui ont changées par rapport à la commande précédente sont utiles) :

```
mysql> change master to master_log_file='mysql-bin.000002', master_log_pos=48044;
Query OK, 0 rows affected (0.10 sec)
```

Démarrer l'esclave :

```
mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

Libérer les écritures sur le maître :

```
mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.00 sec)
```

La seconde erreur fréquente se produit quand il y a un décalage de la valeur des paramètres donnés par la commande « show master status » : cette erreur survient lorsque, par exemple, il y a eu écriture sur une base de l'esclave.

Elle est facilement décelable en comparant :

Sur le maître

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000002 |   444808 | DatabasesName|                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

Sur l'esclave

```
mysql> show slave status \G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 172.16.0.10
                  Master_User: replicateur
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000002
          Read_Master_Log_Pos: 444808
               Relay_Log_File: mysqld-relay-bin.000008
        Relay_Log_Pos: 63459
        Relay_Master_Log_File: mysql-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
            ...
```

Ici, pas d'erreur, les

À noter que d'autres problèmes peuvent survenir comme :

- l'esclave qui ne contacte plus son maître ;
- l'esclave qui n'a pas pu exécuter une requête ;
- etc.

La commande précédente indiquera alors la cause de l'erreur et son numéro de référence MySQL pour dépanner.

Gestion des binlogs

Les relay-logs de l'esclave (fichiers créés dans */var/lib/mysql* et contenant les commandes à exécuter par l'esclave) sont consommés dès que toutes les requêtes de chacun ont été exécutées. Par conséquent, aucune action n'est nécessaire dessus.

Par contre, ceci n'est pas vrai pour les binlogs du maître car celui-ci n'a pas conscience de ses esclaves et les binlogs s'empilent (même si nous les avons un petit peu gérés dans le fichier de configuration).

Afin de supprimer les binlogs qui deviennent consommateurs d'espace disque avec le temps, il convient de faire la requête suivante :

```
mysql>SHOW MASTER LOGS;
mysql>PURGE MASTER LOGS TO 'mysql-bin.000027';
```

À la première ligne, le résultat indique le fichier binlog en cours. **Il ne faut surtout pas le supprimer.**
La seconde ligne supprime tous les binlogs disponibles jusqu'à celui indiqué exclus.
Il est préférable de ne pas supprimer directement les fichiers au niveau système (avec la commande rm) car dans ce cas MySQL ne gérera plus ses binlogs correctement et donc la requête PURGE MASTER ne fonctionnera plus.

Utilisation des fichiers */var/lib/mysql/master.info* et */var/lib/mysql/relay-log.info*

Un serveur de réplication esclave crée deux autres petits fichiers dans le dossier de données. Ces fichiers sont appelés master.info et relay-log.info par défaut. Ils contiennent des informations comme celles affichées par la commande SHOW SLAVE STATUS. **En tant que fichiers disques, ils survivent à l'extinction de l'esclave. Au prochain démarrage de l'esclave, ce dernier pourra lire ces fichiers pour savoir où il en était du traitement des événements du maître et de leur lecture.**

Lorsque l'on sauvegarde les données de l'esclave, il est nécessaire de sauver ces deux fichiers, ainsi que les logs de relais. Ils sont utiles pour reprendre la réplication après une restauration de la base.

En cas de perte des logs de relais, le fichier relay-log.info peut limiter les dégâts, car il est possible de déterminer ce que le thread SQL a traité des logs binaires du maître. Puis, on utilise CHANGE MASTER TO avec les options MASTER_RELAY_LOG et MASTER_RELAY_POS pour dire au thread d'I/O de relire les logs depuis ce point. *Cela impose que ces logs soient toujours disponibles sur le serveur.*