O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on **ONLamp.com** (http://www.onlamp.com/)
http://www.onlamp.com/pub/a/onlamp/2004/12/16/slony_install.html
See this if you're having trouble printing code examples

Building and Configuring Slony

by A. Elein Mustain
12/16/2004

Figure 1. Slony mascot

Editor's note: in Introducing Slony, A. Elein Mustain explained the goals of Slony, the replication project for PostgreSQL. This follow-up explains how to install, configure, and start using it.

Building

I am pleased to report that the basic instructions for the download, build, and install of Slony-I release 1.0.5 were perfect.

Slony-I is fairly version independent, but you still need to build it for each PostgreSQL version (7.3 or later) and installation on each machine participating in the replication. The same technique applies when the installations live on different machines.

On one machine, I run several versions of PostgreSQL, each built from source. My plan is to replicate between my 7.4 installation and my 8.0 installation, so I configured and built Slony-I against each of those source trees. That took less than a minute for both.

Repeat these steps for each source tree and installation:

- `./configure -with-pgsourcetree=/local/src/postgresql-`*`version`*
- `make all`
- `sudo make install`

Setting Up Slony-I

Related Reading

This step-by-step reading of instructions will be applied to replicate a small database named `gb`. The plan is to replicate from a PostgreSQL 7.4 installation to a PostgreSQL 8.0 installation, making it possible to upgrade the database.

Slonik is the command-line interface that defines the replication system. There will be Slonik scripts to create,

update, and change the replication cluster for `gb`. There are also tools under development to simplify the creation of replication systems with Slony-I; however, this description will explore the underlying Slonik requirements. It is important to learn the basic Slonik commands.

About the database

`gb` is a simple eight-table database containing issues and articles for my General Bits web site. The database is normalized, and all tables have natural primary keys.

There are several prerequisites:

- Each installation that will participate in replication must have Slony-I built and installed. The Slony-I Project on GBorg gives instructions for building and installing Slony-I. My experience with building Slony-I from source against PostgreSQL 7.4 and 8.0Beta3 was very good. Following the instructions provided a clean and fast builds.
- You need a set of master database tables to replicate and at least one other installation containing the same schema objects. The other installation will be the replica. To achieve this initially, I dumped and restored the schema for the master database on 7.4 into the 8.0 installation:

  ```
  pg_dump -p 5434 -C -s gb | pgsql -p 5430
  ```

  As you can see, these installations are on the same host and have different port numbers.
- The real-time clocks of the servers hosting the nodes must be in sync. I recommend using NTP.
- The pg_hba.conf files on each installation must allow each machine to contact the other.

Slonik

Slonik is a command-line interface for Slony-I. It can connect to the various databases involved in the replication scheme to perform specific actions. It is an independent helper of Slony-I and of PostgreSQL.

The first commands for most Slonik scripts constitute the identity of a group of databases and servers and the connection parameters for accessing each database in the group. Each database and Slony-I connection is a numbered node. The numbers are simply identifiers. The next parameter is the action you wish to process.

Slonik commands work well when they are embedded in shell scripts, as in this example. (The next section covers the commands to identify the cluster and node connection information.)

```
#!/bin/bash
slonik << _END_
cluster name = gb_cluster;
node 1 admin conninfo = 'dbname=db host=localhost port=5432 user=postgres';
node 2 admin conninfo = 'dbname=db host=localhost port=5430 user=postgres';
```

```
...additional nodes...
...slonik commands...
_END_
```

Both Slonik commands and Slony-I have full sets of commands.

Node networks

A node is the combination of a database in an installation and one `slon` process "belonging to" that database. A cluster is a set of nodes cooperating in a replication scheme.

The documentation suggests that all nodes have a path to all other nodes. With only two nodes, this is simple to describe. With more nodes, be sure to include a path to all other nodes, regardless of whether you expect replication to take the paths.

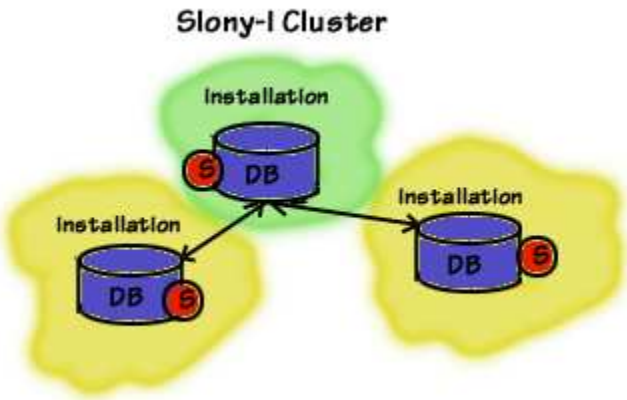

Figure 2. A Slony-I cluster

Our first Slonik script initializes the cluster, defines each node, and defines the paths from each node to every other node. Notice that each node has an identifying number. `init cluster` defines the cluster on the first node. `store node` adds each subsequent node. The user is the slony superuser--in this case, `postgres`. You can choose any privileged user established as the Postgres superuser on each installation.

The `path` is defined by designating one node as a server and the other as a client for messaging. The terminology does not relate to the replicator/replica relationship; instead it references the possible network path. The connection information in each command belongs to the server node. The client's `slon` daemon will connect to the server node using that connection information.

```
#!/bin/bash
#
# 01: Initialize Cluster
#
slonik << _END_
cluster name = gbcluster;
node 1 admin conninfo = 'dbname=gb host=localhost port=5434 user=postgres';
node 2 admin conninfo = 'dbname=gb host=localhost port=5430 user=postgres';

#
# Initialize the cluster and create the second node
#
init cluster (id=1, comment='gb 7.4 5434');
echo 'Initializing gb cluster';
echo 'Node 1 on pgsql74 port 5434 defined';

store node (id=2, comment='gb 8.0 5430');
echo 'Node 2 on pggsql80b port 5430 defined';

#
```

```
# create paths in both directions
#
store path (server=1, client=2, conninfo='dbname=gb host=localhost
     port=5434 user=postgres');
store path (server=2, client=1, conninfo='dbname=gb host=localhost
     port=5430 user=postgres');
echo 'path from server node 1 to client node 2 created.';
echo 'path from server node 2 to client node 1 created.';

_END_
```

Using Slonik's `echo` command can help log and track the commands in any Slonik script.

Listening for events

Events will occur throughout the cluster, and you must tell Slony-I what nodes listen to what nodes to receive these events. The events may be replication information or administrative information that requires propagation throughout the cluster.

In the simple case of two nodes, they listen to each other. In any case, all nodes should be able to listen to all other nodes. The paths' definitions intentionally make this possible.

Specifying the origin identifies which node the receiver is listening for. The origin of an event may or may not provide the event to the receiver; however, the default is to do so. It is possible for node 3 to listen for events initiated on node 1 and have those events provided by node 2 (which, one assumes, is also listening for events from node 1).

In our case, we are having both nodes listen for events on the other, with the events provided by the origin node.

```
#!/bin/bash
#
# 02: Listen
#
slonik << _END_
cluster name = gbcluster;
node 1 admin conninfo = 'dbname=gb host=localhost port=5434 user=postgres';
node 2 admin conninfo = 'dbname=gb host=localhost port=5430 user=postgres';

#
# make the nodes listen on the paths
# in both directions
#
store listen (origin=1, receiver=2, provider=1);
store listen (origin=2, receiver=1, provider=2);

_END_
```

Starting the `slon` processes

Once the nodes can listen to each other for events, start `slon`. Each database participating in the replication needs a `slon` process. Give `slon` a chance to start itself and its threads.

The output in our example goes to two logs, which you can `tail` to watch the activity and look for errors.

`slon` is essentially an event and messaging system. The events involve the replication of data and administrative information to facilitate the replication of data.

```
#!/bin/sh
#
```

```
# 02: Start up Slon processes
#
#
# Start Slony for each node
#
slon gbcluster "dbname=gb user=postgres port=5434 host=localhost" >
     slon_gb_74.out 2>&1 &
slon gbcluster "dbname=gb user=postgres port=5430 host=localhost" >
     slon_gb_80.out 2>&1 &
```

Creating sets

Replication in Slony-I works by subscribing to sets of tables. The set usually should comprise the group of related tables for an application or an entire schema.

To make this work, first define a set and designate the origin for the set. Then add the tables by naming the set ID, the origin of the set, a table ID, the fully qualified table name, and an optional alternate key. Make sure to enter the origin of the set as it was in the set creation (redundantly). All of the tables participating in the replication must have a primary key. If the table does not have one, you can have Slony-I add one for replication purposes only.

Be careful when setting the ID number of a table; it also designates the order in which Slony will lock the tables. This means that master tables should have IDs lower than those of detail tables. The relationship hierarchy of your schema should help you determine the order of the numbers. If the ordering of the table IDs is backward or incorrect, there may be problems with deadlocking the `slon` process or PostgreSQL.

In our example, the `issues` table is the topmost master, followed by `articles`. Each of the other tables are lookup tables for those, so their numbers are higher, accordingly.

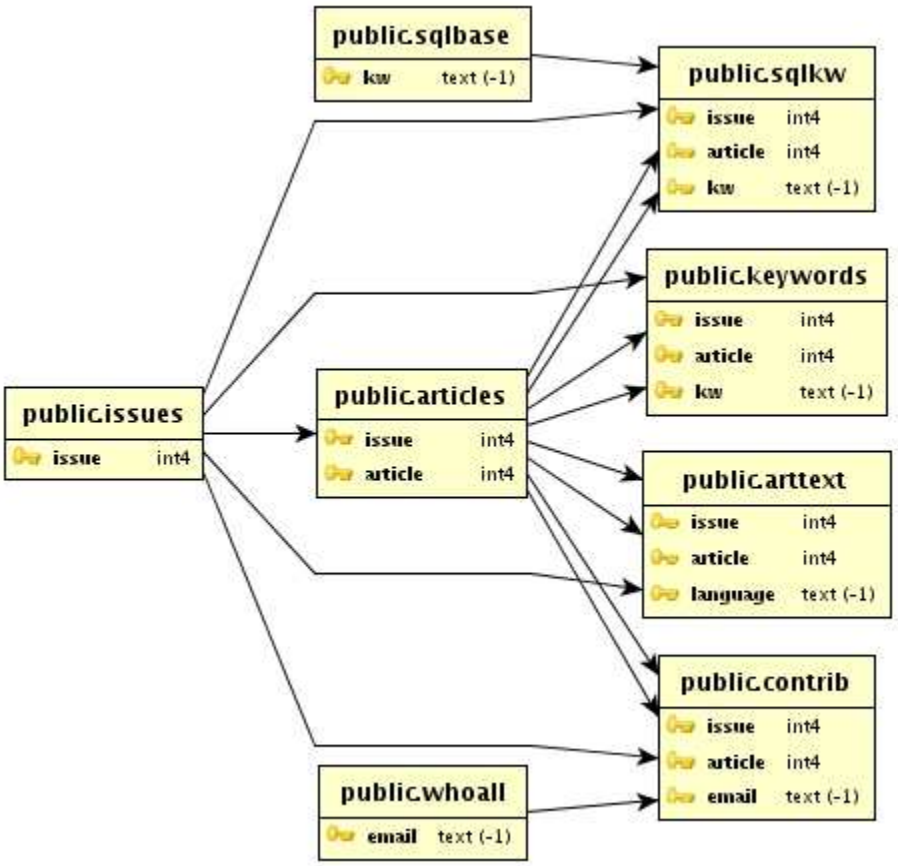

Figure 3. General Bits schema

You can create a set only once, without any active subscribers. To add tables to replication set, create a new set. You can later combine two sets by using Slonik's `MERGE SET` command.

```
#!/bin/sh
#
# 03: Create Set
#
slonik << _END_
#
# Define cluster namespace and node connection information
#
cluster name = gbcluster;
node 1 admin conninfo = 'dbname=gb host=localhost port=5434 user=postgres';
node 2 admin conninfo = 'dbname=gb host=localhost port=5430 user=postgres';

create set (id=1, origin=1, comment='gb tables');
echo 'Set created';
set add table (set id=1, origin=1, id=1,
   full qualified name = 'public.issues', comment='Issues table');
set add table (set id=1, origin=1, id=2,
   full qualified name = 'public.articles', comment='Articles table');
set add table (set id=1, origin=1, id=3,
   full qualified name = 'public.arttext', comment='Article Text table');
set add table (set id=1, origin=1, id=4,
   full qualified name = 'public.sqlbase', comment='Full SQL keywords');
set add table (set id=1, origin=1, id=5,
   full qualified name = 'public.whoall', comment='All contributors');
set add table (set id=1, origin=1, id=6,
   full qualified name = 'public.contrib', comment='Contributors by Article');
set add table (set id=1, origin=1, id=7,
   full qualified name = 'public.keywords', comment='Keywords by Article');
set add table (set id=1, origin=1, id=8,
   full qualified name = 'public.sqlkw', comment='Subset of SQL keywords');
echo 'set 1 of gb tables created';
_END_
```

Subscribing to sets

The nodes can now subscribe to the newly created sets. To subscribe to a set, identify the set, the node that can provide the set, the receiver of the set, and whether the receiver of this set should be able to forward the set to another node. In our case, the origin node of the set is the same as the provider of the set, but for cascading subscriptions that is not necessarily the case.

Even though this replication system has only two nodes, we are saying that the receiving node may forward the set. This is for the case in which we may want to switch masters or add other nodes to the cluster. Here, node 2 is subscribing to set 1. originating on node 1 and provided by node 1.

```
#!/bin/sh
#
# gb_subscribeset.sh
#
slonik << _END_
#
# Define cluster namespace and node connection information
#
cluster name = gbcluster;
node 1 admin conninfo = 'dbname=gb host=localhost port=5434 user=postgres';
node 2 admin conninfo = 'dbname=gb host=localhost port=5430 user=postgres';

subscribe set (id=1, provider=1, receiver=2, forward=yes);
echo 'set 1 of gb tables subscribed by node 2';

_END_
```

Of course, you should assume that these scripts have no typos and that you've run them exactly as intended. Yeah, right. Fortunately, you can recover from mistakes.

Undoing

By this time, you probably have made a typo or two and need to know how to start over. The simplest way of undoing is to start fresh. There are subtler ways of correcting mistakes by updating the underlying tables. However, I don't recommend those unless you have intimate knowledge of the underlying tables.

- To terminate the `slon` processes, list their process IDs and use `kill -TERM` to terminate the oldest of the processes for each node.
- To completely remove all Slony-I definitions from your database, uninstall each node:

```
#!/bin/sh
# gb_uninstallnode.sh
slonik << _END_
#
# Define cluster namespace and node connection information
#
cluster name = gbcluster;
node 1 admin conninfo = 'dbname=gb host=localhost port=5434 user=postgres';
node 2 admin conninfo = 'dbname=gb host=localhost port=5430 user=postgres';
echo 'Cluster defined, nodes identified';

#
# UnInstall both nodes
#
uninstall node (id=1);
uninstall node (id=2);
echo 'Nodes 1 and 2 Removed';
_END_
```

NOTE: `UNINSTALL NODE` removes all definitions, and you must start cleanly after that.

Slony-I schema

The underlying tables for Slony-I are fairly straightforward. The cluster name is the name of the schema in the database in which the Slony tables reside. (Use `set search_path` in `psql`.) You can verify your commands to add nodes, listens, paths, and so on by examining these tables. It also looks tempting to "fix" things by just changing the underlying tables. Resist doing so, however. Use Slonik so that it can trigger the appropriate events to perform the updates in an orderly fashion across all nodes.

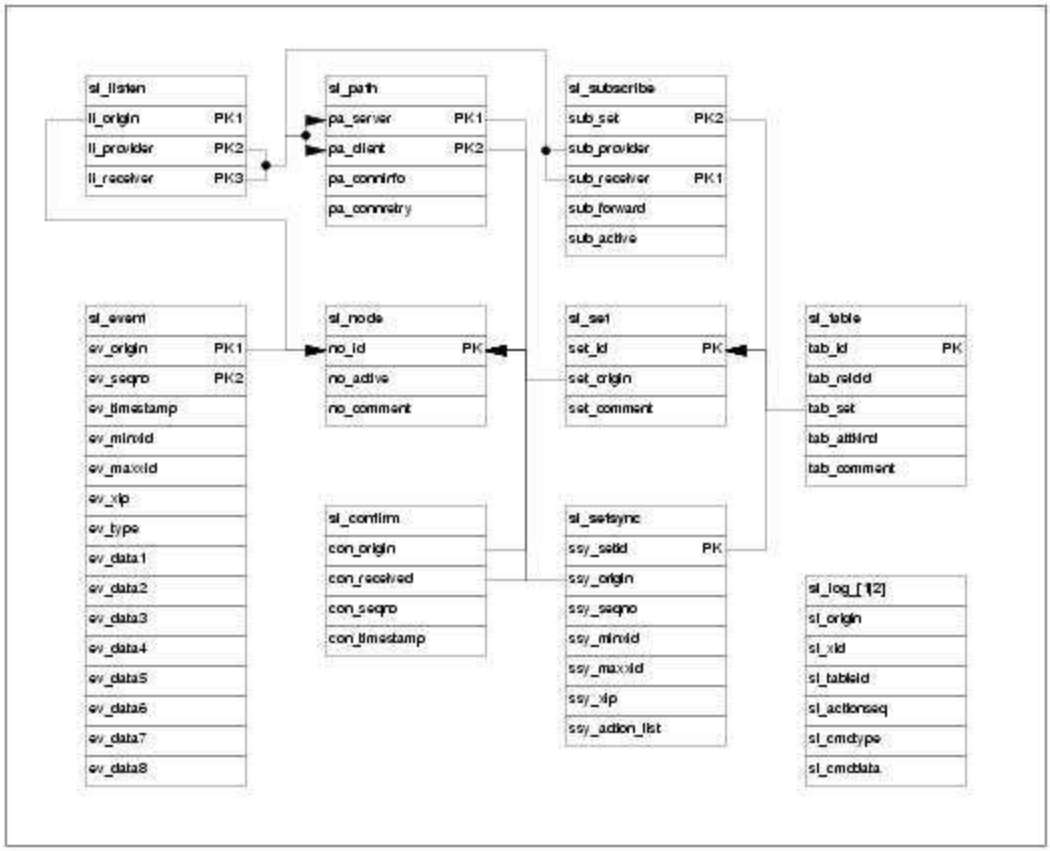

Figure 4. Slony schema

References

- General Bits Slony Articles on Tidbits
- The Slony-I Project documentation on GBorg
- Slonik Commands
- Jan Wieck's Original Slony-I Talk and Scripts July 2004 in Portland, Oregon, sponsored by Affilias Global Registry Services
- Information from IRC #slony on freenode.net
- Mailing List: Slony1-general@gborg.postgresql.org

A. Elein Mustain has more than 15 years of experience working with databases, 10 of those working exclusively with object relational database systems.

Return to ONLamp.com.