Adatbázis másolás Slony-I segítségével

Akár a magas elérhetőség érdekében, akár mentésként vagy leállás nélküli verziófrissítés miatt van szükségünk másolatkészítésre, ez a rugalmas eszköz mindent szinkronban tart.

Az adatbázis kezelő rendszerek jó néhány éve létfontosságú alkotóelemei a különféle megoldásoknak.
A *PostgreSQL* fejlett, objektum-relációs adatbázis kezelő rendszer, amelyet igen gyakran használnak összetett alkalmazások hátttereként. Bár ez az adatbázis kezelő rendszer közismerten stabil, két két nyílt forrású másolatkészítő rendszere az *rserv* és az *ERServer*, komoly problémákkal küzd és helyettesítésre szorul.
Szerencsére mostanában jelent meg egy ilyen helyettesítő rendszer. Nevezetesen *Jan Wieck PostgreSQL*-hez kifejlesztett trigger alapú *Slony-I*, egyetlen mester – több szolga elvű másolatkészítő rendszere. Ez az üzleti kategóriás másolatkészítő megoldás aszinkron elven működik és az adatközpontokban elvárható valamennyi kulcsképességgel rendelkezik. A *Slony-I* legfontosabb felhasználási területei:

- Adatbázis másolatkészítés a központi hivatal adatairól az ágak felé, csökkentve a sávszélesség felhasználást, vagy felgyorsítandó az adatbázis lekérdezéseket.
- Adatbázis másolatkészítés valamennyi példány terhelés-kiegyenlítése érdekében. Különösen hasznos megoldás jelentéskészítések vagy dinamikus weblapok esetében.
- Adatbázis másolatkészítés a magas elérhetőség biztosítása érdekében.
- Forró mentés készenléti kiszolgáló segítségével vagy átállás új *PostgreSQL* verzióra.

Cikkünk végigkíséri az olvasót a *Slony-I* telepítésének lépésein, majd másolatot készítünk a helyi gépen található adatbázisról. Bemutatjuk, miképpen használhatjuk a *Slony-I* rendszerét automatikus hibaelhárításra magas rendelkezésre állású megoldásokban.

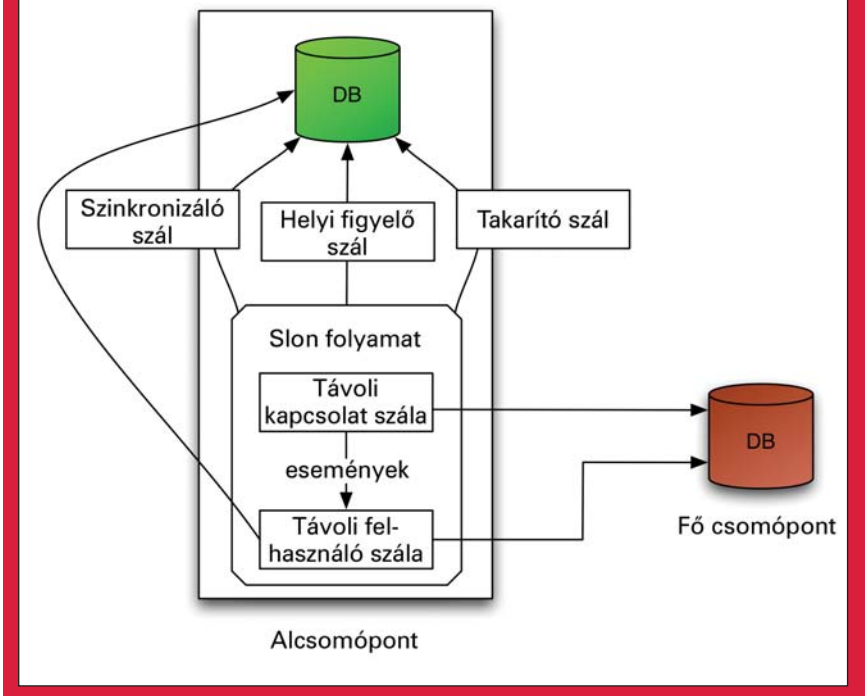
Slony-I telepítése

Amennyiben egy egyszerű adatbázison ki is szeretnénk próbálni a *Slony-I* rendszert először is telepítsük fel forrásból a *PostgreSQL*-t. A *Slony-I* a *PostgreSQL 7.3.2* vagy újabb (7.4. és 8.0) verziókat támogatja és fordítás közben szüksége lesz a *PostgreSQL* forrásfájára. Amennyiben szívesebben használjuk a kedvenc terjesztésünk *PostgreSQL* csomagjait, egyszerűen fordítsuk le őket a forráscsomagból és hagyjuk meg a forrásfát érintetlenül, így fel tudjuk majd használni a Slony-I fordításához. Szerezzük be a legfrissebb *Slony-I* kiadást, azaz az 1.0.5-öst, fordítsuk le és telepítsük.
Ezt a következő parancsokkal tehetjük meg:

```
% tar -zxvf slony1-1.0.5.tar.gz
% cd slony1-1.0.5
% ./configure \
--with-pgsourcetree=/usr/src/redhat/
➥BUILD/postgresql-7.4.5
% make install
```

Példánkban a *Slony-I* `configure` parancsfájlának megadtuk, hogy a */usr/src/redhat/BUILD/postgresql-7.4.5/* helyen kell keresnie a *PostgreSQL* forrást, ugyanis *Red Hat Enterprise Linux* ezt a könyvtárat használja a *PostgreSQL 7.4.5 RPM* állomány fordításakor. Az utolsóparancs lefordítja a *Slony-I*-t és telepíti a következő állományokat:

- *$postgresql_bindir/slonik*: a *Slony-I* adminiszrációs és beállító állománya. A *slonik* egyszerű eszköz és általában héjprogramba ágyazva a *Slony-I* másolatkészítő eszköz beállítására használják. Saját formátummentes parancsnyelve van amelyet a *Slonik Command Summary* dokumentum ismertet részletesen.
- *$postgresql_bindir/slon*: A központi másolatkészítő motor. Ez a többszálú rendszer használja fel a másolatkészítési vázlat információit és kommunikál a többi motorral létrehozva a megoszló másolatkészítő rendszert.
- *$postgresql_libdir/slony1_funcs.so*: C függvények és ravaszok (triggers).
- *$postgresql_libdir/xxid.so*: további adattípusok a tranzakció azonosítók biztonságos tárolásához.
- *$postgresql_datadir/slony1_base.sql*: Másolatkészítési vázlat.
- *$postgresql_datadir/slony1_base.v73.sql*.
- *$postgresql_datadir/slony1_base.v74.sql*.
- *$postgresql_datadir/slony1_funcs.sql*: másolatkészítő függvények.
- *$postgresql_datadir/slony1_funcs.v73.sql*.
- *$postgresql_datadir/slony1_funcs.v74.sql*.
- *$postgresql_datadir/xxid.v73.sql*: A korábban megadott adattípusok betöltését végző parancsfájl.



1. ábra A Slony-I másolatkészítő motor működése mester és szolga adatbázisokkal

egy `contactdb` adatbázist és engedélyezzük a ***plpgsql*** programozási nyelvet a frissen elkészült ***PostgreSQL*** adatbázisunkhoz a következő parancsok kiadásával:

```
% su - postgres
% createuser --pwprompt
➥contactuser
Enter password for user
➥"contactuser": (specify
➥a password)
Enter it again:
Shall the new user be allowed to
➥create databases?
(y/ n) y
Shall the new user be allowed to
➥create more new
users? (y/ n) n
% createdb -O contactuser
➥contactdb
% createlang -U postgres -h
➥localhost plpgsql \
contactdb
```


A ***$postgresql_bindir*** általában a */usr/bin/*, a ***$postgresql_libdir*** a */usr/lib/pgsql/* és a ***$postgresql_datadir*** pedig a */usr/share/pgsql/* könyvtárra mutat. Az `pg_config --configure` parancs használatával megjeleníthetjük a ***PostgreSQL*** fordításakor használt paramétereket és megtudhatjuk milyen megoldást használ a terjesztésünk. A teljes körű ***PostgreSQL*** másolatkészítő motorunk létrehozásához mindössze ezekre a fájlokra van szükségünk.

Amint azt az *1. ábrából* kitalálhatjuk, a ***Slony-I*** központi másolatkészítő motorja a ***slon*** jónéhány szálat használ. A szinkronizációs szál előre beállítható időtartamonként ellenőrzi, hogy van-e valamilyen másolható adatbázis aktivitás, és ha igen, akkor `SYNC` eseményt generál. A helyi figyelő szálak érzékelik az új beállítási eseményeket és ennek megfelelően módosítják a fürt beállításait valamint a ***slon*** folyamat a memóriában található paramétereit.

Mint a neve is mutatja, a ***tisztító (cleanup)*** folyamat a ***Slony-I*** vázlat karbantartását végzi, eltávolítva a régi eseményeket és kiürítve a táblákat. A távoli figyelő szál felkapcsolódik távoli csomópontok adatbázisához, hogy annak eseményküldőjétől eseményeket fogadjon. Amikor eseményt vagy jóváhagyást észlel, kiválasztja a megfelelő információt és a távoli dolgozók szálainak belső üzenetsorába tölti azokat. A másolatkészítési adat, tranzakciócsoportokból áll. A távoli dolgozó szál (minden csomóponton egy-egy) végzi a tényleges másolatkészítést, esemény tárolást és a visszaigazolások elküldését. A szolga minden időpillanatban pontosan tudja, milyen tranzakciókat dolgozott már fel.

Kisebb adatbázis másolata

Először is készítünk egy adatbázist amit majd másolhatunk. Az adatbázis egyetlen táblát és egy szekvenciát tartalmaz. készítsünk egy `contactuser` nevű felhasználót,

Ezek után létrehozzuk a szekvenciát és a táblát a másolandó adatbázisunkban majd némi információt szúrunk a táblába:

```
% psql -U contactuser contactdb
contactdb=> create sequence contact_seq start with 1;
contactdb=> create table contact (
 cid         int4 primary key,
 name        varchar(50),
 address     varchar(255),
 phonenumber varchar(15)
);
contactdb=> insert into contact (cid, name, address,
phonenumber) values ((select
➥nextval('contact_seq')),
'Joe', '1 Foo Street', '(592) 471-8271');
contactdb=> insert into contact (cid, name, address,
phonenumber) values ((select
nextval('contact_seq')),
'Robert', '4 Bar Roard', '(515) 821-3831');
contactdb=> \q
```

Az egyszerűség kedvéért készítsünk még egy adatbázist ugyanezen a rendszeren ahol az első (`contactdb`) adatbázisunk adatait fogjuk duplikálni. A következő parancsok segítségével hozzuk létre az adatbázist, vegyük fel a ***plpgsql*** programozási nyelv támogatást és vigyük fel az `contactdb` adatbázis vázlatát adatok nélkül:

```
% su - postgres
% createdb -O contactuser contactdb_slave
% createlang -U postgres -h localhost plpgsql \
contactdb_slave
```

1. lista cluster_setup.sh

```
#!/bin/sh
CLUSTER=sql_cluster
DB1=contactdb
DB2=contactdb_slave
H1=localhost
H2=localhost
U=postgres
slonik <<_EOF_
cluster name = $CLUSTER;
node 1 admin conninfo = 'dbname=$DB1 host=$H1
➥user=$U';
node 2 admin conninfo = 'dbname=$DB2 host=$H2
➥user=$U';
init cluster (id = 1, comment = 'Node 1');
create set (id = 1, origin = 1,
       comment = 'contact table');
set add table (set id = 1, origin = 1, id = 1,
        full qualified name = 'public.contact',
        comment = 'Table contact');
set add sequence (set id = 1, origin = 1,
➥id = 2,
        full qualified name =
        ➥'public.contact_seq',
        comment = 'Sequence contact_seq');
store node (id = 2, comment = 'Node 2');
store path (server = 1, client = 2,
      conninfo = 'dbname=$DB1 host=$H1
      ➥user=$U');
store path (server = 2, client = 1,
           conninfo = 'dbname=$DB2 host=$H2
           ➥user=$U');
store listen (origin = 1, provider = 1,
➥receiver = 2);
store listen (origin = 2, provider = 2,
➥receiver = 1);
```

```
% pg_dump -s -U postgres -h localhost contactdb | \
psql -U postgres -h localhost contactdb_slave
```

Miután létrehoztuk az adatbázisokat, már készen állunk az adatbázis fürt létrehozására, ahol egy mesterhez mindössze egyetlen szolga tartozik majd. Hozzuk létre a `cluster_setup.sh` parancsfájlt és futtassuk le.
A `cluster_setup.sh` parancsfájl tartalmát az *1. listában* mutatjuk be
Az 1. listában olvasható első `slonik` parancs (a `cluster name`) adja meg azt a névteret, ahol az összes ***Slony-I*** jellegű függvény, eljárás, tábla és szekvencia tárolódik. ***Slony-I*** rendszerében a csomópont egy adatbázis és `slon` folyamatok együttese, a fürt pedig csomópontok csoportja, amelyeket ösvények (`path`) kötnek össze. Megadjuk az 1-es és a 2-es csomópont kapcsolati adatait, majd az első csoportot alaphelyzetbe állítjuk (`init cluster`). Ezek után a parancsfájl létrehoz egy másolandó új készletet,

2. lista subscribe.sh

```
#!/bin/sh
CLUSTER=sql_cluster
DB1=contactdb
DB2=contactdb_slave
H1=localhost
H2=localhost
U=postgres
slonik <<_EOF_
cluster name = $CLUSTER;
node 1 admin conninfo = 'dbname=$DB1 host=$H1
➥user=$U';
node 2 admin conninfo = 'dbname=$DB2 host=$H2
➥user=$U';
subscribe set (id = 1, provider = 1, receiver
➥= 2, forward = yes);
```

amely tulajdonképpen a `public.contact` táblát és a `public.contact_seq` szekvenciát tartalmazza.
A készlet létrehozása után a parancsfájl felveszi a `contact` táblát és a `contact_seq` szekvenciát. A `store node` parancs segítségével alaphelyzetbe állítjuk a második csomópontot (`id = 2`) és felvesszük a fürtbe (`sql_cluster`). Ezek után a parancsfájl meghatározza a 2-es csomópont másolatkészítő rendszere milyen módon csatlakozik az 1-es csomóponthoz. Végül, a szkript mindkét csomópontot utasítja, hogy figyeljék a fürt többi elemének eseményeit (`store listen`).
A parancsfájl végrehajtását követően indítsuk el a ***slon*** másolatkészítő folyamatokat. A mester és a szolga csomóponton is szükségünk lesz egy-egy ***slon*** folyamatra. Példánkban a két szükséges folyamatot ugyanazon a rendszeren indítottuk el. Replikáció közben a ***slon*** folyamatoknak állandóan futniuk kell. Ha valamilyen okból le kellene állítanunk őket, egyszerű újraindítással onnan folytatják a munkát ahol abbahagyták. A másolatkészítő motor indítását a következő parancsok kiadásával végezzük:

```
% slon sql_cluster "dbname=contactdb
➥user=postgres" &
% slon sql_cluster "dbname=contactdb_slave
➥user=postgres" &
```

Következő lépésként fel kell iratkoznunk a frissen elkészített készletre. A készletre feliratkozva a második, azaz a feliratkozó csomópont, elkezdi az első csomópont `contact` táblájának és a `contact_seq` szekvenciájának másolását.
A *2. listában* a feliratkozó parancsfájl tartalmát olvashatjuk.
Az 1. listához hasonlóan a ***subscribe.sh*** is a fürt névtér és a két csomópont kapcsolati adatainak megadásával kezdődik. Ezek után a `subscribe set` parancs hatására az első csomópont ***slon*** folyamatokon keresztül elkezdi az egyetlen táblából és szekvenciából álló készlet másolását a második csomópontra.

3. lista compare.sh

```
#!/bin/sh
CLUSTER=sql_cluster
DB1=contactdb
DB2=contactdb_slave
H1=localhost
H2=localhost
U=postgres
echo -n "Comparing the databases..."
psql -U $U -h $H1 $DB1 >dump.tmp.1.$$
➥<<_EOF_
        select 'contact'::text, cid, name,
        ➥address,
          phonenumber from contact order by
          ➥cid;
_EOF_
psql -U $U -h $H2 $DB2 >dump.tmp.2.$$ <<_EOF_
        select 'contact'::text, cid, name,
        ➥address,
          phonenumber from contact order by
          ➥cid;
_EOF_
if diff dump.tmp.1.$$ dump.tmp.2.$$ >dump.diff ;
➥then
        echo -e "\nSuccess! Databases are
        ➥identical."
        rm dump.diff
else
        echo -e "\nFAILED - see dump.diff."
fi
rm dump.tmp.?.$$
```

Miután a ***subscribe.sh*** parancsfájl lefutott, csatlakozzunk a `contactdb_slave` adatbázishoz és vizsgáljuk meg a `contact` tábla tartalmát. Bármely időpillanatban azt kell látnunk, hogy az adatok pontosan lemásolódtak:

```
% psql -U contactuser contactdb_slave
contactdb_slave=> select * from contact;
 cid |  name  |   address    |  phonenumber
-----+--------+--------------+----------------
   1 | Joe    | 1 Foo Street | (592) 471-8271
   2 | Robert | 4 Bar Roard  | (515) 821-3831
```

Most csatlakozzunk a `/contactdb/` adatbázishoz és szúrjunk be egy sort:

```
% psql -U contact contactdb
contactdb=> begin; insert into contact
➥(cid, name,
 address, phonenumber) values
 ((select nextval('contact_seq')), 'William',
 '81 Zot Street', '(918) 817-6381'); commit;
```

Ha ismét megvizsgáljuk a `contacdb_slave` adatbázis `contact` tábláját, láthatjuk, hogy a sor itt is megjelenik. Most töröljünk egy sort a `/contactdb/` adatbázisból:

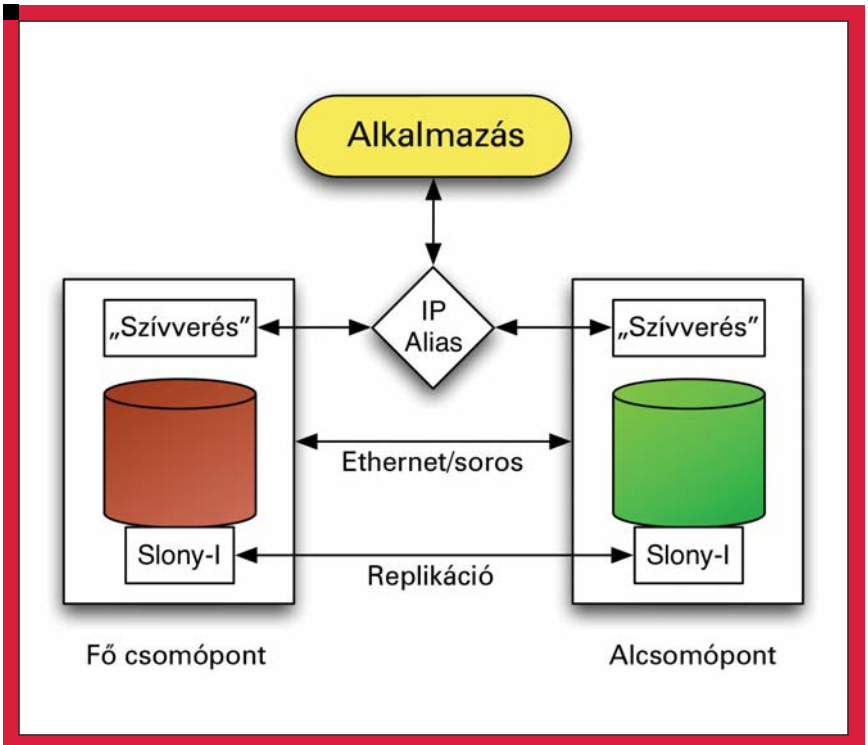
```
contactdb=> begin; delete from contact
 where cid = 2; commit;
```

Akárcsak az előbb, ha megnézzük a `contactdb_slave` adatbázis `contact` tábláját, láthatjuk, hogy a változások hűen követve a sor a szolga csomópontról is törlődött.
A `contactdb` és a `contactdb_slave` kézi összehasonlítgatása helyett egy egyszerű parancsfájl segítségével könnyen automatizálhatjuk ezt a folyamatot. A parancsfájl forrását a *3. listában* olvashatjuk. A parancsfájlt adott időközönként lefuttathatjuk és ellenőrizhetjük, hogy valamennyi csomópont szinkronban fut-e és figyelmeztethetjük az adminisztrátort amennyiben nem ez a helyzet.
Igaz, egyazon rendszeren nincs igazán sok értelme az adatbázis másolatok készítésének, azonban példánk jól szemlélteti milyen egyszerű kialakítani egy ilyen rendszert.
Amennyiben külön gépeken futó csomópontokkal szeretnénk kipróbálni a másolatkészítést, egyszerűen csak a `DB2`, `H1` és `H2` környezetei változókat kell módosítanunk az ***1-3 listákban***. Általában a `DB2` és a `DB1` azonos értékre mutatnak, íg az alkalmazás mindig azonos adatbázisnevet használ. A gazdagép (host) környezetei változókat a két csomópont teljes tartományvére kell állítani. Előfordulhat, hogy arról is meg szeretnénk győződni, hogy a slon folyamatok mindkét gépen futnak. Végül, nem árt, ha valamennyi gép óráját egyeztetjük az ntpd vagy hasonló eszköz segítségével.
Később, ha további táblákat vagy szekvenciákat szeretnénk felvenni az eredeti másolatkészítő rendszerbe, létrehozunk egy új készletet majd beolvasztjuk a `merge set slonik` parancs segítségével. Ezen kívül a készlet megosztásához használhatjuk a `set move table` és a `set move sequence` parancsokat. A ***Slonik Command*** összefoglalóban további információkat találunk ezekről a módszerekről.

Hibakezelés

A mester csomóponton probléma merül fel, például operációs rendszer összeomlás vagy alkatrész meghibásodás miatt, a ***Slony-I*** nem nyújt olyan lehetőséget, amellyel az egyik szolga gépet mesterré léptethetnénk elő. Ez gondot jelenthet, hiszen emberi beavatkozás szükséges a csomópont előléptetéséhez, így magas rendelkezésre állású adatbázisokat követelő szolgáltatásokat nem alapozhatunk rá. Szerencsére számos megoldás létezik amelyeket a ***Slony-I*** rendszerrel ötvözve automatikus hibakezelő rendszert kapunk. Az egyik ezek közül a ***Linux-HA Heartbeat***.
A *2. ábrán* egy mester és egy szolga csomópontot láthatunk ***Ethernet*** és soros kapcsolattal összekötve. Ebben a felállásban a ***Heartbeat*** ezen a két kapcsolaton ellenőrzi a kiszolgáló rendelkezésre állását. Az alkalmazás ***IP*** álnéven keresztül használja a ***PostgreSQL*** adatbázis szolgáltatást, amelyet a ***Heartbeat*** kezdetben a mester csomópontra állít. Amennyiben a ***Heartbeat*** a mester csomópont leállását érzékeli, az ***IP*** álnevet a szolga csomópontra állítja majd végrehajt egy ***slonik*** parancsfájlt, amely a szolga gépet mesterré lépteti elő.

A parancsfájl elég egyszerű. A *4. listában* olvashatjuk a ***slave.example.com*** címen futó szolga csomópontot előléptető parancsfájlt, amely ezáltal átveszi a ***master.example.com*** csomópont által eddig nyújtott adatbázis szolgáltatási feladatokat.
A 4. listában a `failover` ***Slonik*** parancs jelzi, hogy az `id = 1` jelzésű csomópont, azaz a ***master.example.com*** címen futó gép leállt, és a `id = 2` jelzésű csomópont veszi át a kimaradt gép valamennyi készletét. A második parancs a `drop node` feladata az `id = 1` jelzésű csomópont teljes eltávolítása a másolatkészítő rendszerből. Idővel persze szeretnénk a kimaradt csomópontot visszarakni a fürtbe. Ehhez szolgaként kell beállítanunk, majd utasítanunk ***Slony-I*** rendszert, hogy másolja le a hiányzó információkat. Végül az eredeti mester rendszerre visszaváltatáshoz először zároljuk a készletet (`lock set`), kivárjuk valamennyi esemény befejeződését (`wait for event`), átmozgatjuk a készletet az új forráshelyre (`move set`) majd megvárjuk amíg az utolsó parancs kiadásáról megérkezik a visszaigazolás. Ezekről a parancsokról a ***Slonik*** parancsösszefoglalóban találunk további információkat.



2. ábra Heartbeat a szolga csomópontra változtatja az IP álnevet a mester leállása esetén

Összefoglalás

A ***Slony-I*** segítségével viszonylag egyszerű az adatbázis másolatkészítés. A ***Linux-HA Heartbeat*** rendszerrel kombinálva magas rendelkezésre állást biztosíthatunk az adatbázis szolgáltatásainknak. Bár a ***Slony-I*** és a ***Linux HA-Heartbeat*** párosítása vonzó lehetőség, fontos megjegyezni, hogy nem helyettesítheti a kiszolgálóik jó minőségű alkatrészeit.

Igaz, vannak kisebb hiányosságai, hiszen nem tud vázlat változásokat vagy nagy objektumokat másolni, a ***Slony-I*** ma kiváló alternatíva lehet mind az ***rserv***, mind az ***ERServer*** rendszer helyett, sőt tulajdonképpen a ***PostgreSQL*** adatbázisok másolásához ez a leginkább javasolható megoldás.
A szinkronizált több mesteres másolatkészítést is támogató ***Slony-II*** már a tervezőasztalon fekszik.
Végszóként szeretnék köszönetet mondani ***Jan Wieck***-nek, a ***Slony-I*** szerzőjének e cikk lektorálásáért.

Linux Journal 2005. június, 134. szám

4. lista promote.sh

```
#!/bin/bash
CLUSTER=sql_cluster
H1=master.example.com
H2=slave.example.com
U=postgres
DB1=contactdb
DB2=contactdb
su - postgres -c slonik <<_EOF_
cluster name = $CLUSTER;
node 1 admin conninfo = 'dbname=$DB1 host=$H1
➥user=$U';
node 2 admin conninfo = 'dbname=$DB2 host=$H2
➥user=$U';
failover (id = 1, backup node = 2);
drop node (id = 1, event node = 2);
```

KAPCSOLÓDÓ CÍMEK

- www.postgresql.org
- www.slony.info
- developer.postgresql.org/~wieck/slony1/download/slony1-1.0.5.tar.g z
- gborg.postgresql.org/project/slony1/genpage.php?slonik_commands
- gborg.postgresql.org/mailman/listinfo/slony1-general
- www.linux-ha.org/heartbeat

Ludovic Marcotte (ludovic@sophos.ca)

A Montreali Egyetem Informatika Karán szerzett mérnöki diplomát. Jelenleg a montreali központú Inverse, Inc., programtervezője.